Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: Автоматизація процесу створення інтер'єру двокімнатної квартири в 3ds Max із використанням Python та постобробкою у Photoshop.

Виконала: студентка 4 курсу групи КН-41
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Андрусишин М.Я.
(прізвище та ініціали)

Керівник Сторожук О.Л.
(прізвище та ініціали)

Рецензент Душин С.І.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН

Борецька І. Б.
"10" червня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Андрусишин Марії Ярославівні
(прізвище, ім'я, по батькові)

1. Тема роботи *Автоматизація процесу створення інтер'єру двокімнатної квартири в 3ds Max із використанням Python та постобробкою у Photoshop*
керівник роботи *Сторожук Олександр Леонідович, доцент, канд. техн. наук*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *15.11.2024року № С-882*

2. Термін подання студентом роботи *10.06.2025р.*

3. Вихідні дані до роботи: Постановка задачі та її формалізація. Основні параметри та вимоги до розробки інструментів на Python для автоматизації створення інтер'єрної сцени в 3ds Max та обробки зображень у Photoshop.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
1) Стан проблемної області;
2) Інформаційне та математичне забезпечення;
3) Програмне та технічне забезпечення;
4) Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація до диплому.

6. Дата видачі завдання *18.11.2024р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	*Огляд літературних джерел*	*18.11.2024р. – 16.12.2024р.*	*Виконано*
2	*Розділ 1. Стан проблемної області.*	*16.12.2024р. – 01.01.2025р.*	*Виконано*
3	*Розділ 2. Інформаційне та математичне забезпечення.*	*01.01.2025р. – 28.01.2025р.*	*Виконано*
4	*Розділ 3. Програмне та технічне забезпечення.*	*28.01.2025р. – 10.05.2025р.*	*Виконано*
5	*Аналіз отриманих результатів та написання висновків. Оформлення дипломної роботи.*	*10.05.2025р. – 05.06.2025р.*	*Виконано*
6	*Здача пояснювальної записки на перевірку керівнику, виправлення помилок та здача роботи рецензенту.*	*10.06.2025р.*	*Виконано*

Студент _____ **Андрусишин М. Я.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Сторожук О. Л.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 56 сторінок пояснювальної записки, 32 рисунки, 2 додатки, 16 джерел.

У дипломній роботі створено візуалізацію інтер’єру двокімнатної квартири у світлих, ніжних тонах. Для оптимізації процесу роботи над сценою розроблено два скрипти на Python у середовищі 3ds Max: скрипт генерації параметричних килимів з можливістю налаштування форми, розмірів, текстури та декоративних елементів, а також скрипт заміни тимчасових box-об’єктів на деталізовані 3D-моделі з автоматичним збереженням їхнього розташування, масштабу та орієнтації. Результати доцільно використати для підвищення продуктивності при створенні інтер’єрної візуалізації, в освітніх цілях та в професійному середовищі дизайнерів та 3D-художників.

Ключові слова: 3dsMax, Python, візуалізація інтер’єру з Corona Renderer, автоматизація побудови сцени, скрипти, PySide2 інтерфейс.

ABSTRACT

The thesis contains 56 pages of explanatory notes, 32 figures, 2 appendices, 16 sources.

The thesis created a visualization of the interior of a two-room apartment in light, subdued colors. To optimize the process of working on the scene, two Python scripts were developed in the 3ds Max environment: a script for generating parametric carpets with the ability to customize the shape, size, texture, and decorative elements, as well as a script for replacing temporary box objects with detailed 3D models with automatic saving of their location, scale, and orientation. The results are useful for increasing productivity when creating interior visualizations, for educational purposes and in the professional environment of designers and 3D artists.

Keywords: 3dsMax, Python, interior visualization with Corona Renderer, scene construction automation, scripts, PySide2 interface.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити візуалізацію інтер’єрної сцени та набір інструментів, які автоматизують частину процесу її створення в 3ds Max і оптимізують підготовку сцени до фінального рендерингу та подальшої обробки зображення у Photoshop.

Проєкт має включати такі основні компоненти:

- скрипт мовою Python, який дозволяє генерувати килими з можливістю налаштування форми, розмірів, матеріалів і декоративних елементів (китиць) за допомогою параметрів;
- скрипт мовою Python , який забезпечує заміну простих геометричних об'єктів (box) на деталізовані 3D-моделі, забезпечуючи збереження початкового розташування, масштабу та орієнтації об’єкта під час заміни;
- реалізацію інструментів засобами Python у середовищі 3ds Max із використанням внутрішніх можливостей MaxScript/Python API;
- фінальну інтер’єрну сцену, створену з використанням усіх вище описаних інструментів, яка представляє собою візуалізацію двокімнатної квартири, виконаної в ніжних, світлих тонах. Для рендеру і налаштування матеріалів використовувати Corona Renderer;
- постобробку фінальних зображень у Photoshop, включно з автоматизацією типових етапів редагування (корекція кольору, покращення контрасту, різкості тощо).

Пояснювальну записку оформити відповідно до методичних вказівок.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 7
ВСТУП 8
РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ 10
1.1 Актуальність теми 10
1.2 Огляд існуючих підходів і обґрунтування вибраного підходу 11
1.2.1 Вибір середовища візуалізації 11
1.2.2 Вибір мови програмування для написання скриптів 14
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ 16
2.1 Побудова дерева проблем та дерева цілей 16
2.2 Специфікація вимог до програмного забезпечення 19
2.2.1 Характеристики продукту 19
2.2.2 Середовище функціонування 21
2.2.3 Характеристика системи 22
2.3 Структурна схема системи з урахуванням інформаційних потоків 24
2.4 Математичне забезпечення 28
2.4.1 Перевірка введених даних 28
2.4.2 Автоматизоване розташування китиць 31
2.4.3 Заміна умовного об’єкта на 3D-модель 33
РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ 36
3.1 Проектування інтерфейсу 36
3.2 Опис програмної реалізації 38
3.2.1 Налаштування середовища розробки 38
3.2.2 Реалізація генерації параметричних килимів 41
3.2.3 Заміна примітивів на деталізовані моделі 46
3.2.4 Створення інтер’єрної сцени та фінальна обробка 48
ВИСНОВКИ 56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 57
ДОДАТКИ 59
ДОДАТОК А 59
ДОДАТОК Б 71

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface – інтерфейс прикладного програмування; набір інструментів для взаємодії з програмним забезпеченням.

GUI – Graphical User Interface – графічний інтерфейс користувача.

3ds Max – Autodesk 3ds Max – програмне забезпечення для тривимірного моделювання, візуалізації та анімації.

Mtl – Material – матеріал, що використовується для текстурування об’єктів у сцені.

Corona Renderer – фотореалістичний рендер-движок, інтегрований у 3ds Max.

HDRI – High Dynamic Range Image – зображення з розширеним динамічним діапазоном, використовується для освітлення сцен.

DOF – Depth of Field – глибина різкості, ефект у візуалізації для симуляції розмиття.

pymxs – Python MaxScript – модуль для взаємодії з 3ds Max API за допомогою мови Python.

UI – User Interface – користувацький інтерфейс, спосіб взаємодії користувача з програмою.

ВСТУП

Актуальність

У сучасному світі 3D-моделювання займає важливу роль у галузі архітектурної візуалізації, дизайну інтер’єрів та комп’ютерної графіки в цілому. Якісна візуалізація є основою для успішного представлення ідеї, оскільки дозволяє як клієнтам, так і дизайнерам отримати чітке уявлення про вигляд кінцевого простору ще до початку його реалізації. Фотореалістичні зображення здатні точно передавати атмосферу приміщення, стиль, колірні поєднання, розташування меблів і загальне відчуття простору, що є важливим для прийняття правильних рішень під час розробки дизайну.

Створення повноцінної інтер’єрної сцени складається з багатьох етапів, таких як моделювання, налаштування матеріалів, текстурування, розміщення об'єктів та підготовка сцени до рендерингу. Всі ці завдання вимагають значної кількості часу та уваги до деталей. Це особливо критично, коли час обмежений, а вимоги до якості кінцевого зображення високі. Тому кожен етап роботи, від моделювання до остаточної обробки, відіграє важливу роль у досягненні бажаного результату.

Автоматизація рутинних процесів дозволяє значно скоротити час роботи, зменшити ймовірність помилок та знизити навантаження на систему. При цьому, автоматизація не позбавляє дизайнера контролю над фінальним виглядом сцени, а навпаки дає більше можливостей для вдосконалення візуального результату, що є ключовим для досягнення високої якості візуалізації.

Об’єкт дослідження – процес створення інтер’єрної 3D-візуалізації житлового приміщення (двокімнатної квартири) у середовищі 3ds Max, включаючи рендеринг у Corona Renderer і базову постобробку в Adobe Photoshop.

Предмет дослідження – розробка та застосування засобів автоматизації в середовищі 3ds Max за допомогою Python-скриптів з метою підвищення ефективності створення інтер’єрної 3D-візуалізації та зменшення трудомісткості повторюваних операцій.

Метою дипломної роботи є дослідження, проектування та реалізація програмних рішень для автоматизації основних етапів створення інтер’єрної 3D-візуалізації двокімнатної квартири у середовищі 3ds Max. Робота спрямована на підвищення ефективності та оптимізацію повторюваних процесів за допомогою програмної автоматизації, що відповідає сучасним вимогам до розробки прикладних засобів у галузі комп’ютерної графіки.

Для досягнення поставленої мети у дипломній роботі потрібно вирішити наступні **завдання**:

- розробити скрипт мовою Python для генерації килимів, з можливістю налаштування форми, розмірів, матеріалів та декоративних елементів(китиць);
- розробити скрипт мовою Python для заміни простих об'єктів, типу box, на деталізовані 3D-моделі з урахуванням їх розташування, масштабу та орієнтації;
- побудувати інтер’єрну сцену двокімнатної квартири, виконати її рендеринг за допомогою Corona Renderer та провести базову постобробку результату у Photoshop для підвищення візуальної якості.

Практичне значення

Результати роботи можуть бути використані у професійній діяльності дизайнерів інтер’єру, 3D-візуалізаторів, а також для навчальних цілей при вивченні процесів автоматизації у 3ds Max. Запропоновані інструменти автоматизації дозволяють значно скоротити час на повторювані дії, створення сцени та підвищити якість кінцевого результату.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Актуальність теми

Сфера 3D-моделювання та візуалізації активно розвивається, що пов’язано зі зростанням інтересу до максимально правдоподібного відтворення інтер’єрів, архітектурних об’єктів, рекламних матеріалів та віртуальних середовищ. Якісні візуалізації дозволяють ще до початку робіт побачити майбутній вигляд простору, тому вони стали незамінним інструментом для представлення дизайнерських рішень. З кожним роком підвищуються вимоги до точності, рівня деталізації, швидкості створення та загального вигляду таких зображень. Сьогодні 3D-візуалізація є важливою частиною комунікації між дизайнерами, архітекторами, забудовниками та клієнтами.

Однією з найпотужніших програм для створення візуалізацій є 3ds Max, що забезпечує широкі можливості для побудови моделей, нанесення текстур, налаштування сцени та рендерингу [1]. Однак навіть із цим інструментом процес оформлення інтер’єру залишається досить складним і вимагає багато кроків. Часто значна частина з них виконується вручну, що уповільнює роботу і збільшує ризик помилок, особливо при сислих термінах та великих обсягах проєкту.

Автоматизація окремих етапів допомагає прискорити процес, зменшити навантаження на систему і зосередити увагу на творчих завданнях. Завдяки мові програмування Python можна створювати власні інструменти для 3ds Max, які значно полегшують рутинну роботу, зокрема з повторюваними елементами такими як меблі, декор чи килими [2].

У такому підході поєднання моделювання інтер’єру двокімнатної квартири з розробкою автоматизованих скриптів є актуальним і доцільним. Це дозволяє дизайнеру ефективно співпрацювати з клієнтом, швидко показати візуальний результат ще до початку реалізації, а також підвищити якість і результативність проєкту.

1.2 Огляд існуючих підходів і обґрунтування вибраного підходу

1.2.1 Вибір середовища візуалізації

Autodesk 3ds Max є одним із найпопулярніших та найбільш професійних інструментів у сфері тривимірної графіки, особливо у галузях архітектурної візуалізації, дизайну інтер'єрів, створення віртуальних турів і рекламних проєктів. Його висока популярність зумовлена потужними можливостями моделювання, широким спектром інструментів для рендерингу та сильною підтримкою професійної спільноти [3, 16].

Ключові переваги 3ds Max:

- Розширені можливості моделювання. Програма підтримує як полігональне, так і параметричне моделювання, що дозволяє створювати як прості форми, так і надзвичайно деталізовані сцени для високоякісної візуалізації.
- Високі можливості роботи з текстурами та матеріалами. Матеріальний редактор 3ds Max підтримує складні шейдери, фізично коректні матеріали (PBR) і процедурні текстури, що забезпечує створення реалістичних поверхонь і матеріалів.
- Програма має інтеграцію з найпопулярнішими рендер-двигунами, такими як Corona Renderer, V-Ray і Arnold, що дозволяє отримувати фотореалістичні результати з точним налаштуванням освітлення та матеріалів.
- Можливості автоматизації 3ds Max підтримує використання MaxScript, Python і C++ API, що дає змогу автоматизувати рутинні завдання, створювати власні інструменти і розширювати функціонал під специфічні потреби.
- Широкий вибір плагінів (наприклад, Forest Pack, RailClone, FloorGenerator) дає змогу значно розширити стандартні можливості програми.
- Велика кількість форумів, навчальних матеріалів, відеоуроків та офіційної документації робить процес вивчення програми доступним, а вирішення складних питань – швидким і зручним.

- Гнучка робота зі сценами 3ds Max дозволяє ефективно працювати з об’ємними сценами, складними ієрархіями об’єктів, багаторівневим налаштуванням анімацій, освітлення та параметрів рендерингу.

Порівняння з основними конкурентами:

1. Blender

Blender – це потужне, безкоштовне та відкрите програмне забезпечення, яке пропонує широкий набір інструментів для моделювання, анімації, рендерингу та текстурування. Blender постійно розвивається, має велику і активну спільноту та безліч плагінів [4]. Проте для професійної архітектурної візуалізації він має кілька недоліків:

- Blender пропонує меншу кількість готових бібліотек для архітектурних моделей та текстур, що ускладнює створення детальних та складних сцен;
- інтеграція з рендер-рушіями, які активно використовуються в архітектурній візуалізації є менш зручною і безшовною;
- Blender поки не отримав такого широкого визнання серед професійних користувачів у сфері архітектурної візуалізації.

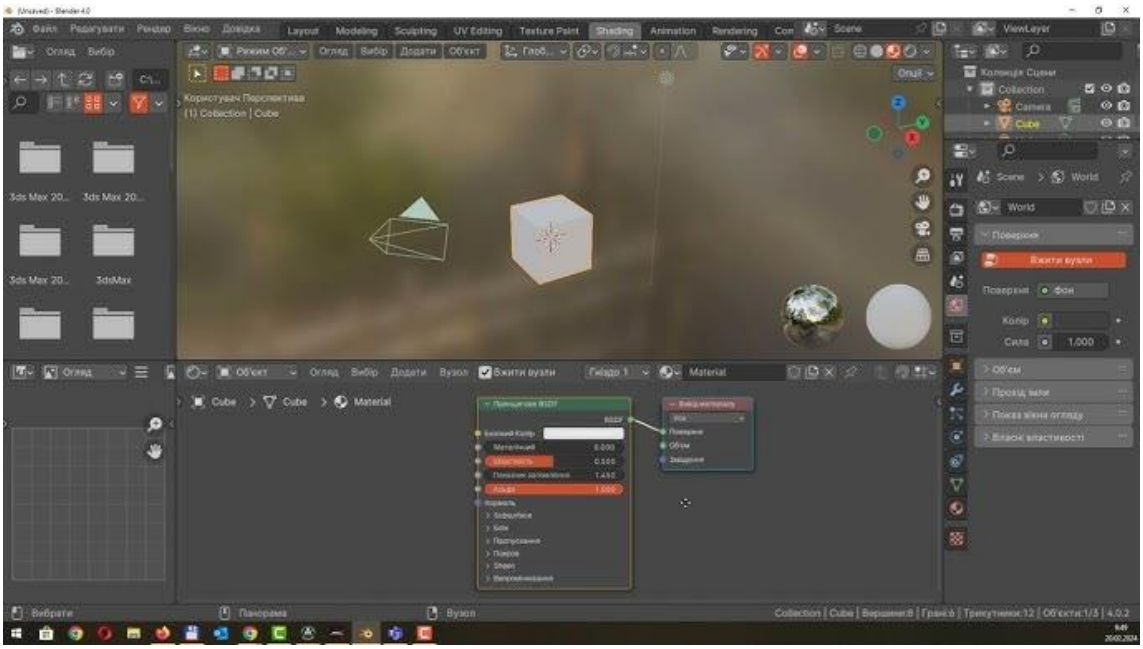


Рисунок 1.1 – Інтерфейс Blender

2. Cinema 4D

Cinema 4D – це професійна програма, яка здобула популярність серед моушн-дизайнерів, аніматорів та рекламних агентств завдяки зручності у створенні анімацій та графіки[5]. Проте для архітектурної візуалізації Cinema 4D має кілька суттєвих обмежень:

- Cinema 4D не має таких потужних інструментів для фотореалістичного рендерингу, як 3ds Max з рендер-рушіями Corona або V-Ray, що обмежує її здатність створювати реалістичні інтер'єри та екстер'єри;
- Cinema 4D більше орієнтована на анімацію і моушн-дизайн, тому вона може бути менш зручною для створення складних статичних візуалізацій інтер'єрів або архітектурних сцен.

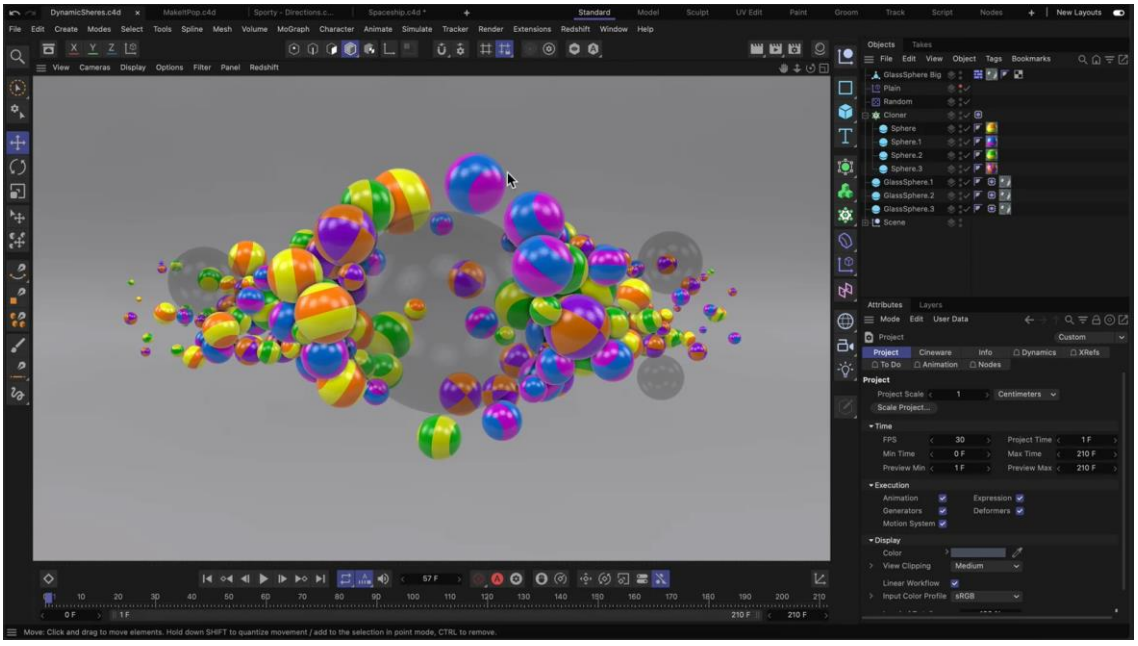


Рисунок 1.2 – Інтерфейс Cinema 4D

3. SketchUp

SketchUp – популярна програма для швидкого створення ескізних 3D-моделей у архітектурі та дизайні [6]. Однак вона має кілька обмежень, що знижують її ефективність для створення високоякісних фотореалістичних візуалізацій:

- SketchUp не підтримує потужні рендери, такі як V-Ray або Corona, що обмежує якість фінальних зображень, особливо для складних сцен;

- SketchUp може мати проблеми з обробкою великих і деталізованих проєктів, що є важливим для більшості архітектурних студій.

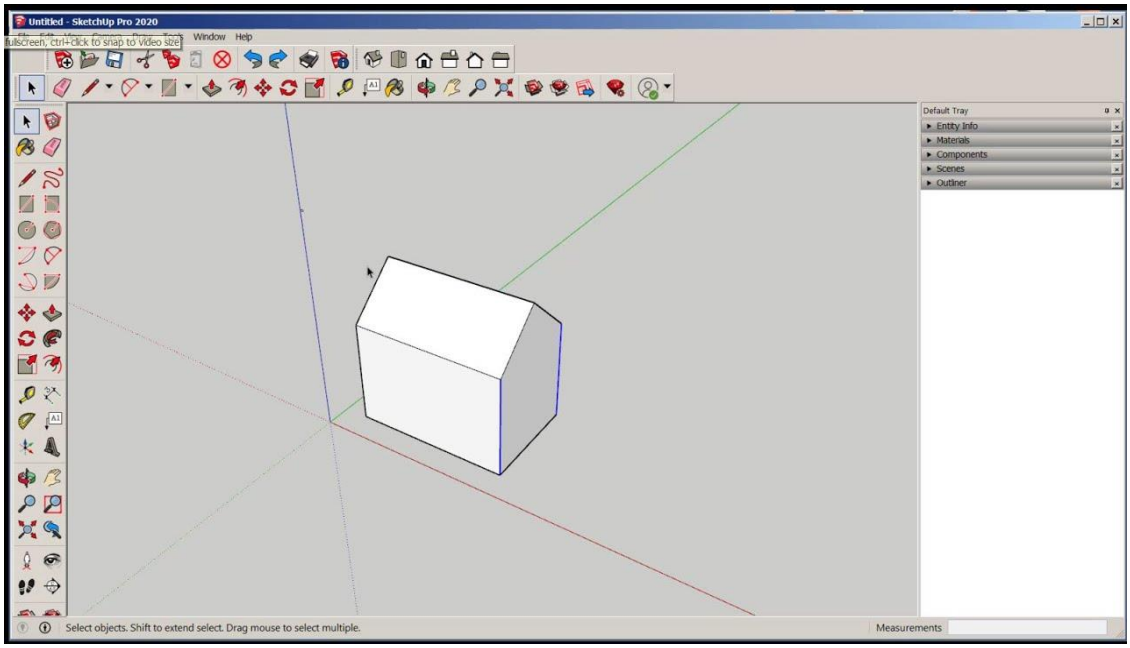


Рисунок 1.3 – Інтерфейс SketchUp

Таким чином, 3ds Max залишається провідним програмним забезпеченням для професійної архітектурної візуалізації завдяки своїм потужним інструментам для моделювання, текстурування та рендерингу, а також можливості інтеграції з такими рендер-рушіями, як V-Ray і Corona.

1.2.2 Вибір мови програмування для написання скриптів

Для розробки інструментів автоматизації в 3ds Max у межах цієї роботи була обрана мова програмування Python. Таке рішення пояснюється поєднанням простоти, гнучкості та великого набору можливостей, які Python надає розробникам [2, 7, 15].

Однією з основних переваг Python є його легкий для розуміння синтаксис, що дозволяє швидко освоїтися навіть тим, хто не має ґрунтовних знань у сфері програмування. У контексті 3D-дизайну це надзвичайно важливо, адже багато дизайнерів та художників шукають способи автоматизувати повсякденні завдання без глибокого занурення в технічні нюанси.

Ще одним вагомим плюсом є велика кількість готових бібліотек і рішень, що доступні для Python. Це суттєво полегшує і пришвидшує процес розробки: стає можливим реалізувати різноманітні функції для роботи зі сценами, файлами, мережами, а також створювати власні інтерфейси безпосередньо в 3ds Max.

Крім того, компанія Autodesk офіційно підтримує Python через спеціалізований API, що відкриває прямий доступ до об'єктів сцени, матеріалів, анімаційних даних та параметрів рендерингу. Завдяки цьому створення скриптів стає значно простішим і зручнішим у порівнянні з використанням інших мов програмування.

Також доцільно розглянути альтернативні варіанти мов програмування, які могли б бути застосовані.

1. MaxScript

MaxScript – це внутрішня мова скриптів для 3ds Max, що дозволяє маніпулювати об'єктами сцени та їхніми властивостями. Однак її синтаксис трохи застарілий і не завжди зручний для реалізації великих чи складних рішень. Крім того, можливості взаємодії з зовнішніми системами через MaxScript досить обмежені, що знижує його гнучкість у порівнянні з Python [8].

2. C++

C++ забезпечує максимальну швидкодію та повний контроль над усіма аспектами програми. Проте розробка навіть простих функцій на цій мові вимагає значних часових витрат, глибоких знань архітектури програм і проходження складного процесу компіляції та відладки. Для задач середньої складності застосування C++ є невиправдано складним і затратним [9].

3. MEL

MEL є мовою скриптів для пакету Maya і не має підтримки в середовищі 3ds Max. Отже, використання MEL для цієї роботи неможливе. Навіть у Maya ця мова поступається Python у гнучкості, а також має обмежений набір інструментів і бібліотек.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Побудова дерева проблем та дерева цілей

Щоб створити справді ефективну систему автоматизації в середовищі 3ds Max, спершу потрібно чітко окреслити всі наявні труднощі та зрозуміти, як із ними можна впоратися. Це можна зробити за допомогою побудови дерева проблем – інструменту, який дає змогу наочно відобразити основні перешкоди, що впливають на якість і швидкість проєктування інтер'єрних сцен. Поруч із ним формується дерево цілей, яке допомагає спланувати конкретні кроки для досягнення бажаного результату [11].

Дерево проблем (рис. 2.1) – це спосіб впорядкувати й логічно зв'язати між собою всі труднощі, що виникають у процесі роботи. Воно дозволяє чітко побачити, з чого почати вирішення проблем, які з них є найкритичнішими, а які можна поки відкласти. Таке структурування дає змогу ефективніше розподілити зусилля і ресурси на розв'язання конкретних задач.

Головна проблема: Процес створення інтер'єрної візуалізації в 3ds Max є повільним, трудомістким і не завжди результативним.

Основна проблема 1. Створення однотипних об'єктів вручну.

Причина 1.1: Відсутність інструментів для автоматичного генерування стандартних елементів.

Причина 1.2: Повторення одних і тих самих дій без застосування автоматизації.

Основна проблема 2. Недостатня ефективність підготовки сцени до фінального рендеру.

Причина 2.1: Неоптимальні налаштування освітлення, камер і матеріалів.

Причина 2.2: Керування світлом виконується вручну, без використання LightMix чи шаблонів.

Основна проблема 3. Помилки у геометричних розрахунках через людський фактор.

Причина 3.1: Помилки при введенні значень або створенні об'єктів вручну.

Причина 3.2: Відсутність перевірки точності розмірів і співвідношення параметрів.

Основна проблема 4. Обмежений функціонал стандартних інструментів.

Причина 4.1: Базові інструменти 3ds Max не мають достатньо можливостей для автоматизації типових процесів.

Причина 4.2: Без створення власних скриптів кастомізація функцій є майже неможливою.

Основна проблема 5. Повільність ручного проєктування.

Причина 5.1: Необхідність вручну моделювати кожен елемент сцени значно уповільнює роботу.

Причина 5.2: Відсутність підходів до повторного використання готових шаблонів або процедурних моделей.

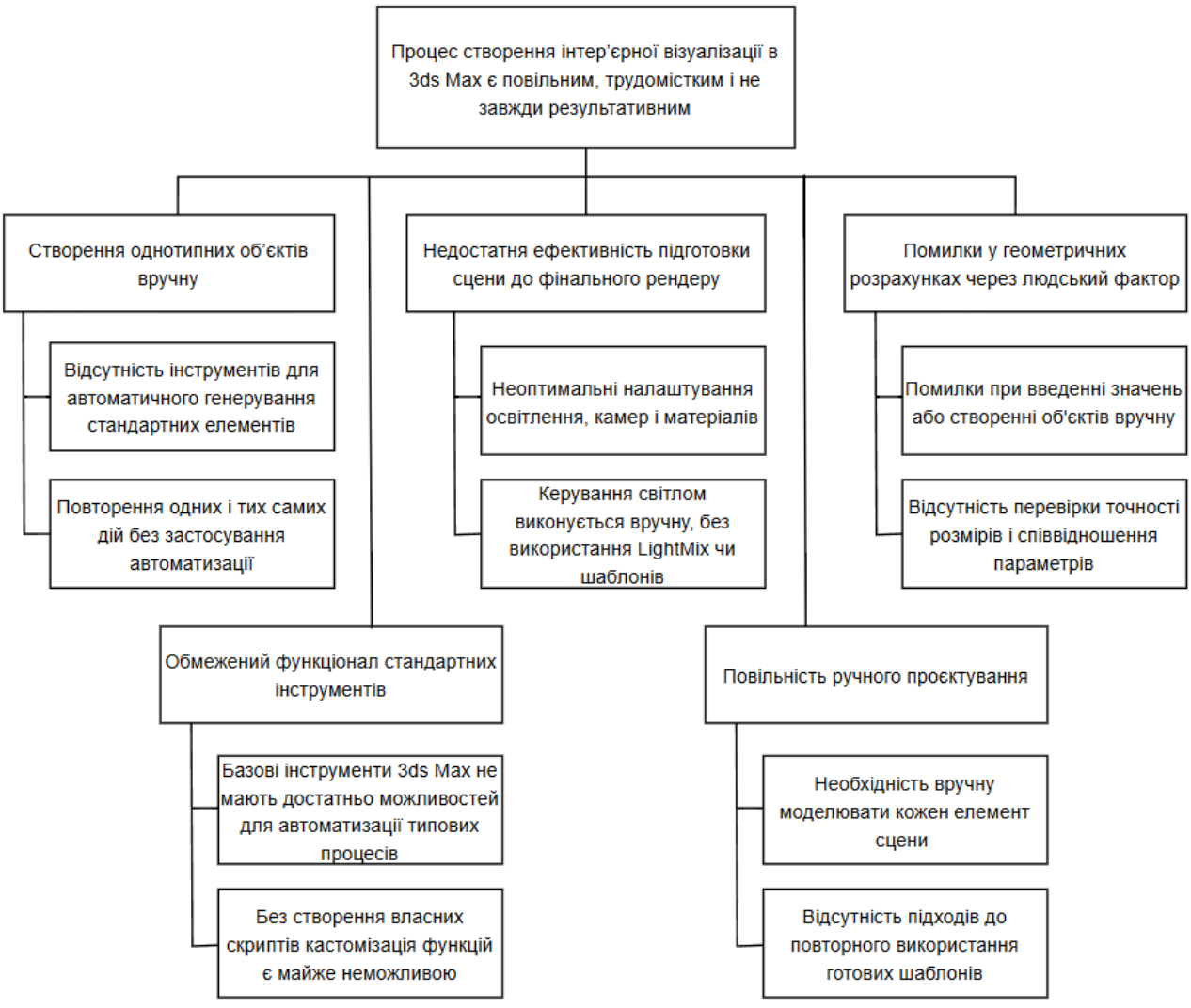


Рисунок 2.1 – Дерево проблем

Дерево цілей (рис. 2.2) є логічним продовженням дерева проблем. Якщо перше допомагає виявити, що саме заважає ефективній роботі, то друге – демонструє, як можна це змінити на краще. Побудова дерева цілей починається з головного завдання, до якого ми прагнемо, а далі розкладається на окремі кроки й дії. Такий підхід дозволяє краще зрозуміти шляхи досягнення результату й оцінити, наскільки вони практичні. Це також допомагає правильно організувати процес, уникнути зайвих труднощів і впевнено рухатись до мети, розуміючи всі етапи роботи.

Головна мета: Поліпшення та автоматизація процесу створення інтер'єрної 3D-візуалізації в 3ds Max для підвищення якості результатів і скорочення часу, що витрачається на рутинні дії.

Основна ціль 1. Автоматичне створення повторюваних елементів декору.

Завдання 1.1: Розробити параметричний генератор інтер'єрних килимів.

Завдання 1.2: Додати функції налаштування форми, розмірів, текстур і декоративних деталей.

Завдання 1.3: Створити простий та зрозумілий інтерфейс для користувачів.

Основна ціль 2. Спрощення процесу заміни тимчасових об'єктів на готові 3D-моделі.

Завдання 2.1: Створити скрипт для автоматичної підстановки простих об'єктів більш детальними моделями.

Завдання 2.1: Забезпечити збереження масштабу, розташування та орієнтації під час такої заміни.

Основна ціль 3. Підвищення точності роботи та зменшення кількості помилок.

Завдання 3.1: Додати механізм перевірки допустимості введених параметрів.

Завдання 3.2: Реалізувати автоматичну перевірку правильності даних для запобігання помилкам.

Основна ціль 4. Розширення стандартного функціоналу 3ds Max

Завдання 4.1: Використати можливості API мови Python для створення нових скриптів і розширень.

Завдання 4.2: Розробити інструменти для автоматизації операцій, які не покриваються стандартними засобами 3ds Max.

Основна ціль 5. Досягнення високої якості візуального результату.

Завдання 5.1: Побудувати сцену інтер’єру двокімнатної квартири, використовуючи всі реалізовані інструменти.

Завдання 5.2: Виконати постобробку у Photoshop для покращення візуального ефекту.

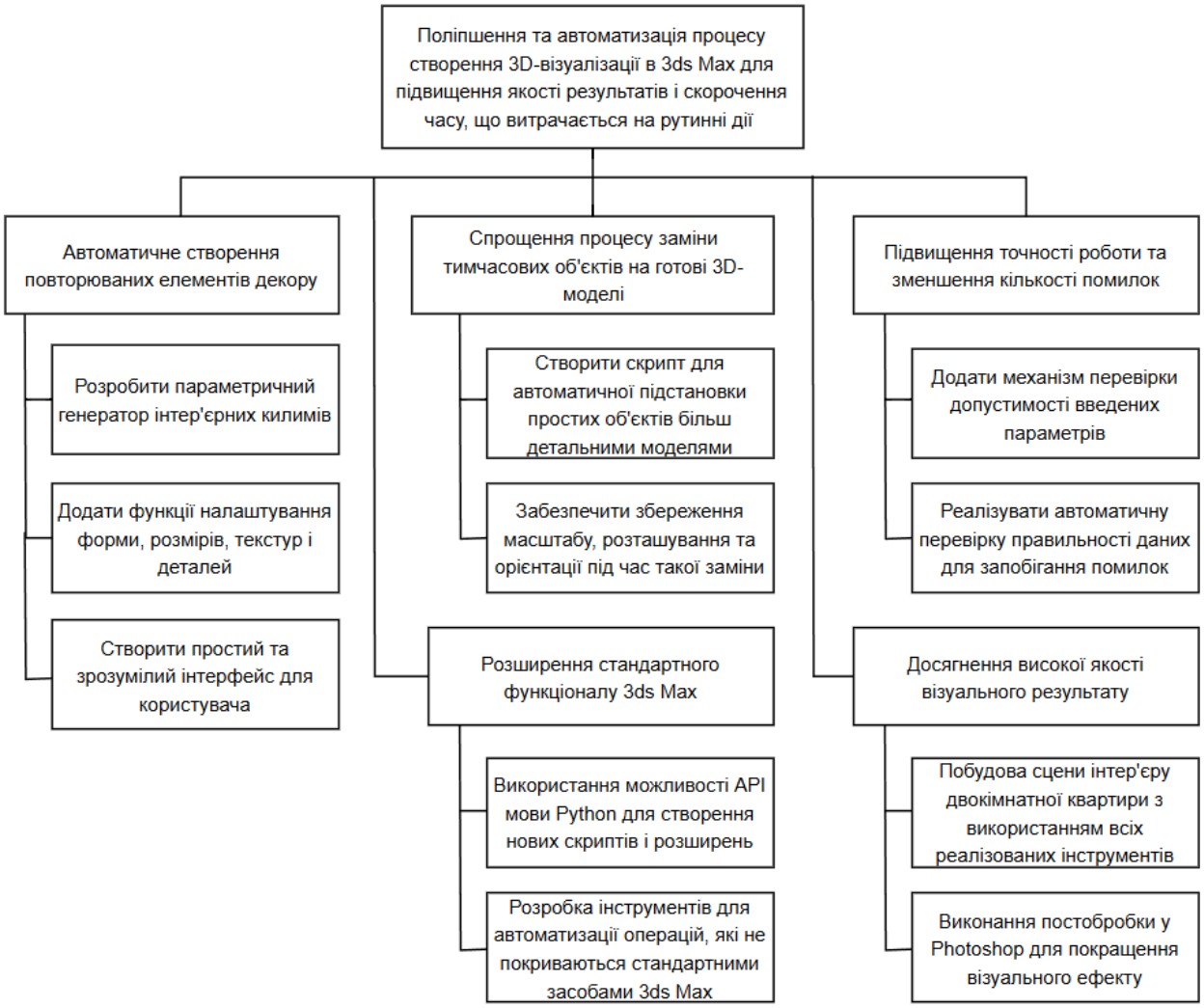


Рисунок 2.2 – Дерево рішень

2.2 Специфікація вимог до програмного забезпечення

2.2.1 Характеристики продукту

Програмне забезпечення розробляється для автоматизації процесу створення 3D-візуалізацій інтер’єрів у 3ds Max. Головною метою є спрощення рутинних завдань, зменшення ймовірності помилок та прискорення роботи над

сценами. Цей інструмент дозволить швидко генерувати повторювані об'єкти, автоматично заміняти тимчасові моделі на завершені 3D-версії, перевіряти коректність введених даних і розширювати стандартні можливості 3ds Max. Інтерфейс має бути у вигляді зручної панелі або вікна в межах 3ds Max, з полями для введення параметрів і кнопками для запуску скриптів.

Функціональні вимоги:

1. Автоматизоване створення параметричних об’єктів:

- формування інтер’єрних килимів з можливістю змінювати форму, габарити, матеріали та декоративні деталі;
- використання заздалегідь підготовлених шаблонів для стилізації.

2. Автоматизована заміна тимчасових об’єктів:

- підміна базових геометричних заготовок на повноцінні деталізовані 3D-моделі;
- збереження розташування, масштабування та орієнтації об’єктів у сцені.

3. Інструменти для перевірки даних:

- контроль правильності введених користувачем параметрів;
- виявлення помилок у геометрії, що можуть виникнути через ручне проєктування.

4. Розширення можливостей 3ds Max:

- підключення до MaxScript та Python API для написання власних рішень;
- опція зберігати й повторно використовувати користувацькі шаблони для пришвидшення роботи.

5. Інтерфейс для користувача:

- зрозуміла форма для введення параметрів під час створення об’єктів;
- можливість візуально оцінити результат без необхідності ручної корекції.

Нефункціональні вимоги:

1. Продуктивність: забезпечення швидкої обробки даних і виконання операцій при стандартному рівні навантаження.

2. Сумісність: підтримка 3ds Max версій 2020–2024 та майбутніх оновлень.

3. Зручність використання: мінімізація необхідності навчання для початкового користувача і простий зрозумілий інтерфейс.

4. Безпека: обробка помилок при введенні недопустимих параметрів і уникнення аварійного завершення сеансу 3ds Max під час роботи скриптів.

2.2.2 Середовище функціонування

Програмне забезпечення:

1. Autodesk 3ds Max (версія 2022 або новіша) з підтримкою MaxScript або Python API.

2. Плагін Corona Renderer для рендерингу.

3. Adobe Photoshop для постобробки.

Операційна система:

1. Windows 10 або Windows 11 (64-bit).

Обладнання:

1. Процесор: не нижче Intel Core i5 / AMD Ryzen 5.

2. Оперативна пам’ять: мінімум 16 ГБ (рекомендовано 32 ГБ).

3. Відеокарта: NVIDIA RTX 2060 або вище з підтримкою CUDA.

4. Вільне місце на диску: від 10 ГБ.

5. Монітор з роздільністю Full HD або вищою.

Вимоги до користувача:

1. Базові знання роботи в 3ds Max, розуміння принципів 3D-моделювання та побудови сцен.

2. Для розширеного використання бажано мати навички роботи зі скриптами MaxScript або Python.

2.2.3 Характеристика системи

Генерація килимів

Опис і пріоритет

Дозволяє автоматично створювати килими з параметрами форми, розміру й текстури. Пріоритет – високий.

Послідовність дій

Користувач запускає скрипт у 3ds Max.

У вікні налаштувань задаються основні характеристики килима:

- форма (прямокутна або кругла);
- розміри(для відповідної форми килима відповідні параметри);
- текстура;
- додавання китиць.

Програма перевіряє коректність введених даних та у випадку помилки система повідомляє користувача про неї і залишає значення за замовчуванням у такому полі. Далі об’єкт генерується і автоматично з’являється в центрі сцени. При необхідності користувач може змінити параметри та повторно згенерувати об’єкт.

Функціональні вимоги

REQ 1: Система повинна створювати 3D-моделі на основі параметрів користувача.

REQ 2: Має бути можливість змінювати форму, розміри, текстуру об'єкта.

REQ 3: Генерація має відбуватись у межах сцени без перезапуску 3ds Max.

REQ 4: Програма має здійснювати перевірку правильності введених даних – щоб були введені саме додатні числа, а не символи чи літери.

Заміна тимчасових боксів на деталізовані моделі

Опис і пріоритет

Автоматично підставляє готові 3D-моделі замість примітиву box із збереженням їхніх параметрів. Пріоритет – високий.

Послідовність дій

Користувач обирає примітив для заміни.

Вказує, яку модель використовувати для підстановки.

Скрипт автоматично підвантажує обрану модель, зберігаючи початкове положення, розміри та орієнтацію в просторі. У разі потреби можна процес.

Функціональні вимоги

REQ 1: Система повинна підтримувати підстановку моделей із даних комп'ютера.

REQ 2: Користувач повинен мати можливість самостійно вибрати шлях до моделі, яку він хоче вставити.

REQ 2: Має зберігатися позиція, розмір та орієнтація об'єкта.

Перевірка коректності введених параметрів

Опис і пріоритет

Перевірка введених даних користувачем для об'єктів сцени. Пріоритет – високий.

Послідовність дій

Користувач вводить значення параметрів у графічному інтерфейсі. Система автоматично перевіряє допустимість і відповідність значень.

У разі помилки виводиться повідомлення про конкретну помилку і попередження, що програма використає значення за замовчуванням у даному полі. Якщо всі значення коректні, програма підтверджує введення і переходить до створення об'єкта.

Функціональні вимоги

REQ 1: Система повинна фіксувати хибні параметри (тип введення даних лише числовий).

REQ 2: Повідомлення про помилку повинно бути зрозумілим.

REQ 3: Параметри повинні перевірятись до створення об'єкта.

REQ 4: Перевірені значення надсилаються для виконання обчислень і побудови об'єкта.

Інтеграція в середовище 3ds Max

Опис і пріоритет

Система працює всередині 3ds Max без потреби окремого встановлення чи зовнішніх додатків. Пріоритет – високий.

Послідовність дій

Користувач відкриває сцену в 3ds Max

Активує скрипт через меню або панель інструментів

Після запуску всі можливості стають доступними безпосередньо в інтерфейсі 3ds Max – без потреби переходити до зовнішніх програм. Створення моделей, заміна об’єктів, перевірка параметрів і редагування відбуваються прямо у поточній сцені. Такий підхід забезпечує безперервне та зручне впровадження інструментів у повсякденну роботу дизайнера чи 3D-візуалізатора.

Функціональні вимоги

REQ 1: Система має запускатися у вигляді скрипта безпосередньо в середовищі 3ds Max.

REQ 2: Повинна підтримувати роботу з 3ds Max версії 2020 або новішими.

REQ 3: Виконання скриптів повинно бути можливим автономно, без необхідності підключення до інтернету.

2.3 Структурна схема системи з урахуванням інформаційних потоків

Система, призначена для автоматизації процесу створення інтер’єрної 3D-візуалізації в 3ds Max, складається з кількох основних модулів, які взаємодіють між собою. Кожен модуль відповідає за окремий етап – від початкового введення даних до формування готової тривимірної моделі. У цьому розділі подано опис основних складових системи та пояснено, як відбувається передача інформації між ними.

Основні компоненти системи:

1. Інтерфейс користувача (UI)

Забезпечує спілкування користувача з усіма можливостями системи через простий та зрозумілий графічний інтерфейс.

Входи:

Дії користувача:

- введення параметрів(ширина, довжина, радіус, відстань між китицями);
- вибір матеріалів, китиць;
- натискання кнопок.

Виходи:

- Передача всіх введених даних у модуль перевірки для оцінки правильності.
- Команди початку генерації або заміни елементів у сцені.
- Повідомлення про помилки або підтвердження дій у вигляді зворотного зв’язку.

2. Модуль перевірки правильності введених даних

Відповідає за автоматичну перевірку введених користувачем параметрів на відповідність вимогам формату та допустимим межам.

Входи:

- Дані, введені користувачем через інтерфейс – числові значення, текстові рядки, вибрані опції з меню.

Виходи:

- Повідомлення про виявлені помилки (незаповнені поля, невірні типи інформації (наприклад, літери замість цифр).
- Список вірно заповнених і перевірених параметрів, які далі використовуються у модулях створення або заміни об’єктів.

3. Модуль створення параметричних об’єктів (килима)

Формує новий тривимірний об’єкт на основі введених характеристик, таких як розміри, форма, матеріал та оздоблення.

Входи:

- Перевірені параметри: довжина, ширина, геометрична форма (круг, прямокутник), матеріал, додаткові декоративні елементи (китиці).

Виходи:

- Згенерований 3D-об’єкт, створений відповідно до заданих налаштувань у вигляді повноцінної геометричної моделі.
- Передача цього об’єкта до модуля виводу для подальшого розміщення у відповідному місці сцени.

4. Модуль заміни тимчасових об’єктів (box)

Виконує підміну простих об’єктів-заглушок у сцені на деталізовані 3D-моделі без втрати розташування чи масштабу.

Входи:

- Координати та типи примітивів (box), які використовуються як тимчасові об’єкти.
- Дані про обрані 3D-моделі, які будуть використовуватися замість примітивів.

Виходи:

- Автоматично виконана заміна, при якій нові моделі вставляються на місце старих з точним збереженням розміру, положення і напряму.
- Оновлена сцена передається у фінальний модуль для додавання результатів у візуальне середовище.

5. Модуль виводу та інтеграції результатів у сцену

Відповідає за додавання нових або замінених 3D-об’єктів до поточної сцени 3ds Max, забезпечуючи їх правильне розміщення та відображення.

Входи:

- Новостворені 3D-об’єкти або замінені моделі, отримані з попередніх етапів – модуля генерації або заміни.

Виходи:

- Інтеграція об’єктів у поточну сцену 3ds Max: розміщення у відповідних координатах з коректним відображенням матеріалів та текстур.
- Оновлення сцени в реальному часі, що дозволяє користувачеві бачити зміни без необхідності перезавантажувати сцену чи вручну додавати елементи.

Інформаційні потоки:

Потік "Введення даних користувачем":

Користувач вводить параметри через інтерфейс – передача даних до модуля перевірки – генерація або заміна об'єктів у сцені.

Потік "Перевірка даних користувача":

Перевірка правильності введених параметрів – сповіщення про помилки або підтвердження – передача коректних даних у модуль створення чи заміни об'єктів.

Потік "Створення 3D моделі килима":

Перевірені параметри – формування 3D моделі – передача моделі в модуль виводу для її розміщення в сцені.

Потік "Заміна тимчасових об'єктів":

Дані про примітиви в сцені – заміна примітивів на детальні 3D моделі – оновлення сцени і передача в модуль виводу.

Потік "Інтеграція результатів у сцену":

Створені чи замінені 3D моделі – розміщення їх в сцені 3ds Max – візуальне оновлення сцени в реальному часі для користувача.

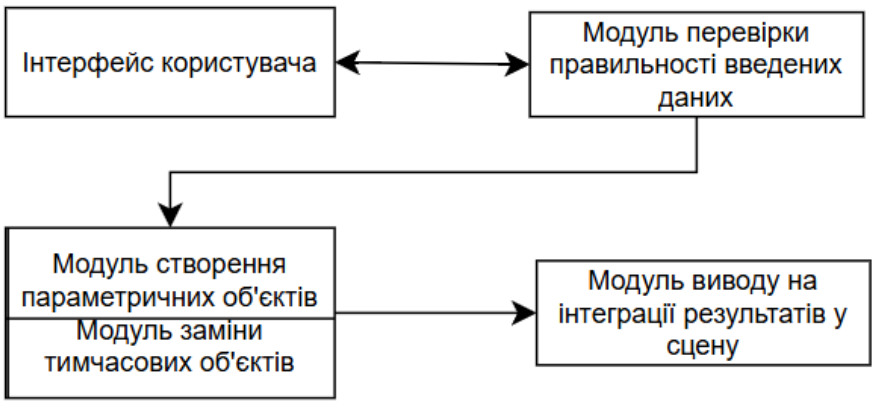


Рисунок 2.3 – Структурна схема системи із урахуванням інформаційних потоків

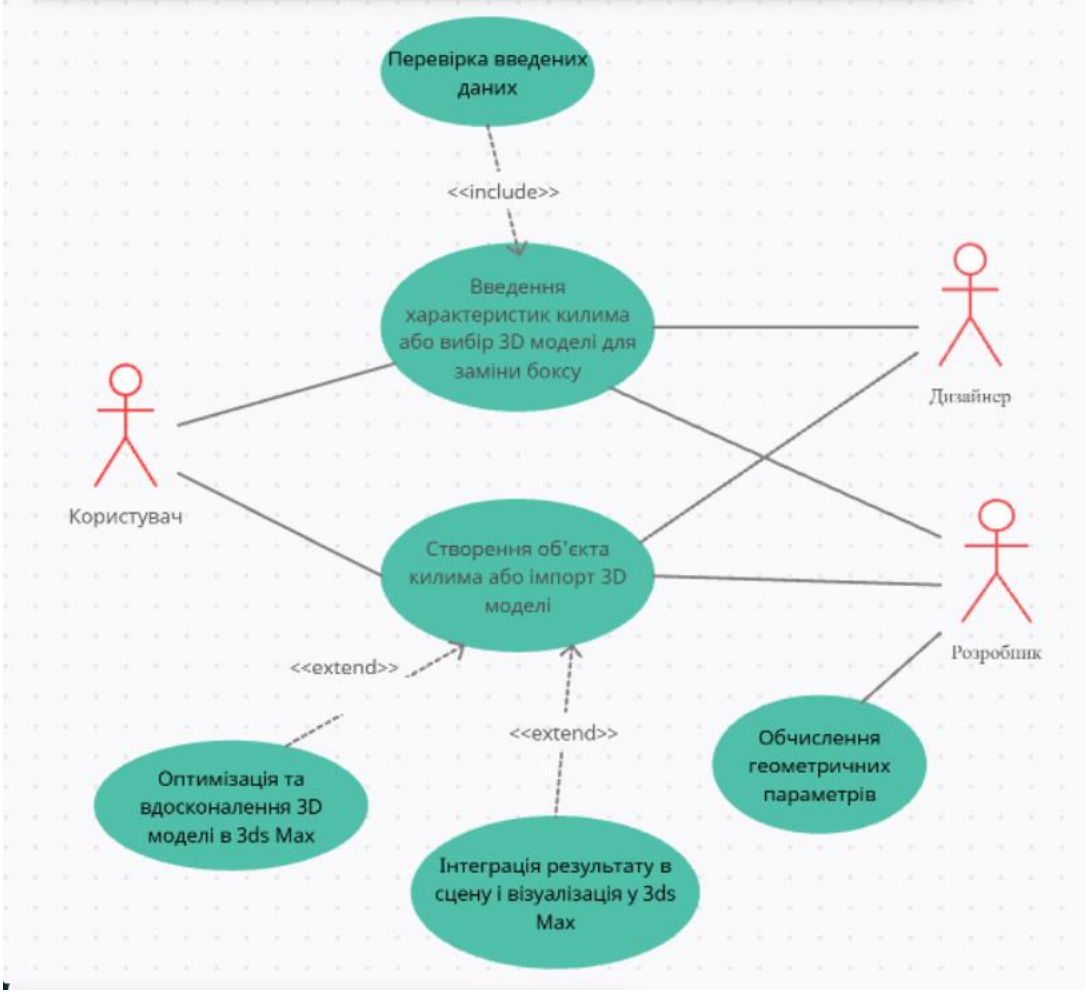


Рисунок 2.4 – Діаграма прецендентів проєктованої системи

Діаграма прецедентів (рис. 2.4) ілюструє взаємодію акторів (користувача, розробника та дизайнера) з системою, визначаючи їх ролі та функції. Вона показує, як кожен актор взаємодіє з різними модулями та етапами роботи системи.

2.4 Математичне забезпечення

Математичне забезпечення цієї системи охоплює виконання різних геометричних операцій, перевірку введених даних, а також здійснення математичних обчислень для створення та редагування 3D моделей в програмному середовищі 3ds Max.

2.4.1 Перевірка введених даних

Функція validate_input реалізована на мові Python призначена для перевірки введених значень. Розглянемо механізм її функціонування.

1. Очищення та перевірка на порожнє значення

На самому початку функції виконується очищення введеного тексту за допомогою методу .strip(), що видаляє зайві пробіли з початку та кінця рядка. Це важливо, щоб уникнути ситуацій, коли користувач залишає пробіли замість значення.

text = input_field.text().strip()

Далі перевіряється, чи є введений текст порожнім:

if text == "":

Якщо текст порожній (тобто користувач не ввів нічого або ввів тільки пробіли), спрацьовує блок, що викликає вікно повідомлення, в якому користувачеві повідомляється про відсутність введеного значення. Замість порожнього значення встановлюється значення за замовчуванням, яке передається як параметр `default_value`.

QtWidgets.QMessageBox.warning(None, "Invalid Input", f"Empty input. Default set to: {default_value}.")

input_field.setText(str(default_value

return default_value

Ця перевірка гарантує, що навіть якщо користувач залишить поле порожнім, система не дозволить відправити порожнє значення та замість цього автоматично підставить значення за замовчуванням.

2. Перевірка на числове значення та конвертація

Далі йде спроба перетворити введений текст на число з плаваючою комою (тип float). Це робиться за допомогою функції float(text):

value = float(text)

- Якщо текст не є коректним числом (наприклад, якщо користувач введе літери або спеціальні символи), цей процес викликає помилку, і буде перехоплено виняток ValueError, що дозволяє обробити помилку:

except ValueError:

- Якщо під час спроби перетворення на число виникає помилка, викликається вікно повідомлення, яке інформує користувача про те, що введене значення некоректне, і також встановлюється значення за замовчуванням:

QtWidgets.QMessageBox.warning(None, "Invalid Input", f"Invalid value. Default set to: {default_value}.")

input_field.setText(str(default_value

return default_value

3. Перевірка на значення більше нуля

Після того, як текст успішно перетворено на число, додається додаткова перевірка: чи це число більше нуля.

if value <= 0:

raise ValueError

Якщо значення не є дійсним числом (наприклад, від’ємне число або нуль), то знову виникає помилка, і запускається блок except, у якому, як і в попередньому випадку, повідомляється про помилку введення та встановлюється значення за замовчуванням:

QtWidgets.QMessageBox.warning(None, "Invalid Input", f"Invalid value. Default set to: {default_value}.")

input_field.setText(str(default_value

return default_value

4. Повернення коректного значення

Якщо всі перевірки пройшли успішно (введене значення є числом, більше нуля і не порожнє), повертається це значення:

return value

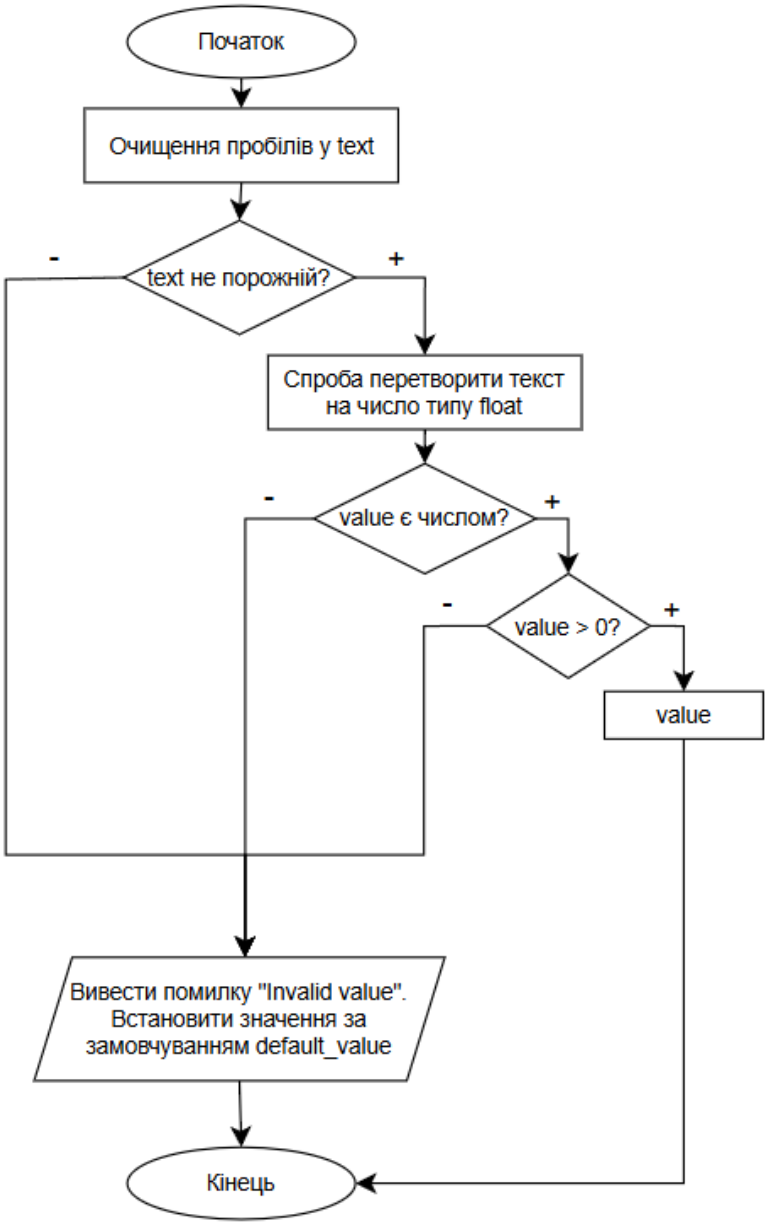


Рисунок 2.5 – Алгоритм роботи функції перевірки validate_input

2.4.2 Автоматизоване розташування китиць

1. Перевірка наявності об'єктів

if not tassel_node or not carpet_node:

QtWidgets.QMessageBox.warning(None, "Error", "Tassel or carpet not available.")

return

Якщо китиці або килим не задані – видається повідомлення і функція завершується.

2. Для прямокутного килима ("Rectangle")

- Отримання розмірів (box_height, box_width ,box_length)

Висота потрібна для розміщення китиць на верхній грані. box_length і box_width задають межі по X та Y.

- Внутрішня функція: distribute_on_edge(...)

Визначає кількість елементів і розміщує їх між start і end координатами, rotation відповідає за правильне орієнтування китиць.

- Розміщення по краях

Всі краї ("All Edges"):

distribute_on_edge(PointA, PointB, Rotation)

4 виклики для кожного з боків килима з відповідним поворотом (0°, 90°, 180°, -90°).

Тільки бокові краї ("Side Edges Only"):

Тільки дві сторони — верх і низ по осі Y.

3. Для круглого килима ("Circle")

- Початкові обчислення

radius = carpet_node.radius

circumference = 2 * math.pi * radius

tassel_count = max(3, int(circumference // tassel_distance))

angle_step = 360.0 / tassel_count

Обчислюється довжина кола і кількість елементів. Кут між кожною китицею розраховується рівномірно.

- Розміщення по колу

for i in range(tassel_count):

angle_deg = i * angle_step

angle_rad = math.radians(angle_deg)

x = cos(angle_rad) * radius

y = sin(angle_rad) * radius

pos = carpet_node.position + Point3(x, y, center_z)

Позиція кожної китиці визначається по колу за допомогою тригонометрії. Китиця створюється копіюванням:

tassel_copy = rt.copy(tassel_node)

tassel_copy.position = pos

- Орієнтація китиць по колу

rt.rotate(tassel_copy, rt.angleAxis(angle_deg, rt.Z_axis))

rt.rotate(tassel_copy, rt.angleAxis(-90, rt.Z_axis))

Кожна копія повертається в напрямку радіуса, потім додатково на -90° для природного вигляду.

Переваги функції

- Універсальність – функція підтримує як прямокутні, так і круглі килими.
- Автоматизація – самостійно обчислює кількість китиць, враховуючи розміри країв і задану відстань.
- Налаштовуваність – дозволяє обирати, чи додавати бахрому на всі краї, чи лише на бокові.
- Точність – китиці розміщуються рівномірно і з правильним обертанням.
- Надійність – перевіряє наявність необхідних об’єктів і повідомляє про помилки.
- Гнучка структура – легко адаптується для нових типів килимів або змін у логіці.
- Охайний результат – забезпечує симетричне та візуально привабливе розміщення без ручного втручання.

2.4.3 Заміна умовного об’єкта на 3D-модель

При розробці скрипта для автоматичної заміни об'єкта в сцені 3ds Max було реалізовано дві основні функції: get_object_dimensions() і scale_object_to_fit(). Вони забезпечують отримання точних розмірів тривимірного об’єкта та його масштабування відповідно до заданих параметрів.

1. Функція get_object_dimensions()

Ця функція використовується для обчислення розмірів об'єкта за трьома осями (X, Y, Z) на основі координат його вершин. Вона приймає як аргумент будь-який 3D-об’єкт у сцені.

Основні кроки роботи:

- Створюється mesh-снапшот переданого об’єкта, щоб отримати доступ до геометричних даних:

  mesh = rt.snapshotAsMesh(obj)

- Зчитуються координати всіх вершин об'єкта по кожній осі:

  verts = [v for v in mesh.verts]

  xs = [v.position.x for v in verts]

  ys = [v.position.y for v in verts]

  zs = [v.position.z for v in verts]

- Визначаються габарити шляхом знаходження різниці між максимальними та мінімальними значеннями по кожній осі:

  width = max(xs) - min(xs)

  length = max(ys) - min(ys)

  height = max(zs) - min(zs)

- Повертається кортеж з трьома значеннями:

  return width, length, height

Ця функція дозволяє отримати точні розміри об’єкта незалежно від його положення чи початкових параметрів.

2. Функція scale_object_to_fit()

Функція відповідає за масштабування одного об’єкта до габаритів іншого. Вона приймає два аргументи – об'єкт, який потрібно масштабувати, та еталонний об’єкт, під який треба підігнати розміри.

Основні кроки роботи:

- Визначаються габарити обох об'єктів, використовуючи get_object_dimensions():

  source_width, _length, _height = get_object_dimensions (source_obj)

target_width, length, target_height = get_object_dimensions (target_obj)

- Обчислюються коефіцієнти масштабування по кожній осі:

  scale_x = target_width / source_width

  scale_y = target_length / source_length

  scale_z = target_height / source_height

- Застосовується масштаб до об’єкта:

  source_obj.scale = rt.Point3(scale_x, scale_y, scale_z)

Масштабування відбувається пропорційно кожній осі окремо, що дозволяє точно вписати об’єкт у заданий простір.

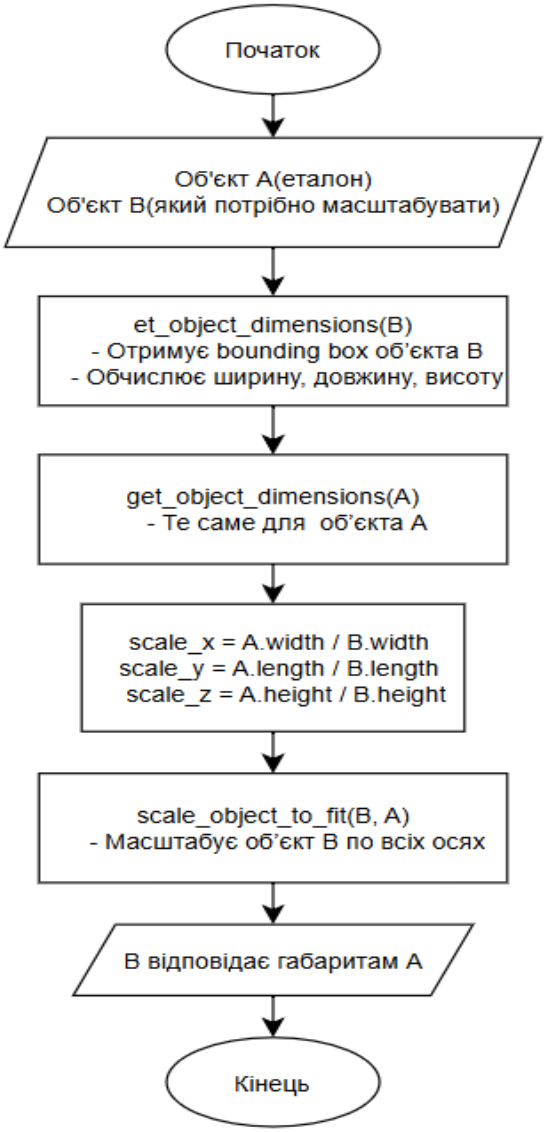


Рисунок 2.6 – Алгоритм роботи скрипта масштабування 3D-моделі за параметрами іншого об’єкта

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Проектування інтерфейсу

Проєктування інтерфейсу є ключовим етапом при створенні програм, що взаємодіють із користувачем. Від зручності, доступності та логіки розташування елементів керування залежить комфорт і ефективність користування програмним продуктом.

Інтерфейс користувача (UI) – це набір візуальних та інтерактивних компонентів, через які відбувається взаємодія користувача з програмою. Під час створення інтерфейсу були враховані такі базові принципи:

- Лаконічність – інтерфейс має бути простим у використанні та не вимагати спеціальних навичок від користувача.
- Узгодженість – усі елементи розміщені логічно та витримані в єдиному стилі оформлення.
- Адаптивність – можливість підлаштувати інтерфейс під різні варіанти килимів.
- Всеосяжність – інтерфейс охоплює всі основні характеристики моделі: геометрію, габарити, інтервали між елементами китиць, вибір сторін для її розміщення.

Практична реалізація

У ході розробки було створено два окремі макети інтерфейсу за допомогою інструменту Figma:

1. Інтерфейс для прямокутного килима (рис. 3.1) – включає поля для введення ширини, довжини та параметрів китиць.

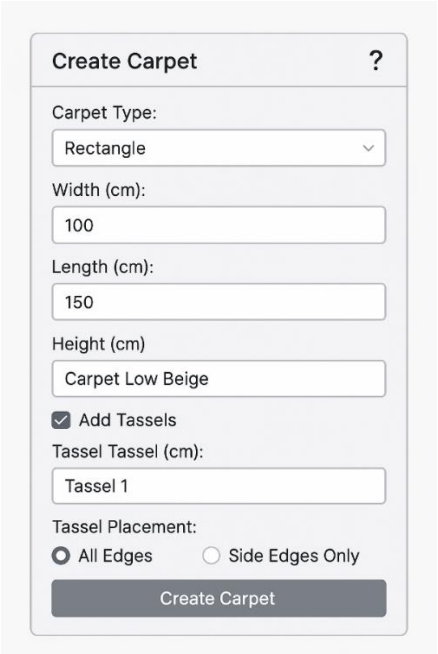


Рисунок 3.1 – Прототип вікна інтерфейсу прямокутного килима

2. Інтерфейс для круглого килима (рис. 3.2) – передбачає введення лише радіуса замість довжини й ширини.

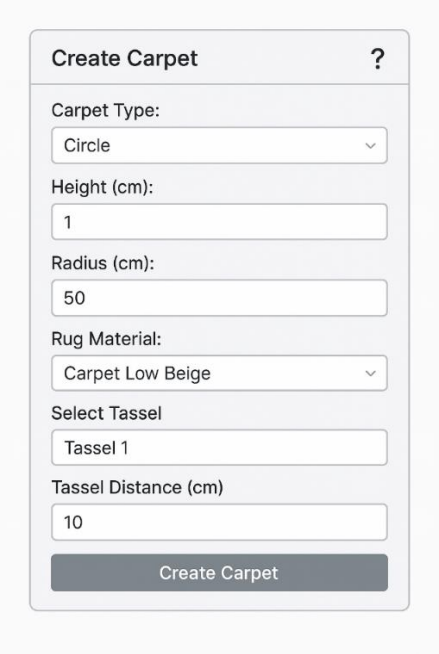


Рисунок 3.2 – Прототип вікна інтерфейсу круглого килима

Крім основних параметрів, обидва шаблони містять такі елементи:

- Меню вибору форми килима (коло або прямокутник)
- Поле для введення інтервалу між елементами бахроми

- Опції вибору сторін, на яких буде розміщено китиці (усі сторони або лише бічні)
- Кнопка запуску генерації моделі

Ці макети слугували базою для подальшої реалізації графічного інтерфейсу.

3.2 Опис програмної реалізації

3.2.1 Налаштування середовища розробки

Для створення інтер'єрної візуалізації і скриптів для автоматизації роботи було застосовано кілька ключових програмних інструментів. На цьому етапі виконується встановлення всіх потрібних інструментів, налаштовується середовище розробки з підтримкою Python, додаються сторонні бібліотеки, а також готуються засоби для візуалізації й редагування графічних ресурсів.

Встановлення базового програмного забезпечення

Початковим етапом є інсталяція основного програмного забезпечення, що буде використовуватись у проєкті. Базовим інструментом для моделювання 3D-об'єктів є Autodesk 3ds Max, адже він підтримує мови скриптів MaxScript і Python, що відкриває можливості для автоматизації внутрішніх процесів у середовищі програми.

Для генерації фотореалістичних візуалізацій застосовується Corona Renderer – популярний рушій візуалізації з підтримкою фізично достовірних матеріалів та джерел світла [12, 16]. Крім того, слід встановити окрему версію Python для розробки та тестування скриптів, а також Adobe Photoshop – для подальшої обробки графічного контенту [13].

Після завантаження Autodesk 3ds Max з офіційного сайту потрібно впевнитися, що під час інсталяції було увімкнено параметр активації Python API. Цей компонент надає змогу використовувати Python безпосередньо у 3ds Max. Для перевірки його наявності після запуску програми потрібно перейти в меню Scripting → Scripting Listener і ввести команду import pymxs.

Якщо команда виконується без помилок – середовище Python налаштоване правильно.

Налаштування Python-середовища

Хоча 3ds Max має вбудований інтерпретатор Python, доцільно встановити окрему версію Python (наприклад, 3.7 або 3.9), щоб отримати доступ до додаткових бібліотек, яких немає у стандартному комплекті Max. Завантажити Python можна з офіційного ресурсу – python.org. Під час встановлення важливо увімкнути опцію "Add Python to PATH", аби операційна система могла розпізнавати команду python з будь-якого терміналу.

Для реалізації графічного інтерфейсу буде використано бібліотеку PySide2 (Qt для Python). Її встановлення здійснюється через командний рядок за допомогою команди pip install PySide2 [14, 15].

Щоб ці бібліотеки стали доступними у середовищі 3ds Max, необхідно додати шлях до їхнього розташування в змінну sys.path. Це забезпечить можливість підключення сторонніх модулів до проєкту.

Для створення нового скрипту потрібно відкрити меню "Scripting" у 3ds Max, обрати пункт "MAXScript", а далі – "New Script".

Підключення Corona Renderer

Наступним кроком є встановлення Corona Renderer. Його потрібно завантажити з офіційного сайту та інтегрувати як плагін до 3ds Max. Після інсталяції Corona автоматично з’явиться у переліку доступних систем рендерингу. Щоб перевірити правильність роботи плагіна, можна створити просту тестову сцену, застосувати до об'єктів матеріал Corona Material і запустити рендер. Якщо зображення відображається коректно, це свідчить про успішну інтеграцію плагіна.

Налаштування параметрів рендерингу

У вікні Render Setup (рис. 3.3) необхідно обрати Corona Renderer як основну рендер-систему. Для якісного освітлення можна використати CoronaSun + CoronaSky або HDRI-карту.

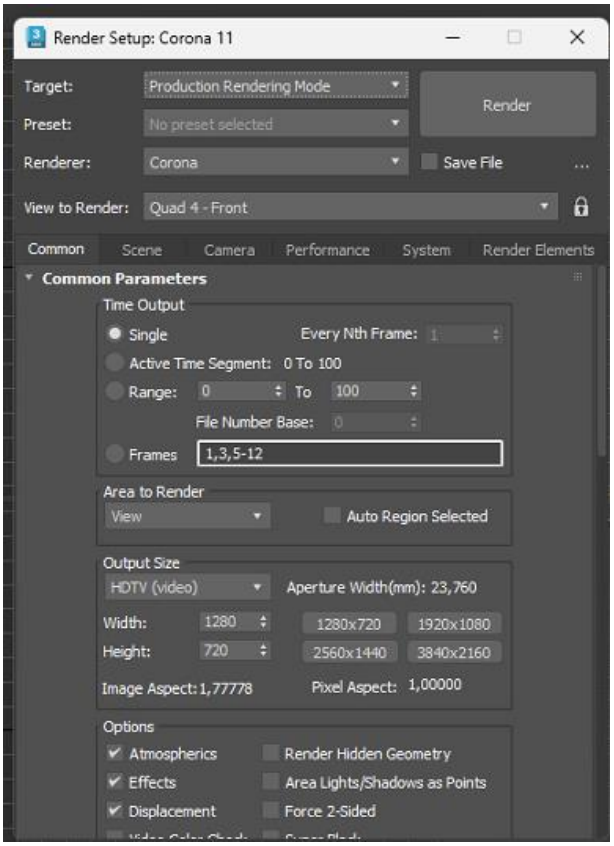


Рисунок 3.3 – Вікно налаштування рендеру

Використання фізичної камери Corona дозволяє точно налаштовувати параметри експозиції, глибини різкості та інші характеристики зображення. Активація функцій глобального освітлення (Global Illumination) та приглушення шумів (Denoising) сприяє отриманню високоякісного та реалістичного фінального рендеру.

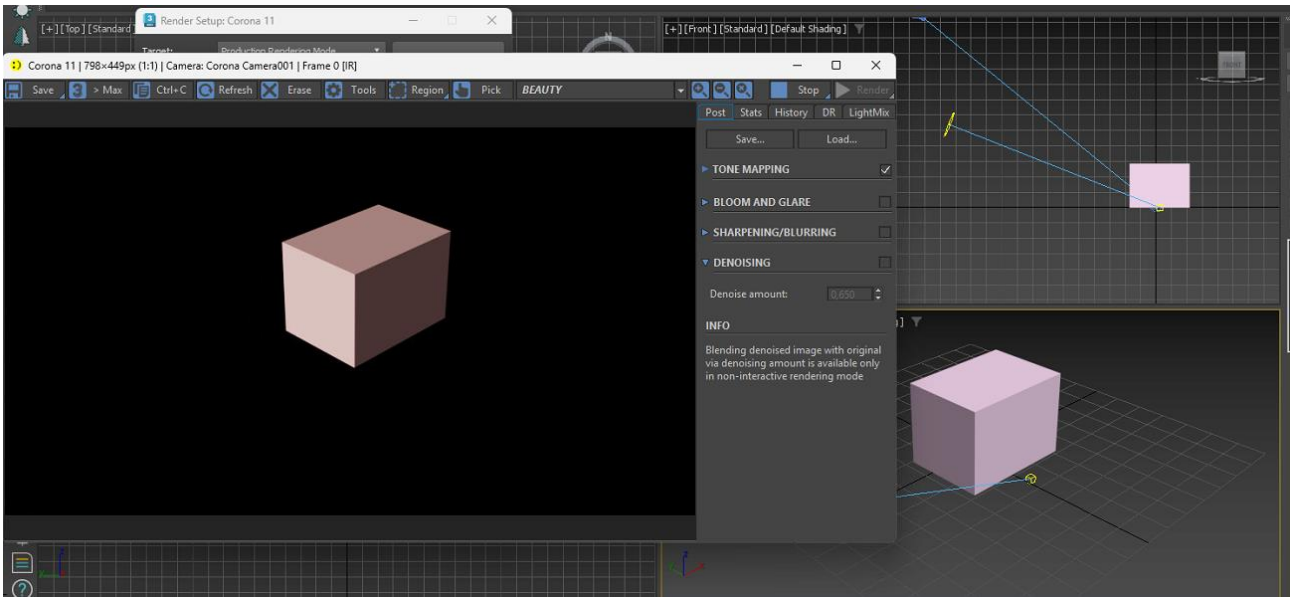


Рисунок 3.3 – Процес рендеру у Corona

3.2.2 Реалізація генерації параметричних килимів

Один із основних функціональних компонентів розробленого програмного інструменту – це скрипт, який автоматизує процес створення 3D-моделей килимів у середовищі Autodesk 3ds Max. Головне призначення цього скрипта – надати користувачу можливість швидко формувати килими з заданими властивостями без потреби в ручному моделюванні. Це значно скорочує час підготовки інтер'єрної сцени та підвищує ефективність роботи.

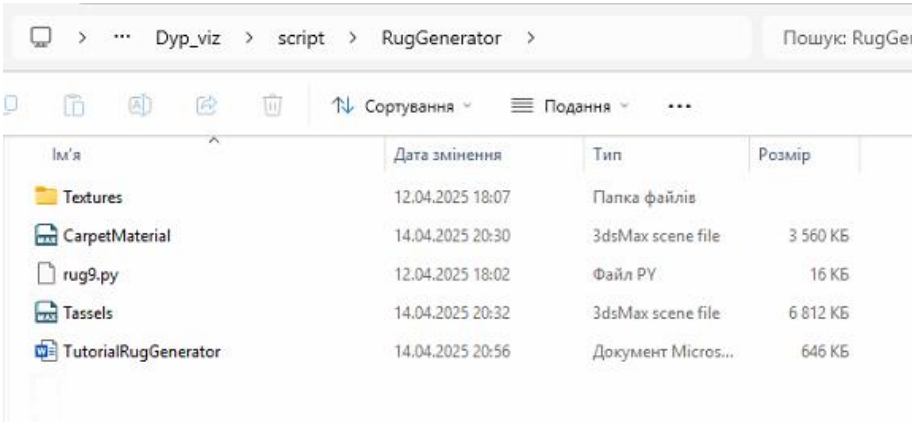


Рисунок 3.4 – Структура папки генератора килимів

Враховані параметри

У скрипті передбачено підтримку наступних налаштувань:

- Тип форми килима –можна вибрати між прямокутною та круглою конфігурацією.
- Розміри – для прямокутного килима вказуються довжина і ширина, для круглого – радіус.
- Матеріал – заздалегідь створений матеріал Corona Material, який автоматично призначається об'єкту після його створення.
- Наявність китиць – передбачена можливість додавання декоративних елементів по вибраних сторонах килима.
- Крок між китицями – задається відстань, яка визначає частоту розміщення елементів по периметру.
- Вибір сторін для китиць – можна вибрати, чи додавати китиці по всьому контуру або лише з коротких сторін.

Структура інтерфейсу

Графічний інтерфейс створено за допомогою бібліотеки PySide2, яка дозволяє будувати зручні користувацькі вікна у Python.

Взаємодія з 3ds Max через API

Коли користувач вводить потрібні параметри, скрипт передає їх у 3ds Max через API pymxs, який дозволяє створювати об’єкти, керувати матеріалами та виконувати інші дії у сцені. Залежно від вибраної форми, скрипт створює або об’єкт типу Box, або циліндричну геометрію.

Фрагмент створення прямокутного килима:

```
rt = pymxs.runtime
def create_rect_carpet(width, length):
    carpet = rt.box(length=length, width=width, height=0.2)
    carpet.name = "ParametricCarpet"
    return carpet
```

Для круглих килимів використовується примітив Cylinder з малою висотою:

```
def create_round_carpet(radius):
    carpet = rt.cylinder(radius=radius, height=0.2)
    carpet.name = "RoundCarpet"
    return carpet
```

Додавання китиць

Китиці створюються у вигляді окремої моделі, яка багаторазово копіюється вздовж вибраного краю. Кількість копій розраховується з урахуванням загальної довжини сторони та встановленого кроку. Для копіювання об’єктів використовується метод copy, а їх розміщення виконується шляхом обчислення позиції кожного наступного елемента.

```
def add_tassels_to_side(start_point, end_point, spacing, tassel_model):
    direction = end_point - start_point
    length = rt.length(direction)
    count = int(length / spacing)
    step = direction / count
```

```
for i in range(count):
    pos = start_point + step * i
    tassel = rt.copy(tassel_model)
    tassel.position = pos
```

Присвоєння матеріалу

Матеріал типу Corona Material, підготовлений заздалегідь, автоматично застосовується до створеної геометрії. Він завантажується з окремої сцени та призначається килиму:

```
material = rt.CoronaMtl()
material.name = "CarpetMaterial"
material.texmap_diffuse = rt.bitmaptexture(filename="path/to/texture.png")
carpet.material = material
```

Обробка помилок та повідомлення

У разі введення некоректних значень (наприклад, від’ємних розмірів або нульового кроку) інтерфейс виводить відповідне повідомлення користувачу за допомогою вікна діалогу (QMessageBox).

Розглянемо роботу цього скрипта

При запуску скрипта з’являється діалогове вікно (рис. 3.5), у якому за замовчуванням обраний тип килима Rectangle і введені стандартні параметри всіх полів.

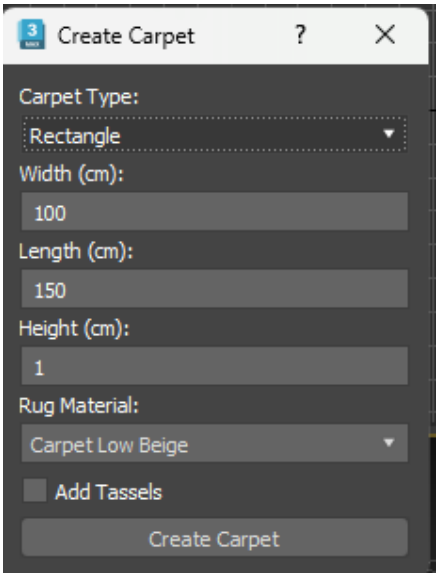


Рисунок 3.5 – Вікно запуску скрипта

У коді автоматично створюється розгортка для можливості накладання матеріалу на килим, і в наступному рядку користувач може у випадаючому меню обрати матеріал зі списку.

Наступним етапом є додавання китиць до килима (якщо немає потреби його можна опустити). Вибираємо чекбокс “Add Tassels” і з’являються додаткові параметри для розкидання китиць: вибір виду китиці(всього 7) і відстані між кожною китицею. Для примокутних килимів є можливість вибору методу розкидання китиць повністю по колу всього ковра (All Edges) або лише по правому і лівому краю(Side Edges Only).

Протягом усіх налаштувань параметрів вікно інтерфейсу автоматично регулюється відповідно до кількості видимих полів на даний момент.

Приклад створення прямокутного килима з китицями по периметру

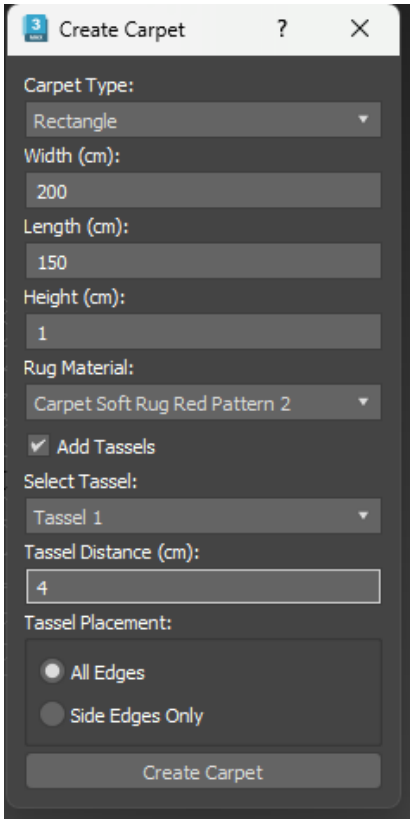


Рисунок 3.6 – Вікно створення прямокутного килима з китицями по периметру

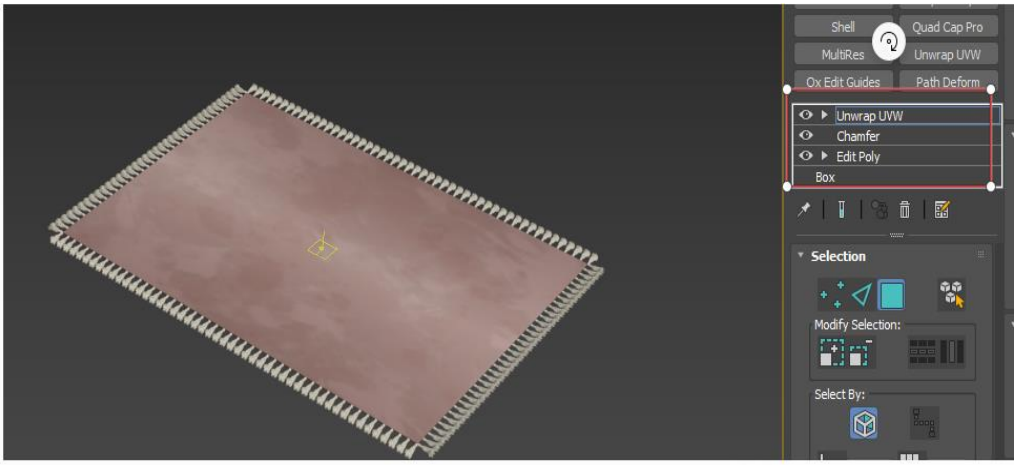


Рисунок 3.7 – Створений прямокутний килим

Створився килим з усіма налаштованими модифікаторами, розгорткою, матеріалом і китицями.

Приклад створення круглого килима

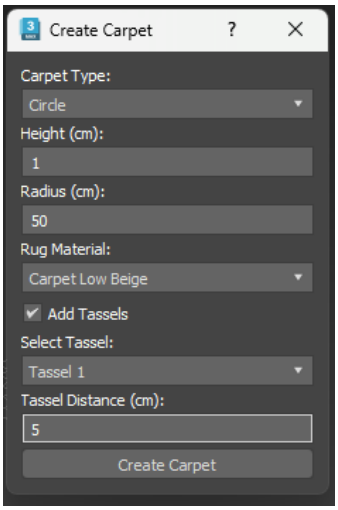


Рисунок 3.8 – Вікно створення круглого килима

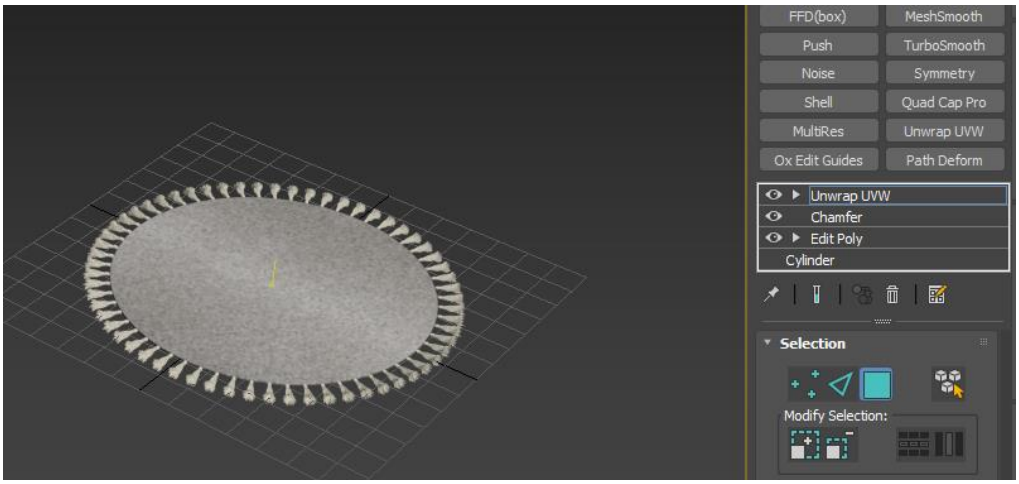


Рисунок 3.9 – Результат створення круглого килима

У результаті реалізація такого генератора параметризованих килимів забезпечує зручний інструмент для створення об’єктів інтер’єру з гнучкими налаштуваннями. Це значно полегшує моделювання сцени, зменшує час розробки та автоматизує повторювані завдання, що робить процес створення 3D-контенту набагато ефективнішим.

3.2.3 Заміна примітивів на деталізовані моделі

Даний модуль виконує функцію автоматичної заміни простих геометричних елементів, типу Box, на деталізовані тривимірні моделі. Такий підхід допомагає знизити ресурсне навантаження на систему під час побудови сцени, використовуючи примітиви як тимчасові заглушки. Уже на фінальному етапі розробки ці об'єкти можуть бути автоматично замінені на більш складні моделі з високою деталізацією без втрати розмірів, положення або орієнтації.

Призначення модуля

- Автоматична підміна геометричного примітиву на обрану користувачем 3D-модель із зовнішнього .max-файлу.
- Збереження усіх просторових характеристик: розміру, координат і напрямку об’єкта.
- Просте та надійне впровадження моделі в поточну сцену в 3ds Max.

Етапи виконання скрипта

1. Перевірка об'єкта визначається, чи активний лише один об'єкт і чи є він примітивом Box.

   ```
   if rt.selection.count == 1 and rt.classof(rt.selection[0]) == rt.Box:
      selected_box = rt.selection[0]
   ```

2. Отримання даних – зчитуються розміри вибраного об'єкта: довжина, ширина і висота.
3. Вибір зовнішньої моделі – користувач за допомогою діалогового вікна обирає файл моделі у форматі .max.

   ```
   model_path = rt.getOpenFileName(caption="Select 3ds Max File", types="*.max")
   ```

4. Імпорт сцени – нові об’єкти додаються до сцени, а система виявляє, який саме елемент було імпортовано.
5. Визначення габаритів – за допомогою функції snapshotAsMesh обчислюється bounding box імпортованої моделі.

   mesh = rt.snapshotAsMesh(imported_object)

   verts = [v.position for v in mesh.verts]
6. Масштабування – модель підганяється за розміром відповідно до параметрів вихідного примітиву.

   scale_vector = rt.Point3(

   width / model_width,

   length / model_length,

   height / model_height

)

   imported_object.scale = scale_vector
7. Розташування – нова модель встановлюється точно в ту позицію, де раніше знаходився Box.

   imported_object.position = selected_box.position
8. Видалення тимчасового об’єкта – початковий примітив видаляється зі сцени.

Розглянемо роботу цього скрипта

Побудуємо бокс розмірами 50*50*30

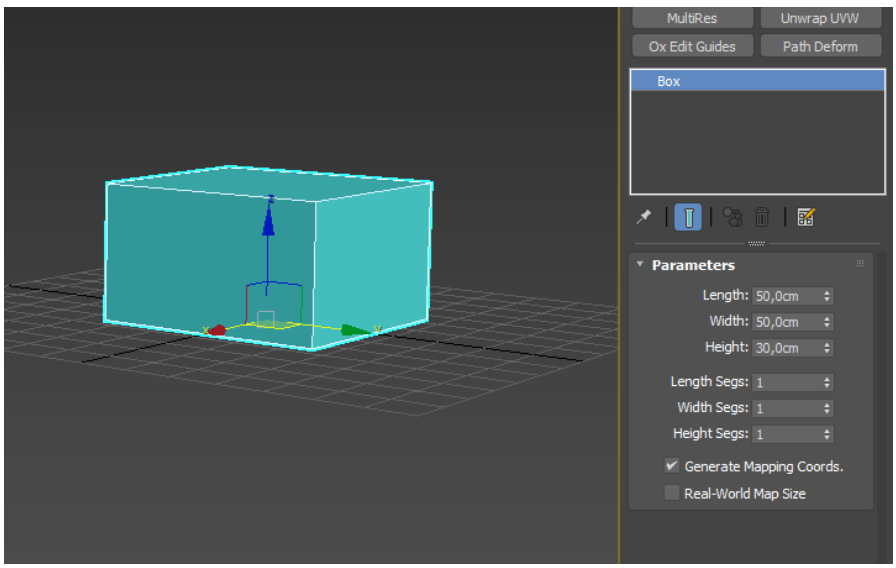


Рисунок 3.10 – Створення початкового боксу

Після запуску скрипта з’являється діалогове вікно, яке дозволяє обрати файл у форматі 3ds Max із моделлю, що буде вставлена у сцену. Це може бути будь-яка 3D-модель, попередньо збережена користувачем на комп’ютері.

І коли користувач обирає потрібну модель і натискає Open, замість боксу вставляється обрана модель з такими ж розмірами і в тому ж місці.

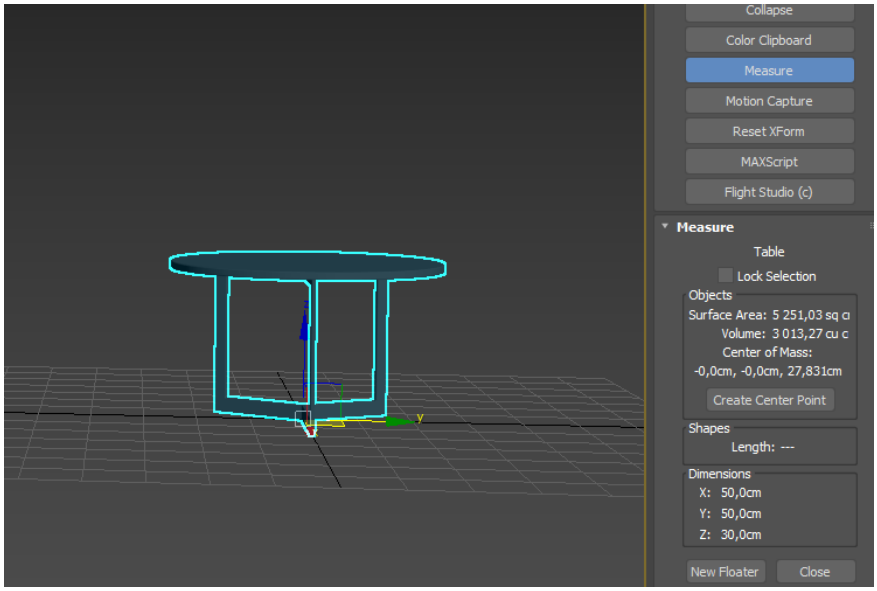


Рисунок 3.11 – Модель столу, яка вставилась замість боксу

Ось таким чином через простий інтерфейс користувач може замінити бокс на модель, зберігаючи її розміри та позицію у сцені.

У підсумку, такий скрипт значно полегшує процес роботи над сценою: під час макетування використовується легка геометрія, а у фіналі – автоматично підставляються складні моделі, що економить час і ресурси.

3.2.4 Створення інтер’єрної сцени та фінальна обробка

Для створення інтер’єрної сцени двокімнатної квартири були виконані такі етапи:

1. Моделювання

Спочатку було створено основні елементи інтер’єру (рис. 3.12), такі як стіни, підлога, стеля, вікна та двері, а також здійснено планування кімнат.

Рисунок 3.12 – Створені основні елементи інтер’єру

2. Розташування об'єктів

Потім були розставлені меблі, освітлювальні прилади та декоративні елементи, зокрема підібрані 3D-моделі для меблів на кухні та вітальні.

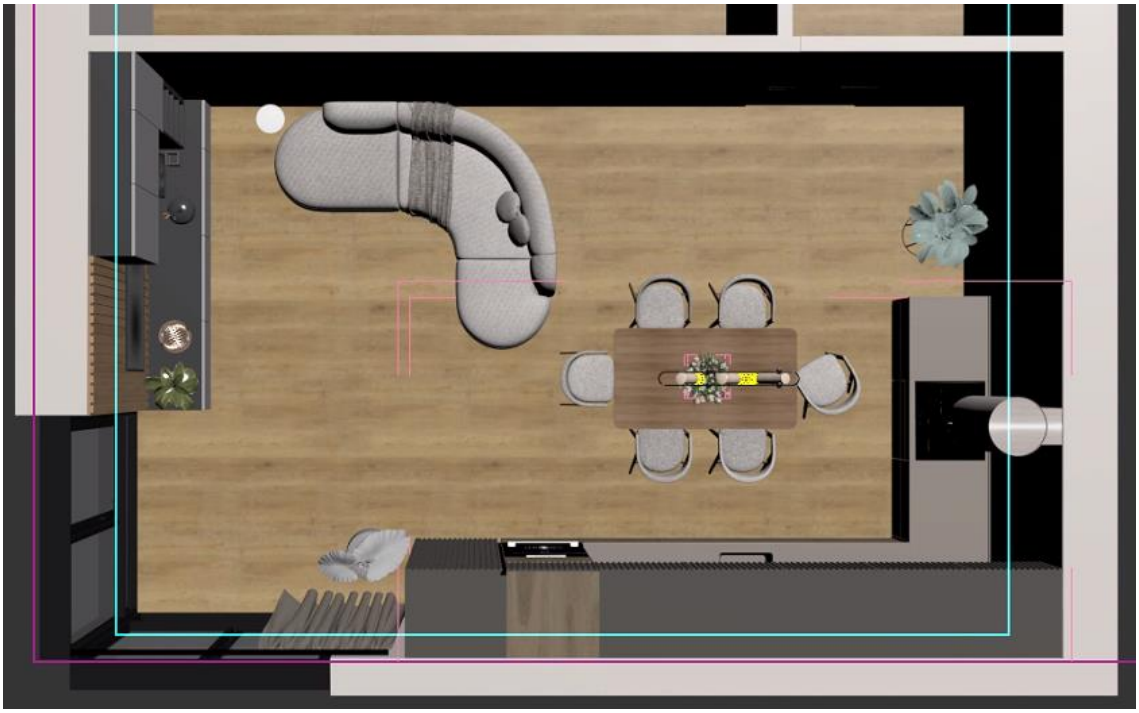


Рисунок 3.13 – Додані меблі, світло, декор до інтер’єру кухні

3. Використання генератора килимів

Завдяки розробленому скрипту в кухні (рис. 3.15), спальні та ванній кімнаті були додані параметричні килими з різними формами, розмірами та матеріалами.

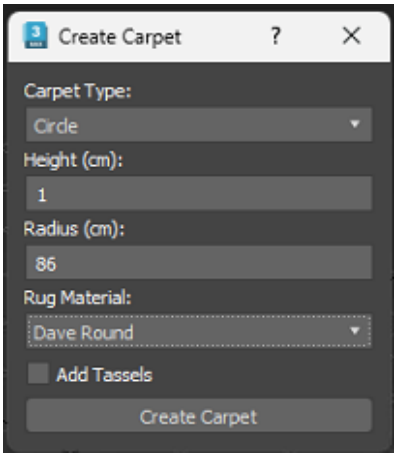


Рисунок 3.14 – Форма створення килима для кухні

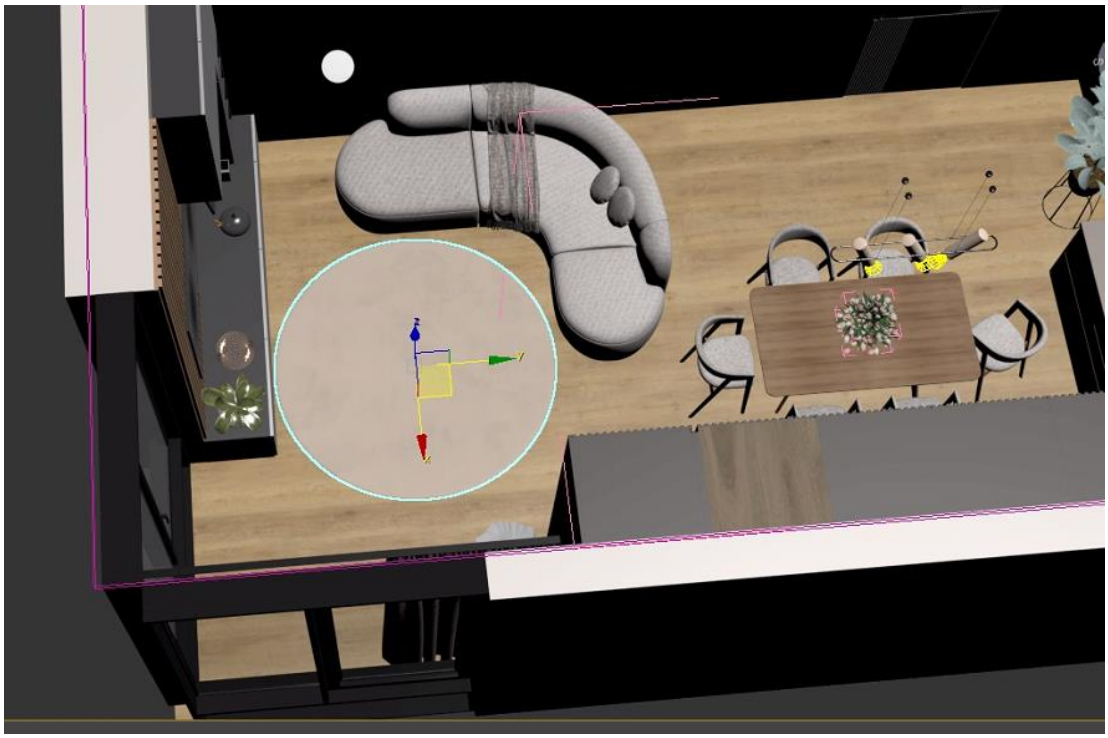


Рисунок 3.15 – Готовий килим в інтер’єрі

4. Заміна тимчасових об’єктів на деталі

Всі тимчасові об’єкти були замінені на високодеталізовані 3D-моделі, враховуючи точні розміри та позиціонування.

5. Освітлення

Для освітлення сцени було використано CoronaSun і CoronaSky для природного освітлення, а також HDRI-текстури для додаткового підсвічування та підвищення реалістичності.

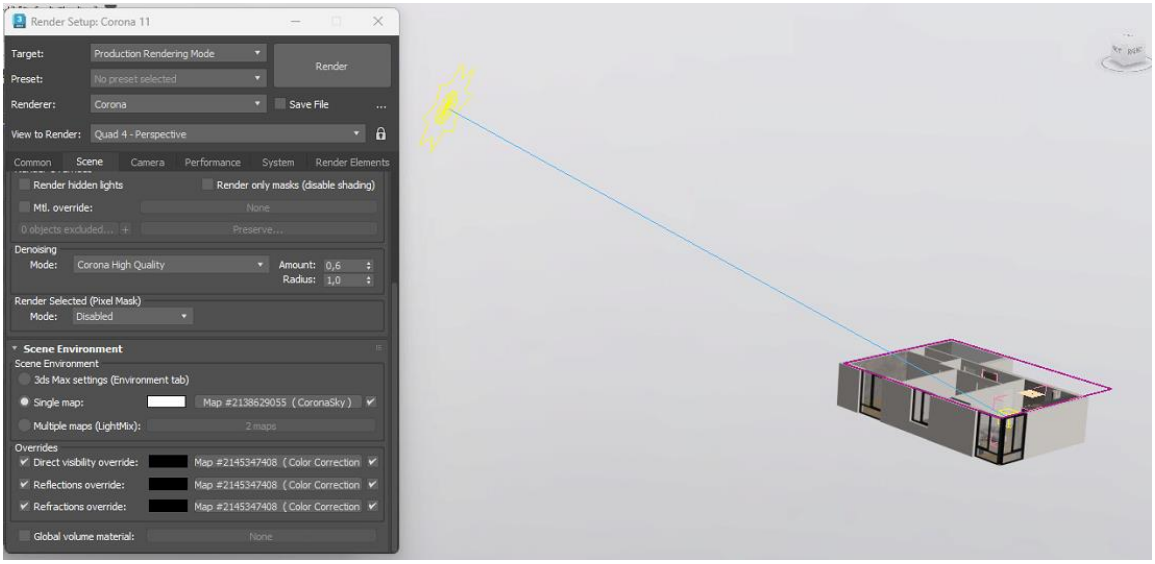


Рисунок 3.16 – Демонстрація використання елементів освітлення Corona

6. Налаштування камери і матеріалів

Були налаштовані камери для рендерингу з урахуванням композиційних принципів, а матеріали були кориговані за допомогою Corona Renderer, включаючи матеріали для підлог, стін, меблів і текстур.

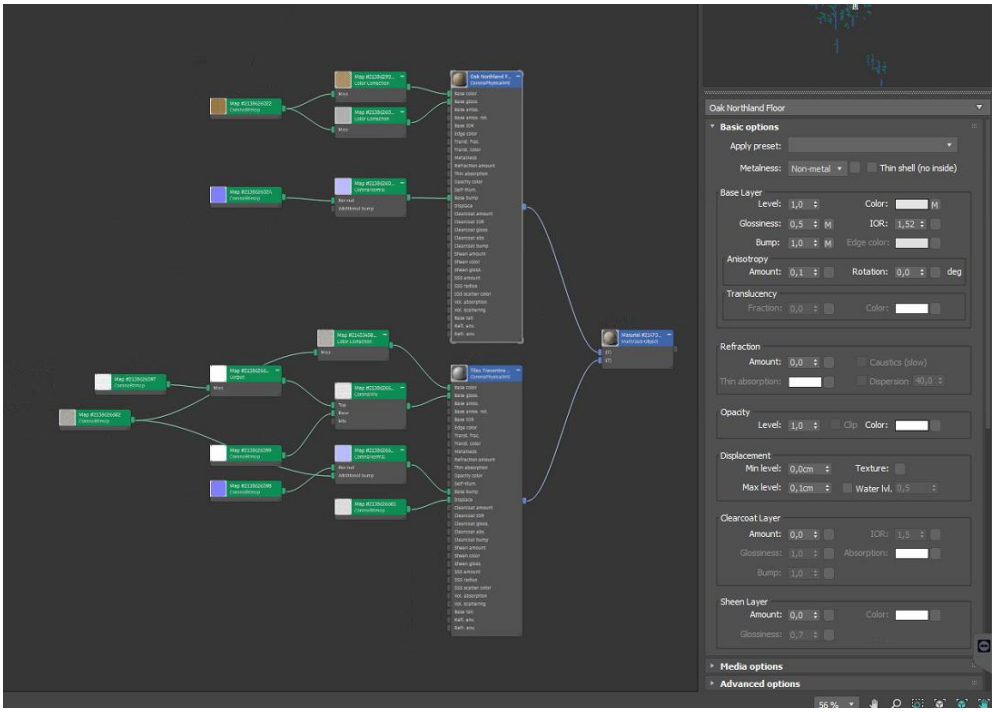


Рисунок 3.17 – Налаштовані матреріали Corona для підлоги

Рисунок 3.18 – План квартири з налаштованими матеріалами у вигляді зверху

7. Після завершення роботи над сценою було виконано налаштування фінального рендеру Corona Renderer

   Була проведена оптимізація рендеру з урахуванням експозиції та налаштувань денойзингу для забезпечення чистоти зображення. Додано ефект глибини різкості для створення більш реалістичного візуального сприйняття простору.

Обробка у Photoshop

Після рендерингу фінальні рендери були перенесені в Photoshop для покращення освітлення, корекції кольорової гами та підвищення різкості.

Для автоматизації і пришвидшення процесу зберігання готових фото було створено Action “300Dpi_JPG_Save” (рис. 3.19), за допомогою якої можна швидко зберегти всі кадри з однаковим розширенням (300 Dpi), у форматі JPEG та у одній папці.

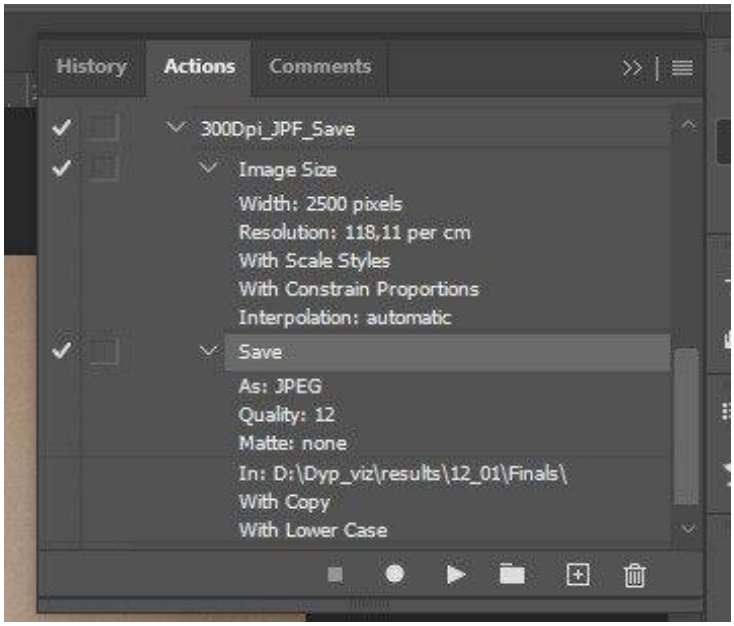


Рисунок 3.19 – Створений Action “300Dpi_JPG_Save”

Розглянемо кілька відредагованих зображень із застосуванням корекцій і фільтрів для досягнення оптимального результату.

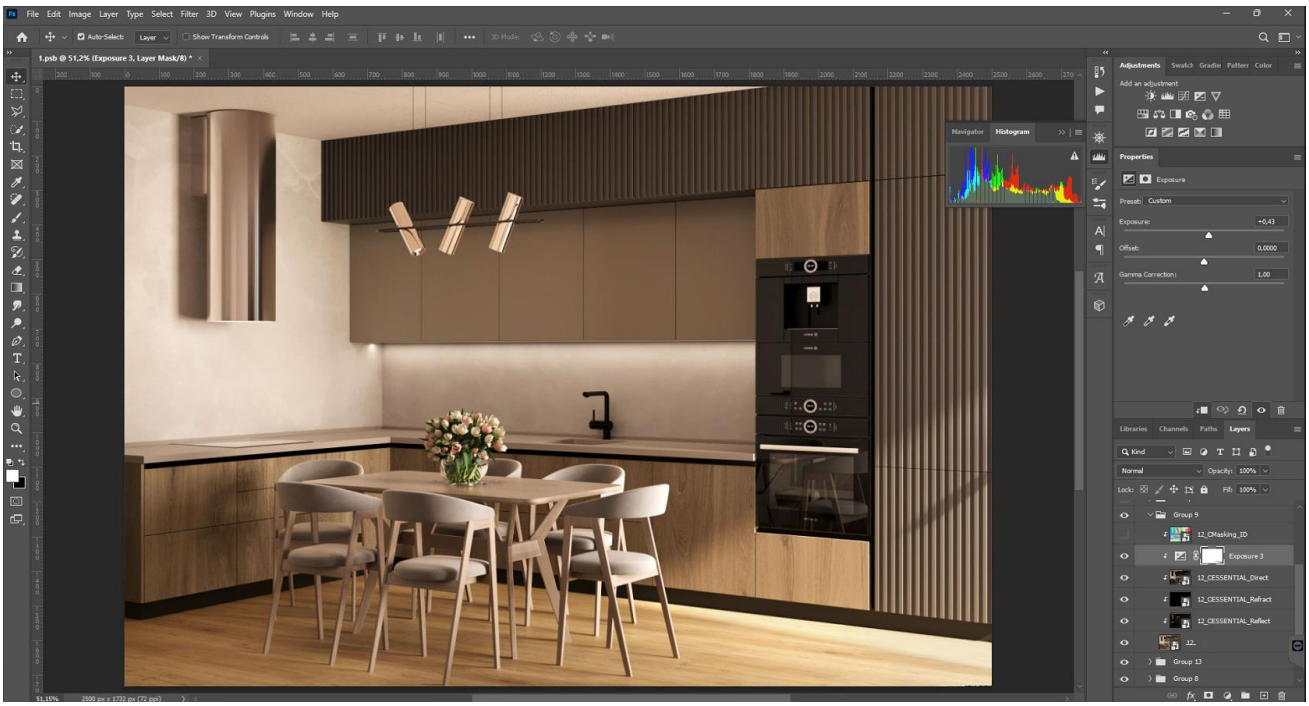


Рисунок 3.20 – Процес обробки рендеру кухні у Photoshop

Також розглянемо кілька фінальних результатів зображень.

Рисунок 3.21 – Фінальний результат (кухня)

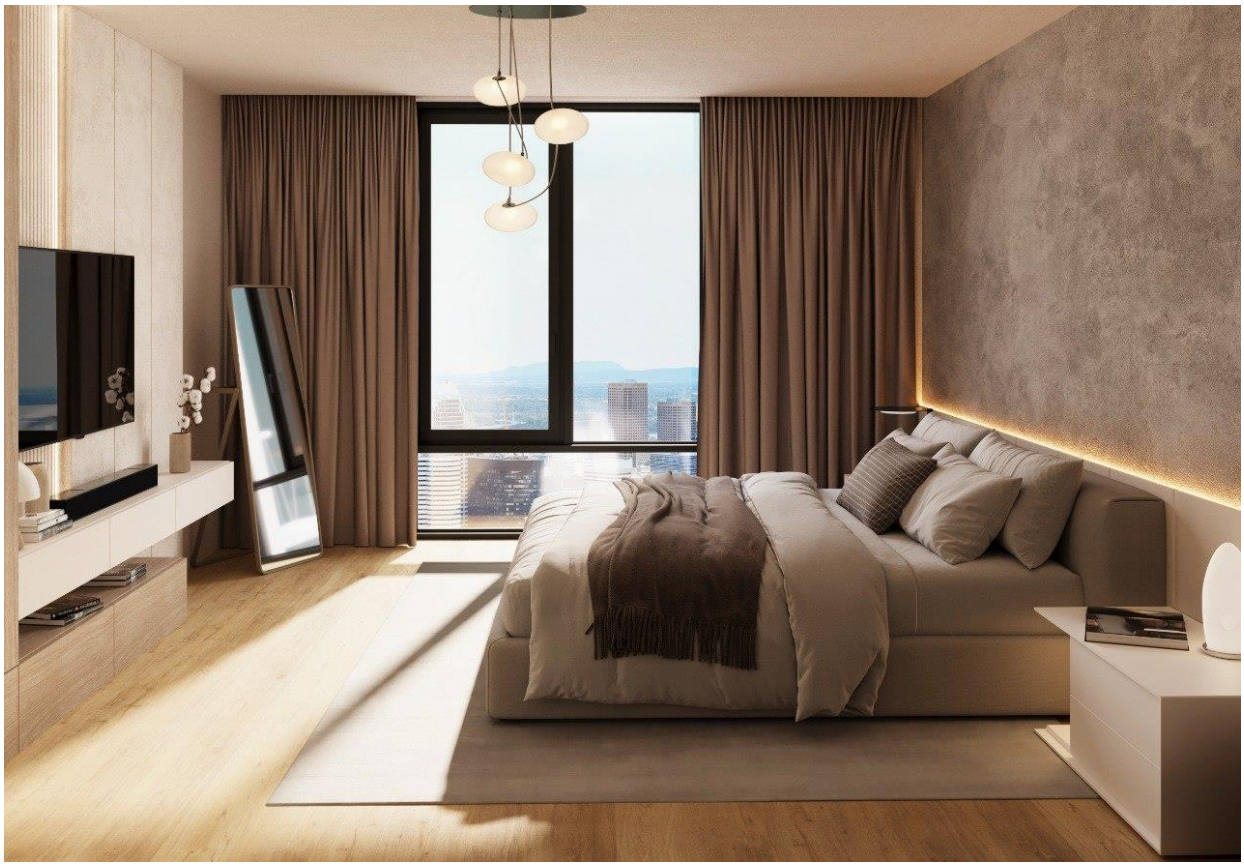


Рисунок 3.22 – Фінальний результат (спальня)

Рисунок 3.23 – Фінальний результат (ванна кімната)

ВИСНОВКИ

У процесі виконання дипломного проєкту було здійснено поглиблене дослідження сучасних методів інтер'єрної візуалізації, з акцентом на автоматизацію побудови сцен у програмному середовищі 3ds Max. Під час аналізу було виявлено, що існуючі інструменти не завжди відповідають вимогам дизайнерів щодо оперативного створення декоративних елементів і ефективного заповнення простору деталізованими моделями. Це стало поштовхом для розробки власного набору рішень, спрямованих на підвищення гнучкості та оптимізацію творчого процесу.

У межах проєкту були реалізовані два функціональні модулі. Перший – генератор параметричних килимів, який дозволяє формувати об'єкти на основі заданих параметрів: форма, розміри, текстури та додаткові елементи, такі як китички. Архітектура інструмента базується на принципах геометричної побудови з використанням PySide2 для створення графічного інтерфейсу та API 3ds Max для інтеграції елементів безпосередньо в сцену.

Другий модуль автоматизує процес заміни стандартних геометричних об'єктів типу Box на детальні 3D-моделі, що завантажуються з зовнішніх .max-файлів. Розроблений алгоритм не тільки імпортує модель, але й точно підганяє її розміри, розташування та орієнтацію відповідно до характеристик початкового примітива. Це дозволило зменшити обсяг рутинної ручної роботи та забезпечити узгодженість у структурі сцени.

На завершальному етапі було побудовано інтер'єрну сцену двокімнатної квартири, у якій запропоновані рішення були перевірені на практиці. Для візуалізації застосовувався Corona Renderer, що забезпечив високу якість фінальних зображень. Згодом результати були доопрацьовані у Photoshop для покращення кольорів, балансу світла та композиції.

Загальні результати роботи підтвердили ефективність використання скриптових інструментів як ефективного підходу в сфері візуалізації, вони успішно поєднують технологічну точність і творчу свободу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Help. Product Documentation | Autodesk Help. URL: https://help.autodesk.com/view/3DSMAX/2025/ENU/ (дата звернення: 06.05.2025).

2. Python 3.13 documentation. Python documentation. URL: https://docs.python.org/3/ (дата звернення: 06.05.2025).

3. 3D MAX - програма №1 для дизайнерів. EDS. URL: https://eds.ua/blog/article/3d-max-programa-nomer-1 (дата звернення: 06.05.2025).

4. blender.org - Home of the Blender project - Free and Open 3D Creation Software. blender.org. URL: https://www.blender.org/ (дата звернення: 06.05.2025).

5. Cineversity. Cineversity. URL: https://cineversity.maxon.net/en (дата звернення: 06.05.2025).

6. 3D Design Software | 3D Modeling & Drawing | SketchUp. 3D Design Software | 3D Modeling & Drawing | SketchUp. URL: https://www.sketchup.com/en?srsltid=AfmBOoqj1ldjtqeXFswggMN8QMaZDuWFbhOajSRPVbwRdgUS6nkUjh3F (дата звернення: 06.05.2025).

7. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/python/ (дата звернення: 06.05.2025).

8. Help. Product Documentation | Autodesk Help. URL: https://help.autodesk.com/view/MAXDEV/2025/ENU/?guid=GUID-55B19ED6-0E18-4BED-B23D-856178AA874B (дата звернення: 06.05.2025).

9. Learn C++ Programming. Programiz: Learn to Code for Free. URL: https://www.programiz.com/cpp-programming (дата звернення: 06.05.2025).

10. MEL - Maya Embedded Language File Format. File Format Docs. URL: https://docs.fileformat.com/programming/mel/ (дата звернення: 06.05.2025).

11. Діагностика та ідентифікація проблем. Stud. URL: https://stud.com.ua/31874/menedzhment/diagnostika_identifikatsiya_problem#goog_rewarded (дата звернення: 06.05.2025).

12. Corona Render + 3DS MAX Rendering Tutorial – Real vs. Rendered. tonytextures.com – graphics for architectural visualization. URL:https://

www.tonytextures.com/3ds-max-corona-tutorial-architecture-rendering-vsphoto/ (дата звернення: 06.05.2025).

13. Основні відомості про робоче середовище. Adobe Help Center. URL: https://helpx.adobe.com/ua/photoshop/using/workspace-basics.html (дата звернення: 06.05.2025).

14. Fitzpatrick M. PySide2 Tutorial 2025, Create Python GUIs with Qt. Python GUIs. URL: https://www.pythonguis.com/pyside2-tutorial/ (дата звернення: 06.05.2025).

15. BRANDT J. Python 101: Basics for Beginners: A Practical Introduction to Python 3. Independently Published, 2022.

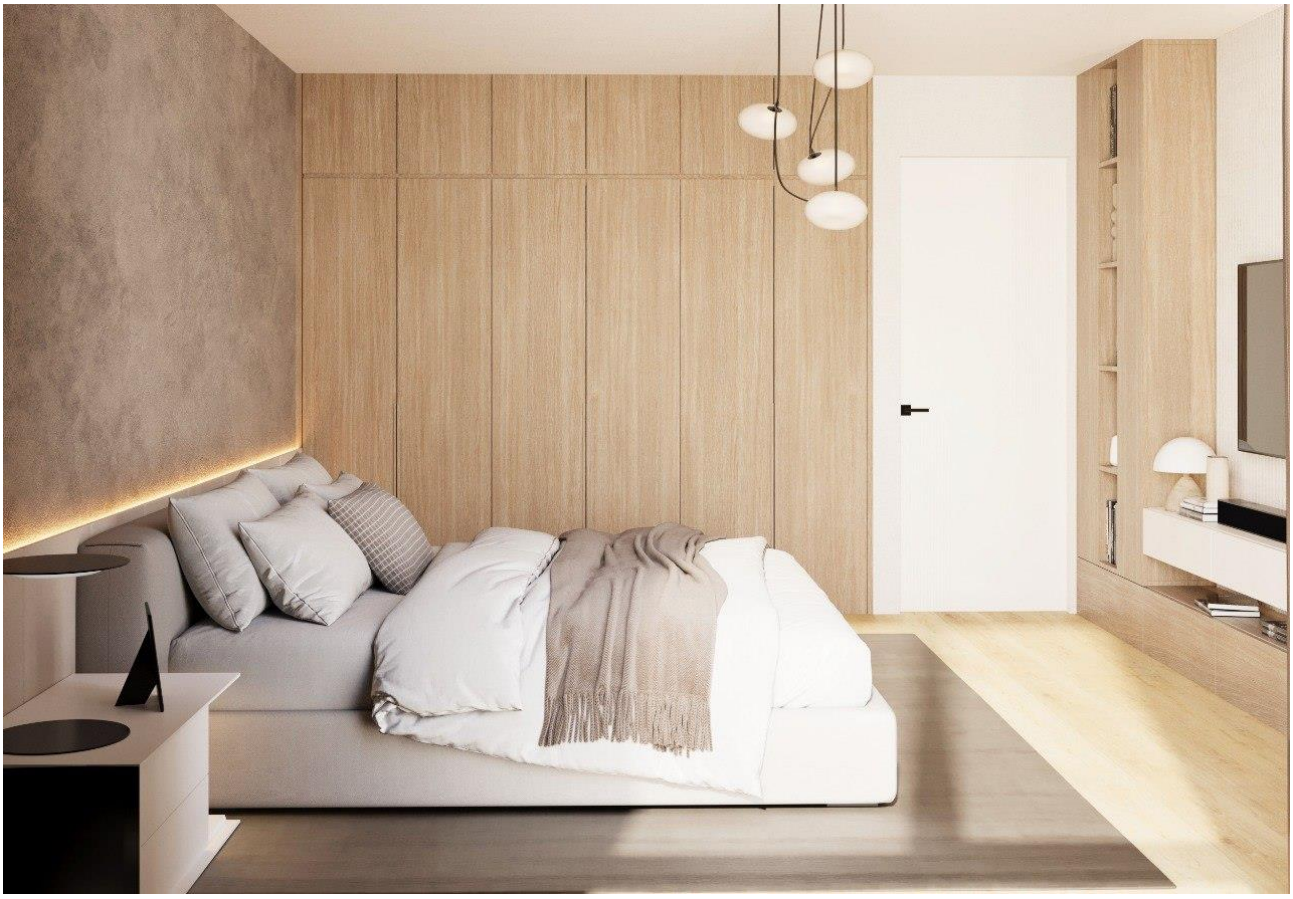
16. Autodesk. 3ds Max 8 Essentials. Taylor & Francis Group, 2017.

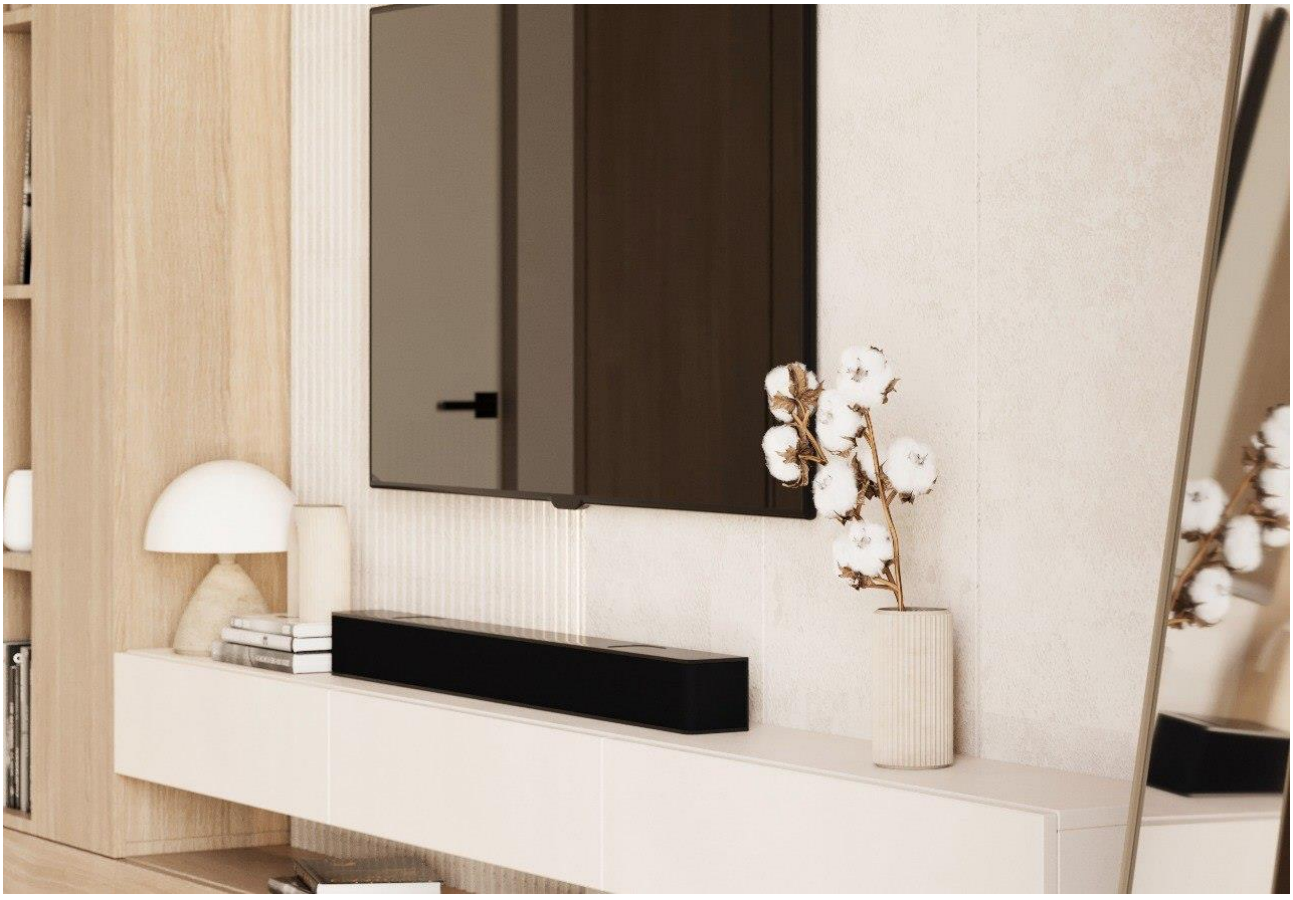
ДОДАТКИ

ДОДАТОК А

Фінальні візуалізації інтер’єру двокімнатної квартири

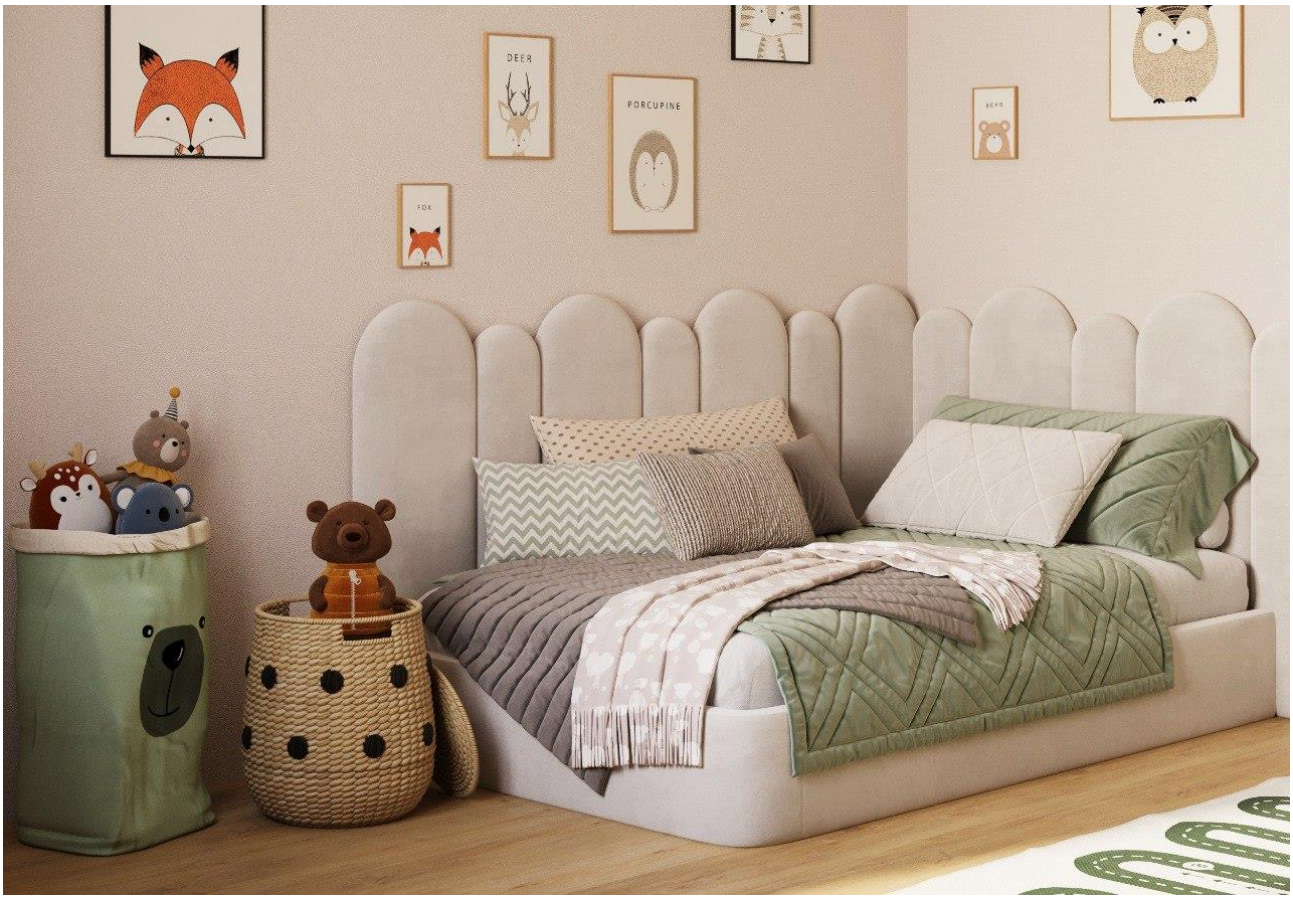
Візуалізація спальні з різних ракурсів: загальний вигляд, ліжко, робочий куточок, зона зберігання.





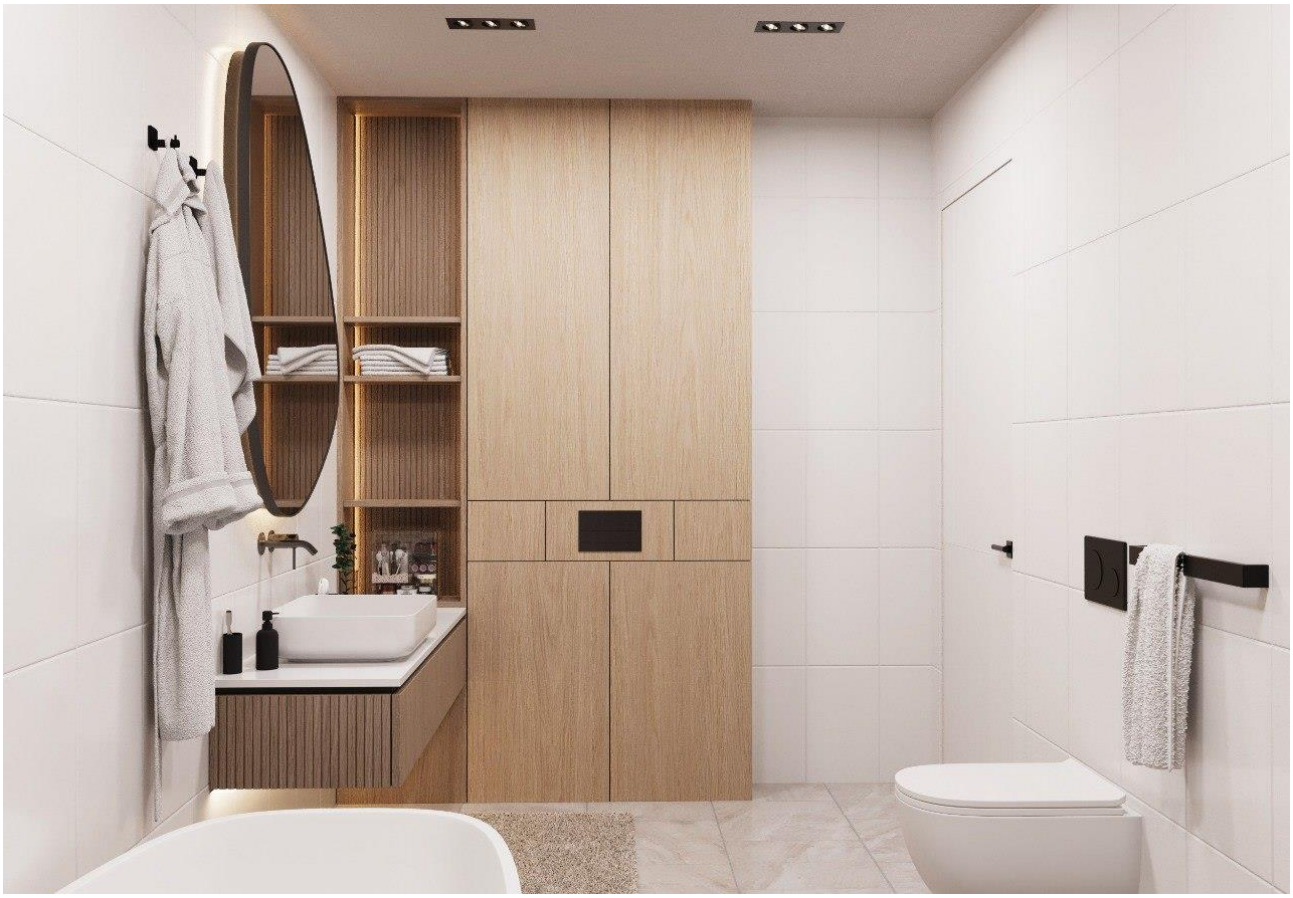
Візуалізація кухонного простору з різних ракурсів: робоча зона, обідній стіл, зона відпочинку.

Рендери дитячої кімнати: ігрова зона, письмовий стіл, місце для сну, декор.




FOX
OWL

Рендери ванної кімнати: душова зона, умивальник, ванна, декоративні елементи.



Візуалізація коридору: вигляд на вхід, шафа, дзеркало, освітлення.

Візуалізація господарського приміщення з пральною машиною та зоною зберігання побутових речей.

ДОДАТОК Б

Код скрипта гернератора килимів:

```
# -*- coding: utf-8 -*-
import pymxs
from PySide2 import QtWidgets, QtGui
import os
import math

rt = pymxs.runtime

material_loaded = False
carpet_material = None
tassel_node = None  # Holds the loaded tassel node
carpet_node = None  # Stores the created carpet
all_carpet_nodes = []

def load_carpet_material():
    global material_loaded, carpet_material
    if material_loaded:
        print("Material already loaded.")
        return

    script_dir = os.path.dirname(__file__)
    material_file_path = os.path.join(script_dir, "CarpetMaterial.max")

    try:
        rt.mergeMAXFile(material_file_path, quiet=True)
        proxy_box = rt.getNodeByName("CarpetBox")
        if proxy_box and proxy_box.material:
            carpet_material = proxy_box.material
            material_loaded = True
            rt.delete(proxy_box)
            print("Material loaded successfully.")
        else:
            QtWidgets.QMessageBox.warning(None, "Error", "CarpetBox or its material not found in
merged file.")
    except Exception as e:
        QtWidgets.QMessageBox.warning(None, "Error loading materials", str(e))

def load_tassel(tassel_index):
    global tassel_node
    try:
        tassel_node = rt.getNodeByName(f"Tassel{tassel_index}")
        if tassel_node:
            print(f"Tassel {tassel_index} already loaded.")
            return
        script_dir = os.path.dirname(__file__)
        tassel_file_path = os.path.join(script_dir, "Tassels.max")
        rt.mergeMAXFile(tassel_file_path, quiet=True)
        tassel_node = rt.getNodeByName(f"Tassel{tassel_index}")
        if not tassel_node:
```

```
                QtWidgets.QMessageBox.warning(None, "Error", f"Tassel {tassel_index} not found in the merged file.")

            except Exception as e:
                QtWidgets.QMessageBox.warning(None, "Error loading tassel", str(e))

        def generate_unique_carpet_name():
            base_name = "Carpet"
            index = 1
            while True:
                carpet_name = f"{base_name}_{index}"
                if not rt.getNodeByName(carpet_name):
                    break
                index += 1
            return carpet_name

        def create_carpet(width, length, height, carpet_type, material_index):
            global carpet_node, carpet_material, all_carpet_nodes
            corner_radius = 0.35
            load_carpet_material()

            if not carpet_material:
                load_carpet_material()

            if not carpet_material:
                QtWidgets.QMessageBox.warning(None, "Error", "Material not loaded!")
                return

            if material_index < 1 or material_index > len(carpet_material):
                QtWidgets.QMessageBox.warning(None, "Error", f"Invalid material index: {material_index}")
                return

            try:
                sub_material = carpet_material[material_index - 1]
                print(f"Using material: {sub_material}")
            except Exception:
                QtWidgets.QMessageBox.warning(None, "Error", f"Invalid material index: {material_index}")
                return

            carpet_name = generate_unique_carpet_name()

            if carpet_type == "Rectangle":
                maxscript_code = f"
                undo on
                (
                    local carpet = box length:{length} width:{width} height:{height} pos:[0,0,0]
                    addModifier carpet (Edit_Poly())
                    addModifier carpet (Chamfer amount:{corner_radius} segments:2)
                    addModifier carpet (Unwrap_UVW())
                    carpet.name = "{carpet_name}"
                )
                "
            elif carpet_type == "Circle":
                maxscript_code = f"
```

```
            undo on
            (
                local carpet = cylinder radius:{width} height:{height} pos:[0,0,0] sides:90
                addModifier carpet (Edit_Poly())
                addModifier carpet (Chamfer amount:{corner_radius} segments:2)
                addModifier carpet (Unwrap_UVW())
                carpet.name = "{carpet_name}"
            )
            '''

        try:
            rt.execute(maxscript_code)
            carpet_node = rt.getNodeByName(carpet_name)
            if carpet_node:
                carpet_node.material = sub_material
                carpet_node.materialID = material_index
                all_carpet_nodes.append(carpet_node)
                print(f"Material applied to {carpet_name}: {sub_material}")
            else:
                print(f"Carpet creation failed: {carpet_name}")
        except Exception as e:
            QtWidgets.QMessageBox.warning(None, "Execution Error", f"Error executing MAXScript:
{str(e)}")

    def distribute_tassels(carpet_node, tassel_distance, carpet_type, edge_option):
        if not tassel_node or not carpet_node:
            QtWidgets.QMessageBox.warning(None, "Error", "Tassel or carpet not available.")
            return

        box_height = carpet_node.height

        if carpet_type == "Rectangle":
            box_width = carpet_node.length
            box_length = carpet_node.width

            def distribute_on_edge(start, end, rotation):
                distance = rt.distance(start, end)
                tassel_count = max(2, int(distance // tassel_distance))
                for i in range(tassel_count):
                    x = start.x + (end.x - start.x) * (i / (tassel_count - 1))
                    y = start.y + (end.y - start.y) * (i / (tassel_count - 1))
                    pos = rt.Point3(x, y, carpet_node.position.z + box_height / 2)
                    tassel_copy = rt.copy(tassel_node)
                    tassel_copy.position = rt.Point3(0, 0, 0)
                    tassel_copy.rotation = rotation
                    tassel_copy.position = pos

            if edge_option == "All Edges":
                distribute_on_edge(rt.Point3(box_length / 2, -box_width / 2, 0), rt.Point3(box_length / 2,
box_width / 2, 0), rt.EulerAngles(0, 0, 90))
                distribute_on_edge(rt.Point3(-box_length / 2, -box_width / 2, 0), rt.Point3(-box_length / 2,
box_width / 2, 0), rt.EulerAngles(0, 0, -90))
                distribute_on_edge(rt.Point3(-box_length / 2, box_width / 2, 0), rt.Point3(box_length / 2,
box_width / 2, 0), rt.EulerAngles(0, 0, 0))
```

```
                distribute_on_edge(rt.Point3(-box_length / 2, -box_width / 2, 0), rt.Point3(box_length / 2, -box_width / 2, 0), rt.EulerAngles(0, 0, 180))
            elif edge_option == "Side Edges Only":
                distribute_on_edge(rt.Point3(-box_length / 2, box_width / 2, 0), rt.Point3(box_length / 2, box_width / 2, 0), rt.EulerAngles(0, 0, 0))
                distribute_on_edge(rt.Point3(-box_length / 2, -box_width / 2, 0), rt.Point3(box_length / 2, -box_width / 2, 0), rt.EulerAngles(0, 0, 180))

        elif carpet_type == "Circle":
            radius = carpet_node.radius
            height = carpet_node.height
            center_z = height / 2.0
            tassel_distance = max(tassel_distance, 1.0)
            circumference = 2 * math.pi * radius
            tassel_count = max(3, int(circumference // tassel_distance))
            angle_step = 360.0 / tassel_count

            for i in range(tassel_count):
                angle_deg = i * angle_step
                angle_rad = math.radians(angle_deg)
                x = math.cos(angle_rad) * radius
                y = math.sin(angle_rad) * radius
                pos = carpet_node.position + rt.Point3(x, y, center_z)
                tassel_copy = rt.copy(tassel_node)
                tassel_copy.position = pos
                rt.rotate(tassel_copy, rt.angleAxis(angle_deg, rt.Z_axis))
                rt.rotate(tassel_copy, rt.angleAxis(-90, rt.Z_axis))

        hide_start_tassels()

    def hide_start_tassels():
        try:
            for i in range(1, 8):
                tassel_node = rt.getNodeByName(f"Tassel{i}")
                if tassel_node:
                    tassel_node.isHidden = True
        except Exception as e:
            QtWidgets.QMessageBox.warning(None, "Error hiding tassels", str(e))

    def validate_input(input_field, default_value):
        text = input_field.text().strip()
        if text == "":
            QtWidgets.QMessageBox.warning(None, "Invalid Input", f"Empty input. Default set to: {default_value}.")
            input_field.setText(str(default_value))
            return default_value
        try:
            value = float(text)
            if value <= 0:
                raise ValueError
            return value
        except ValueError:
            QtWidgets.QMessageBox.warning(None, "Invalid Input", f"Invalid value. Default set to: {default_value}.")
```

```
            input_field.setText(str(default_value))
            return default_value

        def validate_carpet_params(carpet_type, width, length, radius, height):
            if carpet_type == "Rectangle":
                if width <= 0 or length <= 0:
                    QtWidgets.QMessageBox.warning(None, "Invalid Input", "Width and Length must be
greater than zero.")
                    return False
            elif carpet_type == "Circle":
                if radius <= 0:
                    QtWidgets.QMessageBox.warning(None, "Invalid Input", "Radius must be greater than
zero.")
                    return False
            if height <= 0:
                QtWidgets.QMessageBox.warning(None, "Invalid Input", "Height must be greater than zero.")
                return False
            return True

        def run_ui():
            app = QtWidgets.QApplication.instance()
            if not app:
                app = QtWidgets.QApplication([])

            dialog = QtWidgets.QDialog()
            dialog.setWindowTitle("Create Carpet")
            layout = QtWidgets.QVBoxLayout()

            def add_input(label_text, default_value):
                label = QtWidgets.QLabel(label_text)
                input_field = QtWidgets.QLineEdit(str(default_value))
                input_field.setValidator(QtGui.QDoubleValidator(0.01, 9999.99, 2))
                layout.addWidget(label)
                layout.addWidget(input_field)
                return input_field, label

            carpet_type_label = QtWidgets.QLabel("Carpet Type:")
            carpet_type_combobox = QtWidgets.QComboBox()
            carpet_type_combobox.addItems(["Rectangle", "Circle"])
            layout.addWidget(carpet_type_label)
            layout.addWidget(carpet_type_combobox)

            width_input, width_label = add_input("Width (cm):", 100)
            length_input, length_label = add_input("Length (cm):", 150)
            height_input, _ = add_input("Height (cm):", 1)
            radius_input, radius_label = add_input("Radius (cm):", 50)

            material_label = QtWidgets.QLabel("Rug Material:")
            material_combobox = QtWidgets.QComboBox()
            material_combobox.addItems([
                "Carpet Low Beige", "Carpet Soft Rug Beige Pattern 1", "Carpet Soft Rug Dark Grey Pattern
1",
                "Carpet Soft Rug Light Olive Pattern 1", "Carpet Soft Rug Red Pattern 1",
                "Carpet Soft Rug Beige Pattern 2", "Carpet Soft Rug Dark Grey Pattern 2",
```

```
    "Carpet Soft Rug Light Olive Pattern 2", "Carpet Soft Rug Red Pattern 2",
    "Carpet Soft Rug Beige Pattern 3", "Carpet Soft Rug Dark Grey Pattern 3",
    "Carpet Soft Rug Light Olive Pattern 3", "Carpet Soft Rug Red Pattern 3",
    "Fabric Cotton", "Dave Round", "Nilocream"
])
layout.addWidget(material_label)
layout.addWidget(material_combobox)

add_tassels_checkbox = QtWidgets.QCheckBox("Add Tassels")
layout.addWidget(add_tassels_checkbox)

tassel_label = QtWidgets.QLabel("Select Tassel:")
tassel_combobox = QtWidgets.QComboBox()
tassel_combobox.addItems([f"Tassel {i}" for i in range(1, 8)])  # 7 tassel options
layout.addWidget(tassel_label)
layout.addWidget(tassel_combobox)

tassel_distance_input, tassel_distance_label = add_input("Tassel Distance (cm):", 10)

edge_option_label = QtWidgets.QLabel("Tassel Placement:")
edge_option_group = QtWidgets.QGroupBox()
edge_option_layout = QtWidgets.QVBoxLayout()
radio_all_edges = QtWidgets.QRadioButton("All Edges")
radio_front_back = QtWidgets.QRadioButton("Side Edges Only")
radio_all_edges.setChecked(True)
edge_option_layout.addWidget(radio_all_edges)
edge_option_layout.addWidget(radio_front_back)
edge_option_group.setLayout(edge_option_layout)
layout.addWidget(edge_option_label)
layout.addWidget(edge_option_group)

# Helper to show/hide tassel options
def update_tassel_visibility():
    is_rect = carpet_type_combobox.currentText() == "Rectangle"
    is_circle = carpet_type_combobox.currentText() == "Circle"

    if is_rect:
        tassels_enabled = add_tassels_checkbox.isChecked() and is_rect
        tassel_label.setVisible(tassels_enabled)
        tassel_combobox.setVisible(tassels_enabled)
        tassel_distance_label.setVisible(tassels_enabled)
        tassel_distance_input.setVisible(tassels_enabled)
        edge_option_label.setVisible(tassels_enabled)
        edge_option_group.setVisible(tassels_enabled)
    elif is_circle:
        tassels_enabled = add_tassels_checkbox.isChecked() and is_circle
        tassel_label.setVisible(tassels_enabled)
        tassel_combobox.setVisible(tassels_enabled)
        tassel_distance_label.setVisible(tassels_enabled)
        tassel_distance_input.setVisible(tassels_enabled)

    dialog.adjustSize()
    dialog.update()
```

```
def on_type_changed():
    is_rect = carpet_type_combobox.currentText() == "Rectangle"
    width_input.setVisible(is_rect)
    width_label.setVisible(is_rect)
    length_input.setVisible(is_rect)
    length_label.setVisible(is_rect)
    radius_input.setVisible(not is_rect)
    radius_label.setVisible(not is_rect)
    update_tassel_visibility()

carpet_type_combobox.currentIndexChanged.connect(on_type_changed)
add_tassels_checkbox.stateChanged.connect(update_tassel_visibility)
on_type_changed()

def on_create():
    global carpet_node
    height = validate_input(height_input, 1)
    carpet_type = carpet_type_combobox.currentText()
    material_index = material_combobox.currentIndex() + 1

    if carpet_type == "Rectangle":
        width = validate_input(width_input, 100)
        length = validate_input(length_input, 150)
        radius = None
        if not validate_carpet_params(carpet_type, width, length, radius, height):
            return
        else:
            create_carpet(width, length, height, carpet_type, material_index)
    else:
        radius = validate_input(radius_input, 50)
        width = None
        length = None
        if not validate_carpet_params(carpet_type, width, length, radius, height):
            return
        else:
            create_carpet(radius, 0, height, carpet_type, material_index)

    if carpet_type == "Rectangle" and add_tassels_checkbox.isChecked():
        tassel_index = tassel_combobox.currentIndex() + 1
        tassel_distance = validate_input(tassel_distance_input, 10)
        edge_option = "All Edges" if radio_all_edges.isChecked() else "Side Edges Only"
        load_tassel(tassel_index)
        distribute_tassels(carpet_node, tassel_distance, carpet_type, edge_option)

    elif carpet_type == "Circle" and add_tassels_checkbox.isChecked():
        tassel_index = tassel_combobox.currentIndex() + 1
        tassel_distance = validate_input(tassel_distance_input, 10)
        edge_option = "All Edges"
        load_tassel(tassel_index)
        distribute_tassels(carpet_node, tassel_distance, carpet_type, edge_option)

    dialog.accept()

create_button = QtWidgets.QPushButton("Create Carpet")
```

```
    create_button.clicked.connect(on_create)
    layout.addWidget(create_button)
    dialog.setLayout(layout)
    dialog.exec_()

run_ui()
```

Код скрипта заміни об’єктів типу бокс на 3D-моделі:

```
import pymxs

def get_object_dimensions(obj):
    rt = pymxs.runtime
    mesh = rt.snapshotAsMesh(obj)
    verts = [v for v in mesh.verts]

    if not verts:
        return 0, 0, 0

    xs = [v.position.x for v in verts]
    ys = [v.position.y for v in verts]
    zs = [v.position.z for v in verts]

    width = max(xs) - min(xs)
    length = max(ys) - min(ys)
    height = max(zs) - min(zs)

    return width, length, height

def replace_box_with_table_model():
    rt = pymxs.runtime

    if rt.selection.count == 1 and rt.classof(rt.selection[0]) == rt.Box:
        selected_box = rt.selection[0]

        box_width = selected_box.width
        box_length = selected_box.length
        box_height = selected_box.height

        table_model_path = rt.getOpenFileName(
            caption="Select 3ds Max File",
            types="3ds Max Files (*.max)|*.max"
        )

        if table_model_path:
            pre_objects = set(rt.objects)
            rt.mergeMAXFile(table_model_path)

            post_objects = set(rt.objects)
            new_objects = list(post_objects - pre_objects)

            if new_objects:
                table_node = new_objects[-1]

                table_width, table_length, table_height = get_object_dimensions(table_node)
```

```
            if table_width == 0 or table_length == 0 or table_height == 0:
                print("Error: Could not determine table dimensions.")
                return

            table_node.scale = rt.Point3(
                box_width / table_width,
                box_length / table_length,
                box_height / table_height
            )

            table_node.position = selected_box.position
            rt.delete(selected_box)

            print("Box replaced with model successfully.")
        else:
            print("Error: No new objects found after merge.")
    else:
        print("No file selected.")
else:
    print("Error: Please select exactly one Box object.")

replace_box_with_table_model()
```