

**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: Автоматизація процесу створення 3D-моделей у Blender за допомогою скриптів мовою Python

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи ЦДБ-21-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Технології цифрового дизайну»


(підпис)

Давидова Дмитра Євгеновича

(прізвище, ім'я, по батькові)

Науковий керівник


(підпис)

Гладченко Оксана Вікторівна,
к.пед.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми


(підпис)

Ратушняк Тетяна Володимирівна
к.ф.-м.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних
технологій і систем протокол від «13» травня 2025 № 18

Завідувач кафедри


(підпис)

Філоненко Михайло
Миколайович, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Реєстрація № 76

29.05.2025

(дата)



(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

29.05.2025

(дата)

М.М. Філоненко

(підпис)

Результат захисту

4 (добре)
(оцінка)

11.06.2025

(дата)



(підпис)

Голова комісії



(підпис)

В.М. Франчук

Члени комісії:



(підпис)

М.М. Філоненко



(підпис)

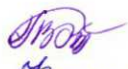
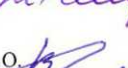
Т.В. Ратушняк



(підпис)

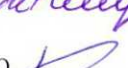
В.О. Ніжегородцев

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	21		Гладченко О.В.  Франчук В.М. 
2. Дотримання вимог щодо оформлення	від 0 до 10	8		Філоненко М.М.  Ратушняк Т.В. 
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		Ніжегородцев В.О.  Суховерхівка Л.П. 
4. Рецензент	від 0 до 5	3		
РАЗОМ БАЛІВ	від 0 до 50	36		

Оцінювання кваліфікаційної роботи науковим керівником – 34 балів.

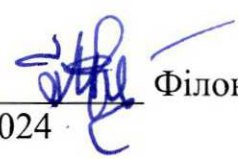
ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
70	C	4 (добре)	11.06.2025	Франчук В.М.  Філоненко М.М.  Ратушняк Т.В.  Ніжегородцев В.О.  Суховерхівка Л.П. 

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Технології цифрового дизайну»

Затверджую
Завідувач кафедри  Філоненко М.М.
«14» листопада 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Давидову Дмитру Євгеновичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Автоматизація процесу створення 3D-моделей у Blender за допомогою скриптів мовою Python.

Затверджена наказом ректора від «13» листопада 2024 № 1717.

2. Строк здачі закінченої роботи «12» травня 2025.

3. Мета роботи Дослідження та практичне застосування мови програмування Python для автоматизації процесів у середовищі Blender з метою підвищення продуктивності та створення нових інструментів.

Об'єкт дослідження Програмне забезпечення Blender як середовище для створення та модифікації 3D-моделей.

Предмет дослідження Застосування мови Python та API Blender для автоматизації, розробки скриптів, оптимізації роботи Blender.

Орієнтовна база дослідження Blender як платформа для розробки, включаючи його API, офіційна документація та приклади використання Blender Python.

4. Вихідні дані до роботи Офіційна документація Blender API, середовище Blender (з відкритим кодом), приклади скриптів на Python, теоретичні відомості про 3D-моделювання, інформація про оптимізацію рендерингу (OpenGL, Vulkan), методи тестування та документування коду.

5. Перелік графічного матеріалу Скриншоти інтерфейсу Blender, графічні ілюстрації, створені за допомогою Python-скриптів у Blender, візуалізації математичних алгоритмів, схеми та діаграми з технічної літератури та онлайн-ресурсів, матеріали, отримані через пошукові системи та офіційну документацію.

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)
ВСТУП. РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ ТА АСПЕКТИ ОСНОВ 3D-МОДЕЛЮВАННЯ. РОЗДІЛ 2. МАТЕМАТИЧНА ОСНОВА РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СКРИПТІВ У BLENDER. ВИСНОВКИ. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 30.10.2024	До 30.10.2024
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	25.12.2024	25.12.2024
3	II розділ (аналіз експериментальної частини про методики досліджень)	15.02.2025	15.02.2025
4	III розділ (проведені теоретичні та (або) експериментальні дослідження та проєктні рішення)	14.04.2025	14.04.2025
5	Висновки	01.05.2025	01.05.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	09.05.2025	09.05.2025
7	Отримання відгуку керівника	03.06.2025	03.06.2025
8	Отримання зовнішньої рецензії	05.06.2025	05.06.2025

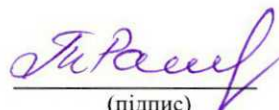
8. Дата видачі завдання «14» листопада 2024

9. Керівник кваліфікаційної роботи


(підпис)

Гладченко
(прізвище, ініціал)

10. Керівник освітньо-професійної програми


(підпис)

Ратушняк Т
(прізвище, ініціал)

11. Завдання до виконання отримав
здобувач вищої освіти


(підпис)

Давидов Д
(прізвище, ініціал)

АНОТАЦІЯ

Автоматизація процесу створення 3D-моделей у Blender за допомогою скриптів мовою Python

Метою роботи є дослідження та практичне застосування мови програмування Python для автоматизації процесів у середовищі Blender з метою підвищення продуктивності та створення нових інструментів.

Основними завданнями роботи є: аналіз базових понять 3D-моделювання та автоматизації в Blender; вивчення архітектури API Blender і особливостей його застосування; розробка скриптів Python для генерації та видалення об'єктів; розкриття математичної сутності; тестування, оптимізація та документування скриптів.

Предмет дослідження – методи автоматизації процесів 3D-моделювання в Blender за допомогою мови програмування Python.

Об'єкт дослідження – програмне забезпечення Blender як середовище для створення та модифікації 3D-моделей.

Базою дослідження є офіційна документація Blender API, приклади скриптів, графічні інтерфейси Blender, а також ресурси з теорії 3D-моделювання.

У результаті досліджень сформульовано підходи до оптимізації 3D-моделювання шляхом автоматизації типових завдань та представлено скрипти, що реалізують ці підходи.

Отримані результати можуть бути використані при розробці аддонів і автоматизації у Blender, як у навчальних цілях, так і в практичній діяльності.

Кваліфікаційна робота містить 50 сторінок, 31 рисунок, список використаних джерел – 43 найменування.

Ключові слова: 3D-моделювання, автоматизація, Blender, Python, API, скрипти, інтерполяція.

Рік виконання кваліфікаційної роботи: 2025.

Рік захисту: 2025.

ABSTRACT

Automating the process of creating 3D models in Blender using Python scripts

Objective: the aim of the work is to study and practically apply the Python programming language to automate processes in the Blender environment in order to increase productivity and create new tools.

The main tasks of the work are: analysis of the basic concepts of 3D modeling and automation in Blender; studying the architecture of the Blender API and the features of its application; development of Python scripts for generating and deleting objects; disclosure of the mathematical essence; testing, optimization, and documentation of scripts.

Subject of research – methods of automating 3D modeling processes in Blender using the Python programming language.

The object of research is Blender software as an environment for creating and modifying 3D models.

The basis of the research is the official documentation of the Blender API, sample scripts, Blender graphical interfaces, as well as resources on the theory of 3D modeling.

As a result of the research, approaches to optimizing 3D modeling by automating typical tasks have been formulated and scripts implementing these approaches have been presented.

The results and recommendations obtained can be used in the development of add-ons and automation in Blender, both for educational purposes and in practice.

The qualification work contains 50 pages, 31 figures, and a list of references includes 43 items.

Keywords: 3D modeling, automation, Blender, Python, API, scripts, interpolation.

Year of completion of the qualification work: 2025.

Year of defense: 2025.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ.....	3
ВСТУП	4
РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ ТА АСПЕКТИ ОСНОВ 3D МОДЕЛЮВАННЯ	6
1.1. Ознайомлення з базовими теоретичними поняттями.....	6
1.2. Обґрунтування вибору мови програмування та оптимізація параметрів продуктивності Blender	15
1.3. Розробка та методи тестування скриптів, ігровий движок та постановка задачі.....	18
РОЗДІЛ 2. МАТЕМАТИЧНА ОСНОВА	24
2.1. Математичні параметри автоматичного створення контенту у скриптах	24
2.2. Анімаційні формули в Python	26
2.3. Метод скінченних елементів (МСЕ) та налаштування середовища розробки	30
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СКРИПТУ	33
3.1. Методи тестування скрипту	33
3.2. Аналіз та тестування роботи скрипту	34
ВИСНОВКИ.....	47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	48

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API (*Application Programming Interface*) – інтерфейс програмування застосунків, набір засобів для взаємодії з програмним забезпеченням.

Blender API – інтерфейс програмування, що дозволяє керувати функціями Blender за допомогою мови Python.

GUI (*Graphical User Interface*) – графічний інтерфейс користувача.

3D – тривимірний простір або об'єкт, що має висоту, ширину й глибину.

MSE – метод скінченних елементів, чисельний метод розв'язання диференціальних рівнянь.

OpenGL (*Open Graphics Library*) – кросплатформний інтерфейс програмування для рендерингу 2D та 3D графіки.

Vulkan – низькорівневий графічний API нового покоління, що забезпечує високопродуктивну візуалізацію.

L-система (*Lindenmayer System*) – формальний граматичний підхід для генерації фракталів і природних структур.

Пустешки – це об'єкти типу EMPTY. Вони не мають геометрії (тобто, їх не видно при рендері), але виконують важливу допоміжну роль у 3D-сцені.

RAM – оперативна пам'ять

ВСТУП

Розвиток сучасних цифрових технологій, зокрема у сфері 3D-моделювання, вимагає все більшої ефективності, точності та гнучкості. Одним із провідних інструментів у галузі тривимірної графіки є Blender – безплатне програмне забезпечення з відкритим кодом, що широко використовується в анімації, геймдизайні, архітектурі та цифровому мистецтві. Його відкритість дозволяє користувачам створювати власні інструменти, аддони й автоматизовані процеси мовою Python [4; 5; 11], що суттєво підвищує продуктивність праці.

Актуальність теми. Сучасні проекти вимагають швидкої генерації великої кількості подібних об'єктів або багаторазового виконання однотипних операцій. Автоматизація процесів у Blender дозволяє значно зменшити витрати часу й зусиль, підвищити якість моделювання та знизити вірогідність помилок. Враховуючи підтримку з боку таких компаній як Epic Games, Nvidia, Intel, Blender активно розвивається, а отже потреба в гнучких інструментах автоматизації постійно зростає.

Стан і ступінь розробки проблеми в спеціальній літературі. У вітчизняній і закордонній літературі активно досліджуються питання автоматизації у 3D-середовищах, зокрема створення скриптів для Blender. Серед закордонних авторів, які здійснили внесок у розвиток тематики, можна відзначити розробників Blender Foundation, фахівців з procedural generation, Python API та прикладного програмування. Серед українських дослідників варто виділити Бурду Ю. І., який активно висвітлює питання комп'ютерної графіки та 3D-моделювання у навчальних посібниках; Чепурнову О. В і Пилипенка І.В., які аналізують застосування Python у навчанні й автоматизації; Воронова В. Ю., який спеціалізується на практичних аспектах моделювання в Blender; а також Рибака І. В. – автора низки публікацій з програмування та скриптових технологій для 3D-середовищ. Однак недостатньо висвітленими у вітчизняному просторі залишаються питання розробки практичних, прикладних скриптів для новачків.

Мета дослідження. Метою даної кваліфікаційної роботи є розробка та демонстрація засобів автоматизації створення 3D-моделей у Blender за допомогою мови програмування Python із застосуванням базових і доступних алгоритмів, які можуть бути використані навіть початківцями.

Завданнями роботи є:

- аналіз базових понять 3D-моделювання та автоматизації в Blender;
- вивчення архітектури API Blender і особливостей його застосування;
- розробка скриптів Python для видалення об'єктів;
- розкриття математичної сутності;
- тестування, оптимізація та документування скриптів.

Об'єкт дослідження. Програмне забезпечення Blender як середовище створення та модифікації 3D-моделей.

Предмет дослідження. Використання мови Python та Blender API для автоматизації, розробки скриптів і оптимізації процесів моделювання.

Методи дослідження. У роботі застосовано аналітичні, індуктивні та порівняльні методи дослідження, а також методи візуалізації та тестування програмного забезпечення.

База дослідження. Основою дослідження є середовище Blender, його офіційна документація, Python API, ресурси на платформах GitHub [11], Stack Overflow [38], документація OpenGL [27] та Vulkan [19].

Наукова та практична значущість. Практична значущість полягає у створенні скриптів, що можуть бути використані як навчальний інструмент або як основа для створення власних аддонів і автоматизації процесів. Наукова новизна полягає у поєднанні теоретичних методів побудови геометрії з практичним застосуванням у Blender.

Особистий внесок здобувача. Усі етапи розробки, тестування, написання коду та оформлення документації виконані здобувачем самостійно. Ідеї щодо використання параметричної генерації та модифікації об'єктів реалізовані на прикладі власноруч створених скриптів.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ ТА АСПЕКТИ ОСНОВ 3D МОДЕЛЮВАННЯ

1.1. Ознайомлення з базовими теоретичними поняттями

Автоматизація – це процес переходу від ручного виконання завдань до використання технічних засобів, програмного забезпечення або алгоритмів, які здатні здійснювати ті ж самі операції автоматично, без постійної участі людини. У сфері інформаційних технологій автоматизація охоплює широкий спектр задач – від рутинних офісних процесів до складного керування виробничими лініями або моделювання складних цифрових середовищ.

Основною метою автоматизації є підвищення продуктивності, точності, надійності та швидкості виконання завдань. Вона дозволяє зменшити людський фактор, мінімізувати помилки, а також значно скоротити час на виконання рутинних чи повторюваних операцій.

У комп'ютерній графіці, зокрема в процесі створення 3D-моделей, автоматизація дозволяє значно оптимізувати робочі процеси, прискорити генерацію моделей, сценаріїв анімації, модифікацій об'єктів тощо. Особливо це актуально при створенні великої кількості подібних об'єктів або при виконанні однотипних дій з різними параметрами.

У сучасній комп'ютерній графіці автоматизація має критичне значення, особливо при роботі з великою кількістю об'єктів, матеріалів, анімаційних сценаріїв. Візуальні редактори, такі як Blender, підтримують автоматизацію за допомогою мови програмування Python, що дозволяє створювати скрипти для: автоматичного створення 3D об'єктів за заданими параметрами; пакетної обробки файлів або сцен; створення процедурної геометрії; налаштування матеріалів, текстур і освітлення; автоматизації рендерингу; імпорту/експорту об'єктів; інтеграції з іншими системами (наприклад, базами даних або вебсервісами).

Таке програмне керування процесом дає змогу не тільки прискорити роботу, а й забезпечити відтворюваність результатів, що є особливо важливим у проєктах, де застосовується конвеєрна обробка.

Тривимірне моделювання (3D моделювання) – це процес створення цифрового подання об'єктів у тривимірному просторі за допомогою комп'ютерного програмного забезпечення. Воно базується на використанні примітивів (базових геометричних форм), таких як куби, сфери, циліндри, конуси тощо, які згодом модифікуються через різноманітні трансформації (масштабування, обертання, деформації), або поєднуються у складніші структури. Користувач активує ці функції, зазвичай вони функціонують, натиснувши комбінацію клавіші на клавіатурі.

Зараз існує багато потужного програмного забезпечення для моделювання 3D, яке дозволяє створювати 3D-активи, спецефекти та візуалізацію зображень. Найпопулярнішими платними програмами є Autodesk Maya, Autodesk 3ds Max і Cinema 4D. Також доступно багато безплатних програм, найпопулярнішою з них є Blender.

Blender – це набір інструментів для створення тривимірної комп'ютерної графіки з відкритим кодом. Він написаний на мовах програмування C, C++ і Python (рис. 1.1). Інтерфейс Blender працюватиме як середовище розгортання та розробки для нашого програмного забезпечення.

Велика частина мети праці – ознайомитися з унікальними міркуваннями щодо програмування та тестування Python, залишаючись в інтерфейсі Blender. Впровадження скриптів в Blender дозволяє розширювати функціональність програми, додаючи нові інструменти, функції, ефекти та можливості до базового функціоналу програми. Це може значно полегшити роботу користувачів, додавши нові можливості та функції, які не доступні в стандартному наборі інструментів Blender.

Автоматизуючи рутинні завдання в Blender, користувач зменшує час та зусилля, потрібні для виконання певних дій. Актуальність даної праці полягає в

тому, що однією з головних особливостей Blender є його відкритий код, що дає можливість розширювати функціональність програми.

Blender Foundation отримала велику фінансову підтримку від Epic Games, Nvidia та Intel завдяки заснуванню Фонду розвитку Blender. Це дозволило компанії наймати нових членів команди та в результаті розвиватись швидше.

Blender Foundation є некомерційною організацією, яка відповідає за розробку Blender. Також блендер розроблений спільнотою, яка створює додаткові плагіни написані на Python (так звані addons). Аддони додають нові можливості або покращують функціональність Blender. Вебсайт blender.org популярніший, ніж будь-коли, він і кілька його субдоменів отримали разом 23 мільйони унікальних відвідувачів. Це на 35% більше, ніж минулого року, і наближається до 2 мільйонів відвідувачів на місяць.

Інтерфейс Blender складається з кількох ключових елементів, які забезпечують зручну та ефективну роботу над 3D проєктами (рис. 1.1): Topbar, Workspaces, Outliner, Properties Editor, Timeline, Status Bar, Sidebar, Toolbar.

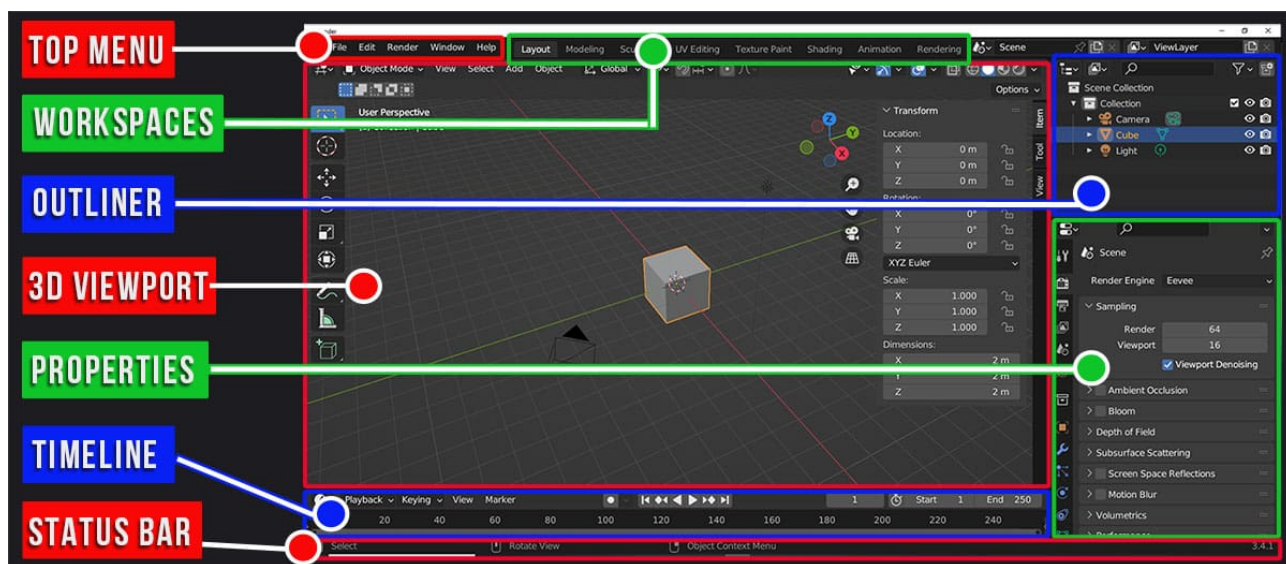


Рис. 1.1 Головний інтерфейс програми Blender

Торбар (верхня панель): розташована у верхній частині вікна, містить основне меню з такими пунктами, як File, Edit, Render, Window, Help. Тут можна

відкривати та зберігати файли, налаштовувати параметри рендерингу, змінювати налаштування програми тощо.

Workspaces (робочі простори): під верхньою панеллю знаходиться ряд вкладок, кожна з яких відкриває окремий робочий простір, налаштований для певного завдання: Layout, Modeling, Sculpting, UV Editing, Texture Paint, Shading, Animation, Rendering, Compositing, Scripting. Це дозволяє швидко перемикатися між різними етапами роботи.

3D Viewport (тривимірне вікно перегляду): центральна частина інтерфейсу, де відбувається основна робота з 3D об'єктами. Тут можна моделювати, скульптувати, анімувати та переглядати сцену з різних кутів. Вікно підтримує різні режими перегляду та інструменти для взаємодії з об'єктами.

Outliner (ієрархія сцени): розташований зазвичай у верхньому правому куті, цей редактор показує структуру сцени у вигляді дерева. Тут можна переглядати, вибирати, приховувати та організовувати об'єкти сцени, а також керувати їхніми властивостями.

Properties Editor (редактор властивостей): знаходиться під Outliner'ом, містить вкладки для налаштування активного об'єкта, матеріалів, модифікаторів, світла, камери та інших параметрів сцени. Це основне місце для детального налаштування властивостей об'єктів.

Timeline (Шкала часу): розташована в нижній частині інтерфейсу, дозволяє керувати анімацією, додаючи ключові кадри та переміщаючись по часовій шкалі. Це важливий інструмент для створення та редагування анімацій.

Status Bar (рядок стану): найнижча частина вікна, де відображається інформація про поточні дії, підказки щодо клавіш швидкого доступу та інші корисні повідомлення.

Sidebar (бічна панель): викликається клавішею N у 3D Viewport, містить додаткові інструменти та налаштування, такі як трансформації об'єктів, інструменти для моделювання та інші параметри, залежно від активного режиму.

Toolbar (Панель інструментів): викликається клавішею T у 3D Viewport, містить інструменти для взаємодії з об'єктами, такі як переміщення, обертання, масштабування, а також інші інструменти, залежно від активного режиму.

Комбінація клавіш Shift + A в середовищі Blender є однією з ключових і базових операцій, що безпосередньо впливає на швидкість та ефективність створення тривимірних сцен. З погляду функціональності, дана команда активує контекстне меню «Add», яке дозволяє користувачеві швидко додати в сцену нові об'єкти різного типу – від базових примітивів до спеціалізованих структур, таких як: кісткові системи, джерела світла, камери, фізичні поля та інстанси колекцій.

З погляду структури цифрового виробництва, меню «Add» (рис. 1.2) є відправною точкою більшості процесів, що стосуються 3D-моделювання, анімації, шейдингу, симуляцій і рендерингу. Саме з використанням цієї команди розпочинається створення нової геометричної форми, яка надалі може бути модифікована, текстурована, анімована або інтегрована в складнішу сцену.

У професійних пайплайнах ця функція має значення також з погляду продуктивності та уніфікації робочого процесу. Комбінація Shift + A забезпечує уніфікований доступ до великої кількості об'єктів незалежно від активного робочого простору (Layout, Modeling, Sculpting, Animation тощо), що спрощує адаптацію користувачів до нових задач і прискорює робочий ритм.

Крім того, з огляду на архітектуру Blender як нелінійного інструменту, контекстне додавання об'єктів через Shift + A дозволяє автоматично враховувати активну точку вставки (3D Cursor), що забезпечує точність у розташуванні нових елементів.

Таким чином, комбінація Shift + A є не лише технічною зручністю, але й відіграє концептуальну роль у дизайні програмного забезпечення Blender, виступаючи інтерфейсним мостом між користувачем і ядром 3D-середовища.

Найголовніша опція контекстного меню «Add» – Mesh (Сітка). Це найчастіше використовувані 3D-об'єкти, з яких будуються моделі. Підменю містить: Plane – плоска площина (2D); Cube – стандартний куб; Circle – плоске коло без наповнення; UV Sphere – куля з рівномірною сіткою, схожа на глобус;

Icosphere – куля, побудована з трикутників (ікосаедрична); Cylinder – циліндр; Cone – конус; Torus – бублик/тор; Grid – сітка (багато полігонів); Monkey (Suzanne) – 3D-голова мавпи для тестування матеріалів та рендеру. Round Cube, Quad Sphere (у новіших версіях) – форми для розширеного моделювання.

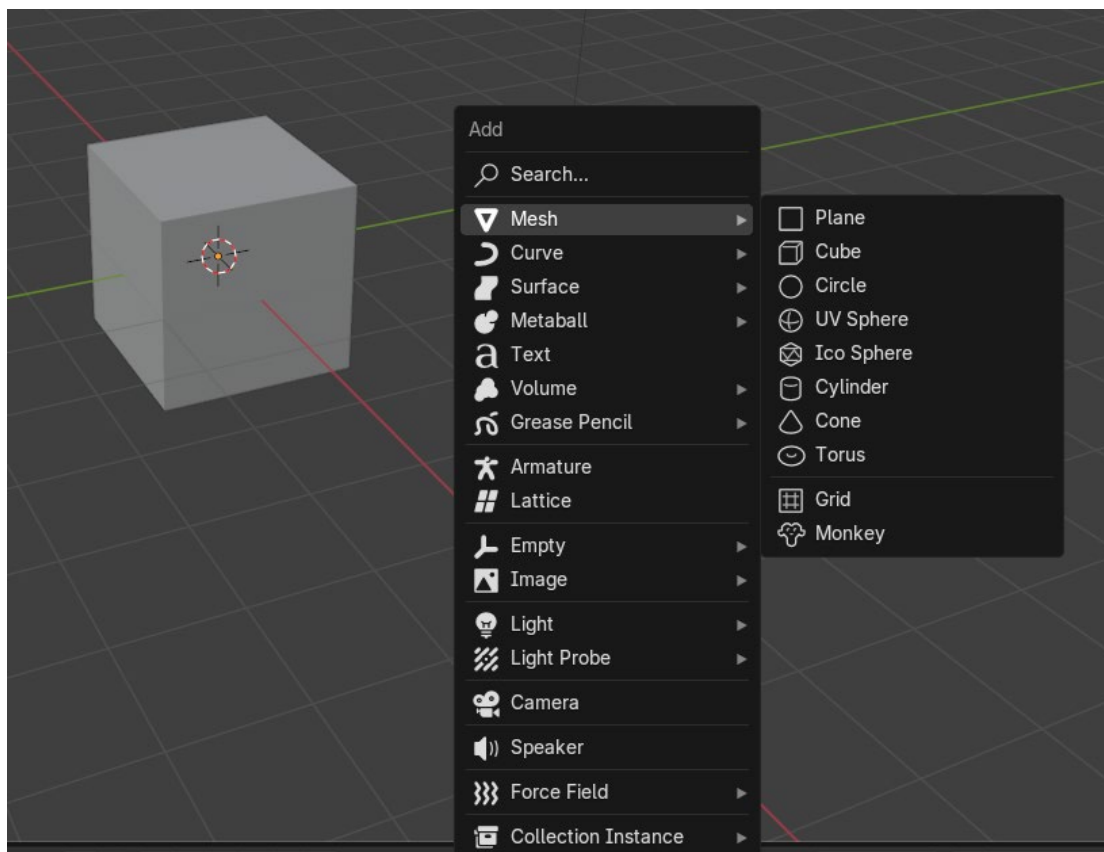


Рис. 1.2 Контекстне меню «Add» і 3D об'єкти опції Mesh

Також коротко про інші опції: «Curve» – криві (Bezier, Circle тощо); «Surface» – поверхні NURBS; «Metaball» – об'єкти зі злиттям форм; «Text» – 3D-текст; «Volume» – томні об'єкти (наприклад, для диму); «Grease Pencil» – об'єкти для 2D-малювання; «Armature» – кістки для анімації; «Lattice» – решітки для деформацій; «Empty» – об'єкти, які не рендеряться, але використовуються як прив'язки; «Image» – додавання зображень у вигляді plane; «Light» – джерела світла (Point, Sun, Spot, Area); «Light Probe» – для роботи з відбиттями та освітленням сцени; «Camera» – камера для рендерингу; «Speaker» – джерело звуку; «Force Field» – фізичні сили (вітри, магніти тощо); «Collection Instance» – інстанси груп об'єктів.

Розділ Scripting (рис. 1.3) в Blender є спеціалізованим середовищем для програмування за допомогою мови Python, що дозволяє створювати власні скрипти та

автоматизувати різні аспекти роботи з 3D сценами. Ось кілька основних особливостей цього розділу: Python console, Text Editor, Scripts and Addons.

Python Console – це інтерактивний термінал, де можна вводити команди Python для маніпулювання об'єктами, змінювати параметри сцени та тестувати різні скрипти. Консоль дозволяє в реальному часі побачити результат виконання команд, що особливо корисно для тих, хто пише або тестує код безпосередньо в Blender.

В Text Editor ви можете писати, зберігати та виконувати скрипти Python. Цей редактор дає змогу створювати складніші програми для автоматизації процесів або додавання нових функцій до Blender.

В редакторі доступні функції підсвічування синтаксису та автодоповнення, що полегшує написання коду.

В Blender можна створювати Scripts and Addons(додаткові модулі), які розширюють стандартний функціонал програми.

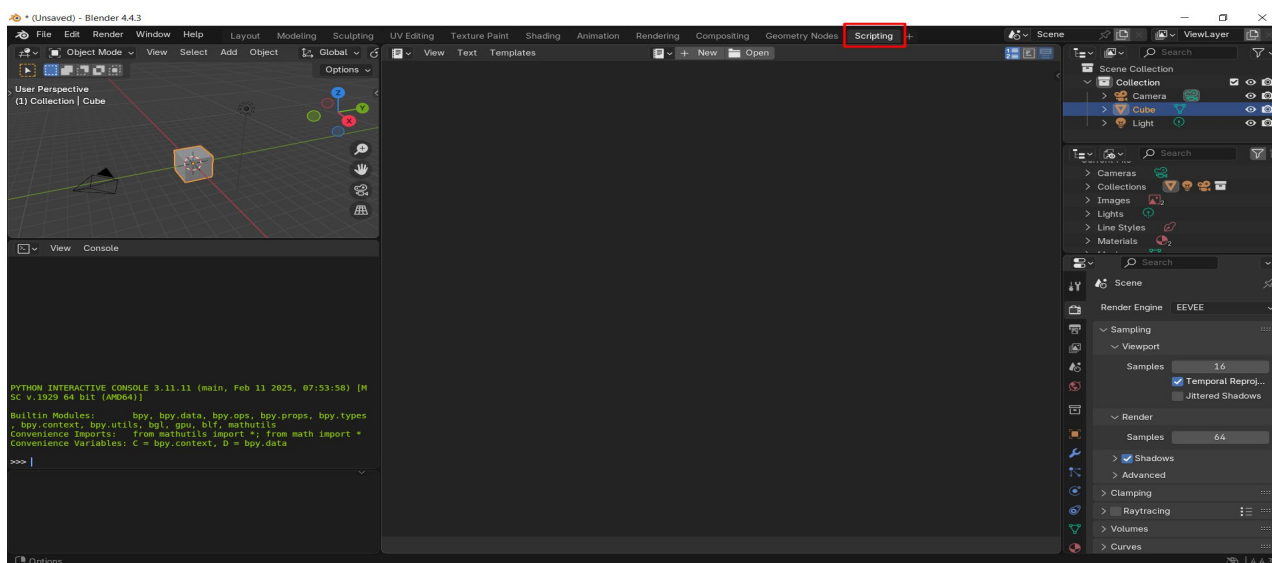


Рис. 1.3 Інтерфейс розділу Scripting

Всі ці додатки зазвичай пишуться на Python і можуть взаємодіяти з інтерфейсом Blender. Аддони можуть бути як простими (наприклад, додавання нових інструментів), так і складними (наприклад, автоматизовані процеси анімації чи рендерингу).

В Blender є потужне Python API (інтерфейс програмування додатків), який надає доступ до майже всіх аспектів програми [4; 5; 11]. Це включає доступ до

об'єктів сцени, матеріалів, камер, світла, анімації тощо. Документація API дуже важлива для тих, хто хоче розширити можливості Blender, використовуючи Python. Вона містить всі доступні класи, функції та методи для програмування.

Завдяки scripting ви можете автоматизувати та кастомізувати повторювані процеси в Blender. Наприклад, можна створити скрипт для імпорту моделей, налаштування освітлення або матеріалів для кількох об'єктів одночасно. Користувачі можуть також створювати інтерфейси для своїх скриптів, що дозволяє додавати спеціалізовані панелі або вікна для зручнішої взаємодії.

Використовуючи Python в Blender, можна також інтегрувати Blender з іншими програмами, такими як Maya, 3ds Max, Unity чи Unreal Engine для створення ігрових сцен або анімацій [3; 39; 40].

Прямий доступ до інтерфейсу Blender: Python у Blender дозволяє створювати нові елементи інтерфейсу, такі як кнопки, поля введення тощо. Це дозволяє користувачам створювати специфічні інструменти або панелі, що значно полегшують роботу.

Scripting дає змогу маніпулювати даними, а також створювати складні анімації на основі скриптів. Наприклад, можна автоматично змінювати параметри об'єктів, додавати ключові кадри або навіть створювати складні анімаційні системи.

Процес створення 3D-моделі в середовищі Blender або аналогічному програмному забезпеченні передбачає кілька ключових етапів:

1. Моделювання – це початковий етап, на якому формується базова геометрія об'єкта. Основними техніками є: полігональне моделювання – створення об'єкта з використанням вершин, ребер і граней; скульптинг – "ліплення" високодеталізованої геометрії на основі цифрового аналога глини; NURBS-моделювання – використання кривих для формування згладжених поверхонь.

2. Ретопологія – це процес оптимізації геометрії моделі для анімації чи інтерактивного використання, коли початкове моделювання здійснювалося методом скульптингу.

3. Розгортка UV: на цьому етапі виконується проєкція поверхні 3D-моделі на 2D-площину для подальшого накладання текстур. Важливість коректної UV-розгортки обумовлена точністю відображення текстурного зображення на поверхні моделі.

4. Текстурування та шейдинг: визначаються матеріали моделі, застосовуються текстури: колір, відбивання, шорсткість, нормалі тощо. У Blender це здійснюється через Shader Editor з використанням нодової системи [9; 15; 18].

5. Освітлення та рендеринг: модель розміщується в сцені з відповідним освітленням, після чого виконується рендеринг (візуалізація) – процес перетворення 3D-сцени у 2D-зображення. Blender використовує такі рушії рендерингу, як Eevee (реального часу) та Cycles (фотореалістичний трасувальник променів) [4; 14; 24].

Щось подібне до зв'язку Blender та API Blender Python рідко зустрічається у світі розробки програмного забезпечення. Для платформ з API характерно розглядати користувачів та розробників як окремі категорії з окремими: інструментами, інтерфейсами та цілями. Blender, з іншого боку, розмиває межі між розробниками та користувачами, дозволяючи користувачам легко виступати в ролі розробників і навпаки.

Тісний взаємозв'язок між розробниками та користувачами є результатом мудрих проєктних рішень, прийнятих основною командою розробників Blender на ранніх етапах. Перед тим, як у серпні 2003 року Blender було випущено як вільне програмне забезпечення з відкритим вихідним кодом версії 2.26, основна команда розробників опублікувала документацію для Python API [4; 5] для тодішньої преміумверсії 2.25. Python 2.0 щойно вийшов у жовтні 2000 року, а Blender вже використовував його для керування викликами з інтерфейсу до структур даних на C-рівні. Це дозволило розробникам взаємодіяти з грою в режимі реального часу. На початку 2010 року художники Blender'a все більше усвідомлювали вплив скриптів Python на процес моделювання. Певні додатки мали стати «must-have» для художників, зацікавлених у певних сферах. Розробники інших програм для 3D моделювання побачили можливість

розробити експортери, які б портували Blender до їхнього програмного забезпечення.

1.2. Обґрунтування вибору мови програмування та оптимізація параметрів продуктивності Blender

Python – потужна мова програмування з великою спільнотою розробників і багатою екосистемою бібліотек, які можна використовувати для розширення функціональності Blender.

Мови програмування C/C++ також використовується для виробництва Addonshis, дозволяє дизайнерам тонко налаштувати графічну систему блендер за допомогою чистого сценарію. Blender також підтримує LUA, що дозволяє розробникам писати сценарії цією мовою. Однак мова сценаріїв Python є найпоширенішим та рекомендованим варіантом для сценаріїв блендер. C/C++ є надійнішим, ніж Python, і прості завдання потребують більшого вихідного коду. Працюючи з інтерфейсом Blender та його функціями, цей вибір піддається збільшенню обмежень у C/C++.

Обробка подій та відповідь користувачів – це два аспекти, які повинні бути ввімкненими вручну. Крім того, створення розширень C/C++ потребує більшого розуміння та досвіду програмування, ніж Python [2]. Це також займе більше часу та ресурсів (рис. 1.4)

Why Python?

C++	Python
<pre> 1 2#include<iostream> 3using namespace std; 4 5int main (int argc, int *argv[]) 6{ 7 cout << "Hello World" << endl; 8 return 0; 9} </pre>	<pre> 1 2 print "Hello World" </pre>
• Complex syntax • Difficult to read	• Minimal Syntax • Easier to read and debug • Faster development time • Increases productivity

4

Рис. 1.4 Порівняння синтаксису C++ та Python

Синтаксис мови Python вважається простішим і легшим для розуміння, ніж мови C/C++. Python має менше синтаксичних вимог, таких як дужки, фігурні дужки та крапки з комою, що полегшує використання для початківців. Що стосується використання розширених бібліотек, Python відомий своїми потужними бібліотеками, такими як NumPy, Pandas, OpenCV і TensorFlow, які можуть ефективно виконувати різні завдання, такі як обробка даних, машинне навчання та наукові обчислення.

Blender – потужний інструмент для творення 3D-моделей, анімацій та візуалізацій. Водночас, подібно до будь-якого програмного забезпечення, Blender здатен стикатися з проблемами продуктивності при роботі з великими проєктами або на малопотужних комп'ютерах.

Одним з аспектів, що дозволяє налаштувати продуктивність у Blender, є конфігурація OpenGL. Налаштування OpenGL впливає на якість візуалізації сцен у Blender і може впливати на загальну ефективність роботи програми. Наприклад, увімкнення проміжного буфера здатне прискорити процес візуалізації сцени Blender, у той час, як застосування візуалізації локального освітлення може подовжити час рендерингу.

OpenGL являє собою багатоплатформний API, призначений для створення 2D та 3D графіки [19; 27]. Він забезпечує програмістам набір функцій для рендерингу графічних об'єктів на екрані. OpenGL – це кросплатформна мова програмування. Як і у випадку з DirectX, API спрощує розробку графічних програм та програмного забезпечення, оскільки їх потрібно адаптувати тільки до стандарту OpenGL, а не до різних операційних систем та вбудованого графічного обладнання.

Стандарт OpenGL визначає близько 250 команд, а інші організації, зокрема виробники графічних карт, мають змогу визначати власні (тобто, пов'язані з виробником) розширення.

У березні 2015 року API Vulkan був представлений як наступник OpenGL на Конференції розробників ігор [19]. Інтерфейс програмування, спочатку

відомий як «Next Generation OpenGL» або «glNext», є відкритим вихідним кодом та також кросплатформним. Відмінність від OpenGL полягає в тому, що програмування більшою мірою зосереджене на апаратному забезпеченні, що суттєво збільшує обчислювальну потужність. Деякі комп'ютерні ігри вже підтримують Vulkan, але більшість використовують DirectX. Vulkan також розробляється групою Khronos.

Для тестування OpenGL може виявитися корисним застосування програми `glxgears` та таких ігор, як `tuxracer` та `armagetron` (пакети мають однакову назву). За умови активації підтримки 3D, ви матимете змогу безперебійно запускати ці три програми на відносно новому комп'ютері. Без підтримки 3D, ці ігри працюватимуть дуже повільно (ефект слайд-шоу). Використовуйте команду `glxinfo`, щоб перевірити активність 3D. Якщо так, вивід міститиме рядок із `direct rendering: Yes` (Прямий рендеринг: Так).

API Blender (інтерфейс прикладного програмування) – це набір функцій та методів, що надаються Blender для взаємодії з його функціями за допомогою програмного коду. API Blender доступний мовами програмування Python та C++. Використовуючи API Python, користувачі здатні створювати сценарії та плагіни для розширення функціональності Blender, автоматизації процесів та спрощення завдань. C++ API часто застосовується для розробки плагінів та складніших інструментів.

Застосування API Blender та OpenGL спільно може бути корисним для створення складніших програм, що потребують значних графічних можливостей [4; 5; 27]. Наприклад, можна використати API Blender для роботи з даними сцени та передавати їх до OpenGL для відображення на екрані. Таким чином, можна створювати якісні та реалістичні 3D-сцени з точнішим управлінням параметрами рендерингу. Іншим параметром продуктивності, який піддається налаштуванню в Blender, є конфігурування циклу рендерингу. Налаштування циклу рендерингу впливає на якість і швидкість рендерингу зображень у Blender. Оптимізація параметрів продуктивності Blender здатна

суттєво підвищити ефективність роботи програми та прискорити процес створення проєктів.

1.3. Розробка та методи тестування скриптів, ігровий движок та постановка задачі

Скрипти для автоматизованого створення моделей являють собою процес програмування у середовищі Blender, спрямований на автоматичну генерацію тривимірних об'єктів. Такий підхід є особливо ефективним при моделюванні однотипних або повторюваних конструктивних елементів, зокрема зубчастих коліс, пластикових компонентів або інших стандартних геометричних форм.

Застосування подібних скриптів дозволяє значно підвищити продуктивність при створенні моделей, зменшуючи витрати часу та мінімізуючи ризики людських помилок.

Під час розробки скриптів для автоматичної генерації моделей доцільно враховувати низку тематичних аспектів. Однією з ключових концепцій є використання генеративних алгоритмів для побудови форм. Зокрема, алгоритм L-систем виступає ефективним інструментом для створення складних геометричних структур на основі простих формальних правил і принципу рекурсивної заміни [31; 37; 42].

Цей метод широко застосовується при моделюванні природних об'єктів, таких як дерева, рослинність, кристалічні структури тощо (рис. 1.5) [42; 31; 37].



Рис. 1.5 Дерева L-системи формують реалістичні моделі природних патернів

Основна ідея L-систем полягає у використанні початкової послідовності символів (аксіоми) та набору правил продукції, які поетапно замінюють символи на нові комбінації. У результаті таких заміन формується новий рядок, який відображає дедалі складнішу структуру [31; 37; 42], придатну для графічної інтерпретації.

Ще одним важливим інструментом у генеративному моделюванні є алгоритм Перліна [10; 29; 35], який застосовується для створення реалістичних текстур, поверхонь та шаблонів, що природно виглядають. Один із найвідоміших різновидів є Perlin Noise, алгоритм згладженого шуму, який широко використовується у процедурній генерації таких об'єктів, як гірські масиви, печери, лісові масиви та інші топографічні або природні утворення [10; 29; 35] (рис. 1.6). Алгоритм ґрунтується на формуванні регулярної сітки контрольних точок, кожній з яких надається псевдовипадкове значення. Далі ці значення інтерполюються з використанням спеціальних функцій згладжування, що дозволяє отримати плавні, без різких переходів, шумові візерунки. Такі візерунки є основою для створення органічних текстур або візуальних ефектів у графіці.



Рис. 1.6 Perlin Noise

Застосування скриптів для повторюваних елементів дозволяє значно оптимізувати процес побудови моделей у Blender. Для цього ефективним є використання модифікатора Array, що дає змогу тиражувати вибраний об'єкт задану кількість разів уздовж певної осі. Поєднання цього модифікатора з іншими, наприклад Mirror (дзеркальне відображення) та Solidify (товщина об'єкта), дозволяє створювати складніші та структурно цілісні сіткові моделі.

Ще одним напрямом автоматизації є написання скриптів для процесу рендерингу, що включає налаштування параметрів візуалізації, матеріалів, джерел освітлення, а також розташування камер. Для цього активно використовується Python API Blender, який надає доступ до всіх аспектів внутрішнього функціонування програми. Крім того, можливе застосування зовнішніх програмних рішень, які мають змогу інтегруватися з Blender для розширення функціоналу, що забезпечує гнучке управління процесом візуалізації та підвищення якості фінального результату.

У цьому підрозділі розглядаються методи тестування скриптів в середовищі Blender з метою виявлення та виправлення помилок, а також забезпечення надійності та стабільності функціональності. Проводиться аналіз різних підходів до тестування скриптів у контексті Blender та розглядаються наступні методи:

- юніт-тестування скриптів: визначення та розробка юніт-тестів для індивідуальних функцій або модулів скриптів з метою перевірки правильності їх роботи. Аналізується використання спеціалізованих бібліотек для автоматизації процесу юніт-тестування в середовищі Blender;
- інтеграційне тестування скриптів: вивчення та застосування методів для виконання інтеграційних тестів скриптів, що передбачають їх взаємодію з різними компонентами та функціональними модулями Blender. Аналізується підхід до створення тестових сценаріїв та використання інструментів автоматизації для виконання інтеграційних тестів;
- валідація та верифікація скриптів: розглядаються методи валідації та верифікації скриптів з метою переконання у відповідності їхнього функціоналу

та роботи заявленим вимогам. Вивчаються підходи до формулювання критеріїв валідації та верифікації, а також використання спеціалізованих інструментів для цих цілей.

У цьому підрозділі також розглядається методика документування скриптів в середовищі Blender з метою забезпечення зрозумілості, доступності та ефективного використання коду для інших розробників [5; 38]. Наступні аспекти документування розглядаються в цьому підрозділі: коментарі та описи, документаційні рядки, приклади використання, зовнішня документація.

Коментарі та описи: аналізується використання коментарів у коді скриптів для пояснення функцій, змінних, класів та інших структурних елементів. Розглядаються рекомендації щодо стилю написання коментарів та їхнього форматування згідно зі стандартами документування.

Документаційні рядки (docstrings): вивчаються рекомендації щодо використання документаційних рядків у функціях, класах та модулях скриптів. Аналізується структура та формат документаційних рядків згідно зі стандартами документування, включаючи вказівку параметрів, повернутих значень та приклади використання.

Розглядається включення прикладів використання скриптів у документацію. Аналізується підхід до створення зрозумілих та детальних прикладів, які демонструють основні функціональні можливості скриптів.

Аналізуються можливості створення зовнішньої документації для скриптів в Blender, наприклад, використання Wiki-сторінок, документаційних сайтів або README файлів. Аналізується структура та зміст зовнішньої документації та її ролі в процесі розробки та спільної роботи.

Методика документування скриптів в середовищі Blender відіграє важливу роль у забезпеченні зрозумілості та ефективного використання коду для інших розробників. Використання коментарів, документаційних рядків та прикладів використання сприяє створенню зрозумілої та доступної документації. Це дозволяє розробникам легше розбиратися у функціональності скриптів, а також забезпечує зручну підтримку та спільну роботу над проектами. Крім того,

зовнішня документація виступає як важливий ресурс для опису функціональних можливостей скриптів та надання контексту їх використання. Застосування методів документування у Blender сприяє покращенню розробки скриптів, забезпечує їхню доступність та сприяє зростанню продуктивності розробного процесу.

Перший виклик, з яким ми стикаємось у розробці ігор, це використання численних інструментів і платформ. Візуальна частина гри у 3D створюється в спеціалізованому програмному забезпеченні, такому як сам Blender, але частина програмування виконується в окремих середовищах. Все зібрано та налаштовано так, що можна запускати анімацію за допомогою спеціального програмного забезпечення під назвою Game Engine. Ігровий движок спрямований на імітацію фізики реального світу ігрового середовища, що зрештою залишає всі складніші взаємодії відповідальними за це програмне забезпечення. Це надзвичайно складне завдання, але залежно від рівня складності використовуваного ігрового движка може досягати дуже реалістичних рівнів.

Ігровий движок – це інтерактивна душа анімації, оскільки саме ця частина програмного забезпечення контролює та дозволяє взаємодіяти з об'єктами у грі. Наприклад, якщо в певній грі ми можемо отримати автомобіль і змусити його врізатися в стіну, в результаті чого стіна зламається на багато маленьких частинок, але в інших іграх це не так. Різниця полягає у двигуні, який він використовує.

Все, що передбачає взаємодію в навколишньому середовищі, від освітлення до розширених ефектів візуалізації, таких як створення дзеркала, керується мотором. Комерційні ігри, які використовуються для створення 3D-ігор, наприклад: CryEngine, Ігрова студія, Blender 3D, Unreal Engine, Unity, Torque. У більшості випадках ці движки відносно дорогі, але є безплатні варіанти та відкрите джерело. Серед відкритих або безплатних варіантів є саме Blender 3D, котрий і дозволяє створювати роботи світового рівня з різних творчих сфер, пов'язаних з анімаціями. Наведемо 2 приклади незалежних робіт:

короткометражний фільм «Агент 327: Операція «Перукарня» та анімаційний фільм «Похід за кавою».

Серед багатьох комерційних ігор, розроблених за допомогою Blender, є проєкти, які використовували програмне забезпечення як графічну платформу для ігор. Є приклади використання Blender через механізми, такі як Unity 3D. Серед них: «Shaolin Soccer», «Ignite», «Saab Virtual Race 91», «Dead Cyborg».

Бурхливий період розробки ігор в кінцевому підсумку сприяв збільшенню використання ігрового двигуна Blender.

Скрипти можуть покращити взаємодію Blender з іншими програмами. Blender підтримує багато форматів файлів, що дозволяє обмінюватися даними з іншими програмами. Написання скриптів може спростити процес імпорту та експорту файлів, а також забезпечити глибшу інтеграцію Blender з іншими програмами.

Метою даної кваліфікаційної роботи є демонстрація можливостей мови програмування Python у розробці скриптів для середовища Blender. Для цього використовуються базові й доступні приклади, зрозумілі навіть пересічному користувачу або початківцю у сфері 3D-модельовання. У рамках реалізації цієї мети передбачено написання простого коду для автоматизації процесів створення 3D-моделей, аналіз цього коду, демонстрація результатів його виконання, а також наведення прикладів практичного застосування таких скриптів у реальних задачах. На підставі отриманих рішень було сформульоване такі дослідницькі завдання: визначити властивості математичних параметрів та дослідити їх, а також написати код для видалення «пустишок»(невидимі об'єкти, які не рендеряться і не мають геометрії).

РОЗДІЛ 2. МАТЕМАТИЧНА ОСНОВА

2.1. Математичні параметри автоматичного створення контенту у скриптах

Математичні формули можуть бути використані для створення складних геометричних форм, таких як фрактали, сфери, тори, а також для виконання операцій над геометрією, таких як злиття, відсікання, деформація та інші. Для обчислення нових координат точок об'єкта при його переміщенні, повороті або масштабуванні можна використовувати матриці Геометричні перетворення. Наприклад, для повороту об'єкта навколо осі Z на кут можна використовувати матрицю (рис. 2.1).

$$\begin{pmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Рис. 2.1 матриця

Форма об'єкта може бути визначена за допомогою математичних формул, таких як криві Безьє, криві Катмулла-Кларка, сферичні координати тощо. Наприклад, можна створити скрипт для створення поверхні Безьє з контрольними точками, використовуючи відповідні формули. Алгоритми заливки: для створення складних текстур та візерунків можна використовувати алгоритми заливання, такі як алгоритм Брезенхема або алгоритм відсікання. Наприклад, для побудови лінії між двома точками (x1, y1) та (x2, y2) можна використовувати алгоритм Брезенхема (рис. 2.2) [10; 20]. Генерація геометрії: для створення складних форм і поверхонь можна використовувати математичні рівняння, наприклад рівняння сфери.

$$x^2 + y^2 + z^2 = r^2$$

Цей скрипт створює сферу із заданим радіусом, а потім обчислює координати вершин сфери, які задовольняють заданій формулі.

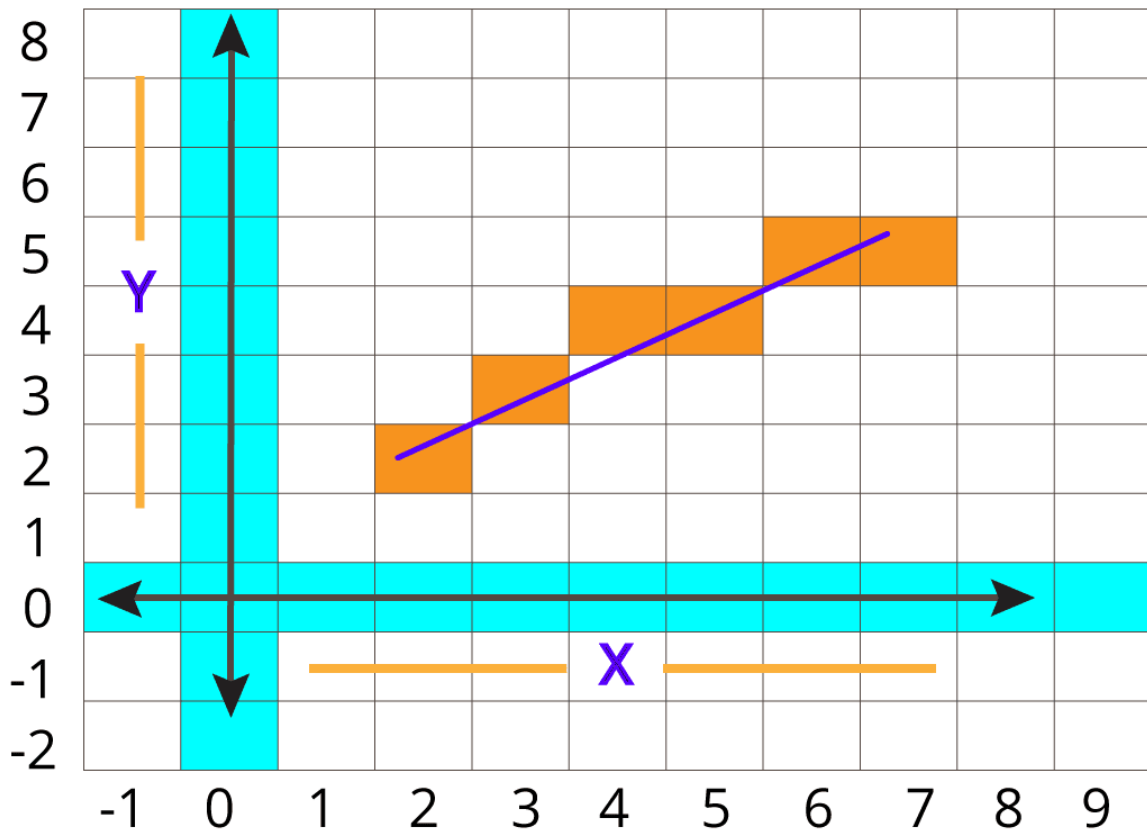


Рис. 2.2 Демонстрація роботи алгоритму Брезенхема

У цьому випадку скрипт виводить координати вершин в консоль. Ці координати можна використовувати для подальшої обробки, наприклад, для створення нових об'єктів на основі знайдених вершин (рис. 2.3).

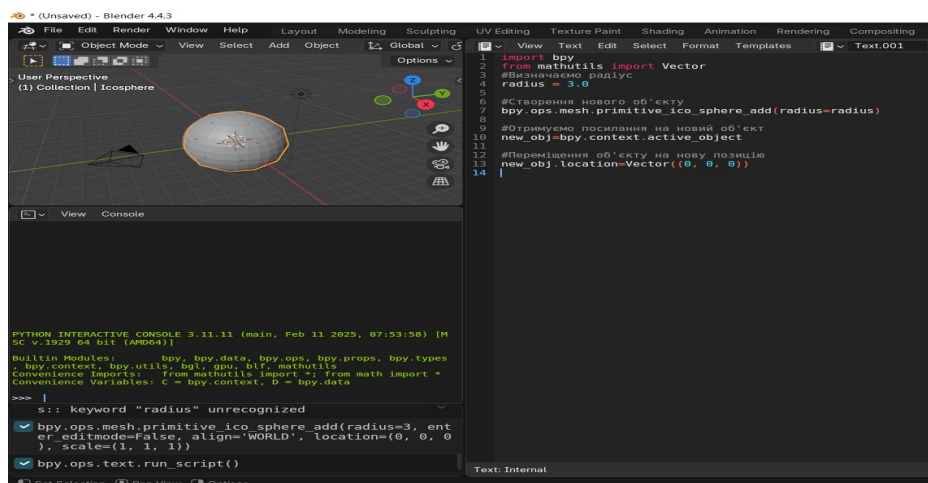


Рис. 2.3 Приклад найпримітивнішого використання коду для створення базового об'єкта

2.2. Анімаційні формули в Python

Геометричні перетворення є одними з найбільш поширених математичних методів для побудови геометричних об'єктів при їх перетворенні. В останній час у геометричному моделюванні все частіше зустрічається контрольоване злиття маніпуляцій геометричних об'єктів довільної форми на основі прототипу, маніпулювання геометричними об'єктами, що не мають точного математичного опису. Ці проблеми можна вирішити запропонованим у роботі методом створення розширеного ресурсу для розробників, які бажають полегшити роботу в програмі Blender, яка популярна для 3D моделювання та анімації.

Використання інтерполяції: для плавного переміщення об'єктів між двома точками можна використовувати методи інтерполяції, наприклад лінійну інтерполяцію. Зокрема, для лінійної інтерполяції між двома точками (x_1, y_1) та (x_2, y_2) можна використовувати таку формулу:

$$x = (1 - t) * x_1 + t * x_2$$

$$y = (1 - t) * y_1 + t * y_2$$

Кусково-лінійна інтерполяція (Piecewise Linear Interpolation) – найпростіший вид інтерполяції. Подібно кусково-лінійній інтерполяції можна використати для наближення на кожному окремому інтервалі. Формула для знаходження значення y на відрізку $[x_1, x_2]$ з використанням кусково-лінійної інтерполяції:

$$y = y_1 + (y_2 - y_1) * ((x_2 - x_1) / (x - x_1))$$

де x_1, y_2 – координати першої точки;

x_1, y_2 – координати другої точки;

x – значення на осі x , для якого потрібно знайти відповідне значення y .

Приклад скрипта мовою Python для використання кусково-лінійної інтерполяції у Blender (рис. 2.4).

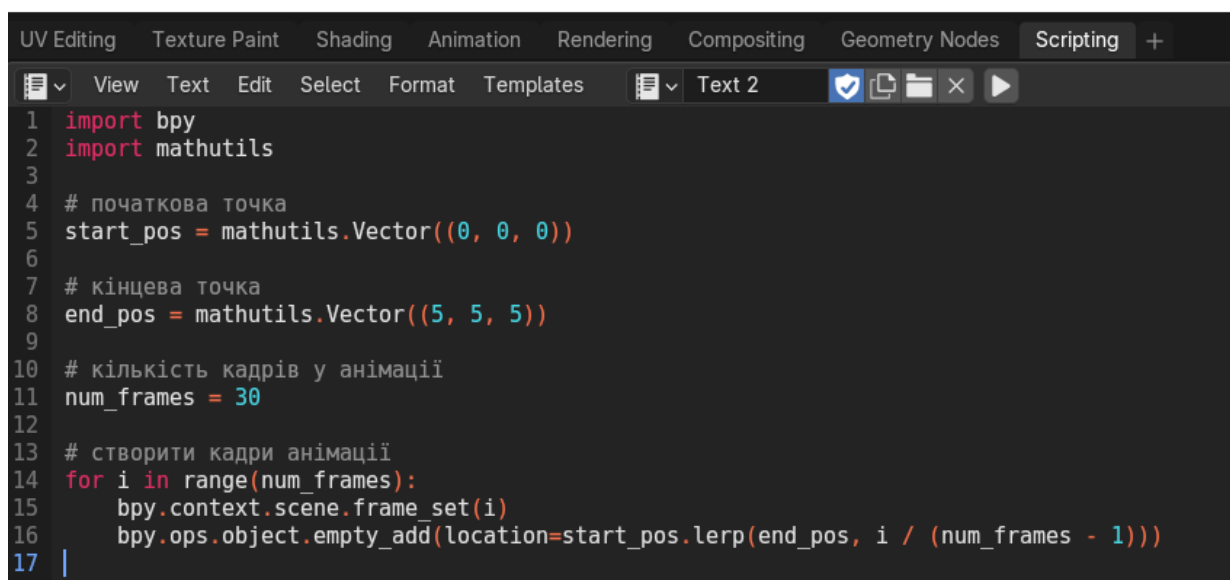


Рис. 2.4 Скрипт з використанням кусково-лінійної інтерполяції

Цей скрипт генерує анімацію, де порожній об'єкт переміщується вздовж прямої, сполучаючи стартову точку (0, 0, 0) з фінальною точкою (5, 5, 5). У цьому скрипті, щоб визначити позицію об'єкта в кожному кадрі анімації, застосовується кусково-лінійна інтерполяція, використовуючи метод `lerp` класу `Vector`. Кожний кадр анімації створюється через метод `frame_set` класу `Scene`, а додавання порожнього об'єкта відбувається за допомогою функції `empty_add` оператора `ops.object`.

Один зі способів інтерполяції на всій ділянці – використання сплайнів. Сплайн, або сплайн-функція – це кусково-поліноміальна функція, що визначена на відрізку і має певну кількість неперервних похідних на ньому. Суть сплайн-інтерполяції полягає у створенні поліномів між парними сусідніми вузлами інтерполяції, причому для кожної пари вузлів будується свій поліном. Найчастіше використовуються кубічні сплайни, тобто кубічні поліноми. Кубічна парабола виглядає так:

$$f(x) = ax^3 + bx^2 + cx + d \quad (2.1)$$

Формули для обчислення інтерполяції кубічного сплайна. Функція (2.1) на проміжку $[x_i, x_{i+1}]$ обраховується наступним чином:

$$f(x) = \frac{(x_{\{i+1\}} - x)^2 * (2(x_i - x) + h) * y_i}{h^3} + \frac{(x_{\{i+1\}} - x)^2 * (x - x_i) * m_i}{h^2} + \\ + \frac{(x_{\{i+1\}} - x)^2 * (2(x_i - x) + h) * y_{\{i+1\}}}{h^3} + \frac{(x_{\{i+1\}} - x)^2 * (x - x_i)^2 * m_{\{i+1\}}}{h^2} \quad (2.2)$$

де

$$m_i = \frac{(y_{\{i+1\}} - y_{\{i-1\}})}{2h} \quad (2.3)$$

$$m_0 = \frac{(-3y_0 + 4y_1 - y_2)}{2h} \quad (2.4)$$

$$m_n = \frac{(3y_n + 4y_{\{n-1\}} - y_{\{n-2\}})}{2h} \quad (2.5)$$

Кубічний сплайн-інтерполяція забезпечує не лише найбільшу однорідність у вузлах. Суміжні параболічні інтерполюючі функції (сплайни), а також їх перша та друга похідні роблять сплайн-інтерполяцію надзвичайно гладкою (рис. 2.5) [11].

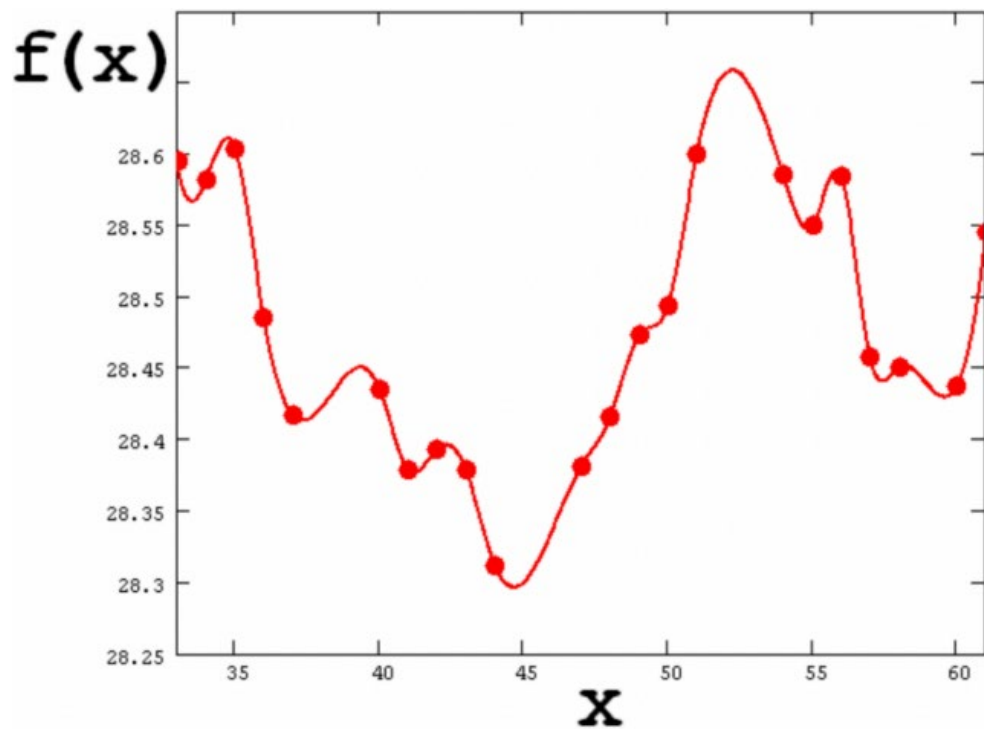


Рис. 2.5 Сплайн-інтерполяція

Рівняння (2.2) – (2.5) не є виключним методом формування кубічних сплайнів, існують інші підходи. На рис. 2.6 зображено графік функції, інтерпольованої двома різними способами [12].

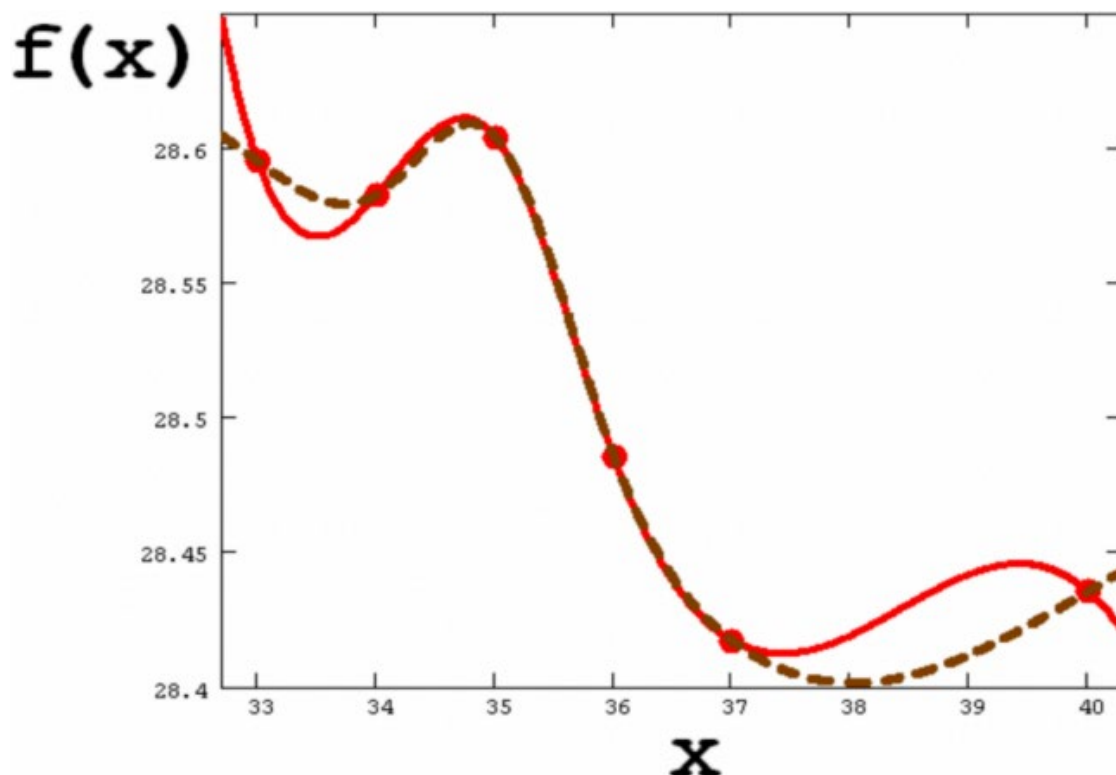


Рис. 2.6 Сплайн-інтерполяція двома способами

Для створення скрипту у Blender, що застосовує сплайн-інтерполяцію, доцільно використовувати бібліотеку SciPy, яка містить реалізацію сплайн-інтерполяції. Скажімо, наведений нижче код створить сплайн-інтерполяцію для визначених точок (рис. 2.7). У Blender ви маєте можливість застосовувати модуль SciPy для реалізації кубічної сплайн-інтерполяції, включно з функціями інтерполяції даних, зокрема, кубічної сплайн-інтерполяції. Спочатку потрібно встановити бібліотеку SciPy, а потім застосувати функцію `scipy.interpolate.CubicSpline`, що забезпечує інтерполяцію кубічного сплайна на основі вказаних точок (рис. 2.7).

```

1 import bpy
2 from scipy import interpolate
3 # задаємо точки для інтерполяції
4 points = [(0, 0, 0), (1, 1, 1), (2, 0, 2), (3, 1, 3)]
5 # отримуємо координати точок в масивах
6 x = [p[0] for p in points]
7 y = [p[1] for p in points]
8 z = [p[2] for p in points]
9 # створюємо сплайн інтерполяцію
10 tck, u = interpolate.splprep([x, y, z], s=0)
11 # генеруємо нові точки на основі сплайт-інтерполяції
12 u_new = np.linspace(u.min(), u.max(), 100)
13 x_new, y_new, z_new = interpolate.splev(u_new, tck)
14 # створюємо об'єкти на основі нових точок
15 for i in range(len(x_new)):
16     bpy.ops.mesh.primitive_cube_add(location=(x_new[i], y_new[i], z_new[i]))

```

Рис. 2.7 Скрипт для реалізації сплайн-інтерполяції

Кусково-постійна двовимірна інтерполяція (англ. *Piecewise constant interpolation*) – це підхід до інтерполяції, що застосовується для відновлення відсутніх значень у наборах даних, які містять значення функції в різних точках площини. Суть методу полягає в припущенні, що в межах кожної окремої області площини функція має незмінне значення (рис. 2.8) [13].

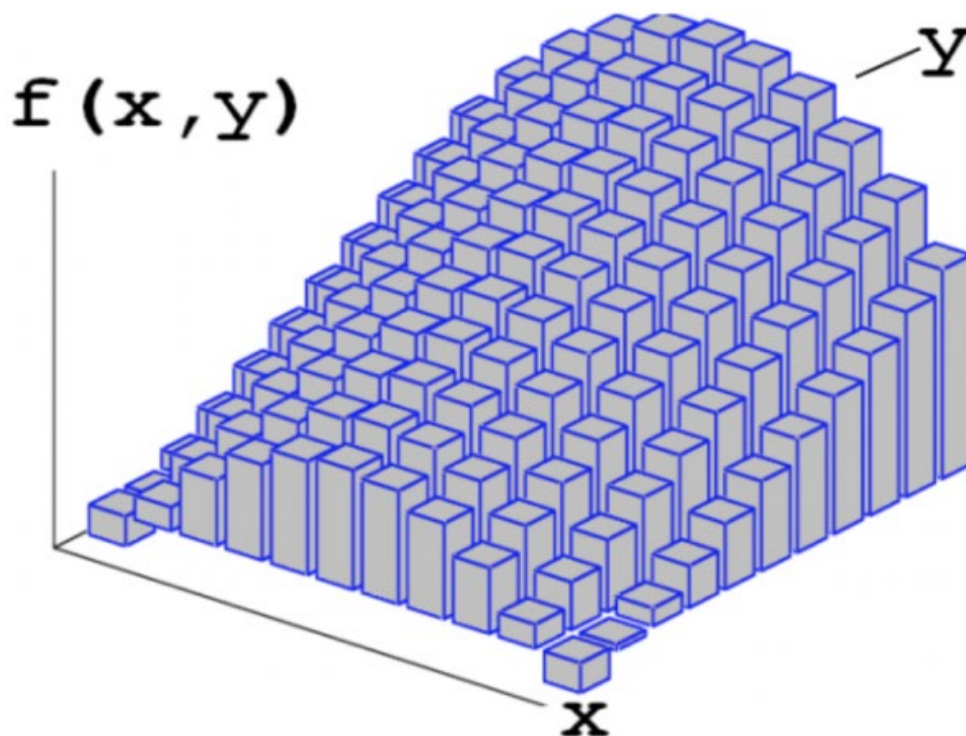


Рис. 2.8 Кусково-постійна двовимірна інтерполяція

У Blender для кусково-постійної двовимірної інтерполяції можна скористатися функцією `bpy_extras.interpolation.interpolate_2d_constant()` з модуля `bpy_extras`.

2.3. Метод скінченних елементів (МСЕ) та налаштування середовища розробки

Метод скінченних елементів (МСЕ) є чисельним інструментом, що застосовується для розв'язання задач математичної фізики та механіки суцільних тіл. Суть цього підходу полягає в розбитті складної геометричної області на невеликі скінченні елементи. Кожен з таких елементів має вузли, в яких

здійснюється обчислення значень поля. У тривимірному моделюванні зазвичай використовуються елементи типу кубів, призм, тетраедрів і пірамід (рис. 2.9).

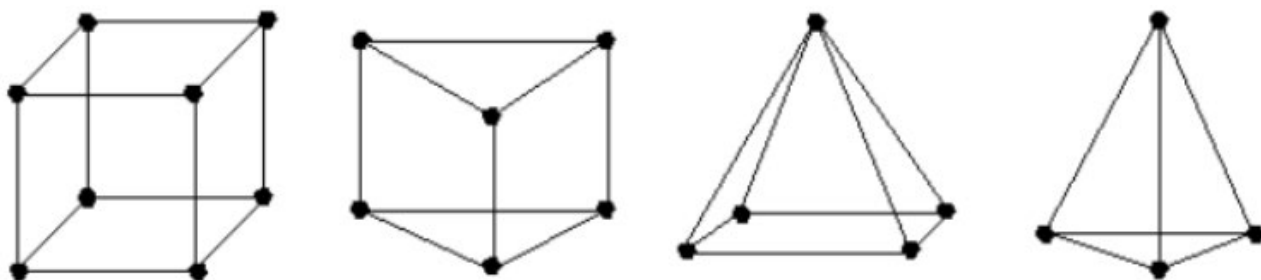


Рис. 2.9 Лінійні тривимірні скінчені елементи

Нижче представлено лінійні елементи, де кожне ребро задається двома вузлами. Це означає, що під час деформації форма ребер залишається лінійною, що зображено на рис. 2.10 [14].

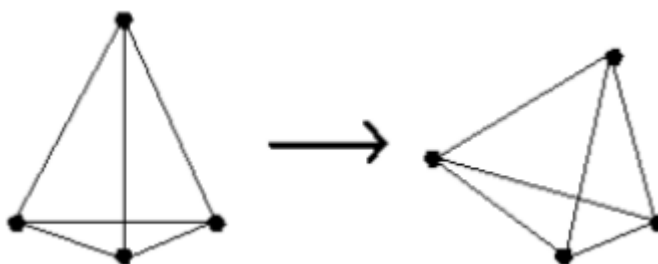


Рис. 2.10 Лінійна тетраедральна деформація кінцевих елементів

Окрім лінійних, використовуються також квадратичні елементи, які мають додатковий вузол на кожному ребрі, а кубічні по два додаткові вузли. Приклад наведено на рис. 2.11.

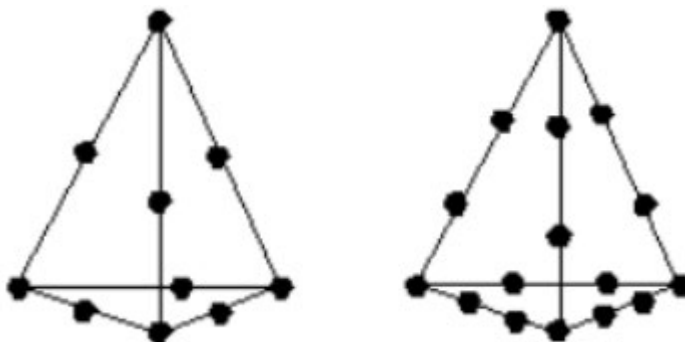


Рис. 2.11 Квадратичні та кубічні скінченні елементи

У Blender Python MCE може бути застосований для реалізації різних задач, зокрема для створення скриптів, які моделюють фізичну поведінку об'єктів з урахуванням матеріалів, освітлення та текстур. Такий підхід дозволяє писати сценарії, що автоматично коригують форму чи положення об'єкта згідно з умовами задачі [15].

Знання бібліотек Python дає змогу глибше взаємодіяти зі сценами, що суттєво пришвидшує процес створення скриптів і розширює функціональні можливості. Серед основних бібліотек, які можуть стати в пригоді: bpy, mathutils, numpy, scipy, OpenCV. Bpy - основна бібліотека для керування об'єктами, властивостями, анімаціями, матеріалами та ін. у Blender. Mathutils містить інструменти для роботи з векторами, матрицями, кватерніонами й іншими елементами лінійної алгебри. Numpy надає широкі можливості для роботи з багатовимірними масивами, виконання математичних операцій і обробки даних. Scipy включає функціонал для вирішення диференціальних рівнянь, оптимізації, обробки сигналів та інших наукових задач. OpenCV використовується для задач комп'ютерного зору, зокрема для обробки зображень, виявлення об'єктів і аналізу відео.

Ці бібліотеки значно розширюють стандартні можливості Blender, надаючи додаткові інструменти для роботи з 3D-сценами, математикою, візуалізацією та аналітикою [19].

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СКРИПТУ

3.1. Методи тестування скрипту

У цьому розділі здійснюється аналіз процесу програмного випробовування скрипту, створеного у межах кваліфікаційної роботи. Перевірка передбачає тестування їх функціональних можливостей, ефективності та надійності з метою виявлення потенційних недоліків і їх усунення [20].

Для перевірки скриптів у середовищі Blender доцільно застосовувати різноманітні методи тестування, що дозволяють оцінити коректність роботи, функціональність і продуктивність. До основних методів, які можуть бути використані, належать такі тестування: модульне, інтеграційне, системне, навантажувальне, безпеки.

Модульне тестування передбачає окрему перевірку кожного модуля скрипту на відповідність встановленим функціональним вимогам. Можна реалізувати автоматизовані тести для перевірки окремих функцій або класів. Це дозволяє оперативно виявляти помилки в ізольованих частинах коду, що сприяє підвищенню надійності та стабільності програмного продукту.

Інтеграційне тестування спрямоване на перевірку взаємодії між окремими модулями скрипту. Варто створювати тести, що аналізують передачу даних між модулями та правильність їх взаємодії. Це дає змогу виявити помилки, що можуть виникнути при спільному функціонуванні компонентів та гарантує коректну інтеграцію.

Системне тестування охоплює перевірку повної функціональності скрипту в умовах, наближених до реального використання. Доцільно здійснювати тестування, яке включає повний спектр передбачених функцій: маніпуляції зі сценами й об'єктами, взаємодію з інтерфейсом користувача тощо. Це дозволяє впевнитися у правильності роботи скрипту за різних сценаріїв використання та відповідності очікуваним результатам.

Навантажувальне тестування дає змогу визначити межі продуктивності скрипту та його стійкість до підвищених навантажень. За допомогою

інтенсивного виконання повторюваних дій можна оцінити, як скрипт справляється з обробкою великих обсягів даних або складних операцій. Це дозволяє виявити слабкі місця з погляду продуктивності та забезпечити ефективну й стабільну роботу.

Тестування безпеки спрямоване на виявлення можливих вразливостей у коді та підвищення рівня захисту скрипту. Доцільно здійснювати аналіз на предмет потенційних загроз, контролювати обробку зовнішніх вхідних даних, обмежувати доступ до критичних ресурсів і забезпечувати безпечну взаємодію з Blender та іншими компонентами системи.

Застосування вищенаведених методів тестування дозволяє забезпечити високу якість, надійність та ефективність скриптів у середовищі Blender. Рекомендується комбінувати ці підходи з урахуванням специфіки проєкту, а також вимог до якості та безпеки програмного забезпечення [21].

3.2. Аналіз та тестування роботи скрипту

У рамках автоматизації процесів очищення сцени в середовищі тривимірного моделювання Blender особливу увагу слід приділити управлінню допоміжними об'єктами типу "Empty" (так званими «пустишками»). Дані об'єкти часто використовуються для організації сцени, контролю модифікаторів, анімації та побудови ієрархічних зв'язків між об'єктами. Проте у випадках, коли такі об'єкти втрачають свою функціональність, наприклад, не мають дочірніх елементів і не використовуються в жодному з модифікаторів інших об'єктів, вони перестають відігравати конструктивну роль у сцені та можуть лише ускладнювати структуру проєкту.

Для розв'язання цієї проблеми було розроблено Python-скрипт (рис. 3.1), що використовує API Blender (bpy) для автоматичного виявлення та видалення «неактивних» пустишок. Алгоритм передбачає перевірку кожного об'єкта сцени на тип EMPTU, перевірку наявності дочірніх об'єктів, а також аналіз участі у модифікаторах інших об'єктів. Об'єкти, що не відповідають жодному з вказаних

критеріїв активного використання, вважаються непотрібними та видаляються зі сцени.

Застосування цього підходу дозволяє ефективно очищувати сцену від зайвих елементів, що своєю чергою сприяє підвищенню продуктивності роботи, спрощенню структури сцени та покращенню керованості проектом у Blender.

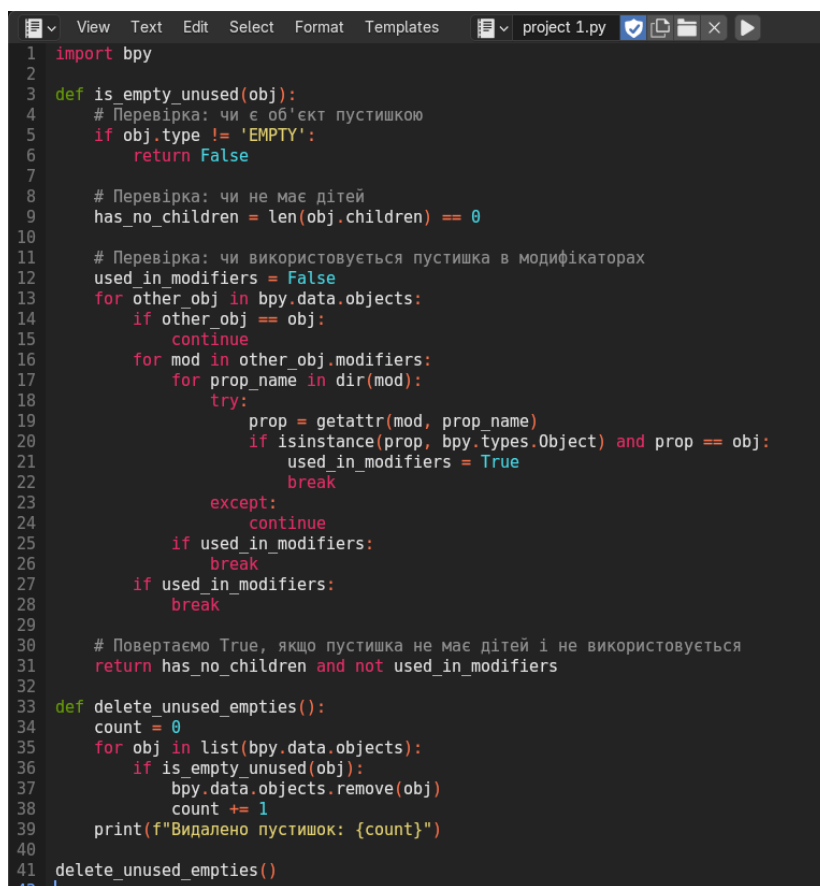
У цьому скрипті використано дві функції:

1. *is_empty_unused(obj)*

Ця функція виконує перевірку, чи є переданий об'єкт невикористовуваною пустишкою:

- *obj.type != 'EMPTY'*: якщо об'єкт не є типу EMPTY, функція одразу повертає False, тобто він не підлягає обробці.
- *has_no_children = len(obj.children) == 0*: перевіряється, чи об'єкт не має дочірніх елементів (тобто, не виконує жодної ієрархічної ролі у сцені).

Далі перевіряється, чи об'єкт використовується в модифікаторах інших об'єктів:



```

1 import bpy
2
3 def is_empty_unused(obj):
4     # Перевірка: чи є об'єкт пустишкою
5     if obj.type != 'EMPTY':
6         return False
7
8     # Перевірка: чи не має дітей
9     has_no_children = len(obj.children) == 0
10
11     # Перевірка: чи використовується пустишка в модифікаторах
12     used_in_modifiers = False
13     for other_obj in bpy.data.objects:
14         if other_obj == obj:
15             continue
16         for mod in other_obj.modifiers:
17             for prop_name in dir(mod):
18                 try:
19                     prop = getattr(mod, prop_name)
20                     if isinstance(prop, bpy.types.Object) and prop == obj:
21                         used_in_modifiers = True
22                         break
23                 except:
24                     continue
25             if used_in_modifiers:
26                 break
27         if used_in_modifiers:
28             break
29
30     # Повертаємо True, якщо пустишка не має дітей і не використовується
31     return has_no_children and not used_in_modifiers
32
33 def delete_unused_empties():
34     count = 0
35     for obj in list(bpy.data.objects):
36         if is_empty_unused(obj):
37             bpy.data.objects.remove(obj)
38             count += 1
39     print(f"Видалено пустишок: {count}")
40
41 delete_unused_empties()
42

```

Рис. 3.1 Скрипт для видалення пустишок

- Проходить цикл по всіх об'єктах сцени.
- Для кожного об'єкта перевіряється, чи має він модифікатори (modifiers), які посилаються на даний EMPTY-об'єкт.
- Проходить цикл по всіх об'єктах сцени.
- Для кожного об'єкта перевіряється, чи має він модифікатори (modifiers), які посилаються на даний EMPTY-об'єкт.
- Якщо знайдено хоча б один такий модифікатор, встановлюється прапорець `used_in_modifiers = True`.

Функція повертає `True` лише тоді, коли об'єкт: типу `EMPTY`, не має дочірніх елементів та не використовується в модифікаторах.

Таким чином, вона визначає, що об'єкт дійсно не виконує жодної функції у сцені.

2. *delete_unused_emptyies()*

Ця функція реалізує основну логіку очищення сцени від невикористовуваних пустишок: ініціалізується змінна `count = 0`, яка буде підраховувати кількість видалених об'єктів. За допомогою `list(bpy.data.objects)` проходиться цикл по всіх об'єктах у сцені. Використовується `list(...)` для уникнення проблем при видаленні елементів під час ітерації. Для кожного об'єкта викликається `is_empty_unused(obj)`. Якщо повертається `True`, об'єкт видаляється з `bpy.data.objects.remove(obj)`, а лічильник `count` збільшується. Після завершення роботи функція виводить повідомлення з кількістю видалених об'єктів: `print(f'Видалено пустишок: {count}')`.

За замовчуванням при запуску Blender створюється сцена з кубом, камерою та світлом. У багатьох випадках при імпорті моделей або при роботі зі складними ієрархіями додаються об'єкти типу `EMPTY`. Вони слугують як: точки контролю, порожні вузли, орієнтири та групувальні об'єкти.

Проте після редагування сцени багато з них можуть втратити своє призначення. Видалення таких об'єктів: зменшує кількість зайвих елементів у сцені, спрощує структуру об'єктів та покращує продуктивність.

Важливо зазначити, що об'єкти типу EMPTY, які використовуються як контрольні точки або орієнтири, не будуть видалені, якщо вони використовуються хоча б в одному модифікаторі або мають дочірні об'єкти. Відповідно до цього, перевірку скрипту проводимо в декілька етапів для перевірки на наявність помилок в роботі коду.

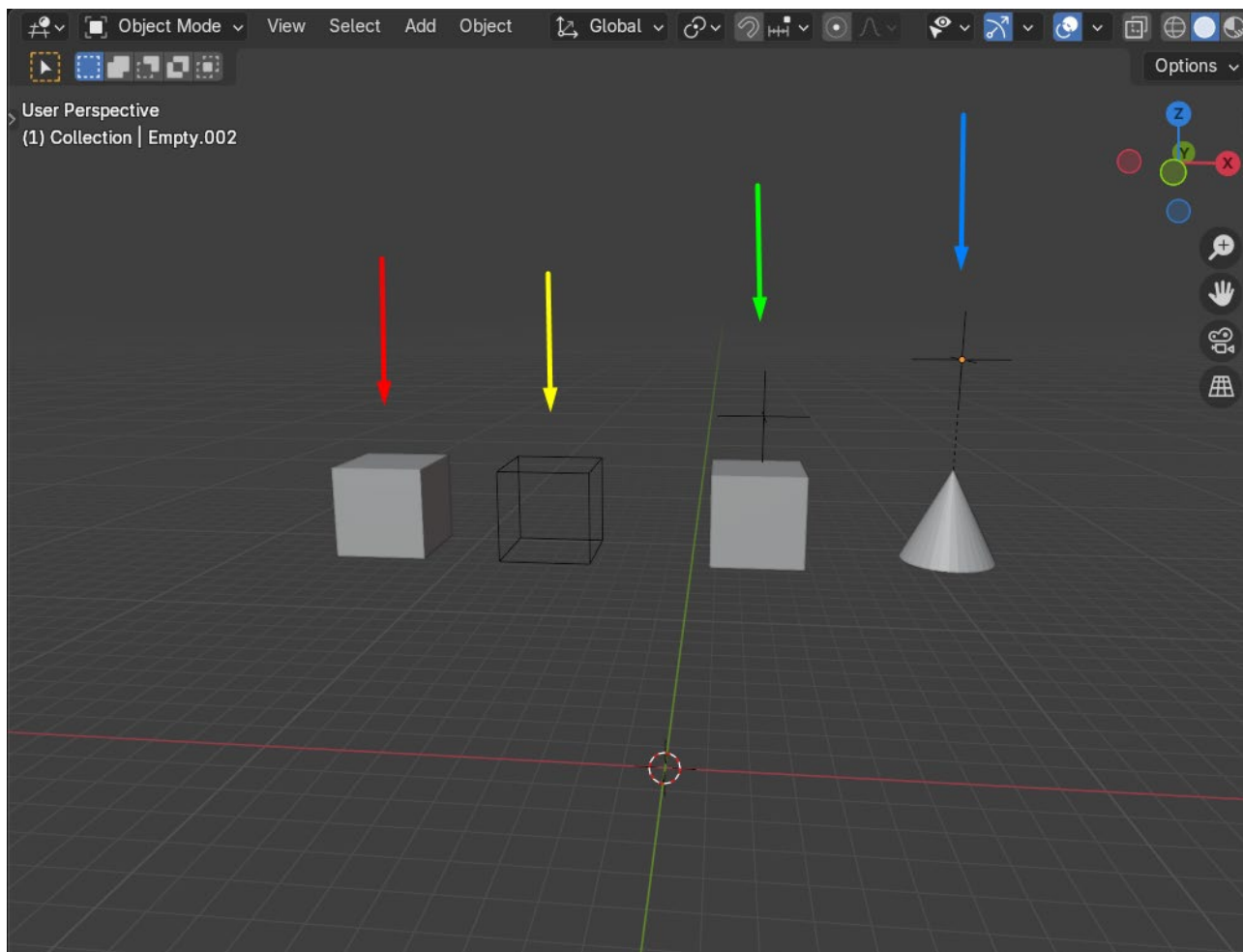
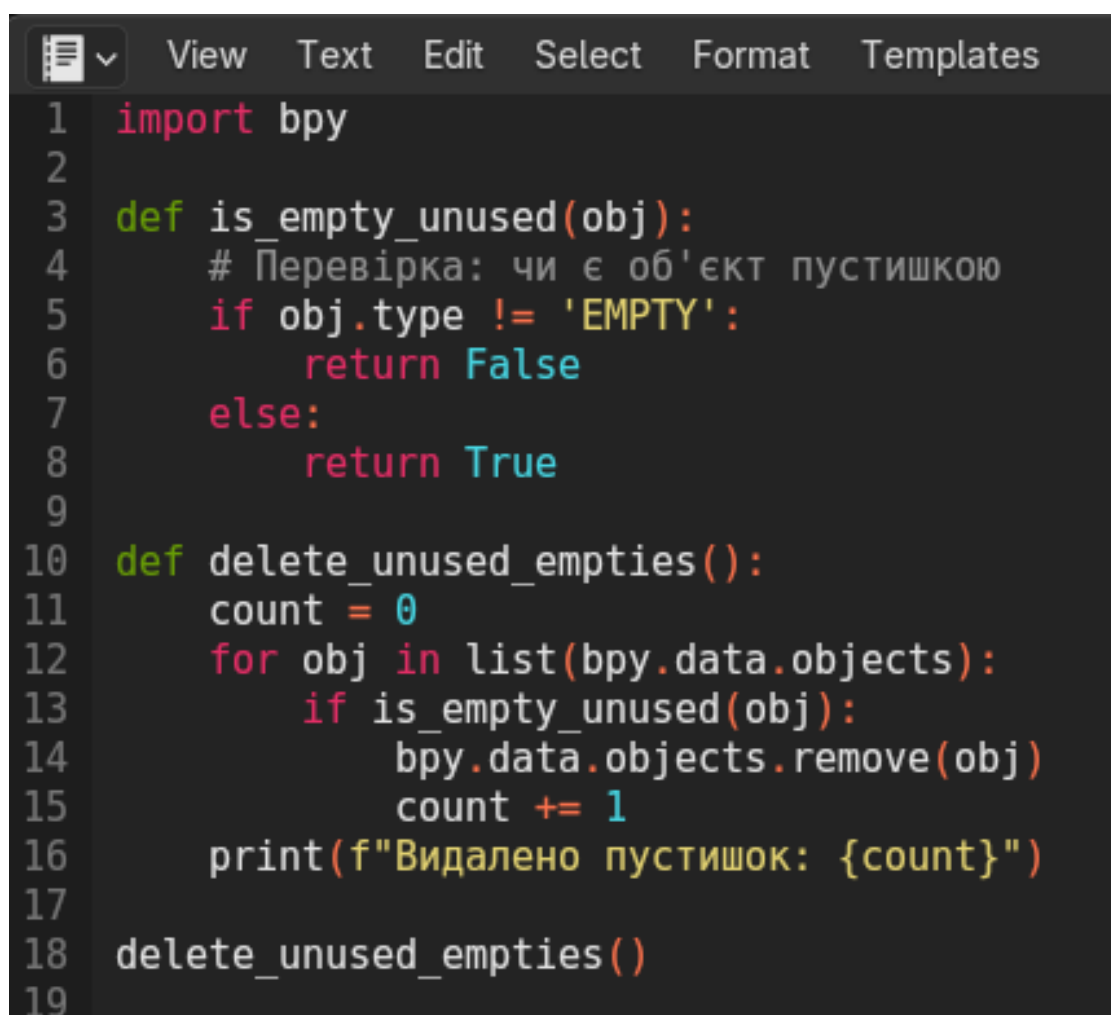


Рис. 3.2 Об'єкти для тестування коду

Для всіх етапів тестування ми використовуємо 4 об'єкти (рис. 3.2): базовий Mesh об'єкт (суцільний куб, на рисунку позначений червоною лінією), пустишка (каркас куба, на рисунку позначений жовтою лінією), пустишка, котра використовується в модифікаторі Array базового Mesh об'єкту (на рисунку позначена зеленою лінією) та пустишка, котра прив'язана до Mesh об'єкту – конуса (на рисунку позначена синьою лінією).

Методом для перевірки ефективності даного скрипта вибрано поетапне додавання трьох частин коду одне до одного та їх застосування для поступового відображення їх роботи. Тестування проводимо на об'єктах рис. 3.2, котрі включають всі потрібні для перевірки типи пустишок. Даний спосіб поетапно продемонструє результат застосування трьох окремих частин скрипта, допоможе перевірити їх взаємодію та запобігти виникненню помилок.

Перша частина скрипту (рис. 3.3) видаляє всі об'єкти типу Empty, незалежно від того, використовуються вони чи ні, та підраховує і виводить кількість видалених пустишок.



```

1  import bpy
2
3  def is_empty_unused(obj):
4      # Перевірка: чи є об'єкт пустишкою
5      if obj.type != 'EMPTY':
6          return False
7      else:
8          return True
9
10 def delete_unused_empties():
11     count = 0
12     for obj in list(bpy.data.objects):
13         if is_empty_unused(obj):
14             bpy.data.objects.remove(obj)
15             count += 1
16     print(f"Видалено пустишок: {count}")
17
18 delete_unused_empties()
19

```

Рис. 3.3 Перша функція скрипту

Запускаємо код і отримуємо результат (рис. 3.4).

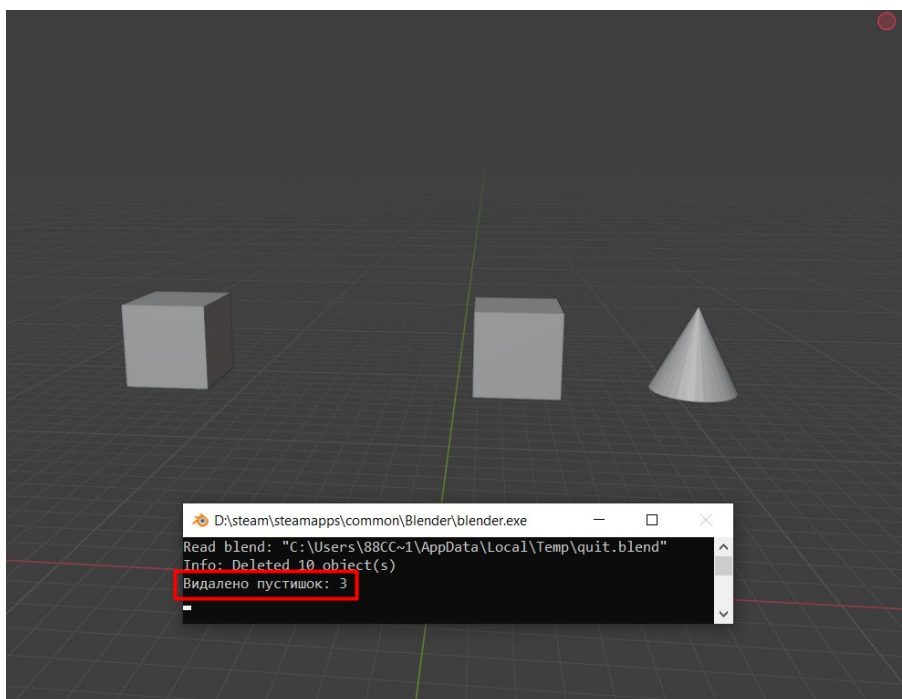


Рис. 3.4 Результат застосування першої функції скрипту

Ми досягли бажаного результату на цьому етапі, тому додаємо другу функцію скрипту до першої для перевірки їх взаємодії (рис. 3.5).

```

1 import bpy
2
3 def is_empty_unused(obj):
4     # Перевірка: чи є об'єкт пустишкою
5     if obj.type != 'EMPTY':
6         return False
7
8     # Перевірка: чи не має дітей
9     has_no_children = len(obj.children) == 0
10
11     # Повертаємо True, якщо пустишка не має дітей
12     return has_no_children
13
14 def delete_unused_empties():
15     count = 0
16     for obj in list(bpy.data.objects):
17         if is_empty_unused(obj):
18             bpy.data.objects.remove(obj)
19             count += 1
20     print(f"Видалено пустишок: {count}")
21
22 delete_unused_empties()
23

```

Рис. 3.5 Об'єднані перша та друга функції скрипту

Запускаємо код і отримуємо результат (рис. 3.6).

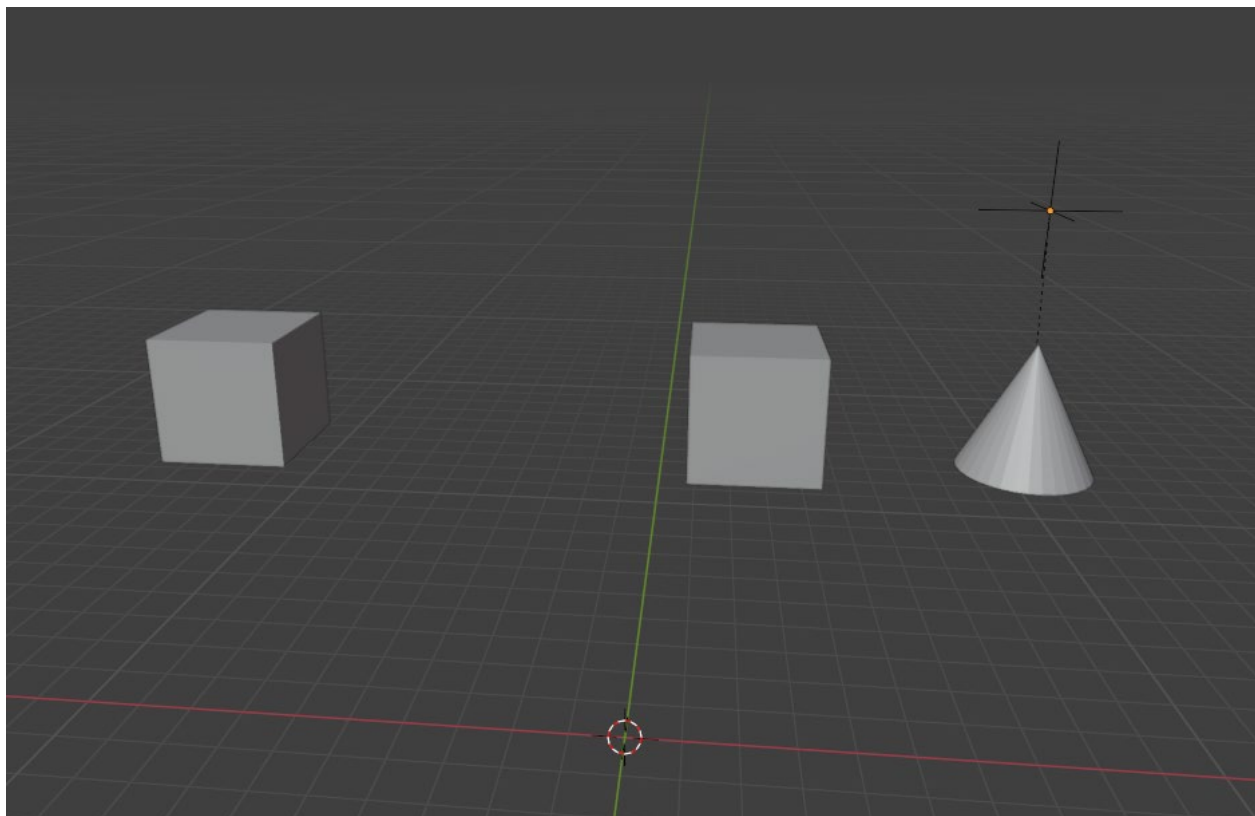


Рис. 3.6 Результат роботи 1 та 2 функцій скрипту

Перший та другий скрипти успішно пройшли тест на сумісність і чітко виконали свої функції, видаливши всі пустишки окрім тієї, у котрої присутній дочірній об'єкт. Під час тестування не було виявлено жодних помилок, а отже можемо перейти до останньої перевірки повноцінного скрипту (рис. 3.1).

Для підвищення унікальності перевірки також додаємо ще один об'єкт типу Mesh і використовуємо пустишку в його модифікаторі Cast (рис. 3.7, зелена стрілка). Таким чином в нас на сцені 2 таких об'єкти, але на них застосовані модифікатори різного типу, адже Cast – це модифікатор типу Deform, а Array – типу Generate (рис. 3.7, червона стрілка). Також генеруємо більше звичайних пустишок для наочності, адже необхідним кінцевим результатом для скрипту є видалення усіх пустишок, котрі не виконують жодних функцій під час роботи та погіршують продуктивність роботи.

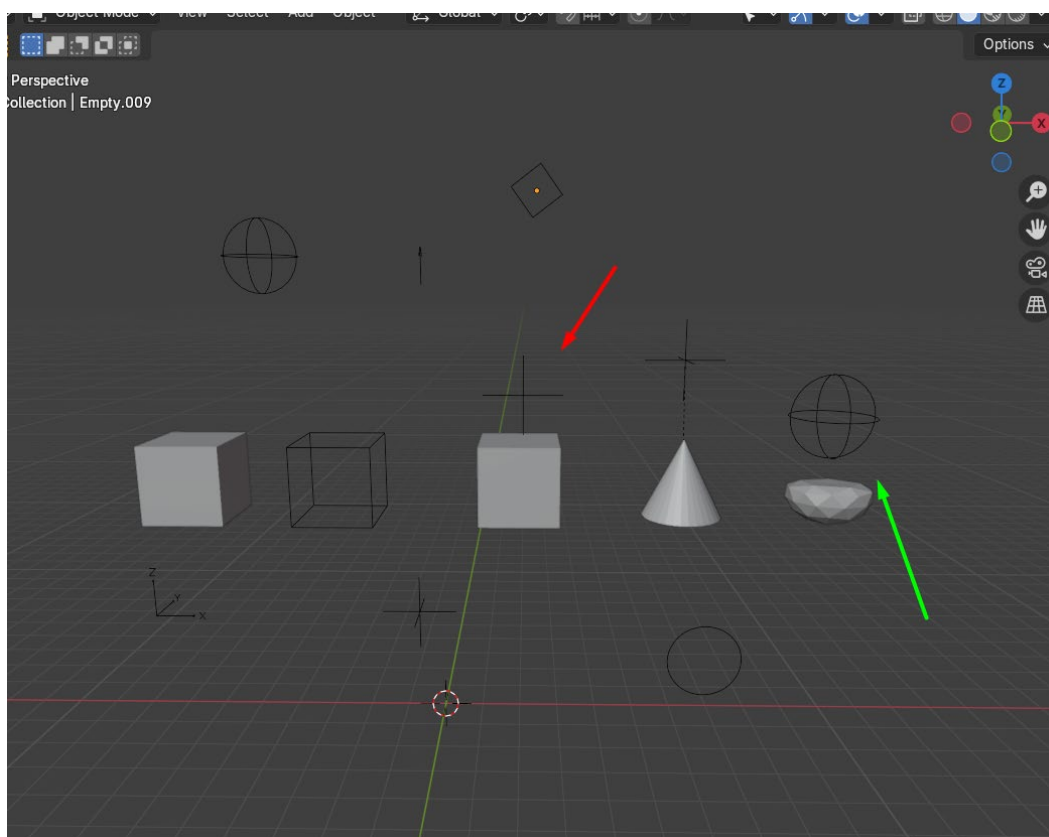


Рис. 3.7 Новостворений Mesh-об'єкт з пустишкою в його модифікаторі Cast

Активуємо скрипт і отримуємо результат (рис. 3.8)

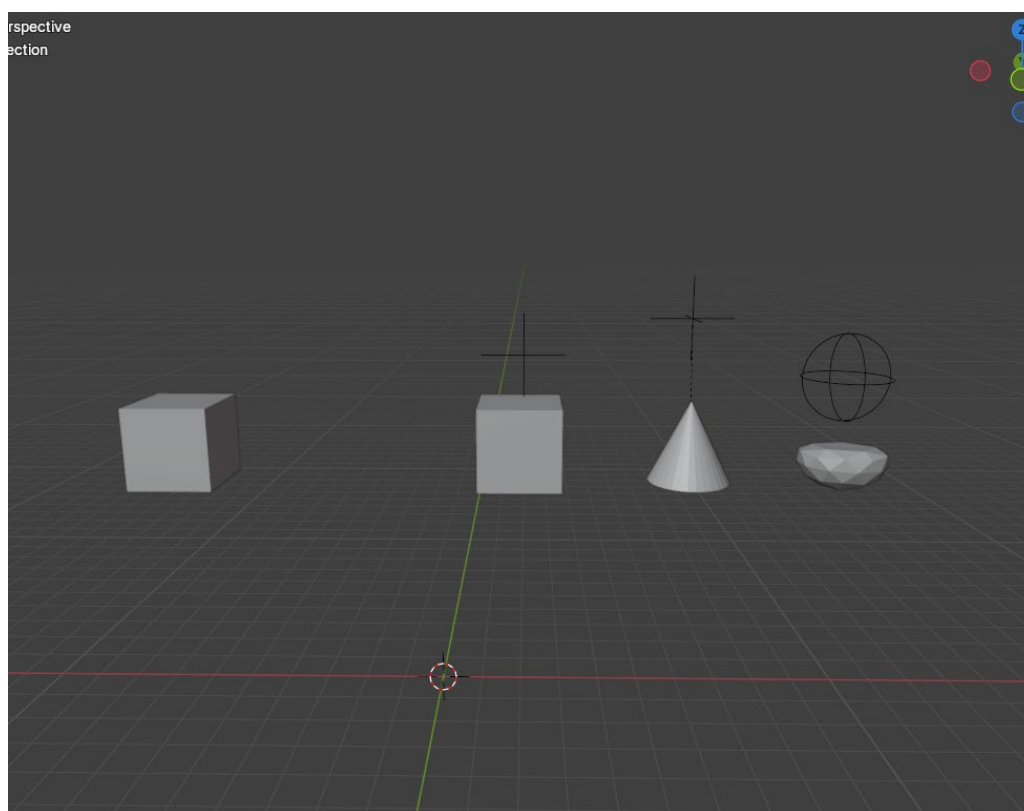


Рис. 3.8 Результат тестування повного скрипту

Після успішного функціонального тестування розробленого скрипту на контрольному наборі даних, який складався з обмеженої кількості об'єктів типу «Empty», було підтверджено коректність його роботи. Зокрема, скрипт правильно ідентифікував та видалив лише ті пустишки, які не мали дочірніх елементів та не використовувались у жодному з модифікаторів інших об'єктів сцени. Об'єкти, що були прив'язані до модифікаторів або мали дочірні зв'язки, залишилися недоторканими, що свідчить про правильну реалізацію логіки перевірки.

З метою перевірки ефективності, масштабованості та продуктивності роботи алгоритму, було вирішено провести додаткове тестування на сцені з високою щільністю пустишок. Для цього пишемо Python скрипт (рис. 3.9), який створює 2000 пустишок, що дозволяє оцінити продуктивність скрипту за умов високого навантаження. Такий підхід дозволяє встановити, наскільки ефективно та стабільно скрипт працює у великих проєктах, характерних для професійного виробничого середовища.

```

1  import bpy
2
3  def create_dense_emptys(count=2000):
4      spacing = 0.25 # Менша відстань між пустишками для щільності
5      for i in range(count):
6          x = (i % 50) * spacing # 50 об'єктів у рядку
7          y = ((i // 50) % 40) * spacing # 40 рядків у шарі
8          z = (i // (50 * 40)) * spacing # Наступний шар по висоті
9          bpy.ops.object.empty_add(type='PLAIN_AXES', location=(x, y, z))
10
11  create_dense_emptys()
12  |

```

Рис. 3.9 Скрипт для створення 2000 пустишок

Нотуємо початкові дані ресурсоспоживання програми Blender (рис. 3.10). Головним показником продуктивності вибрано обсяг оперативної пам'яті (виділено чорним на рис. 3.10), котру потребує програма Blender для виконання поточних задач.

Ім'я	Стан	3% ЦП	25% Пам'ять
Програми (5)			
> Blender (2)		0%	258,1 МБ

Рис. 3.10 Обсяг RAM ПК, котру використовує відкрита Blender з початковою сценою

Запускаємо скрипт для створення 2000 пустишок в Blender і спостерігаємо підвищення задіюваної оперативної пам'яті (рис. 3.11).

Ім'я	Стан	3% ЦП	25% Пам'ять
Програми (5)			
> Blender		0,1%	342,6 МБ

Рис. 3.11 RAM, що використовує Blender при активних 10000 пустишках

На основі порівняння двох наведених знімків екрана, можна зробити висновок про суттєву різницю у споживанні оперативної пам'яті програмою Blender залежно від обсягу даних, що містяться у відкритому проєкті. На першому зображенні (рис. 3.10), що демонструє запуск стартової сцени Blender (типова сцена з кубом, камерою та джерелом світла), програма використовує 259,6 МБ оперативної пам'яті. На другому скриншоті, після створення 2000 пустишок, обсяг використаної оперативної пам'яті складає 342,6 МБ, тобто приблизно у 1,3 раза більше.

Цей приріст у споживанні пам'яті має важливе значення в контексті масштабних проєктів у Blender, оскільки великі обсяги даних (у вигляді тисяч об'єктів, незалежно від їх складності) можуть значно впливати на продуктивність системи. Наприклад, при відкритті бенчмарк-файлу, який містить 43200 об'єктів класу Empty, обсяг цієї пам'яті виросте до 1451 МБ, приблизно в 5.5 разів у порівнянні зі стартовою сценою. Збільшення використаної оперативної пам'яті

призводить до: зниження загальної швидкодії програми, що виражається у затримках при навігації сценою, повільному оновленні Viewport або тривалому виконанні скриптів і операцій; підвищення навантаження на ресурси комп'ютера, що може унеможливити одночасну роботу з іншими додатками; зростання часу збереження та відкриття проекту, особливо при роботі зі складними сценами, а також ризик аварійного завершення роботи у разі перевищення доступного обсягу оперативної пам'яті.

З огляду на значне споживання оперативної пам'яті, доцільним є повторне застосування раніше розробленого скрипту для автоматичного видалення невикористовуваних пустишок у розмірі 2000 штук. Це дозволить оцінити ефективність інструменту в умовах високої щільності об'єктів та перевірити його працездатність на масштабному прикладі.

Запускаємо скрипт для створення пустишок (рис. 3.9) і отримуємо результат (рис. 3.12).

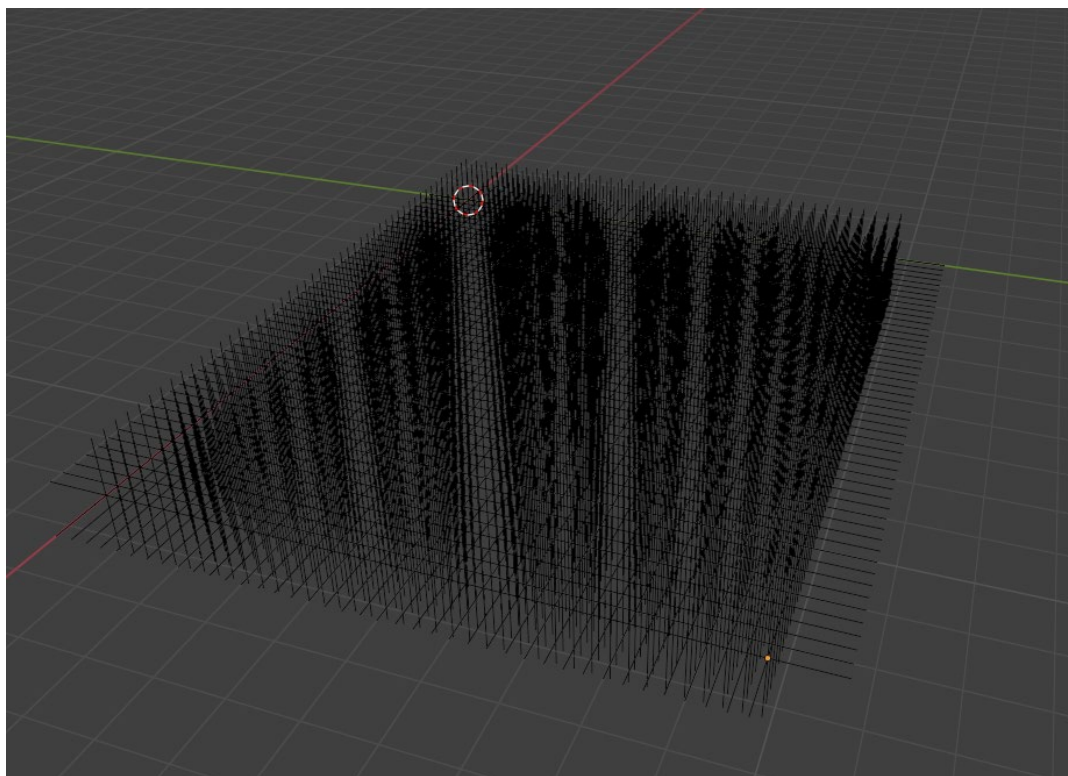


Рис. 3.12 Результат скрипту для створення 2000 пустишок

До 2000 звичайних додаємо одну пустишку з дочірнім об'єктом та другу, що використовується у модифікаторі Mesh-об'єкта, для перевірки роботи скрипта в умовах вищої навантаженості (рис. 3.13).

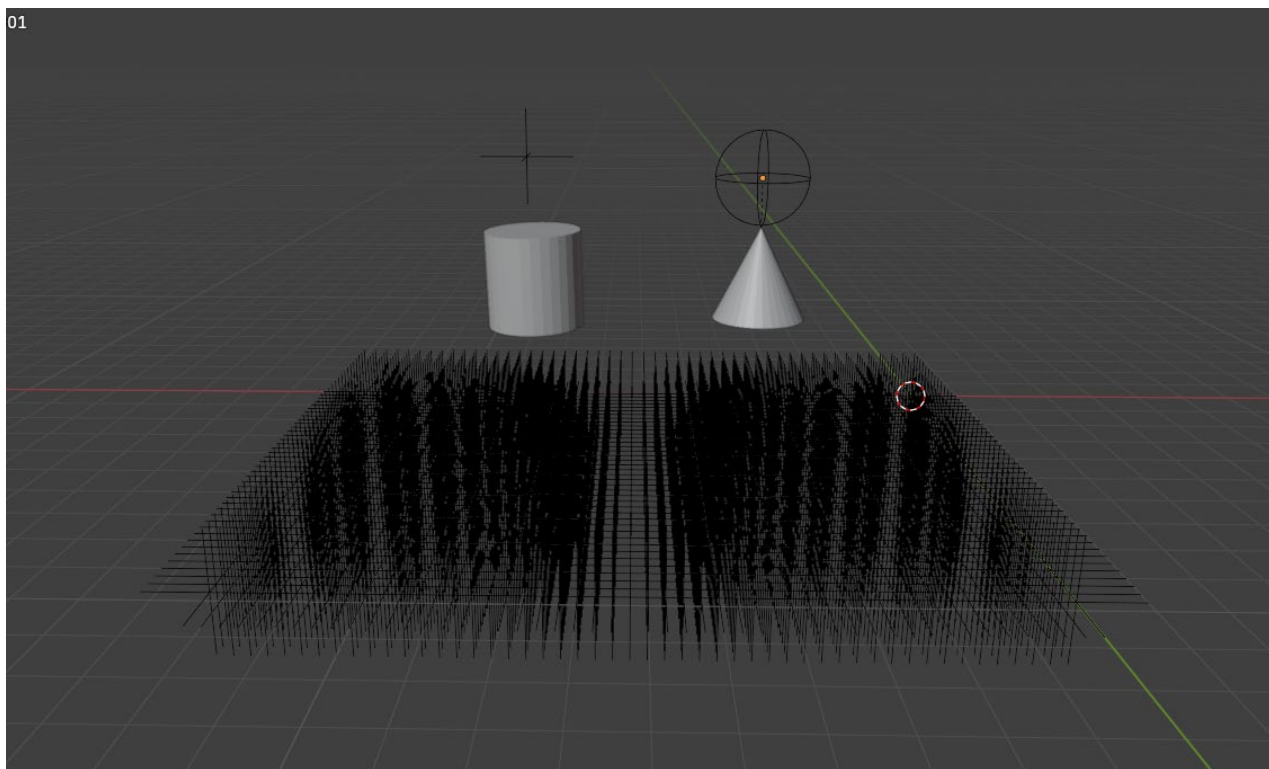


Рис. 3.13 2002 пустишки

Активуємо скрипт для видалення пустишок і отримуємо результат (рис. 3.14)

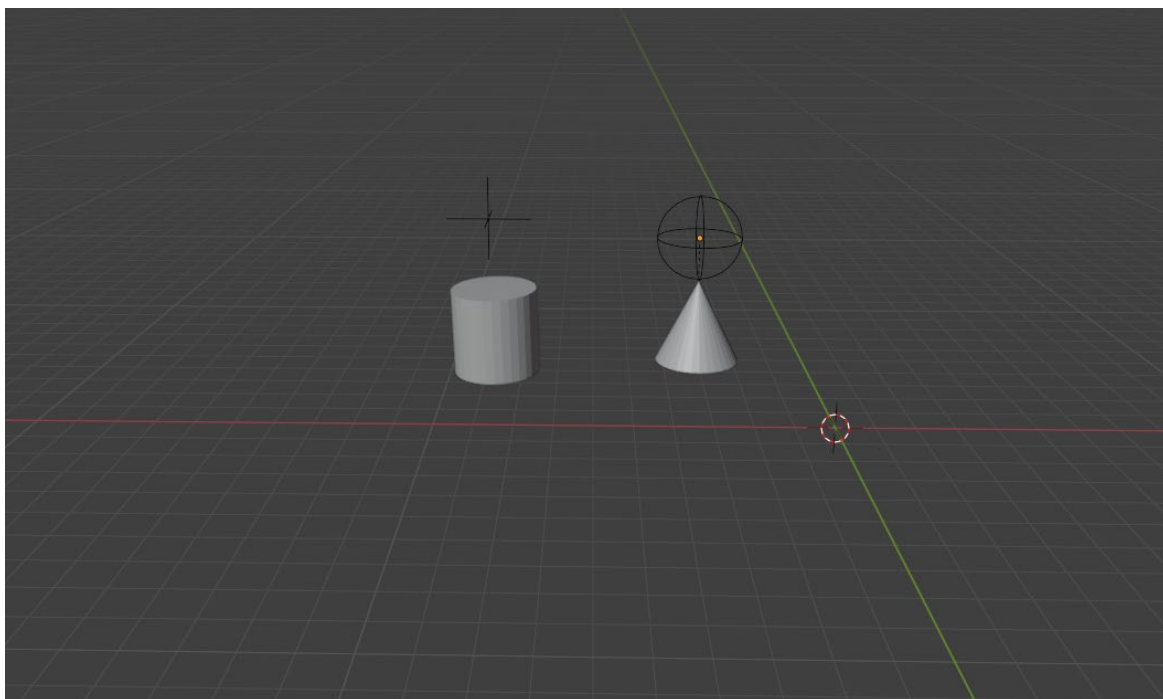


Рис. 3.14 Результат роботи скрипту для видалення пустишок

Одержані результати тестування скрипта з видалення невикористовуваних пустишок у сцені з 2000 об'єктами демонструють його ефективність і точність. Зокрема, час виконання операції склав приблизно 2,1 секунди, що вказує на задовільну швидкодію алгоритму в умовах сцени середньої складності. Варто зазначити, що з-поміж усіх пустишок були коректно збережені ті, що мають функціональне призначення: одна – як батьківський елемент для mesh-об'єкта, інша – як елемент модифікатора. Це підтверджує правильність реалізації умов перевірки та дотримання логіки фільтрації об'єктів, яка ґрунтується на наявності дочірніх елементів або участі в структурі модифікаторів інших об'єктів сцени.

З технічного погляду, виявлена продуктивність є достатньою для інтеграції подібного рішення в реальні проекти різних масштабів, особливо у випадках, коли необхідна оптимізація сцени перед фінальним рендерингом чи експортом проекту. Таким чином, отриманий результат свідчить про релевантність і прикладну цінність створеного інструмента для автоматизованого очищення сцен у середовищі Blender.

ВИСНОВКИ

На підставі обраних методичних засад, було обрано комплексну схему тестування скриптів, що включає модульне, інтеграційне тестування та верифікацію на основі емпіричних даних. Окрему увагу було приділено перевірці скрипта на предмет відповідності функціональним вимогам – видалення лише тих об'єктів типу Empty, які не мають функціонального навантаження в структурі сцени Blender. До таких об'єктів належать ті, що не мають дочірніх елементів і не використовуються в жодному з модифікаторів інших об'єктів сцени.

Реалізовано скрипт, здатний автоматично визначати пустішки, які не виконують жодної активної ролі, та видаляти їх. Для перевірки його працездатності була змодельована сцена з 2000 службових об'єктів типу Empty, з яких дві пустішки виконували виняткові функції: одна мала дочірній mesh-об'єкт, а інша використовувалась у модифікаторі типу Array.

Тестування проводилось за допомогою практичного запуску скрипта в реальному середовищі Blender. У результаті виконання скрипта було коректно виявлено всі пустішки, що не використовувались за призначенням, і здійснено їх видалення. При цьому дві спеціально додані пустішки були збережені в сцені, оскільки задовольняли умови виключення з очищення. Загальний час виконання операції становив близько 2,1 секунди, що вказує на високу швидкодію алгоритму навіть при опрацюванні тисяч об'єктів.

Порівняльний аналіз ресурсного навантаження на систему перед та після виконання скрипта показав помірне, але стабільне зменшення обсягу використовуваної оперативної пам'яті, що є прямим наслідком оптимізації сцени. Це демонструє не лише функціональну коректність, але й практичну доцільність застосування скрипта у виробничих середовищах, зокрема у проєктах, що містять велику кількість службових або дубльованих об'єктів.

Таким чином, результати тестування підтверджують придатність розробленого інструменту до широкого використання, а також його відповідність критеріям ефективності, точності та стабільності, визначеним у методичних засадах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ACM SIGGRAPH. Digital Library. URL: <https://dl.acm.org/conference/siggraph> (дата звернення: 15.04.2025).
2. Aggarwal C. C. Neural Networks and Deep Learning, Cham: Springer, 2018, 497 с.
3. Autodesk. Maya Technical Docs. URL: <https://help.autodesk.com/view/MAYAUL/> (дата звернення: 24.04.2025).
4. Blender Foundation. Blender Manual. URL: <https://docs.blender.org/manual/en/latest/> (дата звернення: 24.03.2025).
5. Blender Foundation. Blender Python API Documentation. URL: <https://docs.blender.org/api/current/> (дата звернення: 25.03.2025).
6. Blender Foundation. Issue #43785: Curve radius not exported with FBX. URL: <https://projects.blender.org/blender/blender/issues/43785> (дата звернення: 27.03.2025).
7. BlenderNation. Blender News and Tutorials. URL: <https://www.blendernation.com/> (дата звернення: 28.03.2025).
8. Bærentzen J. A., Nielsen M. B. GPU-Based Real-Time Surface Reconstruction // The Visual Computer, 2014, Vol. 30, № 6–8, С. 847–856.
9. Disney Research. Physically Based Shading at Disney // ACM SIGGRAPH, 2012. URL: https://disney-animation.s3.amazonaws.com/library/s2012_pbs_disney_brdf_notes_v2.pdf (дата звернення: 29.03.2025).
10. Ebert D. S., Musgrave F., Peachey D., Perlin K., Worley S. Texturing and Modeling: A Procedural Approach, 3rd ed., San Francisco: Morgan Kaufmann, 2002, 552 с.
11. GitHub. Blender Python Scripts. URL: <https://github.com/blender/> (дата звернення: 26.03.2025).
12. Godot Engine. Godot Documentation. URL: <https://docs.godotengine.org/> (дата звернення: 30.04.2025).
13. Google Developers. WebGL Fundamentals. URL: <https://webglfundamentals.org/> (дата звернення: 12.05.2025).
14. Haines E. Real-Time Rendering Techniques and Applications, Wellesley: A K Peters, 2019, 320 с.

15. Hanrahan P., Lawson J. A Language for Shading and Lighting Calculations // ACM SIGGRAPH, 1990, Vol. 24, № 4, С. 289–298.
16. ISO/IEC 14496-10:2014. Coding of audio-visual objects – Part 10: Advanced Video Coding. URL: <https://www.iso.org/standard/55980.html> (дата звернення: 05.05.2025).
17. ISO/IEC 19510:2013. BPMN 2.0 Standard. URL: <https://www.iso.org/standard/62652.html> (дата звернення: 06.05.2025).
18. Kajiya J. T. The Rendering Equation // ACM SIGGRAPH, 1986, Vol. 20, № 4, С. 143-150.
19. Khronos Group. Vulkan API Specification. URL: <https://www.khronos.org/vulkan/> (дата звернення: 02.04.2025).
20. Kleiner K. Algorithms for Procedural Textures, Berlin: Springer Vieweg, 2013, 231 с.
21. McKinney W. Python for Data Analysis, 2nd ed., Sebastopol: O'Reilly Media, 2018, 544 с.
22. Mitchell D. P., Hanrahan P. Illumination from Curved Surfaces // ACM SIGGRAPH, 1992, Vol. 26, № 2, С. 283–291.
23. Ng W., Sun J. 3D Modelling with Blender for Beginners, Singapore: Springer, 2022, 206 с.
24. NVIDIA. Ray Tracing Gems II, Redmond: Apress, 2021, 651 с.
25. OpenAI. Cookbook Examples for GPT and Embedding APIs. URL: <https://github.com/openai/openai-cookbook> (дата звернення: 13.05.2025).
26. OpenCV. Open Source Computer Vision Library. URL: <https://opencv.org/> (дата звернення: 01.04.2025).
27. OpenGL Architecture Review Board. OpenGL Programming Guide, 9th ed., Boston: Addison-Wesley, 2016, 984 с.
28. Parberry I. Introduction to Game Physics with Box2D, Boca Raton: CRC Press, 2013, 290 с.
29. Perlin K. An Image Synthesizer // ACM SIGGRAPH Computer Graphics, 1985, Vol. 19, № 3, С. 287–296.

30. Piskin S. Procedural World Generation Techniques, Helsinki: Aalto University, 2021, 85 с.
31. Prusinkiewicz P., Lindenmayer A. The Algorithmic Beauty of Plants, New York: Springer-Verlag, 1990, 228 с.
32. Python Software Foundation. Python 3.12 Documentation. URL: <https://docs.python.org/3/> (дата звернення: 14.05.2025).
33. Real-Time Rendering. Web Resource. URL: <https://www.realtimerendering.com/> (дата звернення: 10.05.2025).
34. Reas C., Fry B. Processing: A Programming Handbook for Visual Designers and Artists, Cambridge: MIT Press, 2014, 672 с.
35. RTOUTI. Perlin Noise Algorithm. URL: <https://rtouti.github.io/graphics/perlin-noise-algorithm> (дата звернення: 30.03.2025).
36. Shiffman D. The Nature of Code, New York: Daniel Shiffman, 2012, 498 с.
37. Smith R. Procedural Modeling with L-systems, Utrecht: Utrecht University, 2008, 79 с.
38. Stack Overflow. URL: <https://stackoverflow.com/> (дата звернення: 11.05.2025).
39. Unity Technologies. Unity Manual. URL: <https://docs.unity3d.com/Manual/index.html> (дата звернення: 08.05.2025).
40. Unreal Engine. Unreal Engine Documentation. URL: <https://docs.unrealengine.com/> (дата звернення: 09.05.2025).
41. VanderPlas J. Python Data Science Handbook, Sebastopol: O'Reilly Media, 2016, 548 с.
42. Wikipedia. L-system. URL: <https://en.wikipedia.org/wiki/L-system> (дата звернення: 30.03.2025).
43. Williams L. Casting Shadows in Computer Graphics // ACM Computer Graphics, 1978, Vol. 12, № 3, С. 270–274.