КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

РОЗРОБКА СЕРЕДОВИЩА ДЛЯ 3D-МОДЕЛЮВАННЯ НА ОСНОВІ ФУНКЦІЇ ВІДСТАНІ

Виконала студентка 4-го курсу
Валерія ЯРОШЕНКО _____
(підпис)

Науковий керівник:
доцент, кандидат фізико-математичних наук
Оксана ШКІЛЬНЯК _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«____» _____ 2021 р.,
протокол № ___
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ - 2021

РЕФЕРАТ

Обсяг роботи 45 сторінок, 17 ілюстрацій, 12 джерел посилань.

3D, МОДЕЛЮВАННЯ, 3D-МОДЕЛЮВАННЯ, ФУНКЦІЯ ВІДСТАНІ, SDF, 3D-ОБ’ЄКТ, MARCHING CUBES, K-D-ДЕРЕВО, MESH, ПОЛІГОНАЛЬНА СІТКА, STL, ПОВЕРХНЯ.

Об’єктом роботи є процес дослідження функції відстані як методу аналітичного представлення об’єктів у задачах 3D-моделюванні.

Предметом роботи є прикладний програмний додаток для 3D-моделювання, на базі якого можна провести аналіз ефективності функції відстані як методу аналітичного представлення об’єктів у 3D просторі.

Метою роботи є дослідження аналітичних методів представлення 3D-об'єктів та аналіз їхньої ефективності у задачах 3D-моделювання.

Інструменти розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки Microsoft Visual Studio 2017, мова програмування C++ (17 стандарт), фреймворк для розробки графічного інтерфейсу Qt Framework (версія 5.13.0).

Результати роботи: готовий прикладний додаток для 3D-моделювання; висновки щодо ефективності функції відстані як методу аналітичного представлення об’єктів у 3D просторі.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – Application Program Interface

IDE – Integrated Development Environment

SDF – Signed Distance Function

GUI – Graphic User Interface

CAD – Computer Aided Design

SAH – Surface Area Heuristic

STL – Standard Triangle Language

ASCII – American Standard Code for Information Interchange

UML – Unified Modeling Language

ВСТУП

Оцінка сучасного стану об’єкта розробки. На сьогоднішній день 3D-візуалізація відіграє важливу роль в житті суспільства. 3D-моделювання активно використовується у наступних галузях:

- CAD-системи: для створення твердотільних елементів: будівель, деталей машин, механізмів.
- Розважальна: комп'ютерні ігри, елемент кінематографа, телебачення, друкованої продукції.
- Медична: при хірургічному втручанні використовують 3D-графіку для більш детального та якісного планування операцій.
- Виробництво меблів і комплектуючих. Компанії з виробництва меблевої продукції нерідко вдаються до використання тривимірної візуалізації своїх проектів.
- Реклама і маркетинг. Найчастіше рекламним агентствам потрібні нестандартні зображення, що привертають увагу, з чим вдало справляється тривимірна візуалізація.
- Геодезія. 3D-моделювання дозволяє розмістити і закріпити об'єкт в будь-якій системі координат з масштабною адаптацією. Можливість отримання просторових координат будь-якого вузла моделі проекту багаторазово спрощує процес виносу проектної моделі в натуру [1].
- 3D друк. Моделювання є невід’ємною складовою підготовки моделі до 3D друку – технології, що стає все більш поширеною останнім часом.

Актуальність роботи та підстави для її виконання. Наразі існує багато широко відомих застосунків для 3D-моделювання, які є у тому числі лідерами ринку у цій галузі. Однак, майже у кожному з них в реалізації використовуються «традиційні» та глибоко досліджені методи представлення 3D-об’єктів (а саме полігональна сітка з трикутників в більшості випадків).

Альтернативні методи в таких застосунках як правило не представлені, однак немає і чіткого обґрунтування, чому такі методи є гіршими за інші. З цього можна зробити висновок, що вони є ще недостатньо глибоко дослідженими, тож з огляду на все вищезгадане, робота є доволі актуальною.

Мета й завдання роботи. Метою роботи є дослідження аналітичних методів представлення 3D-об'єктів та аналіз їхньої ефективності у задачах 3D-моделювання. Для досягнення цієї мети поставлено наступні завдання:

- дослідження методів представлення примітивних геометричних 3D-фігур та базових операцій над ними функцією відстані;
- розробка прикладного програмного додатку для 3D-моделювання на базі аналітичних методів представлення об’єктів;
- аналіз ефективності аналітичних методів представлення об’єктів у задачах 3D-моделювання.

Об'єкт, методи й засоби розроблення. Об’єктом роботи є процес дослідження функції відстані як методу аналітичного представлення об’єктів при 3D-моделюванні.

Для розв’язання поставлених завдань використовувалися наступні методи дослідження: теоретичні (порівняння, аналіз) і емпіричні (проектування, реалізація алгоритмів, розробка).

В якості інструментів створення програмного засобу було обрано:

- Мову програмування C++, що є найбільш ефективною серед інших з точки зору швидкості виконання. Цей фактор грає важливу роль при роботі з геометрією та алгоритмами комп’ютерної графіки. Крім того, C++ надає можливість ефективного управління пам’яттю на низькому рівні, що є значною перевагою при роботі з великими за розміром даними (наприклад великим набором трикутників).
- Qt фреймворк для створення графічного користувацького інтерфейсу (GUI). Перевагами фреймворку є доступність (бібліотеки

безкоштовні та з відкритим кодом), гарна документованість і наявність готових компонентів для роботи з 3D-графікою.

- Microsoft Visual Studio 2017 – інтегроване середовище розробки (IDE) для мови C++. Перевагами середовища є потужний та зручний механізм для відлагодження та профайлінгу, а також сумісність з бібліотеками Qt (є можливість легкого завантаження та налаштування через NuGet пакет).

Можливі сфери застосування. Розроблений додаток може бути використаний для модулювання нескладних або суто аналітичних 3D-об’єктів для різноманітних цілей. Для спеціалістів-початківців з 3D-моделювання додаток може виступати у якості навчального для розвинення просторового мислення.

Результати аналізу ефективності використання функції відстані можуть слугувати базою для подальших досліджень альтернативних методів представлення 3D-об'єктів.

РОЗДІЛ 1

АНАЛІТИЧНІ МЕТОДИ ПРЕДСТАВЛЕННЯ ТА ПЕРЕТВОРЕННЯ 3D-ОБ'ЄКТІВ

1.1 Функція відстані у метричному просторі

У математиці та її застосуваннях функція відстані (або орієнтована функція відстані) (Signed Distance Function) множини Ω у метричному просторі X визначає відстань даної точки $x \in X$ від межі Ω, причому знак визначається тим, чи належить точка x до Ω. А саме, функція має позитивні значення в точках x всередині Ω, вона зменшується у значенні, коли x наближається до межі Ω, де функція відстані дорівнює нулю, і вона приймає негативні значення поза множиною Ω. [2] Однак іноді замість цього також приймається альтернативна конвенція (тобто негативна всередині Ω і позитивна зовні). [3]

Якщо Ω є підмножиною метричного простору X з метрикою d, то функція відстані f визначається як

$$f(x) = \begin{cases} d(x, \partial\Omega), & x \in \Omega \\ -d(x, \partial\Omega), & x \in \Omega^c \end{cases}$$

де $\partial\Omega$ позначає межу множини Ω.

Для будь-якого $x \in X$

$$d(x, \partial\Omega) := inf\ d(x, y), y \in \partial\Omega$$

де inf позначає infimum.

1.2 Представлення примітивних геометричних об'єктів функцією відстані

Наведемо функції відстані для деяких відомих примітивних геометричних об'єктів.

1.2.1 Сфера

Сфера однозначно задається у просторі центральною точкою (позначимо як $O = (O_x, O_y, O_z)$) та радіусом (позначимо як R).

Якщо S – множина точок сфери, то функція відстані S дорівнює

$$f_S(P) = \left|\overline{O - P}\right| - R$$

Псевдокод задання сфери через функцію відстані:

функція порахувати_відстань(P, O, R):

len ← відстань_між_точками(O, P)

повернути len – R

1.2.2 Прямокутний паралелепіпед

Прямокутний паралелепіпед однозначно задається у просторі центральною точкою (позначимо як $O = (O_x, O_y, O_z)$)) та довжинами трьох попарно перпендикулярних ребер (позначимо як L). Будемо вважати, що ребра паралельні до осей координат відповідно.

Псевдокод задання прямокутного паралелепіпеду через функцію відстані:

функція порахувати_відстань(P, O, L):

R ← L/2

d ← abs(P – O) – R

повернути |max(d, 0)| + min(0, max(d.x, d.y, d.z))

1.2.3 Тор

Тор однозначно задається у просторі центральною точкою (позначимо як $O = (O_x, O_y, O_z)$)), відстанню від цієї точки до осі обертання (позначимо як R) і радіусом твірного кола (позначимо як r). Будемо вважати, що вісь обертання паралельна площині O_{xy}.

Псевдокод задання тору через функцію відстані:

функція порахувати_відстань(P, O, R, r):

v ← P – O

q ← ($|v_x v_z|$ – R, v_y)

повернути |q| - r

1.2.4 Інші примітивні об’єкти

Окрім вищенаведених примітивів, функцію відстані можна легко обчислити для наступних об’єктів:

- полігональний тор (з довільною кількістю граней)
- циліндр;
- правильна призма (з довільною кількістю граней);
- конус (у тому числі усічений);
- правильна піраміда (у тому числі усічена та з довільною кількістю граней).

1.3 Булеві операції над 3D-об’єктами

Побудова більш складних об'єктів відбувається шляхом застосування до примітивів булевих (двійкових) операцій на множинах — об'єднання, перетин і різниця.

1.3.1 Операція об’єднання

Будемо вважати, що об’єднання двох або більше геометричних фігур у 3D просторі є окремим об’єктом. Тоді такий об’єкт можна представити у вигляді функції відстані. Для обчислення цієї функції повинні бути відомі (обчисленні) усі функції відстані об’єктів, до яких потрібно застосувати операцію об’єднання.

Якщо $\Omega_1, \Omega_2, \dots, \Omega_n$ є підмножинами метричного простору X з метрикою d, а $f_{\Omega_1}, f_{\Omega_2}, \dots, f_{\Omega_n}$ – їхні функції відстані відповідно, то функція відстані множини $\Omega_1 \cup \Omega_2 \cup \dots \cup \Omega_n$ визначається як

$$f_{\Omega_1 \cup \Omega_2 \cup \dots \cup \Omega_n} = \min(f_{\Omega_1}, f_{\Omega_2}, \dots, f_{\Omega_n})$$

Як видно з означення, операція є асоціативною та комутативною.

1.3.2 Операція перетину

Будемо вважати, що перетин двох або більше геометричних фігур у 3D просторі є окремим об’єктом. Тоді такий об’єкт можна представити у вигляді функції відстані. Для обчислення цієї функції повинні бути відомі (обчисленні) усі функції відстані об’єктів, до яких потрібно застосувати операцію перетину.

Якщо $\Omega_1, \Omega_2, \dots, \Omega_n$ є підмножинами метричного простору X з метрикою d, а $f_{\Omega_1}, f_{\Omega_2}, \dots, f_{\Omega_n}$ – їхні функції відстані відповідно, то функція відстані множини $\Omega_1 \cap \Omega_2 \cap \dots \cap \Omega_n$ визначається як

$$f_{\Omega_1 \cap \Omega_2 \cap \dots \cap \Omega_n} = \max(f_{\Omega_1}, f_{\Omega_2}, \dots, f_{\Omega_n})$$

Як видно з означення, операція є асоціативною та комутативною.

1.3.3 Операція різниці

Будемо вважати, що різниця двох геометричних фігур у 3D просторі є окремим об’єктом. Тоді такий об’єкт можна представити у вигляді функції відстані. Для обчислення цієї функції повинні бути відомі (обчисленні) обидві функції відстані об’єктів, до яких потрібно застосувати операцію різниці.

Якщо Ω_1 та Ω_2 є підмножинами метричного простору X з метрикою d, а f_{Ω_1} та f_{Ω_2} – їхні функції відстані відповідно, то функція відстані множини $\Omega_1 \backslash \Omega_2$ визначається як

$$f_{\Omega_1 \backslash \Omega_2} = \max(f_{\Omega_1}, -f_{\Omega_2})$$

Як видно з означення, операція не є ані асоціативною, ані комутативною.

1.4 Афінні операції над 3D-об’єктами

В евклідовій геометрії афінне перетворення або спорідненість (від лат. affinis, «пов’язаний з») – це геометричне перетворення, яке зберігає лінії та паралельність (але не обов’язково відстані та кути).

Більш загально, афінне перетворення – це автоморфізм афінного простору (евклідові простори – це специфічні афінні простори), тобто функція, яка відображає афінний простір на себе, зберігаючи при цьому і розмірність будь-якого афінного підпростору (це означає, що воно відображає точки у точки, прямі у прямі, площини у площини тощо) та відношення довжин відрізків паралельних прямих. Отже, набори паралельних афінних підпросторів залишаються паралельними після афінного перетворення. Афінне перетворення не обов'язково зберігає кути між прямими або відстані між точками, хоча воно зберігає співвідношення відстаней між точками, що лежать на прямій.

Якщо X – набір точок афінного простору, то кожне афінне перетворення на X може бути представлене як композиція лінійного перетворення на X і

зсуву X. На відміну від суто лінійного перетворення, афінне перетворення не повинно зберігати початок афінного простору. Отже, кожне лінійне перетворення є афінним, але не кожне афінне перетворення є лінійним.

Більш формально,

Афінне перетворення – відображення $f: R^n \rightarrow R^n$, яке можна записати у вигляді $f(x) = M * x + v$,

де M – невироджена матриця і $v \in R^n$.

Розміщення примітивів у різних місцях та орієнтація у просторі є фундаментальною операцією при моделюванні. Отже розглянемо детальніше базові афінні операції – зсув, обертання та масштабування.

1.4.1 Зсув та обертання

Будемо вважати, що у результаті застосування до об'єкту операцій зсуву або обертання утворюється новий окремий об'єкт. Тоді такий об'єкт можна представити у вигляді функції відстані. Для обчислення цієї функції повинна бути відома (обчислена) функція відстані початкового об'єкта, до якого потрібно застосувати операцію зсуву або обертання, а також матриця трансформації, яка кодує зсув або обертання.

Оскільки зсув та обертання не стискають і не розширюють простір, все, що нам потрібно зробити, це просто застосувати до обраної точки простору обернену трансформацією, яка використовується для розміщення об'єкта на сцені.

Якщо Ω є підмножиною метричного простору X з метрикою d, $x \in X$, f_Ω – її функція відстані, а T – матриця трансформації, то функція відстані множини $T(\Omega)$ визначається як

$$f_{T(\Omega)}(x) = f_\Omega(T^{-1}x)$$

1.4.2 Масштабування

Будемо вважати, що у результаті застосування до об’єкту операції масштабування утворюється новий окремий об’єкт. Тоді такий об’єкт можна представити у вигляді функції відстані. Для обчислення цієї функції повинна бути відома (обчислена) функція відстані початкового об’єкта, до якого потрібно застосувати операцію масштабування, а також коефіцієнт масштабування.

Якщо Ω є підмножиною метричного простору X з метрикою d, $x \in X$, f_Ω – її функція відстані, а s – коефіцієнт масштабування (може бути скаляр або вектор), то функція відстані множини Ω_s визначається як

$$f_{\Omega_s}(x) = f_\Omega(x)s$$

1.5 Дерево перетворень

У моделюванні складні об’єкти утворюються шляхом застосування операцій до більш простих, примітивних об’єктів (основні примітиви та операції над ними було розглянуто вище). Аналітичний спосіб представлення складних об’єктів вимагає збереження усього ланцюга перетворень, які були застосовані до об’єкта. Найбільш ефективна структура даних для зберігання таких ланцюгів є орієнтоване дерево (або ациклічний орграф), де листя відповідають примітивам, а усі інші вузли (у тому числі корень) – операціям над об’єктами, що відповідають дочірнім вузлам [4]. На рисунку 1 наведено умовну схему дерева.

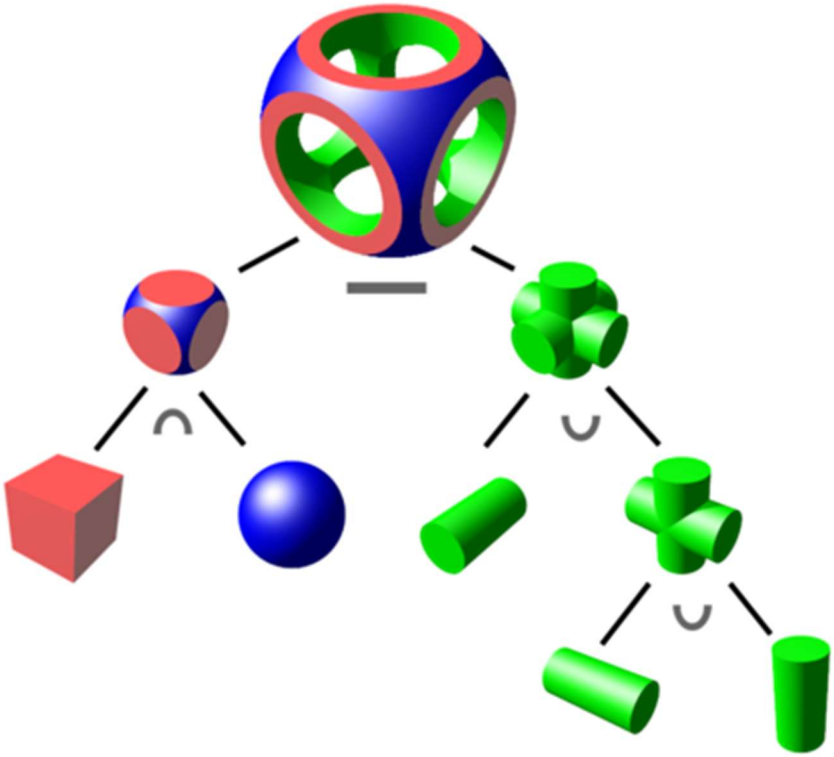

Рис. 1. Схема дерева перетворень для представлення складних об'єктів.

При цьому, будь-яке піддерево буде відповідати окремому об’єкту у 3D просторі. Для того, щоб обчислити функцію відстані деякого піддерева (або 3D-об’єкта), потрібно спочатку обчислити функцію відстані для його дочірніх піддерев, а потім застосувати до них відповідну операцію. Таким чином, обчислення функції відстані для дерева починається знизу, з листків, йде догори, і закінчується у корені. Результуюча функція і буде функцією відстані для складного 3D-об’єкта, заданого деревом.

Таке представлення складних об’єктів надає прозорості щодо способу їхньої побудови, а також дозволяє легко заміняти одні операції та/або примітиви іншими, без необхідності перебудовувати весь об’єкт.

РОЗДІЛ 2

ВІЗУАЛІЗАЦІЯ АНАЛІТИЧНО ПРЕДСТАВЛЕНИХ 3D-ОБ'ЄКТІВ

При аналітичному представленні 3D-об'єктів постає проблема зручної їхньої візуалізації для користувача на сцені. Одним із можливих рішень є створення власного програмного компонента для рендерингу об'єктів, заданих аналітично функцією відстані, у зображення. Але таке рішення є досить складним та дорогим з огляду на час розробки у порівнянні з іншими методами візуалізації і не розглядається у рамках даної роботи.

Іншим більш простішим рішенням є генерація полігональної сітки об'єкта на основі його функції відстані. Оскільки, такий спосіб представлення 3D-об'єктів є найбільш поширеним, існує багато готових рішень для рендерингу, якими можна скористатись. Зокрема Qt фреймворк надає функціонал для рендерингу через API класу Qt3DRender [5].

2.1 Полігональна сітка

Полігональна сітка (англ. Polygon mesh) — це набір вершин, ребер, та граней, що описують форму багатогранного об'єкта в тривимірній графіці та твердотілому моделюванні. Грані зазвичай складаються з трикутників (сітка з трикутників), чотирикутників, чи інших опуклих многокутників, що спрощує їхній рендеринг, хоча можуть використовуватись і загальніші, неопуклі многокутники, чи багатокутники з дірками.

Дослідження полігональних сіток — це великий підрозділ комп'ютерної графіки та геометричного моделювання. Різні представлення полігональних сіток використовуються для різних цілей та застосунків. Серед операцій, які можна виконувати над сітками, можуть бути операції булевої алгебри, згладжування, спрощення та багато інших. Для передачі полігональних сіток по мережі використовуються мережеві представлення, такі як «потокові» та

«прогресивні» сітки. Об'ємні сітки відрізняються від полігональних тим, що вони явно представляють поверхню та об'ємні структури, тоді як полігональні сітки явно представляють лише поверхню, а об'єм залишається неявним. Оскільки полігональні сітки широко використовуються в комп'ютерній графіці, для них розроблені алгоритми трасування променів, виявлення зіткнень та динаміки твердих тіл.

Об'єкти створені за допомогою полігональних сіток повинні зберігати різні типи елементів, такі як вершини, ребра, грані, многокутники (полігони) та поверхні. У багатьох випадках зберігаються лише вершини, ребра і або грані, або многокутники. Рендерер може підтримувати лише трьох-сторонні грані, так що полігони повинні бути побудовані з їхньої множини. Однак багато рендерерів підтримують многокутники з чотирма та більше сторонами, або вміють триангулювати многокутники в трикутники на льоту, роблячи необов'язковим зберігання сітки в тріангульованьому вигляді.

Необхідні типи елементів:

- Вершина. Окрім координат, містить також іншу інформацію, таку як колір, нормальний вектор та координати текстури.
- Ребро з'єднує дві вершини.
- Грань це набір ребер, в якому трикутна грань має три ребра та чотирикутна — чотири. Багатокутник (Полігон) містить довільну кількість граней. У системах, які підтримують багатосторонні грані, полігони та грані еквівалентні. Однак, більшість апаратного забезпечення для рендеринга підтримує лише грані з трьома або чотирма сторонами, так що полігони представлені як множина граней. Математично, полігональна сітка може бути представлена у вигляді неструктурованою сітки, або неорієнтованого графа, з додаванням властивостей геометрії, форми та топології.

Полігональні сітки можуть бути представлені багатьма методами, використовуючи різні способи зберігання вершин, ребер і граней. У них входять:

- Список граней: опис граней відбувається за допомогою покажчиків на список вершин.
- «Крилате» представлення: у ньому кожна точка ребра вказує на дві вершини, дві грані і чотири (за годинниковою стрілкою і проти годинникової) ребра, яким вона належить. Крилате подання дозволяє обійти поверхню за сталий час, але у нього великі вимоги по пам'яті для зберігання даних опису сітки.
- Півреберні сітки: спосіб схожий на «крилате» представлення, за винятком того, що використовується інформація обходу лише половини грані.
- Чотириреберні сітки, які зберігають ребра, півребра і вершини без будь-якої вказівки на полігони. Полігони прямо не виражені в представленні, і можуть бути обчислені при обході структури.
- Таблиця кутів, які зберігають вершини в зумовленій таблиці, такій, що обхід таблиці неявно задає полігони. По суті, це «віяло трикутників», що використовується в апаратному рендерингу. Представлення більш компактне і більш ефективне для знаходження полігонів, але операції по їхньої зміні повільні. Більш того, таблиці кутів не подають сітки повністю. Для представлення більшості сіток потрібно кілька таблиць кутів (віял трикутників).
- Вершинне представлення: представлені лише вершини, що вказують на інші вершини. Інформація про грані та ребра виражена неявно в цьому поданні. Однак, простота представлення робить операції над сіткою доволі ефективними.

Кожне з представлень має свої переваги і недоліки [6]. Вибір структури даних визначається застосуванням, необхідною швидкодією, розміром даних,

операціями, які будуть виконуватися. Наприклад, легше мати справу з трикутниками, ніж з багатокутниками загального вигляду, особливо в обчислювальній геометрії. Для певних операцій необхідно мати швидкий доступ до топологічної інформації, такої як ребра або сусідні грані; для цього потрібні більш складні структури, такі як «крилате» представлення. Для апаратного рендерингу потрібні компактні, прості структури; тому в API низького рівня, такі як DirectX і OpenGL зазвичай включена таблиця кутів (віяло трикутників).

2.2 Marching cubes алгоритм генерації полігональної сітки

Marching cubes — алгоритм комп'ютерної графіки, вперше опублікований SIGGRAPH у 1987, розроблений Лоренсеном та Кляйном [7], для генерації полігональної сітки ізоповерхні з тривимірного скалярного поля (яке іноді називають вокселями). Еквівалентний двовимірний метод називають алгоритмом marching squares.

Алгоритм потребує на вхід функцію, що описує об’єкт і яку можна дискретизувати у просторі. Такою функцією цілком може виступати функція відстані. У результаті ми зможемо встановити, у яких точках функція робить перехід з додатної у від’ємну і навпаки.

Для створення полігональною сітки потрібно обчислити межу функції відстані, тобто точки між її додатними та від’ємними значеннями, де вона перетинає нуль. Алгоритм Marching cubes створює апроксимацію такої межі, яка потім використовується для рендерингу. У 2D просторі ця межа буде неперервною лінією, а у 3D – полігональною сіткою.

Для більшої наочності та розуміння, розглянемо спочатку роботу алгоритму у випадку 2D простору.

2.2.1 Алгоритм Marching cubes у 2D просторі

Розіб’ємо простір на квадрати одного розміру. Далі для кожного квадрата і кожної з його чотирьох вершин з’ясуємо за допомогою функції відстані, знаходиться дана вершина всередині або ззовні об’єкта (у точках, які знаходяться всередині об’єкта функція відстані від’ємна і навпаки).

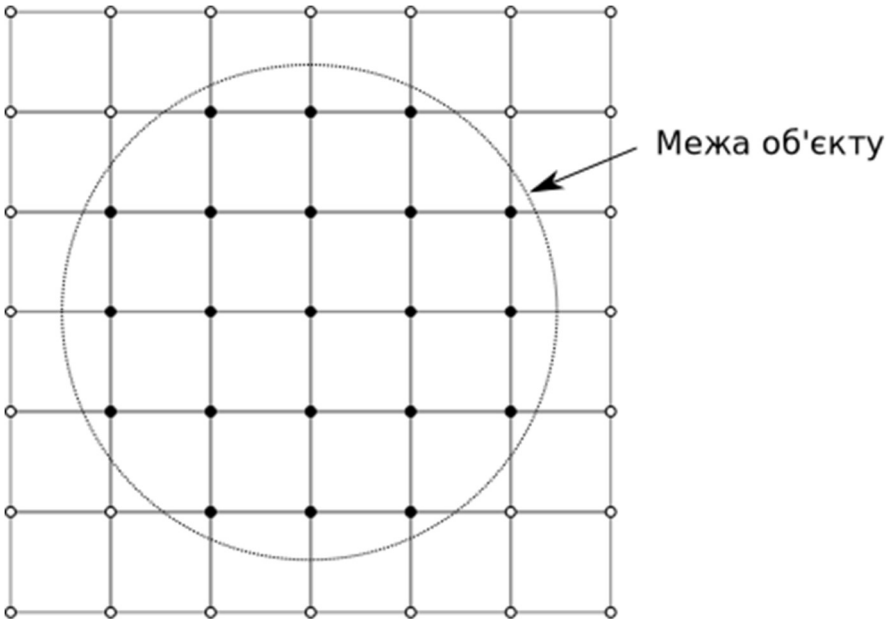

Рис. 2. Графік функції у 2D просторі, що аналітично описує коло.

На рисунку 2 зображено графік функції, що задає коло, тобто межу цього кола. Зафарбованими точками зазначено усі вершини квадратів, у координатах яких функція набуває від’ємного значення, а порожніми точками – вершини, у координатах яких функція набуває додатного значення.

Далі ми обробляємо кожен квадрат незалежно, заповнюючи його ребрами в залежності від комбінації його зафарбованих та порожніх вершин (рис. 3).

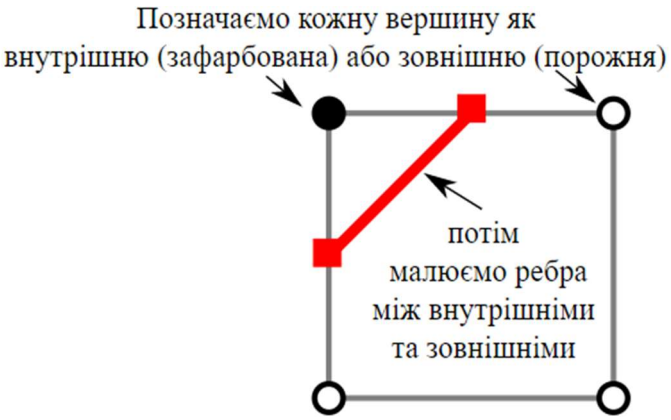

Рис. 3. Заповнення квадрату простора ребрами.

Усього існує $2^4 = 16$ можливих комбінацій. Відповідні для кожної комбінації ребра представлені на рисунку 4.

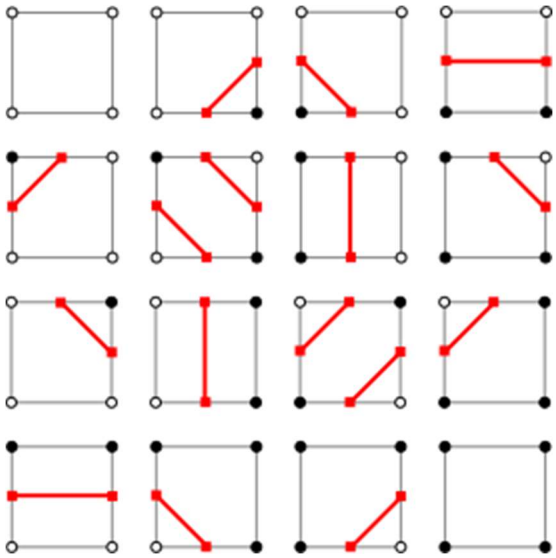

Рис. 4. Можливі комбінації внутрішніх та зовнішніх вершин квадрату.

Після обробки усіх квадратів, усі результуючі ребра з’єднуються і утворюють неперервний контур, не дивлячись на те, що кожен квадрат розглядався окремо і незалежно від інших.

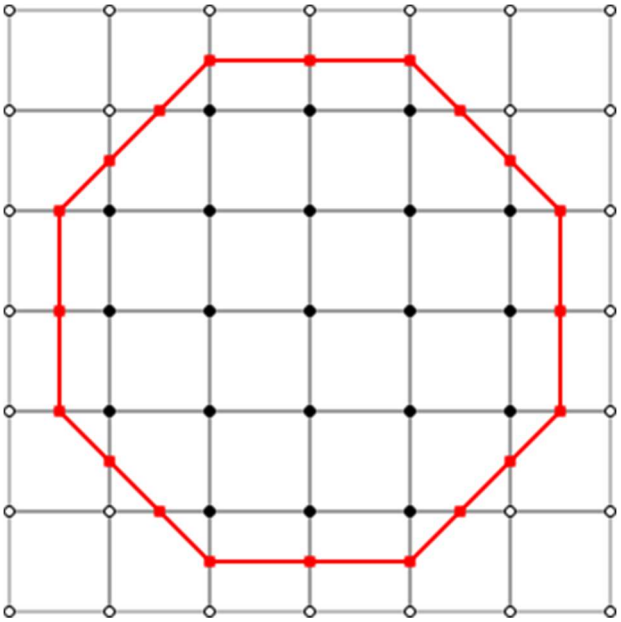

Рис. 5. Контур отриманий у результаті неадаптивного алгоритму Marching Cubes.

Отриманий контур (зображено на рисунку 5) у цілому є подібним до початкового, що заданий функцією, але він все ще має чітко виражені кути під 45 градусів, що ускладнює розпізнавання гладких контурів. Такий результат був отриманий, тому що ми обирали такі вершини межі, щоб вони були рівновіддалені від вершин квадрата.

Для того, щоб покращити візуалізацію гладких контурів застосовується адаптивний алгоритм Marching cubes – більш вдосконалена версія початкового алгоритму [8]. Його суть полягає у тому, що замість вибору точок по центру ребра квадрата, вони обирається на ребрі таким чином, щоб найбільше відповідати реальній функції. Для цього вже недостатньо функції, яка з’ясовує лежить точка простору всередині контуру або ззовні. Потрібна більш строга функція, яка б обчислювала наскільки глибоко точка лежить всередині або ззовні контуру. Очевидно, функція відстані задовольняє вимогам, тож ми можемо застосувати адаптивний алгоритм Marching cubes для генерації полігональної сітки з об’єктів, описаних функцією відстані.

Для будь-якої точки ребра квадрата можна обчислити наскільки близько ця точка лежить до межі об’єкта (через функцію відстані). Таким чином, можна знайти таку точку на стороні квадрата, в якій функція приймає нуль, тобто через яку проходить контур (рис. 6).

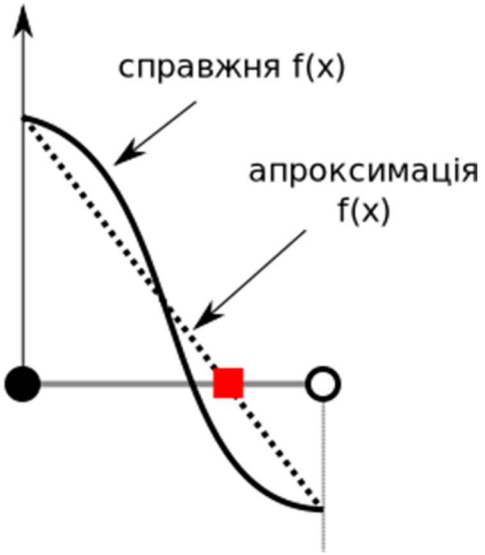

Рис. 6. Апроксимація контуру об’єкта.

Після з’єднання усіх отриманих точок на ребрах квадрата, утворюється більш точна апроксимація початкового контуру (зображено на рисунку 7).

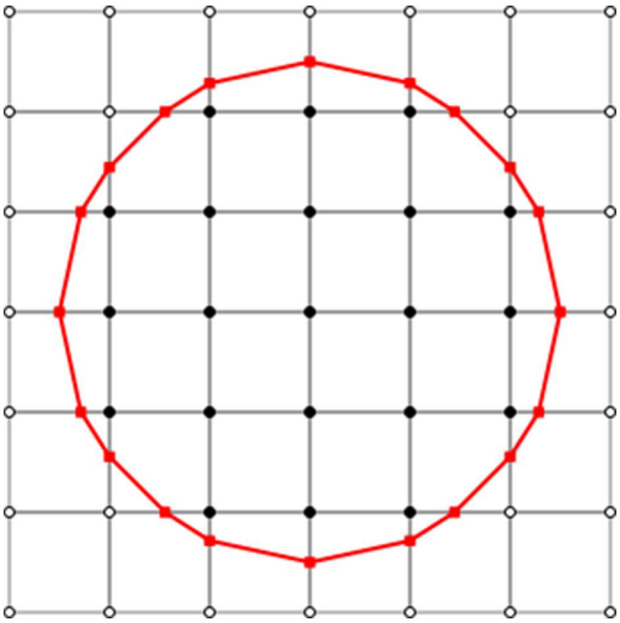

Рис. 7. Контур отриманий у результаті адаптивного алгоритму Marching Cubes.

Не дивлячись на те, що у вихідного контуру така ж сама кількість вершин і ребер, що і у контуру, отриманого у результаті неадаптивного алгоритму, він набагато точніше апроксимує гладку функцію.

2.2.2 Алгоритм Marching cubes у 3D просторі

Принцип роботи алгоритму у 3D просторі є аналогічним 2D випадку. Простір розбивається на куби однакового розміру. Далі для кожного куба і кожної з його восьми вершин з'ясуємо за допомогою функції відстані, знаходиться дана вершина всередині або ззовні об'єкта (у точках, які знаходяться всередині об'єкта функція відстані від'ємна і навпаки).

Далі ми обробляємо кожен квадрат незалежно, заповнюючи його трикутниками в залежності від комбінації його зафарбованих та порожніх вершин. Усього існує $2^8 = 256$ можливих комбінацій. Але це не означає, що треба розглядати кожну з цих комбінацій окремо. Справді, багато з них є дзеркальними відображеннями або поворотами один одного.

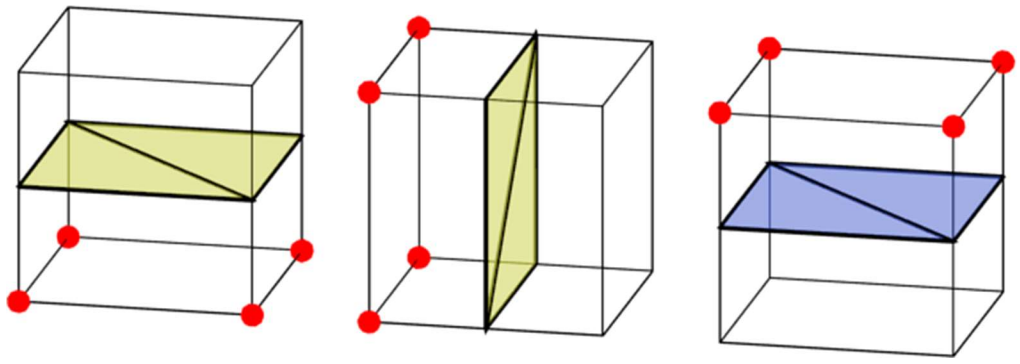

Рис. 8. Приклад комбінацій, які є поворотами один одного.

На рисунку 8 зображено три різні комбінації внутрішніх та зовнішніх вершин куба (у даному прикладі красні вершини відповідають внутрішнім, а усі інші – зовнішнім). У першому випадку нижні вершини є внутрішніми, а верхні – зовнішніми, отже для правильної генерації полігональної сітки потрібно розмежувати куб вертикально (у даному прикладі зовнішня сторона

сітки пофарбована у жовтий колір, а внутрішня – у синій). Легко бачити, що два інших випадка можна звести до першого звичайним обертанням.

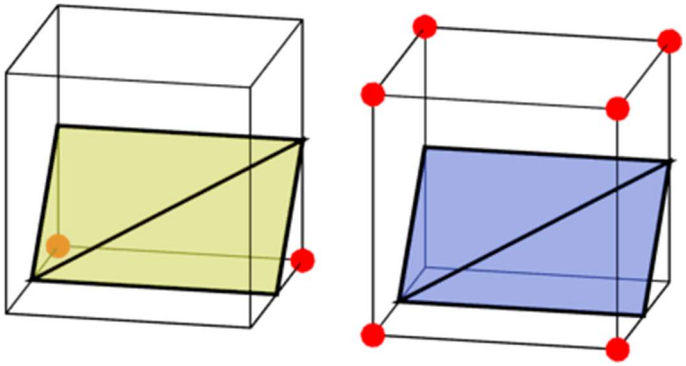

Рис. 9. Приклад комбінацій, які є протилежними.

Дві інші аналогічні комбінації зображені на рисунку 9. Такі випадки є протилежними один одному, тобто внутрішні вершини одного куба є зовнішніми для іншого, і навпаки. Можна легко звести один випадок до іншого, вони будуть мати однакову результуючу поверхню (однакові трикутники).

Беручи до уваги усі випадки, описані вище, існує усього 15 дійсно унікальних комбінацій, які треба розглянути окремо. Усі ці комбінації зображено на рисунку 10.

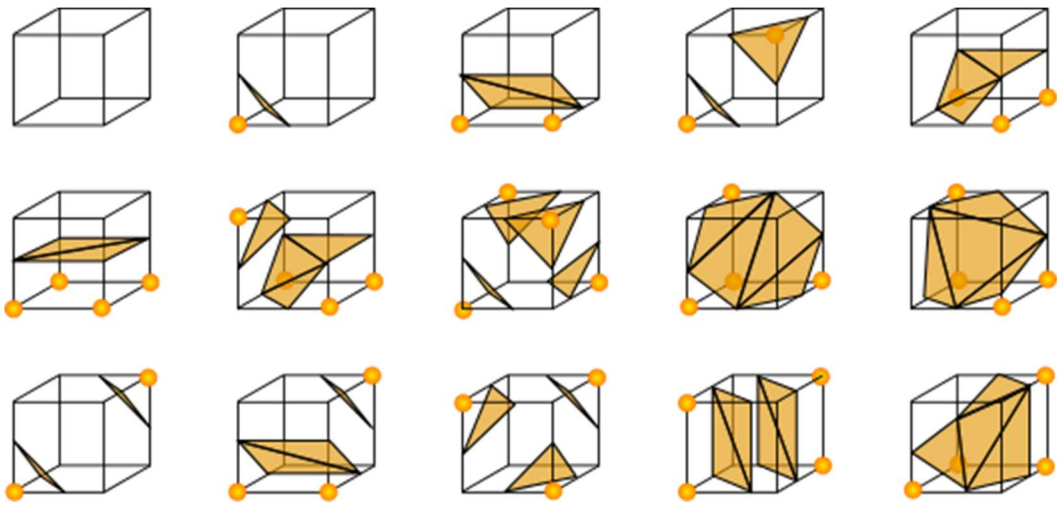

Рис. 10. Унікальні комбінації внутрішніх та зовнішніх вершин.

Далі, як і у випадку з 2D простором, усі утворені трикутники об’єднуються і утворюють неперервну полігональну сітку.

2.3 Аналітичне представлення полігональної сітки на основі k-d-дерева

При створенні додатку для моделювання постає також задача, обернена до попередньої, а саме представлення об’єкту, заданого полігональною сіткою у вигляді функції відстані. Це може бути корисно, якщо у користувача з’явиться потреба працювати не з геометричними примітивами, а з уже існуючими 3D-моделями, завантаженими ззовні.

Постає задача швидкого обчислення відстані від будь-якої точки простору до межі заданого полігональною сіткою 3D-об’єкта. Очевидно, що рішення «грубої сили», тобто перебір усіх трикутників сітки, є занадто повільним, адже має лінійну складність. Враховуючи те, що кількість точок, для яких потрібно порахувати функцію відстані може досягати мільйонів, а кількість трикутників у сітці – десятків тисяч, лінійна складність для обробки однієї точки нас не влаштує. Більш оптимальне рішення забезпечує спеціальна структура даних – k-d-дерево.

k-d-дерево (k-мірне дерево) – структура даних, яка дозволяє розбити k-мірний простір на «менші частини», за допомогою перетину цього самого простору гіперплощинами ($k > 3$), площинами ($k = 3$), прямими ($k = 2$), і в разі одновимірного простору – точкою (виконуючи пошук в такому дереві, отримуємо щось схоже на бінарний пошук) [9][10].

Таке розбиття зазвичай використовують для звуження діапазону пошуку в k-вимірному просторі. Наприклад, пошук ближнього об'єкта (вершини, сфери, трикутника і т.д.) до точки, проектування точок на трьохвимірну сітку, трасування променів (активно використовується в Ray Tracing) і т.п. При цьому об'єкти простору поміщаються в спеціальні обмежуючи паралелепіпеди, сторони яких паралельні осям координат. Обмежуючим

паралелепіпедом будемо називати такий найменший можливий паралелепіпед, який повністю включає в себе початкову множину об'єктів або сам об'єкт, якщо ми будуємо обмежуючий паралелепіпед лише для нього. У точок в якості обмежуючого паралелепіпеду береться паралелепіпед з нульовою площею поверхні і нульовим обсягом.

2.3.1 Побудова дерева

При побудові дерева відбувається поділ k-мірного простору на так звані «вузли». Всі об'єкти поміщаються в один великий обмежуючий паралелепіпед (при цьому кожен об'єкт все ще обмежений своїм власним паралелепіпедом), який потім ділиться площиною, паралельної одній з його сторін, на два. Відповідно, у дерево додаються два нових вузла. Таким же чином кожен з отриманих паралелепіпедів розбивається на два і т.д. Процес завершується або за спеціальним критерієм або при досягненні певної глибини дерева, або ж при досягненні певного числа елементів всередині вузла дерева. Деякі елементи можуть одночасно входити як в лівий, так і в правий вузли (наприклад, коли в якості елементів дерева розглядаються трикутники).

2.3.2 Стратегії поділу вузлів

Велике значення для ефективності роботи дерева має вибір площини для поділу вузлів. Існує багато стратегій вибору, нижче приведено деякі, найбільш поширеніші на практиці (передбачається, що початкові об'єкти поміщені в один великий обмежуючий паралелепіпед, а поділ відбувається паралельно одній з осей координат):

- Вибрати найбільшу сторону обмежуючого паралелепіпеду. Тоді січна площина буде проходити через середину обраної сторони.

- Розсікати по медіані: відсортуємо всі елементи по одній з координат, а медіаною назвемо елемент (або центр елемента), який знаходиться на середній позиції в відсортованому списку. Січна площина буде проходити через медіану так, що кількість елементів зліва і справа буде приблизно рівною.
- Чергувати сторони при розбитті: на початковій глибині проведемо січну площину через середину сторони паралельної OX, на наступному рівні глибини – через середину сторони паралельної OY, потім OZ і т.д. Після того, як розподіл проведений по всім осям, починаємо знову з першої (OX).
- Найбільш оптимальний на практиці спосіб – використовувати SAH (Surface Area Heuristic) функцію оцінки поділу обмежуючого паралелепіпеду. SAH також надає універсальний критерій зупинки поділу вузла.

Стратегія вибору на основі найбільшої сторони та стратегія чергування є досить ефективними на практиці з огляду на швидкість побудови дерева (це обумовлено тим, що операція пошуку та проведення січної площини через середину обмежуючого паралелепіпеду є тривіальною та швидкою). У той же час, доволі часто при застосуванні цих стратегій простір розбивається нещільно, що негативно впливає на ефективність основних операцій в k-d-дереві.

Стратегія вибору медіани дозволяє досягти гарних результатів, але вимагає чимало часу на сортування елементів вузла. Як і у випадку стратегій 1 та 3, метод досить простий у реалізації.

Найбільший же інтерес представляє стратегія з використанням «розумної» SAH евристики. Обмежуючий паралелепіпед вузла дерева поділяється N площинами, паралельними осям координат, на N + 1 рівні частини по кожній із сторін. Насправді, кількість площин для кожної сторони може бути будь-якою, але для простоти та ефективності зазвичай обирається

константа. Далі, в можливих точках перетину площини з обмежуючим паралелепіпедом обчислюють значення спеціальної функції: SAH. В результаті розподіл проводиться площиною з найменшим значенням SAH функції. Наприклад, на рисунку 11 мінімум досягається в SAH під номером 7, отже, розподіл буде проводитися саме цією площиною.

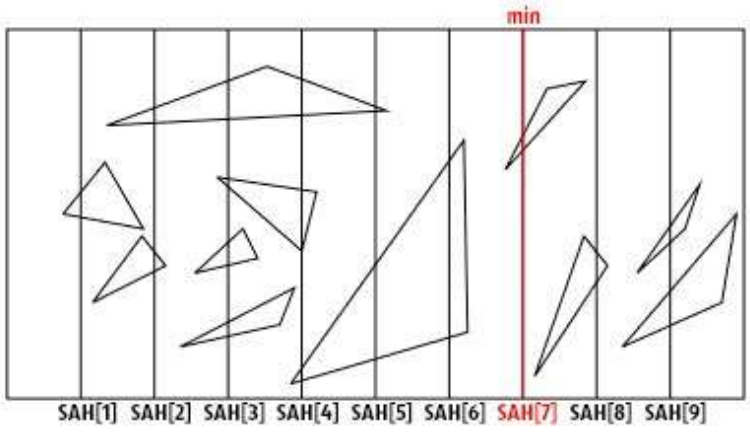

Рис. 11. Значення SAH функції по різним площинам.

Значення SAH функції для поточної площині обчислюється наступним чином:

$$f(x) = C_t + C_i * \frac{SA_L(x) * N_L(x) + SA_R(x) * N_R(x)}{SA_{parent}}, \text{де:}$$

C_t – вартість простеження променю всередині вузла;

C_i – вартість перетину трикутника променем;

SA_L, SA_R – площа поверхні лівого та правого вузлів відповідно, що утворюються в результаті можливого ділення батьківського вузла поточною площиною;

N_L, N_R – кількість елементів лівого та правого вузлів відповідно, що утворюються в результаті можливого ділення батьківського вузла поточною площиною;

SA_{parent} – площа поверхні батьківського вузла.

Перебираючи константи C_i і C_t, можна досягти більш щільної структури дерева (або ж навпаки), жертвуючи часом роботи алгоритму. У багатьох статтях, присвячених в першу чергу побудові k-d-дерева для Ray Tracing рендеру, автори використовують значення C_i = 1, C_t = [1, 2]: чим більше значення C_t, тим швидше будується дерево, і тим гірше результати простеження променю в такому дереві.

Для ефективного обчислення значення SAH функції необхідно вміти швидко визначити, скільки вузлових елементів знаходяться ліворуч та праворуч від даної площини. Час побудови дерева буде незадовільний, якщо використовувати підхід «грубої сили» з квадратичною асимптотикою. Для покращення часу виконання можна застосувати метод «кошика»: для кожної з N + 1 частини обмежуючого паралелепіпеду обчислимо одноразово кількість елементів, які повністю або частково (тобто перетинаючи) знаходяться всередині цієї частини. Далі обчислимо одноразово частинні-постфікс та частинні-префікс суми на основі інформації про кількість елементів. Маючи частинні суми, можна швидко (за O(1)) обчислити кількість елементів ліворуч та праворуч від довільної січної площини з N.

Критерій зупинки поділу вузла у випадку методу вибору через SAH функцію:

$$C_i * N < f(x)$$

Іншими словами, якщо вартість простежування дочірніх вузлів більше вартості простежування батьківського вузла, то розподіл потрібно припинити.

2.3.3 Пошук найближчого до точки елементу

Алгоритм пошуку найближчого елемента дерева до заданої точки простору передбачає наступні кроки:

- Крок 1. Знайдемо найближчий листовий вузол до заданої точки і позначимо його як nearestNode. В кожному вузлі, як відомо, зберігається обмежуючий паралелепіпед, тож відстань до його центру можна легко обчислити як (pointMax + pointMin) * 0.5.
- Крок 2. Методом перебору заходимо серед усіх елементів знайденого вузла (nearestNode) такий, відстань від якого до заданої точки є мінімальною. Отриману відстань позначимо як minDist.
- Крок 3. Побудуємо сферу з центром у заданій точці і радіусом довжини minDist. Перевіримо, чи лежить ця сфера повністю всередині вузла nearestNode. Якщо лежить, то найближчий елемент знайдено, якщо ні – то перейдемо до наступного кроку.
- Крок 4. Починаючи з кореня дерева, спробуємо знайти найближчий елемент в радіусі: спускаючись вниз по дереву, перевіряємо, чи перетинає побудована сфера правий або лівий вузли. Якщо який-небудь вузол був пересічений, то аналогічну перевірку виконуємо і для дочірніх вузлів цього ж вузла. Якщо ми потрапили у листовий вузол, то знайдемо перебором найближчий до точки елемент в цьому вузлі і порівняємо отриманий результат з довжиною радіуса сфери. Якщо радіус сфери більше знайденої відстані, оновимо довжину радіуса значенням мінімуму.
- Крок 5. Елемент, відстань від якого до точки дорівнює оновленому востаннє радіусу сфери і є найближчим елементом до заданої точки.

На рисунку 12 зображено наступну теоретично можливу ситуацію: припустимо, ми знайшли найближчий листовий вузол (позначено блакитним) до даної точки (позначена червоним), тоді, скориставшись пошуком найближчого елемента у вузлі, отримаємо, що таким є трикутник із індексом 1, однак, це не так. Коло з радіусом R перетинає суміжний вузол, отже, пошук потрібно виконати і в цьому вузлі, а потім порівняти новий знайдений мінімум

із поточним. Як підсумок, стає очевидним, що найближчим є трикутник з індексом 2.

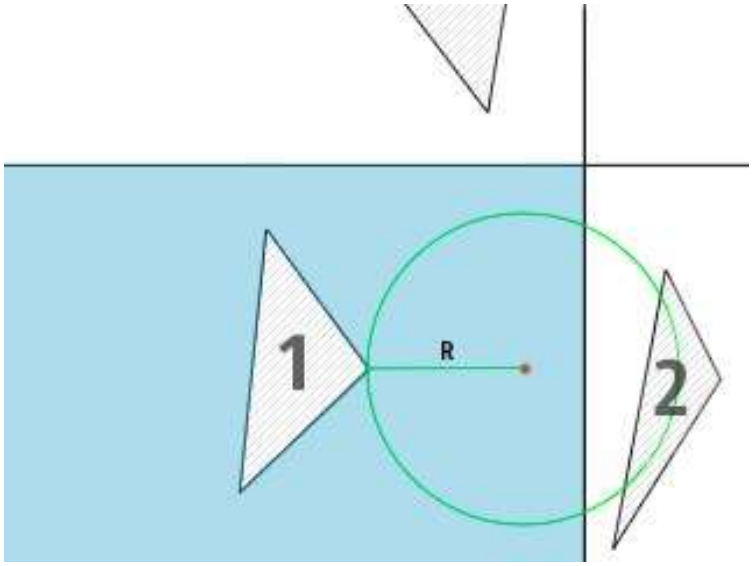

Рис. 12. Найближчий трикутник знаходиться не у найближчому листовому вузлі.

РОЗДІЛ 3

ПРОЕКТУВАННЯ ТА РОЗРОБКА ДОДАТКУ ДЛЯ 3D-МОДЕЛЮВАННЯ

3.1 Архітектура додатку

3.1.1 Plugin-based архітектура

Задля забезпечення більшої гнучкості та можливості розширення, було прийнято рішення використовувати plugin-based архітектуру. Усі компоненти для представлення 3D-об’єктів, тобто примітивні геометричні фігури та операції реалізовано як плагіни. Кожен окремний компонент реалізує необхідні інтерфейси та в результаті компілюється у бібліотеку динамічного компонування dll. У подальшому усі такі бібліотеки динамічно завантажуються у додаток спеціальним модулем (PluginLoader). Таким чином, користувач має змогу створити та завантажити свої власні плагіни, що описують специфічні об’єкти та операції для більш вузькоспеціалізованих потреб.

На рисунку 13 зображена UML діаграма класів, що описує інтерфейси, які необхідно реалізувати компонентам, а також приклад деяких класів, що реалізують ці інтерфейси (примітив сфери і булева операція об’єднання).

Інтерфейс IObjectDescription надає загальну інформацію про компонент: його назву, ID, іншу додаткову інформацію.

Інтерфейс IObjectCalculator дозволяє обчислювати математичні характеристики, такі як функція відстані, баундінг бокс та інші. Очевидно, компонент не повинен реалізовувати усі методи інтерфейсу, а тільки ті, які є релевантними для цього компоненту (інші залишаються із дефолтовою реалізацією). Наприклад клас SphereCalculator реалізовує метод CalculateSDF, тому що сфера – примітивний геометричний об’єкт, тож вона має свою власну функцію відстані, і не реалізовує метод CalculateFromChildrenSDF, тому що

будучи листовою вершиною дерева обчислень не очікує наявність дочірніх вузлів. UnionOperationCalculator навпаки – реалізує метод CalculateFromChildrenSDF замість CalculateSDF, тому що операція об’єднання не має своєї власної функції відстані і має сенс тільки при наявності дочірніх вузлів.

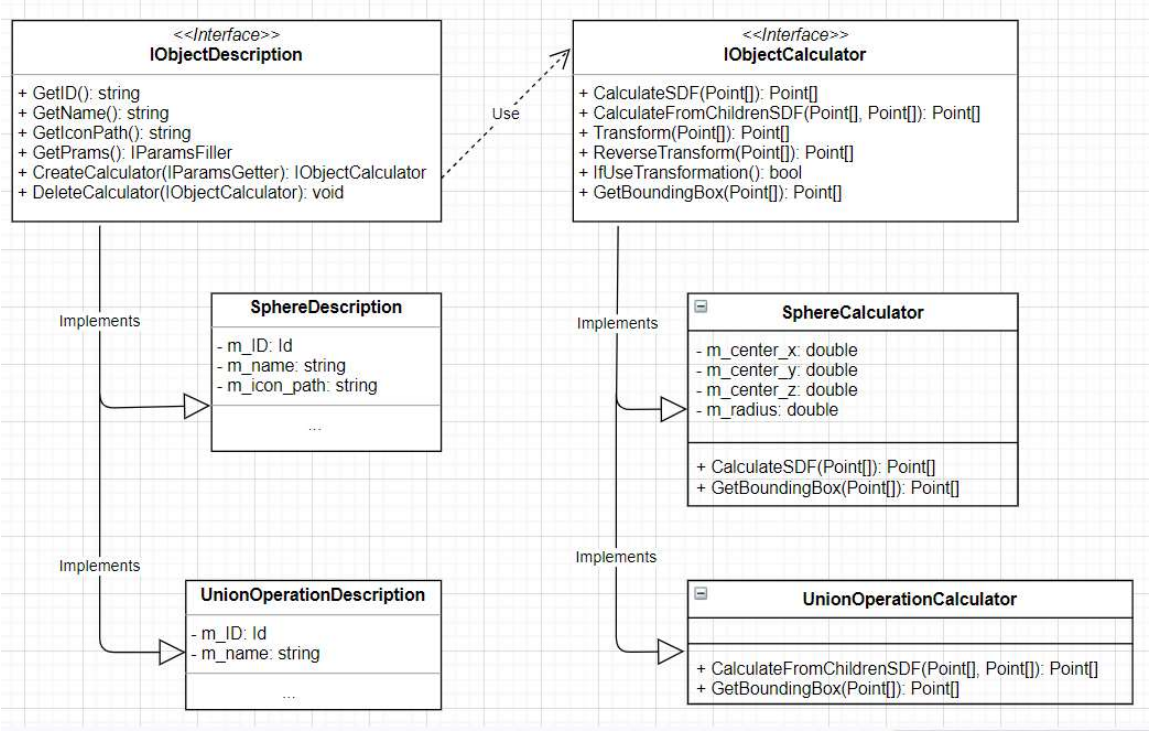

Рис. 13. UML діаграма класів.

Окрім наведених реалізацій, були також створені наступні:

- BoxDescription та BoxCalculator;
- ConeDescription та ConeCalculator;
- CylinderDescription та CylinderCalculator;
- TorusDescription та TorusCalculator;
- PolygonalTorusDescription та PolygonalTorusCalculator;
- PrismDescription та PrismCalculator;
- PyramidDescription та PyramidCalculator;
- MeshDescription та MeshCalculator;

- IntersectOperationDescription та IntersectOperationCalculator;
- SubtractOperationDescription та SubtractOperationCalculator;
- RotateOperationDescription та RotateOperationCalculator;
- ScaleOperationDescription та ScaleOperationCalculator;
- TranslateOperationDescription та TranslateOperationCalculator;
- PlaneCutOperationDescription та PlaneCutOperationCalculator;
- OffsetOperationDescription та OffsetOperationCalculator.

3.2 Можливості системи

3.2.1 Імпорт 3D-моделей у форматі STL

У межах роботи над додатком було реалізовано програмний модуль, що відповідає за імпорт сторонніх 3D-моделей у форматі STL.

STL (від англ. Stereolithography) - формат файлу для зберігання тривимірних моделей об'єктів, широко використовується в адитивних технологіях та CAD-додатках [11][12]. Абревіатура STL має кілька розшифрувань, таких як «Standard Triangle Language» («Стандартна мова трикутника») та «Standard Tessellation Language» («Стандартна мова тесселяції»). Інформація про об'єкт зберігається як список трикутних граней, які описують його поверхню, і їхніх нормалей. Формат STL підтримує як текстові (ASCII), так і двійкові подання. Бінарні файли є більш поширеними, оскільки вони є більш компактними.

Текстовий STL файл починається з рядка:

solid name,

де name – це необов'язковий параметр (але якщо name опущено, все одно повинен бути пробіл після solid). Файл продовжується довільним числом трикутників, що описуються в такий спосіб:

facet normal $n_i\ n_j\ n_k$

outer loop

vertex $v1_x\ v1_y\ v1_z$

vertex $v2_x\ v2_y\ v2_z$

vertex $v3_x\ v3_y\ v3_z$

endloop

endfacet

де кожне n і v - число з плаваючою точкою в форматі: знак, мантиса, «e», знак, експонента. Файл завершується рядком:

endsolid name

Оскільки текстовий файл ASCII STL може бути дуже великим, існує двоїчна версія цього формату. Файл починається з заголовка довжиною 80 символів (який зазвичай ігнорується, але не повинен починатися з «solid», так як з цієї послідовності починається файл ASCII STL). Після заголовка йде 4-байтне беззнакове ціле число, яке вказує кількість трикутних граней в цьому файлі. Після цього йдуть дані, які характеризують кожен трикутник.

Трикутник описується дванадцятьма 32-бітними числами з плаваючою комою: 3 числа для нормалі і по 3 числа на кожну з трьох вершин для координат X, Y, Z. Після йдуть 2 байти беззнакового «short», який називається «attribute byte count».

Опис бінарного STL файлу:

UINT8[80] – Заголовок

UINT32 – Кількість трикутників

foreach triangle

REAL32[3] – Вектор нормалі

REAL32[3] – Вершина 1

REAL32[3] – Вершина 2

REAL32[3] – Вершина 3

UINT16 – Attribute byte count

end

Реалізований модуль STLImporter підтримує обидва (текстовий і бінарний) представлення STL формату.

Для впровадження стороннього об’єкту у модель даних було реалізовано структуру k-d-дерево (детальний опис роботи структури було наведено вище). На рисунку 14 зображено скріншот додатку, а саме сцени, на якій розташовано імпортовану 3D-модель футбольного м’яча.

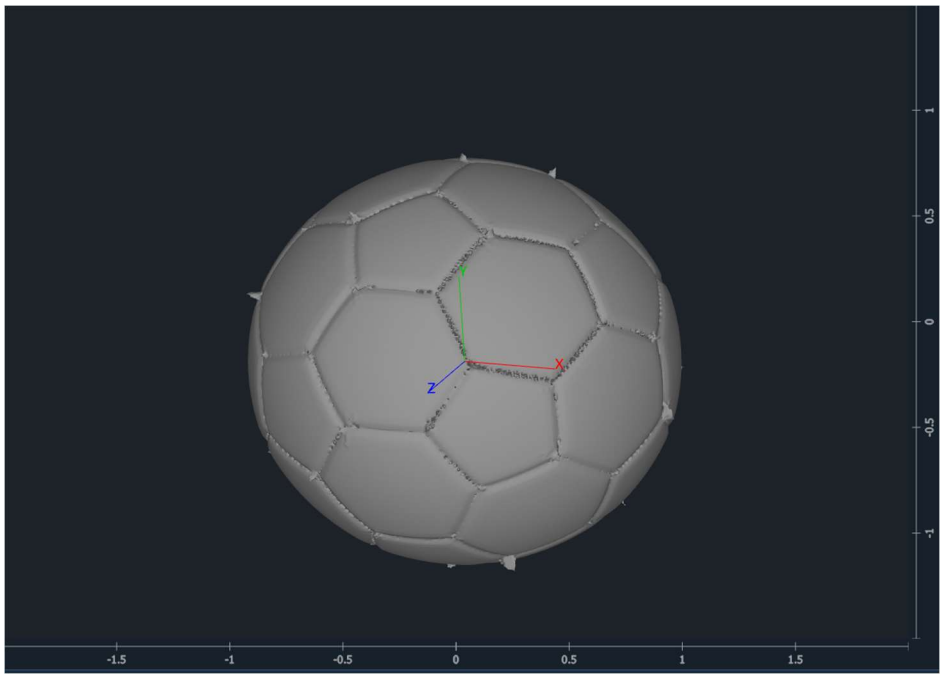

Рис. 14. Скріншот додатку. Приклад роботи з імпортованим об'єктом.

3.2.2 Налаштування якості полігональної сітки

Особливістю алгоритму Marching cubes, що використовується для перетворення функції відстані у полігональну сітку, є його гнучкість. Якість результуючої полігональної сітки можна регулювати шляхом змінення

розміру кубів, на які поділяється простір. Чим менше розмір кубів, (а отже більше кількість цих кубів), тим отримана оболонка ближче до реальної. Але з якістю результату зростає також і час виконання.

Алгоритм і графічний інтерфейс додатку було реалізовано з закладенням можливості налаштування якості результуючої полігональної сітки, аби надати користувачу більшої гнучкості та дати змогу обирати на свій розсуд між часом виконання та якістю (рис. 15, 16).

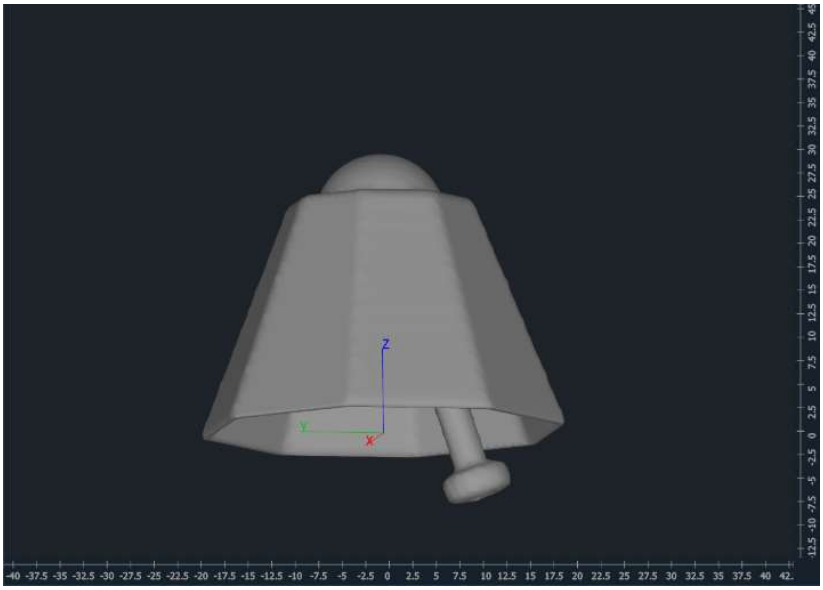

Рис. 15. Скріншот додатку. Приклад низької якості візуалізації моделі.

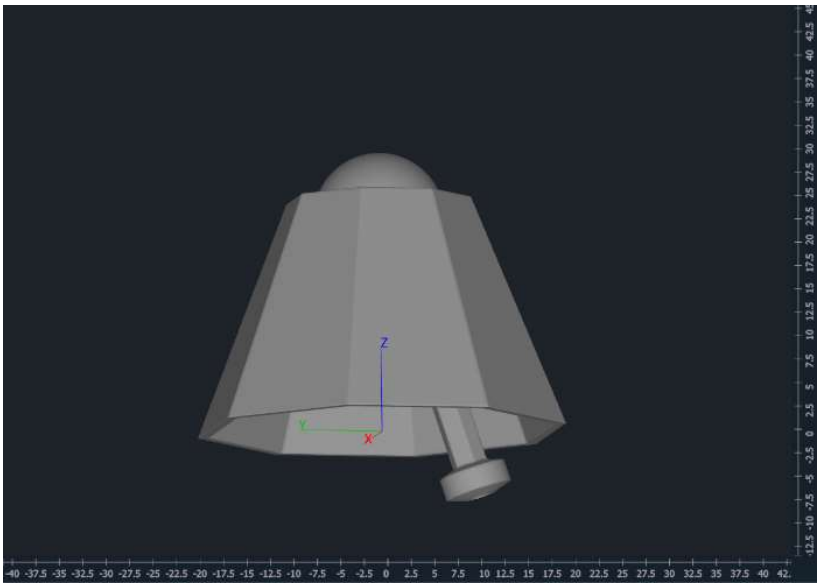

Рис. 16. Скріншот додатку. Приклад високої якості візуалізації моделі.

3.3 Розробка графічного користувацького інтерфейсу

Проектування та розробка графічного інтерфейсу відбувалась з огляду на зручність користування та принципи мінімалізму.

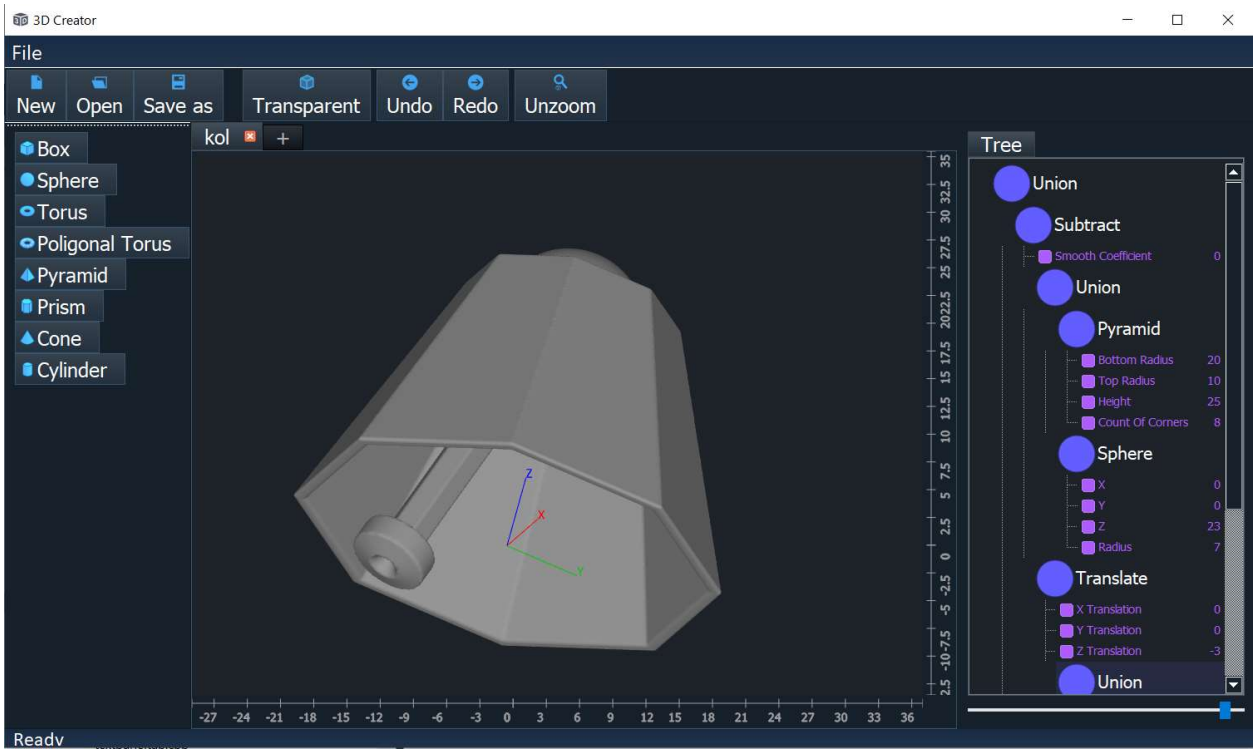

Рис. 17. Скріншот додатку. Приклад графічного користувацького інтерфейсу.

На рисунку 17 зображено головне вікно додатку. Інтерфейс візуально можна розділити на 4 складові:

- верхня – панель інструментів;
- ліворуч – панель швидкого доступу до створення примітивних геометричних фігур;
- по центру – 3D-сцена, на якій візуалізовано об’єкти, що моделюються; цей компонент також дозволяє керувати камерою;
- праворуч – панель, яка відображає інформацію про 3D-об’єкт у вигляді дерева перетворень; дозволяє також змінювати об’єкт шляхом виконання відповідних операцій над деревом (є доступним через контекстне меню).

3.4 Висновки щодо ефективності представлення 3D-об’єктів функцією відстані

Базуючись на досвіді розробки додатку, а також його використанні, можна сформувати деякі висновки щодо ефективності та доцільності використання функції відстані.

До переваг підходу можна віднести наступне:

- Надзвичайно проста реалізація – функції відстані обчислюються доволі легко для більшості примітивних об’єктів і для всіх базових операцій – на відміну від випадку зберігання об’єктів у вигляді полігональної сітки.
- Мінімальні втрати точності, якщо в моделі присутні тільки аналітично задані об’єкти. Точність обмежується тільки можливостями рендерингу.
- Теоретично, при відповідному підході до рендерингу, можна отримати, по суті, векторне зображення.
- Більш прозоре розуміння для користувача способу побудови результуючої моделі.
- Можливість налаштування співвідношення між якістю результуючої полігональної сітки та часом її побудови.
- Менший розмір файлу із збереженим об’єктом.
- Можливість генерувати 3D-об’єкти програмними засобами.

Недоліки підходу:

- Зі зростанням складності об’єкту, що моделюється, нелінійно зростає кількість примітивів та операцій, що описують цей об’єкт. У той же час для людини ускладняється сприйняття та розуміння структури об’єкта.

- Щільна залежність між об’єктами змушує завжди тримати в пам’яті як було отримано той чи інший складений об’єкт. Із зростанням кількості примітивних фігур та операцій, це стає нереальним.
- Швидкість перебудови полігональної сітки задовільної якості занадто низька, особливо якщо у моделюванні задіяно імпортований об’єкт.
- Перебудова усієї полігональної сітки часто є не ефективною, оскільки частіше за всього зміна структури дерева впливає не на весь об’єкт, а тільки на деякі його окремі частини.

ВИСНОВКИ

Отже, в роботі було досліджено аналітичні методи представлення 3D-об'єктів та було проведено аналіз їхньої ефективності у задачах 3D-моделювання.

По-перше, було досліджено методи представлення примітивних геометричних 3D-фігур та базових операцій над ними функцією відстані. Було обчислено та описано функції відстані для деяких примітивних геометричних фігур та операцій. Було досліджено та описано метод представлення складних 3D-об’єктів через структуру дерева операцій.

По-друге, було спроектовано та розроблено прикладний програмний додаток для 3D-моделювання на основі функції відстані. Було досліджено та реалізовано деякі алгоритми та структури даних, що використовуються у комп’ютерній графіці. Було розроблено графічний користувацький інтерфейс.

Нарешті, було проведено аналіз ефективності аналітичних методів представлення об’єктів у задачах 3D-моделювання. Було сформовано основні недоліки та переваги використання вищеописаного підходу відносно більш стандартних методів.

ПЕРЕЛІК ДЖЕРЕЛ

1. Скоренов С.Я. Актуальность 3D-моделирования в геодезии / Скоренов С.Я., Репин А.С. – К.: Интерэкспо гео-сибирь. – 2016. – Т. 1. – № 1. – С. 64-67.
2. Level set based shape prior segmentation. IEEE Computer Society Conference on Computer Vision and Pattern Recognition / Chan, T., Zhu, W. – 2005. doi:10.1109/CVPR.2005.212
3. Shape modeling with front propagation: a level set approach. IEEE Transactions on Pattern Analysis and Machine Intelligence / Malladi, R., Sethian, J.A., Vemuri, B.C. – 1995. – 17 (2): С. 158–175. doi:10.1109/34.368173
4. Computer Graphics: Principles and Practice / Foley, James D.; Van, Foley Dan; Dam, Andries Van; Feiner, Steven K.; Hughes, John F.; Angel, Edward; Hughes, J. – Addison-Wesley Professional, 1996. – 1175 С.
5. Документація класу Qt3DRender фреймворку Qt [Електронний ресурс] – Режим доступу до ресурсу: https://doc.qt.io/qt-5/qt3drender-module.html
6. Colin Smith. On Vertex-Vertex Meshes and Their Use in Geometric and Biological Modeling / Colin Smith [Електронний ресурс] – Режим доступу до ресурсу: http://algorithmicbotany.org/papers/smithco.dis2006.pdf
7. Marching cubes: A high resolution 3D surface construction algorithm / Lorensen, William E.; Cline, Harvey E. // ACM SIGGRAPH Computer Graphics, 1987. – volume 21, issue 4 – С. 163–169. doi:10.1145/37402.37422
8. Dual Contouring of Hermite Data / Tao Ju, Frank Losasso, Scott Schaefer, Joe Warren [Електронний ресурс] – Режим доступу до ресурсу: https://www.cse.wustl.edu/~taoju/research/dualContour.pdf
9. An intoductory tutorial on k-d-tree / Andrew W Moore [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20110716085331/http://www.autonlab.org/auto

nweb/14665/version/2/part/5/data/moore-tutorial.pdf?branch=main&language=en

10. Highly Parallel Fast KD-tree Construction for Interactive Ray Tracing of Dynamic Scenes / Shevtsov M., Soupikov A., Kapustin A. // Computer Graphics Forum. – 2007. – volume 26, issue 3 – C. 395-404. doi:10.1111/j.1467-8659.2007.01062.x
11. 3D Systems Inc. Stereolithography Interface Specification / 3D Systems Inc. – Valencia, CA. – 1988.
12. Neil Sclater. Mechanisms and Mechanical Devices Sourcebook, 5th Edition / Neil Sclater. – McGraw Hill Professional, 2011. – C. 489.