**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри
_____Віталій РОМАНКЕВИЧ

“___” червня 2020 р.

# Дипломний проект

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Спеціалізовані комп’ютерні системи»**

зі спеціальності **123 «Комп'ютерна інженерія»**

на тему: Графічний редактор для побудови 3D-моделі Voxel

Виконав: студент IV курсу, групи КВ-62
(шифр групи)

Буковський Олександр Миколайович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доц. каф. СПіСКС к.т.н., доцент Орлова М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали ) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«___» _____20___ р.

**ЗАВДАННЯ**
**на дипломний проєкт студента**

Буковського Олександра Миколайовича

1. Тема проєкту «Графічний редактор для побудови 3D-моделі Voxel»,

керівник проєкту доц. каф. СПіСКС к.т.н., доцент Орлова М.М.,

затверджені наказом по університету від «__» _____ 20__ р. №_____

2. Термін подання студентом проєкту 20.05.2020

3. Вихідні дані до проєкту Назва. Графічний редактор для побудови 3D моделі Voxel.

4. Зміст пояснювальної записки

1. Аналіз існуючих рішень та проблеми побудови 3D моделі.

2. Розробка графічного редактору для побудови 3D моделі voxel.

3. Опис розробленого програмного продукту

4. Побудова моделі та аналіз результатів

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

Алгоритм обробки буферів. Схема структурна.

Алгоритм видалення вузлів. Схема структурна.

Алгоритм побудови дерева. Схема структурна.

Узагальнена схема роботи візуалізатора. Схема структурна.

Презентація.

6. Консультанти розділів проєкту*

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| Нормоконтроль | Ярослав КЛЯТЧЕНКО, к.т.н., доцент | | |
| | | | |

7. Дата видачі завдання 17.09.2019

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1. | Видача завдання на дипломне проектування | 16.10.2019 | |
| 2. | Розробка технічного завдання | 30.10.2019 | |
| 3. | Аналіз існуючих рішень | 13.11.2019 | |
| 4. | Вибір середовища розробки | 04.12.2019 | |
| 5. | Розробка програмного продукту | 18.03.2020 | |
| 6. | Відлагодження програмного продукту | 22.04.2020 | |
| 7. | Підготовка пояснювальної записки | 02.05.2020 | |
| 8. | Оформлення матеріалів проекту | 15.05.2020 | |
| 9. | Попередній огляд матеріалів диплому на кафедрі | 20.05.2020 | |

Студент _____ _Олександр БУКОВСЬКИЙ_
Керівник проєкту _____ _Марія ОРЛОВА___

# АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (54 с., 19 рис., 2 табл., список використаної літератури з 15 найменувань, 3 додатки).

Метою дипломного проєкту є розробка простого графічного редактору з використанням 3D моделі Voxel, який дозволяє дослідити можливості використання даної технології на сучасних персональних комп'ютерах.

Розроблений графічний редактор дозволяє: створювати воксельну модель з існуючої моделі іншого типу; візуалізувати створені воксельні моделі; виконувати перетворення воксельних моделей з використанням буферів глибини. В процесі розробки були використані власні реалізації технології Voxel, розрідженого воксельного дерева та стандарт графічних адаптерів OpenGL 3.3.

В ході розробки:

- реалізовано об'єкт Voxel, що інкапсулює нюанси реалізації технології в умовах відсутності апаратної підтримки;
- реалізовано розріджене воксельне дерево, що використовується для представлення об'єкту типу Voxel;
- реалізовані алгоритми побудови, візуалізації та редагування воксельних об'єктів;
- реалізовано простий графічний редактор з підтримкою розроблених алгоритмів.

Результати дипломного проєкту можуть бути використані для оцінки можливостей воксельної технології та доцільності її використання.

**Ключові слова:**

3D, VOXEL, SVO, ГРАФІЧНИЙ РЕДАКТОР, РОЗРІДЖЕНЕ ВОКСЕЛЬНЕ ДЕРЕВО, БУФЕРИ ГЛИБИНИ, OPENGL.

# SUMMARY

Qualification work includes an explanatory note (54 p., 19 figs., 2 tables., list of references with 15 items, 3 appendices).

The purpose of the thesis project is to develop a simple graphic editor using the 3D model Voxel, to explore the possibility of using technology on modern PCs.

Developed graphic editor allows you to: create a voxel model from an existing model of another type; visualize the created voxel models; perform voxel model transformations using depth buffers. In the development process, we used our own implementations of Voxel technology, sparse voxel octree and the OpenGL 3.3 graphics adapter standard.

During development:

- implemented Voxel object, which encapsulates the nuances of technology implementation in the absence of hardware support.

- sparse voxel octree used to represent a Voxel object is implemented.

- implemented algorithms for building, visualizing and editing voxel objects.

- implemented a simple graphical editor with support for developed algorithms.

The results of the thesis project can be used to assess the capabilities of voxel technology and the feasibility of its use.



KEYWORDS:

3D, VOXEL, SVO, GRAPHIC EDITOR, SPARSE VOXEL OCTREE, DEPTH BUFFERS, OPENGL.

| Поз. | Формат | ПОЗНАЧЕННЯ | НАЙМЕНУВАННЯ | Кількість аркушів | № прим. | Примітки |
|---|---|---|---|---|---|---|
| | А4 | ІАЛЦ.045480.002 ТЗ | Графічний редактор для | 3 | | |
| | | | побудови 3D моделі Voxel. | | | |
| | | | Технічне завдання | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.003 ТП | Графічний редактор для | 1 | | |
| | | | побудови 3D моделі Voxel. | | | |
| | | | Відомість технічного | | | |
| | | | проекту | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.004 ПЗ | Графічний редактор для | 54 | | |
| | | | побудови 3D моделі Voxel. | | | |
| | | | Пояснювальна записка | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.005 Д1 | Алгоритм обробки буферів. | 1 | | |
| | | | Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.006 Д1 | Алгоритм видалення | 1 | | |
| | | | вузлів. Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.007 Д1 | Алгоритм побудови | 1 | | |
| | | | Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.008 Д1 | Узагальнена схема роботи | 1 | | |
| | | | візуалізатора. Схема | | | |
| | | | структурна | | | |

| Змін. | Арк. | № докум. | Підпис | Дата | ІАЛЦ.045480.001 ОА | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Буковський О.М. | | | *Графічний редактор для побудови 3D-моделі Voxel* | Літ. | Аркуш 1 / Аркушів 2 |
| Перевірив | | Орлова М.М. | | | | | |
| Консульт. | | | | | | КПІ | |
| Н. контроль | | Клятченко Я.М. | | | ***Опис альбому*** | ім. Ігоря Сікорського, | |
| Зав. каф. | | Романкевич В.О. | | | | | |

| Поз. | Формат | ПОЗНАЧЕННЯ | НАЙМЕНУВАННЯ | Кількість аркушів | № прим. | Примітки |
|---|---|---|---|---|---|---|
| | | Диск CD-ROM | Текст пояснювальної | 1 | | |
| | | | записки. | | | |
| | | | Графічний матеріал | | | |

| Змін. | Арк. | № докум. | Підпис | Дата | ІАЛЦ.045480.001 ОА | Арк. 2 |
|---|---|---|---|---|---|---|

**ЗМІСТ**



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.045480.002 ТЗ | | |
| Змін. | Арк. | № докум. | Підпис | Дата | | | |
| Розробив | | Буковський О.М. | | | *Графічний редактор для побудови 3D-моделі Voxel* | Літ. | Аркуш 1 / Аркушів 3 |
| Перевірив | | Орлова М.М. | | | | | |
| Н. контроль | | Клятченко Я.М. | | | ***Технічне завдання*** | КПІ ім. Ігоря Сікорського, ФПМ КВ-62 | |
| Зав. каф. | | Романкевич В.О. | | | | | |

## 1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Графічний редактор для побудови 3D моделі Voxel»».

Галузь застосування: комп’ютерна графіка, автономне проектування, медицина.

## 2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

## 3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення програмного продукту здатного до створення воксельних моделей, їх редагування та візуалізації.

## 4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації в періодичних виданнях та електронні статті у мережі Інтернет.

## 5. ТЕХНІЧНІ ВИМОГИ

### 1.1 Вимоги до програмного продукту, що розробляється

- можливість створення воксельних моделей;
- можливість візуалізації створених моделей;
- можливість редагування створених моделей.

### 5.2 Вимоги до апаратного забезпечення

- оперативна пам’ять: 2 Гб;

- наявність графічного прискорювача з підтримкою стандарту OpenGl 3.3 або сучасніше.

### 5.3 Вимоги до програмного та апаратного забезпечення користувача

- операційна система Linux;
- інстальовані бібліотеки qt, mesa, glfw, glew, glm;
- наявність компілятора мови c++.

## 6. ЕТАПИ РОЗРОБКИ

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту |
|---|---|---|
| 1. | Видача завдання на дипломне проектування | 16.10.2019 |
| 2. | Розробка технічного завдання | 30.10.2019 |
| 3. | Аналіз існуючих рішень | 13.11.2019 |
| 4. | Вибір середовища розробки | 04.12.2019 |
| 5. | Розробка програмного продукту | 18.03.2020 |
| 6. | Відлагодження програмного продукту | 22.04.2020 |
| 7. | Підготовка пояснювальної записки | 02.05.2020 |
| 8. | Оформлення матеріалів проекту | 15.05.2020 |
| 9. | Попередній огляд матеріалів диплому на кафедрі | 20.05.2020 |

| Поз. | Формат | ПОЗНАЧЕННЯ | НАЙМЕНУВАННЯ | Кількість аркушів | № прим. | Примітки |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | Документація загальна | | | |
| | | | Новорозроблена | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.004 ПЗ | Графічний редактор для | 54 | | |
| | | | побудови 3D моделі Voxel. | | | |
| | | | Пояснювальна записка | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.005 Д1 | Алгоритм обробки буферів. | 1 | | |
| | | | Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.006 Д1 | Алгоритм видалення | 1 | | |
| | | | вузлів. Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.007 Д1 | Алгоритм побудови | 1 | | |
| | | | Схема структурна | | | |
| | | | | | | |
| | А4 | ІАЛЦ.045480.008 Д1 | Узагальнена схема роботи | 1 | | |
| | | | візуалізатора. Схема | | | |
| | | | структурна | | | |
| | | | | | | |
| | | Диск CD-ROM | Текст пояснювальної | 1 | | |
| | | | записки. | | | |
| | | | Графічний матеріал | | | |
| | | | | | | |

| Змін. | Арк. | № докум. | Підпис | Дата | ІАЛЦ.045480.003 ТП | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Буковський О.М. | | | *Графічний редактор для побудови 3D-моделі Voxel* | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Орлова М.М. | | | | | |
| Консульт. | | | | | | КПІ | |
| Н. контроль | | Клятченко Я.М. | | | ***Відомість проєкту*** | ім. Ігоря Сікорського, | |
| Зав. каф. | | Романкевич В.О. | | | | | |

# Пояснювальна записка
## до дипломного проекту

на тему: Графічний редактор для побудови 3D-моделі Voxel

Київ – 2020 року

ЗМІСТ

**Додаток 1. Копії графічних матеріалів**

- ІАЛЦ.045480.005 Д1. Алгоритм обробки буферів. Схема структурна.
- ІАЛЦ.045480.006 Д1. Алгоритм видалення вузлів. Схема структурна.
- ІАЛЦ.045480.007 Д1. Алгоритм побудови дерева. Схема структурна.
- ІАЛЦ.045480.008 Д1. Узагальнена схема роботи візуалізатора. Схема структурна.

**Додаток 2. Лістинг програмного коду**

**Додаток 3. Презентація**

# ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ГП – Графічний процесор

ГР – Графічний редактор

КГ – Комп’ютерна графіка

ОСГ – Об’ємна семпльована (англ. Sample) геометрія

ПЗ – Програмне забезпечення

ПК – Персональний комп’ютер

САПР – Систе́ма автоматизо́ваного проєктува́ння

3D(англ. 3-dimensional) – тривимірна графіка

Ball pivoting – алгоритм КГ, для побудови трикутної сітки інтерполяцією заданої хмари точок.

Digital sculpting - вид образотворчого мистецтва, твори якого мають об'ємну форму і створюються за допомогою спеціального комп'ютерного програмного забезпечення, яке імітує інструменти та поводження.

Level of detail - прийом в програмуванні 3D графіки, що полягає в створенні декількох варіантів одного об'єкта з різними ступенями деталізації, які перемикаються в залежності від відстані між об’єктом та віртуальною камерою.

Marching cubes – алгоритм КГ, для виділення полігональної сітки ізоповерхні з тривимірного скалярного поля.

NURBS (англ. Non-uniform rational B-spline) - математична форма, що застосовується в КГ для генерування та подання кривих та поверхонь. Як видно з назви, є частковим випадком B-сплайнів, причому, дуже поширеним через свою стандартизованість та відносну простоту.

Point cloud — набір даних про точки в деякій системі координат.

Polygonal modeling – модель побудови тривимірної графіки за допомогою багатокутників.

Ray casting – один з методів рендеринга в комп'ютерній графіці, при якому сцена будується на основі вимірів перетину променів з поверхнею, що візуалізується

Ray tracing – один з методів геометричної оптики - дослідження оптичних систем шляхом відстеження взаємодії окремих променів з поверхнями.

Sparse voxel octree – програмна технологія, що дозволяє робити ефективну деталізацію об'єктів, що візуалізуються, і ефективну обробку хмар точок.

Spline – функція, область якої розбита на проміжки, на кожному з яких вона співпадає з деяким алгебраїчним многочленом.

Voxel (від англ. Volume та англ. pixel) - елемент простору, позначає значення певної величини в клітинках рівномірної просторової ґратки.

# ВСТУП

Сучасне суспільство характеризується швидкоростучим поширенням інформації та вдосконаленням інформаційних технологій. Інформація стала товаром, котрий можна продавати та купувати. Модернізація існуючих та розробка нових комп'ютерних технологій стала невід'ємною частиною сучасного світу. Ці зміни торкаються всіх сфер життя людини [1].

Описані тенденції визначають напрямки розвитку сучасних технологій. В той же час, попри постійне покращення існуючих технологій та вдосконалення технологічного процесу, більшість сучасних тенденцій не лише тримають свої позиції, а й збільшують свою актуальність. Таким чином, це може свідчити про необхідність пошуку альтернативних технологій для вирішення тих чи інших суспільних потреб.

В наш час, однією з технологій, що бурхливо розвиваються є комп'ютерна графіка (КГ). Її популярність обумовлена діапазоном областей застосування: реклама, мистецтво, наука, медицина, бізнес, розваги, навіть спілкування та багато інших.

В даній роботі під терміном КГ розглядається область діяльності, в якій комп'ютери використовуються як інструмент для створення і редагування зображень, а також їх візуалізації. КГ є одним з наймолодших напрямків комп'ютерних технологій, вона існує близько сорока років [2].

Не дивлячись на відносно молодий вік КГ можна розділити на наступні категорії: векторна графіка, растрова графіка, тривимірна графіка (3D), фрактальна графіка. Векторні зображення представляють собою набір простих геометричних фігур (точки, прямі, кола та прямокутники), їм присвоюється колір, товщина ліній та інші характеристики. Растрова графіка оперує двовимірними масивами пікселів, кожному з яких належить певне значення кольору, прозорості і яскравості. 3D широко застосовується в кіно і комп'ютерних іграх, але область її застосування значно ширша. Фрактальна

графіка являє собою об’єкти, що складаються з елементів однієї спорідненої структури. Деталі будуються за алгоритмом, який можна описати кількома математичними рівняннями.

Окремої уваги заслуговує такий розділ КГ як 3D, яка представляє собою інструмент, що дозволяє створювати візуалізації, що найбільш наближені до реального життя.

Першим повноцінним прикладом 3D була модель обертання супутника, продемонстрована в 1963 році [3]. З того часу КГ стрімко розвивалась, активно розроблювались алгоритми побудови, обробки та збереження зображень та нарощувались обчислювальні потужності комп’ютерів. Проте, незважаючи на величезну різноманітність запропонованих рішень, що були розроблені на початку розвитку КГ, в наш час активно розвиваються лише деякі з них. Така ситуація обумовлена з одного боку як історичними обмеженнями на обчислювальні потужності перших графічних процесорів (ГП), так і комерційною незацікавленістю виробників цих самих процесорів.

Попри вказані недоліки, довгий час дана стратегія давала задовільний результат, відповідаючи постійно зростаючим вимогам суспільства, нарощуючи обчислювальні потужності у відповідності до закону Мура [4], але суттєво не змінюючи технології побудови 3D зображень. Як наслідок такої стратегії, в наш час комп’ютерна графіка стикнулася з серйозною проблемою, коли подальше збільшення обчислювальної потужності апаратних компонентів не надає пропорційного покращення візуальної якості зображення або і взагалі не здатне його покращити. Звісно кожен із існуючих методів побудови зображень не позбавлений своїх недоліків, проте на фоні досягнення порогу якості зображень, що здатна надати полігональна модель (англ. polygonal modeling), збільшують свою актуальність інші проблеми даної моделі.

Однією з існуючих замін для полігональної моделі є використання сплайнів (англ. spline). Сплайни частково вирішують таку проблему побудови 3D, як ступінчастість при приближені до об’єктів. Недоліком сплайнів є

непомірне збільшення вимог до обчислювальних потужностей. Таким чином, не дивлячись на явні переваги використання сплайнів, вони не можуть бути заміною полігональній моделі, через неспроможність побудови 3D зображень у реальному часі.

Іншим недоліком як полігональної, так і сплайнової моделей побудови зображення є пустотілість побудованих моделей, тобто вони являють собою лише контури об’єкту і позбавлені цілого ряду фізичних характеристик, що у свою чергу ускладнює або навіть унеможливлює використання їх при моделюванні реалістичної поведінки в 3D сценах реального часу. Найголовнішим недоліком для побудови таких сцен є неможливість зміни редагування самих об’єктів в реальному часі, тому сцени фактично будуються заміною одних об’єктів іншими, або елементарними їх перетвореннями у просторі на кшталт повороту чи зміни лінійних розмірів. Для рішення цих проблем може бути використана так звана об’ємна семпльована геометрія (ОСГ) [5].

Прикладами ОСГ є вокселі (англ. Voxel) та хмари точок (англ. point cloud). Об’ємна семпльована геометрія є графічною моделлю, найбільш наближеною до реального світу, кожній точці простору в цій моделі відповідає елементарний об’єкт – аналог атому чи молекули. Таким чином, в будь який момент часу 3D сцена може бути представлена скінченим набором елементарних об’єктів, які в свою чергу можуть бути частинами тих чи інших об’єктів. Такий підхід максимально спрощує редагування об’єктів в реальному часі, адже геометричне положення елементарних об’єктів завідомо відоме і достатньо лише змінити необхідні характеристики.

Головним, і можливо єдиним, недоліком використання ОСГ, є відносно велике використання пам’яті ГП. Цей фактор став вирішальним при виборі розвитку КГ на початку її розвитку. Як наслідок ні один з сучасних виробників ГП не підтримує дану технологію на апаратному рівні, проте вона впроваджується все частіше у сучасні ГР, САПР та інше.

На сьогоднішній день існують деякі професійні комерційні ГР, що цілком побудовані на даній технології [6], а також численні гібридні рішення з її використанням для виправлення недоліків основної графічної моделі.

Мета даної роботи – це побудова нескладного графічного редактора, що базується на воксельній технології, абстрагуючись від деталей реалізації сучасних графічних процесорів, а також аналіз його можливостей для побудови 3D сцен з використанням простих фігур, що можуть бути описані математичними формулами.

# 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПРОБЛЕМИ ПОБУДОВИ 3D МОДЕЛІ

Проаналізуємо основні існуючі методи побудови 3D моделі об’єкту та основні проблеми, які виникають при цьому.

## 1.1.Аналіз існуючих методів побудови 3D моделі.

Основними актуальними методами побудови є:

- полігони;
- сплайни;
- фрактали;
- хмари точок;
- вокселі.

Полігональні моделі (полігони), або полігональні сітки, знаходять в КГ найширше застосування. Вони представляють поверхні геометричних об'єктів у вигляді набору зістикованих один з одним плоских полігонів. Традиційний для КГ опис полігональної моделі об'єкта є ієрархічним і включає список вершин, список ребер і список полігонів об'єкта.

Основний недолік полігональних моделей - необхідність створення великої кількості полігонів для представлення складних, особливо криволінійних, поверхонь. Реалістичне відображення просторової сцени може вимагати числа полігонів, що доходить до мільйона і більше. Це означає, що при синтезі динамічних зображень геометричні параметри потрібно перераховувати з великого числа примітивів в режимі реального часу [7].

Загальний вигляд зображення, побудованого з використанням описаної моделі, представлено на рис. 1.1. Незважаючи на той факт, що кількість полігонів в сучасних 3D моделях може вимірюватись в мільйонах, ця модель все ще

залишається практично монополістом в цій сфері КГ завдяки простоті побудови користувачем та ГП, що фактично проектуються для її використання.

Іншим прикладом побудови поверхонь геометричних об’єктів є використання сплайнів. Вони є більш сучасним підходом, що обумовлений ростом обчислювальних потужностей цифрової техніки.

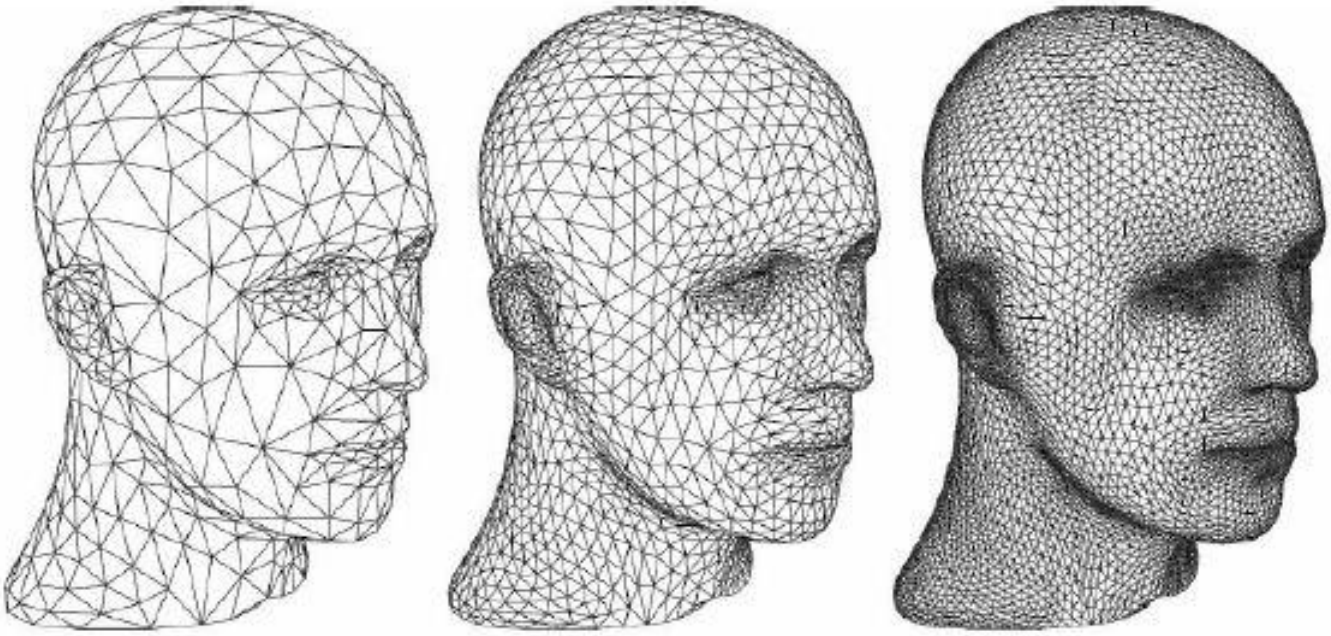

Рисунок 1.1 - Полігональна модель обличчя людини

Сплайни являють собою криві, що описуються функціями, заданими на проміжках та описаними деяким поліномом на кожному з них. Поверхні при побудові за допомогою сплайнів фактично представляють з себе величезну кількість кривих, що утворюють візуально цілісне зображення. Головним недоліком сплайнів являється значна обчислювальна складність, тому їх використання для 3D моделювання в реальному часі не представляється можливим, хоча вони активно використовуються при монтуванні фільмів та технічному проєктуванні, а також при цифровому скульптингу (англ. Digital sculpting). Приклад зображення побудованого за допомогою цієї технології зображено на рис.1.2.

Описані вище способи  побудови 3D моделей є побудовами поверхонь. Вони дають гарний результат при візуалізації моделей, але фактично об’єкти

позбавлені цілісності. Такі об’єкти збираються з елементарних одиниць і насправді представляють собою колекції кривих чи полігонів. Як наслідок, операції зі зміни одного об’єкту перетворюється в велику кількість операцій зміни групи елементів, що значно збільшує обчислювальну складність і ставить під питання можливість використання в реальному часі.

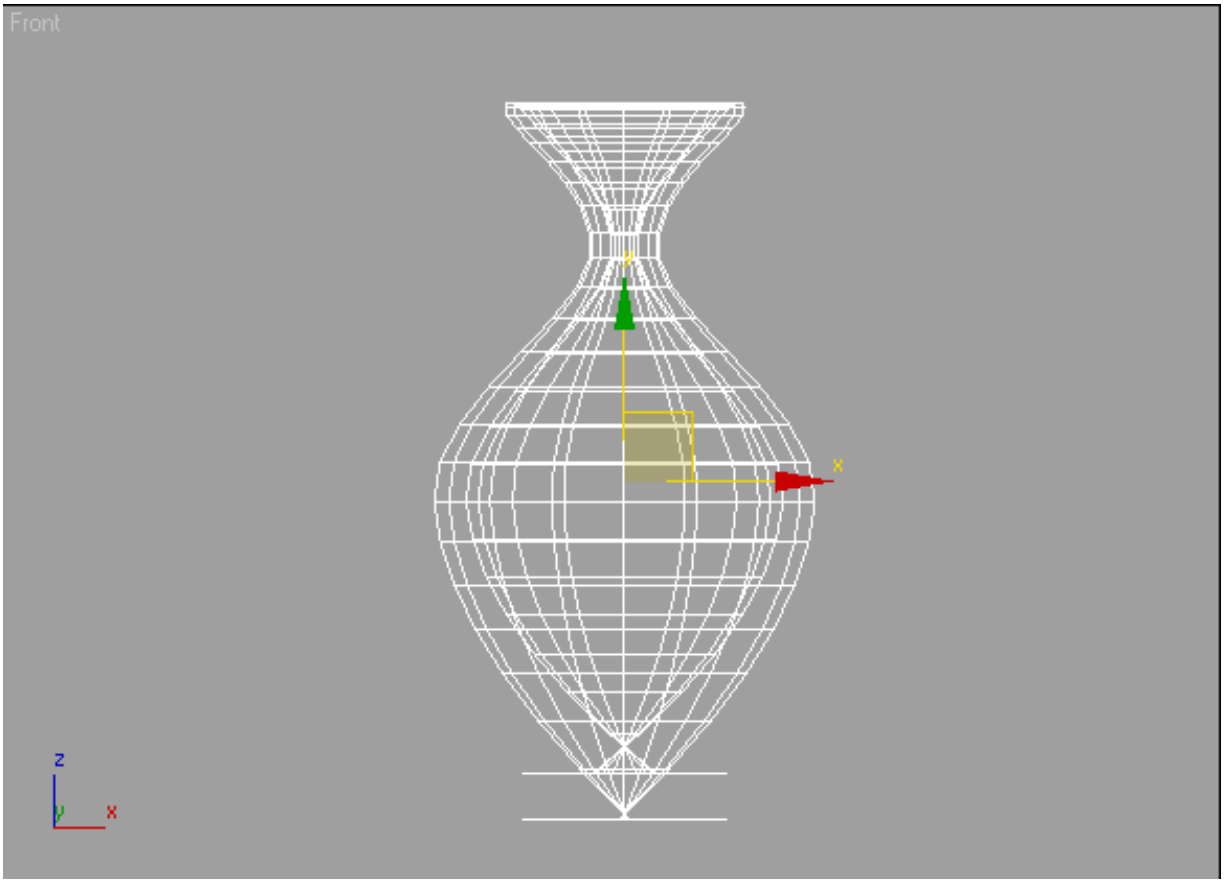


Рисунок 1.2 – Приклад побудови за допомогою сплайнів

Часто відображаються об'єкти, особливо природні, що мають досить складну форму, яка не допускає універсального аналітичного опису в цілому. Їх форма задається набором характерних (опорних) точок, що належать поверхні об'єкту. В якості прикладу можна навести складену геодезистами карту висот ділянки земної поверхні. В процесі геометричного моделювання вихідна поверхня повинна бути відновлена з заданою точністю і повинна проходити якомога ближче до опорних точок, а краще - через них.

Фрактальні ландшафти можуть створюватися з фракталів методом випадкового зміщення середньої точки. Приклад побудови представлено на рис. 1.3. Середні точки сторін трикутника (a) зміщуються вгору або вниз від площини зображення і з'єднуються з вершинами (b). При цьому виникають чотири менших трикутника, до яких повторно застосовується та ж процедура. Функція розподілу ймовірності визначає величину зсуву і, отже, ступінь гладкості фрактального ландшафту. Потім графічна програма комп'ютера зафарбовує трикутники, створюючи різні відтінки (d). В результаті виходить дуже реалістична картина (e).

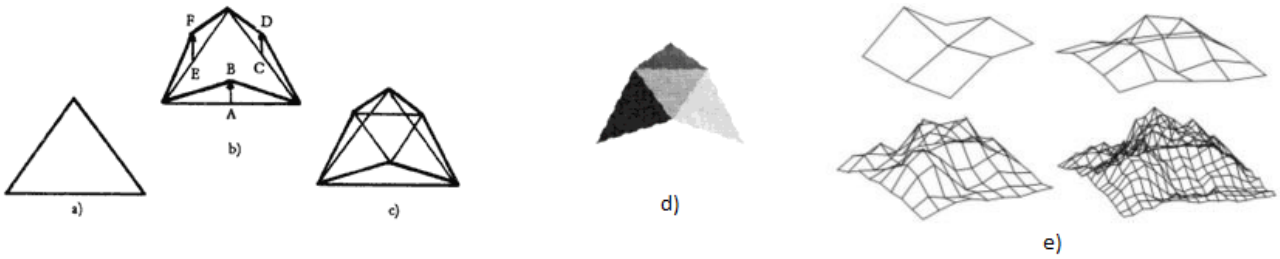


Рисунок 1.3 – Побудова фрактального рельєфу.

Таким чином, фрактали являють собою досить простий спосіб генерації різноманітних складних структур, що характеризується ще і масштабованістю. В той же час, фрактальна модель є ситуативною. Дійсно, в природі часто зустрічаються об’єкти, що можуть бути описані фракталами, наприклад, гори, річки, морські узбережжя, рослини тощо. Проте для більшості природніх об’єктів фрактальна побудова неможлива, тому дана технологія розглядається виключно як доповнення до інших.

Хмари точок - відносно молода технологія, що породжена розвитком цифрової техніки. Вони являють собою результати роботи 3D-сканерів та використовуються з різною метою, зокрема для створення 3D моделей для виробництва деталей, контролю якості та для використання у чисельних застосунках візуалізації, анімації, рендерінгу. Хоча хмари точок можуть бути

безпосередньо відрендерені та перевірені, їх зазвичай не використовують для побудови 3D моделей. Тому вони, як правило, перетворюються в моделі з полігональною, у NURBS (Non-uniform rational B-spline) модель або САПР модель за допомогою процесу так званої реконструкції поверхні. Існує багато методів перетворення хмари точок у 3D поверхню. Зокрема, у статті М. Берже і групи авторів наведена порівняльна таблиця з 35 методів відтворення поверхні з хмари точок [8]. Деякі наближення, такі як тріангуляція Делоне, Альфа-форма та метод поворотних куль (англ. *ball pivoting*) [9], будують трикутну або полігональну сітку по вже наявним вершинам хмари точок, а інші наближення будують об'ємні таблиці відстаней або реконструюють неявну поверхню за допомогою алгоритму крокуючих кубиків (англ. *marching cubes*). Приклад побудови за допомогою описаної моделі зображено на рис.1.4.

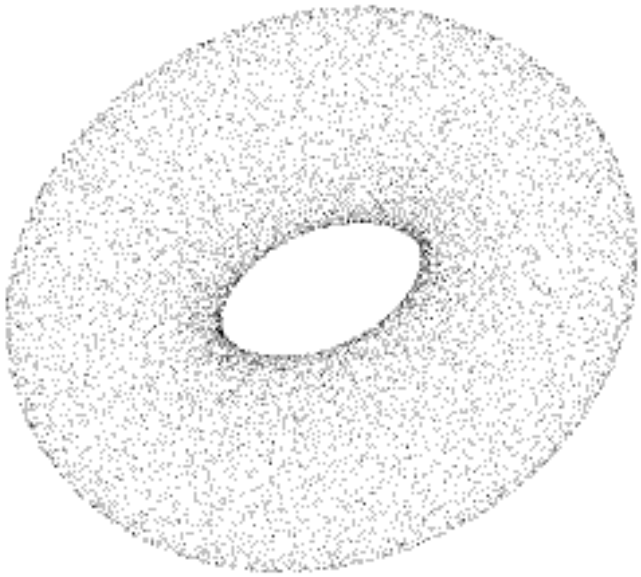

Рисунок 1.4 – Хмара точок моделі тора

Обмежена ділянка простору, що охоплює весь модельований об'єкт, вважається розбитим на велику кількість кубічних комірок (вокселей). У

найпростішому випадку ребра куба дорівнюють умовній одиниці вимірювання довжини. Структура даних представляється тривимірної матрицею, в якій кожен елемент відповідає просторової осередку. З одного боку, осередковий метод має суттєві переваги, в першу чергу простоту, з іншого - недоліки, пов'язані з великим обсягом пам'яті, необхідної для запису об'єкта з високою роздільною здатністю. Так, для збереження моделі, що має роздільну здатність 1024*1024*1024 вокселів, при кодуванні кожного 3 байтами для RGB палітри та ще одним байтом для прозорості, буде потрібно 4 Гб пам’яті, що в свою чергу в десятки чи навіть сотні разів більше ніж при використанні інших методів побудови 3D моделі.

Не дивлячись на такі оцінки по обсягу необхідної для побудови моделей пам’яті, реальне використання технології можливе. І в якості прикладу цього може служити комп’ютерна гра Outcast, що була випущена в 1999 році [10]. Дійсно, в цій грі використовувалась дуже спрощена воксельна модель карти висот для побудови ландшафту, а моделі персонажів були полігональні. Іншим прикладом може слугувати демонстрація Джона Оліка з конференції SIGGRAPH 2008 [11]. На ній була показана повноцінна 3D воксельна модель, що налічувала 35,184,372,088,832 вокселів, і все це на ГП NVIDIA GeForce GTX280, що мав 1Гб відео пам’яті. Приклад простої воксельної моделі наведено на рис.1.5.

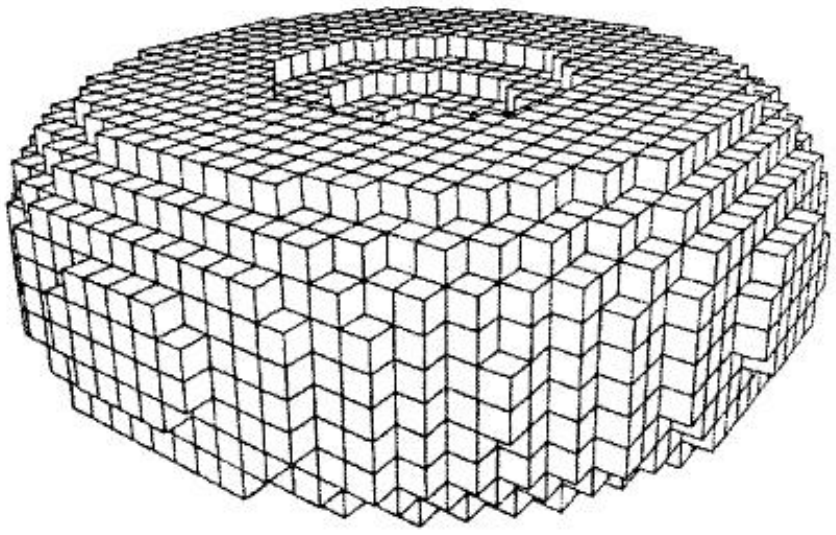

Рисунок 1.5 – Воксельна модель тору

## 1.2.Аналіз методів побудови 3D моделі Voxel

Серед всіх описаних раніше моделей – вокселі є моделлю, найбільш наближеною до реальної. Простір представляється у вигляді сітки з кубів одиничного розміру. Дійсно, при кодуванні кожної комірки простору потрібні величезні обсяги пам’яті, але об’єкти рідко складаються виключно з незв’язаних вокселів, крім того немає сенсу зберігати дані про вільний простір. Ці принципи закладені в спеціальні структури для збереження та побудови воксельних моделей.

Однією з новітніх перспективних технологій, що дозволяє робити ефективну деталізацію воксельних об'єктів, є розріджене воксельне октодерево (англ. sparse voxel octree). Взагалі дерева, як структура даних, дуже часто використовуються в програмуванні, оскільки вони дозволяєть ієрархічно організовувати дані. Якщо взяти структуру дерева файлової системи, то можна виділити корінь на жорсткому диску, в якому буде присутні декілька дочірніх гілок у вигляді папок, які в свою чергу можуть утримувати свої дочірні гілки (вкладені папки) і так далі, поки не дійде черга до листя (тобто самих файлів). Цей приклад показує одну з переваг використання дерев: дістатися до частини організованого дерева буває набагато швидше, ніж якщо гілки розташовані довільно.

Вузли дерева можуть мати різну кількість гілок. Наприклад, якщо гілок у вузла максимум дві, то говорять про бінарне дерево, а кількість гілок від нуля до чотирьох породжує квадродерево (дерево квадрантів), нарешті, дерево з числом гілок від нуля до восьми є октодеревом.

Октодерева дозволяють використовувати пам'ять більш ефективно, оскільки вони переходять до більш детального розширення тільки там, де це необхідно. Для кращого розуміння процесу побудови октодерева в просторі, розглянемо на прикладі квадродерева на площині.

При стандартному підході до побудови ( рис.1.6(a)), площина розбивається на однакові комірки, далі визначається які з них належать фігурі і відповідно відбувається їх заповнення. При такому підході в пам’яті має зберігатися інформація про всі можливі комірки та їх стану. Побудова квадродерева зображена на рис 1.6(b). Для побудови дерева потрібно почати з базового зображення, після чого поділити його на два в обох напрямках, що дасть чотири квадранта. Якщо квадрант порожній або повністю заповнений, то алгоритм для нього припиняється. Якщо квадрант заповнений частково, то він поділяється на два в обох напрямках і так далі. Алгоритм завершується, коли всі квадранти однорідні (тобто повністю заповнені або повністю порожні), або коли досягнута певна глибина (на прикладі показано, як на дереві з глибиною чотири, що дало поділ на 16 в кожному напрямку). Як можна бачити, навіть в такому простому прикладі кінцевий результат виявився набагато ближче до оригінального кола, при цьому використано менше даних (97 вузлів або осередків проти 122 у звичайній координатній сітці). А октодерево переносить цей принцип з площини в простір.

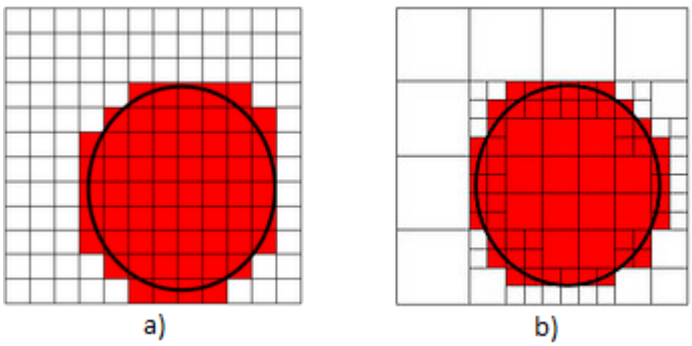


Рисунок 1.6 – Варіанти заповнення

На практиці виграш за використаним об’ємом виявляється дещо меншим, ніж може здатись на перший погляд. Це пов'язано з тим, що в разі звичайної координатної сітки розташування вокселей визначено з самого початку. Навпаки, в разі октодерева кожен вузол повинен містити прив'язку до всіх

дочірніх об'єктів. На практиці це відображається в необхідності зберігати в кожному вузлі вісім вказівників на дочірні елементи в додаток до кольору і нормалі вокселя.

Варто відмітити переваги та недоліки використання октодерев при побудові воксельних моделей. Першою перевагою є прокладка променів (англ. ray casting). Подібне трасування променів (англ. ray tracing), прокладка променів ґрунтується на проведенні променя від кожного пікселя картинки. Але відмінність в наступному, що як тільки буде знайдено перетин, то алгоритм на цьому завершується, вторинних променів не прокладається.

Отже, прокладка променів виконується швидше трасування променів. Більш того, зазвичай не потрібно створювати додаткові структури даних, щоб прискорити обчислення, пов'язані з перетинами. Октодерево є одночасно і даними (геометрія і текстури), і структурою прискорення.

Розглянемо тепер переваги воксельного октодерева при рендерінгу. Основна перевага цієї структури даних полягає в тому, що вона забезпечує досить простий та цікавий шлях вирішення проблеми рівня деталізації (англ. level of detail) текстур, а також і геометрії - все це за одним простим алгоритмом. Причина криється в тому, що, як згадано раніше, кожен вузол октодерева містить кольорову інформацію, тому можна обійтися без 2D-текстур в їх традиційному розумінні. Або, якщо бути більш точним, октодерево містить як текстурну інформацію, так і геометричну.

Отже, проблема рівня деталізації, яку доводилося зовсім по-різному вирішувати для геометрії і для текстур, тепер зводиться до однієї простої системи: вирішення проблеми рівня деталізації для октодерева. А це можна зробити дуже просто, за принципом, який подібний текстуруванню mip-рівня. Мета створення mip-рівнів полягає в збереженні розміру текселей (елементи текстури) максимально близько до розміру пікселів екрану. Для цього текстура прораховується і зберігається в декількох дозволах, а графічний процесор вибирає той чи інший mip-рівень в залежності від розміру текстури на екрані.

Щось подібне можна зробити і з воксельного октодерева, коли рівень деталізації буде вибиратися автоматично. Якщо розмір вокселя менше розміру пікселя, то проходження дерева на цьому закінчується. Все, що потрібно - зберегти середнє від інформації, що міститься в дочірніх гілках кожного вузла, і таким чином отримується дуже простий спосіб вирішення проблеми рівня деталізації. Цей механізм ідеально підходить для потокових систем візуалізації. У відеопам'яті зберігаються тільки ті частини октодерева, які потрібні, деякі частини розташовуються в основній оперативній пам'яті, щоб прискорити подальший доступ. Але більша частина октодерева просто знаходиться у сховищі.

В результаті можна отримати майже безмежну кількість геометрії (і текстур, як відзначено). Октодерево може бути настільки деталізовано, наскільки потрібно в кожному конкретному випадку, і під час роботи обсяг пам'яті і час виведення на дисплей залишаються (відносно) постійними. Єдиними обмеженнями є кількість часу, яке художники можуть витратити на створення октодерев, і фізичні обмеження накопичувачів в комп'ютерах наступного покоління.

Серед недоліків використання цієї моделі варто відмітити спільну проблему всіх потокових систем візуалізації. Як вже показано раніше, дана технологія рендерінгу добре підходить для поступового збільшення рівня деталізації по мірі наближення до об'єкта, коли система буде на фоні завантажувати дані з октодерева. Через спосіб організації даних це можна зробити плавно. Але що станеться, коли раптово необхідно переміститися в іншу частину сцени? Геометрія при цьому буде унікальна, і потокова система буде поставлена занадто завантажена. Тому можна очікувати виведення деякого часу дуже простий геометрії, з поступовим поліпшенням деталізації.

Ця проблема не є унікальною для воксельних октодерев, оскільки всі потокові системи будуються на принципі щодо безперервної зміни рівня деталізації, і тому часто вони не справляються з раптовими змінами. Але на

сьогоднішній день така проблема стоїть тільки з текстурами. І якщо бачити текстуру низького розширення секунду або дві не дуже приємно, то октодерево з низьким розширенням буде візуально просто жахливо.

### 1.3.Обґрунтування теми дипломної роботи

Як було вказано у вступі, сучасні моделі побудови 3D зображень вже досягнули свого порогу якості зображень, при цьому так і не вирішивши цілий ряд проблем, що носять фундаментальний характер для комп’ютерної графіки. В той же час сучасні графічні процесори та носії інформації відкривають нові можливості для використання альтернативних методів побудови 3D зображень.

Необхідність пошуку альтернатив обумовлена постійно зростаючим попитом суспільства на інтерактивні розваги. В наш час навіть просте спілкування перетворюється в синтез віртуального оточення, а кіно знімають без акторів.

Таким чином, було прийнято рішення написання простого графічного редактора з використанням альтернативного підходу до побудови 3D зображень. Вибір пав на воксельну технологію, як найбільш перспективну в плані вирішення проблем редагування об’єктів, прокладання променів та моделювання суцільних тіл.

Дослідження альтернативних методів побудови зображення та можливості їх використання може стати відправним пунктом в подальшому розвитку КГ. Важливість даного дослідження обумовлена нинішньою неспроможністю комп’ютерної графіки та характеристиками графічних процесорів, адже виробники дуже довгий час не вносили суттєвих апаратних змін у нові покоління своєї продукції в наслідок відсутності розвитку актуальної технології.

## 1.4. Обґрунтування вибору інструментів розробки

Оскільки розробка графічного редактору для побудови 3D моделі Voxel включає розробку складних структур даних і роботу з пам’яттю, то вибір мови програмування пав на C++.

Для компіляції програмного коду був використаній вільний компілятор g++ в реалізації для linux.

Оскільки розроблена модель воксельного представлення зображень являється насправді лише абстракцією над полігональною обробкою вершин в ГП в наслідок апаратної реалізації останніх, рендерінг здійснювався у відповідності до відкритого стандарту OpenGL 3.3, що підтримується усіма сучасними графічними процесорами.

Для побудови користувацького інтерфейсу з урахуванням використання C++ в якості мови написання ГР, та необхідності підтримки OpenGL контестів, був вибраний вільний фреймворк Qt.

За середовище розробки, з урахування вбудованої підтримки вибраного фреймворку, використання вказаної мови програмування та підключення потрібного компілятора, було вибрано IDE QtCreator.

# 2. РОЗРОБКА ГРАФІЧНОГО РЕДАКТОРУ ДЛЯ ПОБУДОВИ 3D МОДЕЛІ VOXEL

## 2.1 Етапи побудови графічного редактора

На першому кроці розробки графічного редактора варто визначитись, що собою буде представляти розроблювана модель. Як вже було сказано, ні один з сучасних виробників графічних процесорів не реалізує підтримку вокселів апаратно, тому розроблювана модель може бути представлена лише як абстракція над теоретично описаною. Втім дане обмеження не є суттєвою перепоною для дослідження можливостей технології, адже важливим є саме сприйняття вокселів як цілісних об’єктів, а не методи їх відображення.

Отже першим кроком розробки, є реалізація class Voxel, що інкапсулює деталі реалізації та методи відображення вокселя на екрані користувача. Сам воксел, практично, буде представляти з себе набір вершин, що відповідають полігональній моделі куба одиничного розміру, а методи його рендерінгу будуть відповідати рендерінгу полігональної моделі з використанням апаратного прискорення.

Наступним кроком розробки є визначення методів побудови та обробки початкового об’єкту. Побудова об’єкту може з самого початку здійснюватись за допомогою використання вокселів, тобто подібно збиранню об’єкту з кубиків або з використанням будь-якої іншою технології з подальшою вокселізацією моделі для можливості подальшої обробки та візуалізації у графічному редакторі.

В даній розробці вибір був наданий початковій побудові об’єкту з використанням інших методів та подальшою вокселізацією отриманої моделі, як більш складній та гнучкій у порівнянні з базовою. Об’єкти можуть бути описані як за допомогою наборів вершин, що відповідають полігонам, так і аналітично за допомогою математичних функцій, що описують поверхні, якими обмежується

об’єкт. Побудова навіть простої полігональної моделі потребує часу та навичок роботи зі спеціальним програмним забезпеченням (ПЗ), тому в даній роботі було прийняте рішення обмежитись використанням простих об’єктів, що можуть бути описані аналітично. Вказаний підхід не вносить суттєвих коректив у розробку, проте сильно зменшує затрати часу на виконання дій, що не мають безпосереднього відношення до розробки.

Після визначення з методом представлення початкового об’єкту постає задача його вокселізації. Ця проблема носить доволі суттєвий характер, адже на відміну від воксельної моделі, полігональна використовується лише відображення і не несе в собі ніякої інформації про заповнення простору. Отже не існує тривіальних рішень даної задачі.

Основною ідеєю більшості алгоритмів вокселізації, є перевірка приналежності кожного вокселу до того чи іншого об’єкту і присвоєння 1 чи 0 в залежності від результату такої перевірки. Це досягається шляхом дослідження, чи лежить центр вокселя всередині досліджуваного об’єкту або вибором всіх вокселів, які перетинаються об’єктом. Більш складні алгоритми генерації гладкої моделі без контурних нерівностей виконують фільтрацію об’єму [12], ділення початкового об’єму [13], або розрахунок відстаней від вокселя до поверхні об’єкта [14].

В даній роботі для рішення задачі вокселізації використано метод, що базується на створенні об’ємних даних з використанням інформації про глибину різних положень об’єкта і може використовуватись, як застосування апаратно підтримуваного z-буфера [15]. Об’єкт оточений трьома парами ортогональних z-буферів, кожна пара зберігає дані про глибину об’єкта з різних кутів огляду. Вокселізація здійснюється шляхом сканування об’ємного простору і перевірки, чи лежить кожен воксель між границями, заданими буфером.

Після визначення з методом подання початкового об’єкту та методом його вокселізації можна приступати безпосередньо до створення описаної у розділі 1

структури – розрідженого воксельного дерева, що призначена для збереження воксельного представлення об’єкту.

Останнім кроком розробки є обрамлення описаної розробки за допомогою графічного інтерфейсу користувача, що пришвидшує та полегшує роботу з програмою. Вказаний інтерфейс має надавати змогу задавати початкову аналітичну функцію для опису об’єкту, візуалізації моделі, вибір параметрів візуалізації та додаткові можливості, що наглядно демонструють використання описаних раніше підходів.

## 2.2 Опис алгоритму побудови аналітичної моделі та її подальшої вокселізації

Як вже було сказано, в даній роботі використаний метод вокселізації, оснований на інформації про глибину, тобто проекціях об’єкту на ортогональні площини. Першим шагом метода є представлення об’єкта на сцені, як це зазвичай робиться при візуалізації, незалежно від типу об’єкту. Для того, щоб перейти безпосередньо до візуалізації, необхідно створити три пари буферів глибини ($[x_1, x_2]$, $[y_1, y_2]$, $[z_1, z_2]$), направлені вздовж осей X, Y, Z, які визначають глобальну систему координат. Кожна пара з двох буферів, перпендикулярних одній і тій же осі, але знаходиться на протилежних сторонах об’єкту, таким чином, що вони дивляться одне на одного, як показано на рисунку 2.1.

Буфери тієї ж пари зберігають протилежні погляди на об’єкт, що відповідають одній і тій же осі перегляду. Перший буфер кожної пари $x_1$, $y_1$, $z_1$ (R, U, F на рисунку 2.1) зберігає найбільш близьку до користувача глибину, в той час як друга пара буферів $x_2$, $y_2$, $z_2$ (L, D, B на рисунку 2.1) зберігає максимальну відстань від користувача до відповідної осі.

Оскільки дані про глибину об’єкту зазвичай зберігаються в Z-буфері, тобто вздовж осі Z, в нашому випадку необхідний об’єкт, що буде обертатись двічі, для отримання пари X та Y буферів. Не потрібно ніяких інших додаткових

перетворень для отримання другого буфера кожної пари, адже вони можуть бути отримані шляхом модифікації функції порівняння, що використовується в z-буфері, для збереження найбільшої відстані замість найменшої. Контроль за функцією порівняння надається більшістю розповсюджених API, таких як OpenGL.

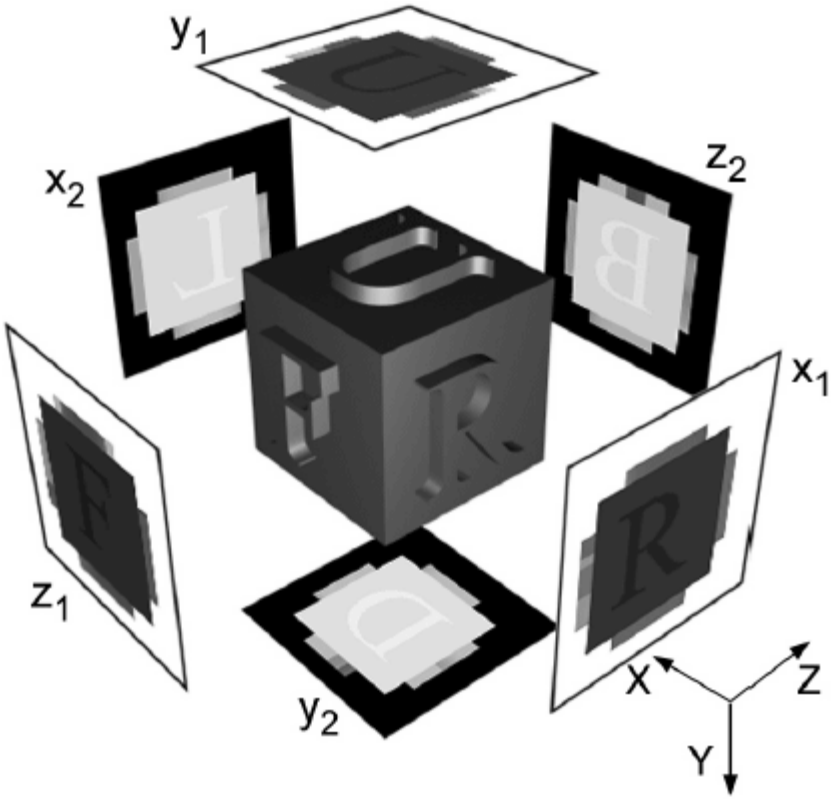


Рисунок 2.1 – Буфери глибини об’єкту

На рисунку 2.1 більше значення (світлий колір) вказує на більш віддалену точку, в той час як менше значення (темніше) вказує на точку, що розташована ближче до користувача. Точки на фоні інтерпретуються як віддалені точки в першій групі буферів і як ближні в другій групі буферів, оскільки критерій порівняння обернений.

Після того як всі буфери створені, можна переходити безпосередньо до кроку вокселізації. Кінцева воксельна модель обмежена розширенням буфера,

тому потрібно вибирати буфер розміру N на N для $N^3$ воксельного кубу. Орієнтація куба визначається трьома глобальними осями X, Y, Z.

Перетворення в об’ємі даних складається з потрійного циклу для покриття воксельного простору. Воксель v(i,j,k) зв’язаний з шістьма проекціями точок на буфери глибини, координати яких на осях являются пары (j,k), (k,i) и (j,i) відповідно. Глибина задана відповідними буферами $x_1(j,k)$, $x_2(j,k)$, $y_1(k,i)$, $y_2(k,i)$, $z_1(j,i)$, $z_2(j,i)$.

Вокселізація, тобто рішення чи належить воксель об’єкту (внутрішньому об’єму чи поверхні) або ні еквівалентно перевірці, чи лежить воксель на межі, визначеної відповідним чином буферів глибини, як це представлено на рисунку 2.2.

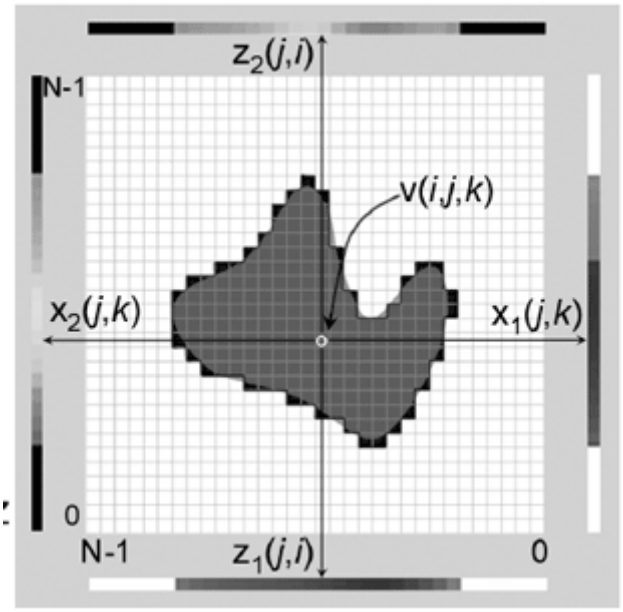


Рисунок 2.2 – Перевірка належності вокселу до об’єкту

Оскільки воксельні координати визначаються відносно початкового розміру воксельного кубу (на інтервалі [0 , N)), а значення z-буфера лежить в діапазоні [$depth_{min}$ , $depth_{max}$] (що залежить від реалізації), для порівняння необхідні деякі перетворення. Позиція v(i,j,k), перетворюється в значення з діапазону z-буфера:

$$(x, y, z) = \frac{(i, j, k) * (depth_{max} - depth_{min})}{(N - 1)}$$

v(i,j,k) знаходиться всередині об'єкта, тільки якщо виконується умова:

$$x_1(j,k) \leq x \leq x_2(j,k)$$

$$AND\ y_1(k,i) \leq y \leq y_2(k,i)$$

$$AND\ z_1(j,i) \leq z \leq z_2(j,i)$$

Описаний алгоритм має дві головних переваги: він дуже простий в реалізації і дуже швидкий. Вокселізація об’єкта в межах розумного просторового розширення може виконуватись в режимі реального часу навіть на комп’ютерах середньої потужності та без використання спеціалізованого обладнання. Також варто відмітити, що практично всі сучасні комп’ютери оснащені відео картами з апаратною підтримкою z-буферів.

Іншою важливою перевагою алгоритму є його гнучкість. Алгоритм може бути використаний для вокселізації будь-якого типу 3D об’єктів (полігональних, аналітичних чи навіть об’ємних) без модифікації та збільшення обчислювальної складності, на відміну від більшості аналогів.

Проте суттєвим недоліком використання даного об’єкту є неможливість обробки об’єктів, що мають скриті порожнини, а також наборів об’єктів, що перекривають одне одного.

В даній роботі було прийнято рішення використання даного методу, незважаючи на неможливість його використання для деяких об’єктів. Дане спрощення вважається можливим з урахуванням обмеженості розробки в часі та ресурсах.

Відповідно до описаного підходу в розробці були реалізовані алгоритми для побудови розрідженого воксельного дерева та подальшої його візуалізації, що ґрунтуються на апаратному буфері глибини. В якості буферів використовувались двомірні масиви чисел з плаваючим знаком, розмірності

котрих збігатись з розмірністю області візуалізації. Схеми алгоритмів наводяться в додатку Д1.

## 2.3 Методи редагування воксельних зображень

Одним з головних рушіїв, що підштовхують до вивчення можливостей воксельної технології, є перспектива редагування форм в реальному часі. Вказана характеристика дуже затребувана в сфері КГ. Цей попит частково обумовлений розвитком сфери цифрової графіки, що потребує все більшого реалізму та розширення можливостей, та частково відсутністю такої можливості у сучасного підходу до побудови 3D сцен.

Серед методів редагування воксельної графіки, можна виділити метод, що базується на прокладанні променів. Визначаючи координати точки перетину променю та моделі можнв зпівставити їх з вузом в деревовидній структурі і відповідно перетворити її, видаливши або додавши, черговий воксель в структуру.

Механізм прокладання променів дійсно непогано зістикується з деревовидною структурою, проте подібні перетворення все ще досить затратні за обчислювальними потужностями з урахуванням відсутності апаратної підтримки воксельної технології. До того ж прокладаючи промінь за раз можна редагувати лише один воксель, що з урахуванням можливого представлення моделі мільйонами чи навіть мільярдами вокселів очевидно є нераціональним.

В даній розробці запропонований інший підхід до редагування воксельних зображень. Цей також базується на використанні буферів глибини, і переймає недоліки подібних методів. Проте простота використання і швидкодія, а також його сумісність з запропонованими раніше рішеннями є вирішальними для його використання.

Суть методу полягає у зміні буферів глибини при вокселізації, шляхом додавання до них матриць того ж розміру, що містять певний шаблон. Таким

чином, можна виконувати перетворення початкового об’єкту шляхом імітації відтисків певної форми на об’єкті.

# 3. ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Опис інтерфейсу розробленої програми

Програмний інтерфейс розробленого продукту представлений декількома вікнами, що мають різні призначення і змінюють одне одного по мірі виникнення необхідності в їх представленні. Таким чином, початкове вікно представлене на рисунку 3.1 відповідає за визначення поверхні та параметри її візуалізації.

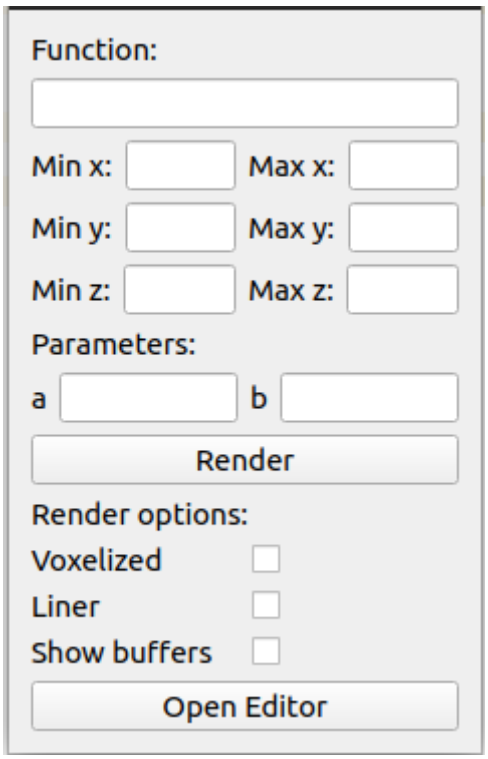


Рисунок 3.1 – Початковий інтерфейс програми

Секція Function, головного вікна, відповідає за аналітичне визначення описаної поверхні, встановлення її параметрів та визначення області визначення змінних, таким чином, повністю визначаючи її побудову.

Секція Render options головного вікна, дає змогу визначити необхідні режими для відображення заданої функції. За замовчуванням аналітична функція буде представлена за допомогою хмари точок.

- Опція **Voxelized** встановлює режим вокселізації вказаної поверхні з її подальшою візуалізацією. Ця опція є основним режимом для розробленого програмного продукту, адже мета даної роботи полягає саме в дослідженні використання воксельної графіки в сучасних персональних комп'ютерах.
- Опція **Liner** є допоміжною при відображені моделі. Використання даної опції задає режим побудови трафаретів, що в свою чергу доволі зручно при необхідності дослідження побудованого дерева, проте підключення даної опції має сенс лише з використанням опції Voxelized.
- Опція **Show buffer** відповідає за відображення буферів глибини. Це зручно для кращого розуміння принципів роботи алгоритмів побудови та обробки розріджених дерев.

Після закінчення вибору необхідних опцій та визначення поверхні можна переходити до її візуалізації. Для переходу до вікна візуалізації необхідно натиснути кнопку Render на панелі головного інтерфейсу. Приклад візуалізації показано на рисунку 3.2.

Для вікна візуалізації реалізована складна рухома камера, тобто можливі вільні зміни кутів огляду моделі, а також приближення. Камера реалізована за допомогою використання кутів Ейлера, а конкретніше, кута прецесії та кута власного обертання. Рух з використанням кута прецесії реалізовано з використанням клавіш A та D. Рух з використанням кута власного обертання реалізовано за допомогою використання клавіш W та S. Приближення та віддалення камери реалізовано з використанням повзунка мишки. Також можливий рук камери з використання рухів мишки з зажатою лівою кнопкою. Відповідно, горизонтальні рухи реалізують повороти навколо оси Y, а вертикальні - навколо осі X.

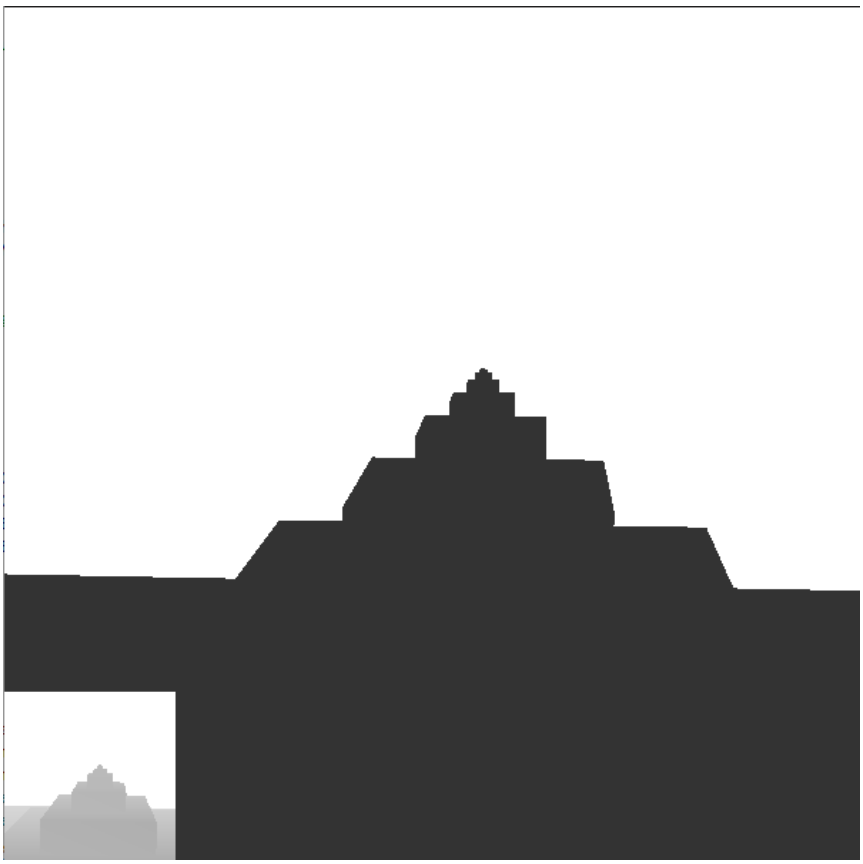

Рисунок 3.2 – Приклад візуалізації з відображенням буферу глибини

Реалізована програма також підтримує використання методу редагування воксельного зображення, описаного в пункті 2.3 розділу 2 даної роботи. Для відредагування зображення необхідно відкрити вікно редактора, натиснувши кнопку Editor в головному вікні. Вікно редагування надає можливість завантаження шаблонів для їх подальшого суміщення з буфером глибини. Вигляд вікна зображено на рисунку 3.3.

Для коректної роботи редактора необхідно попередньо визначити аналітичну функцію, а також її параметри та область визначення, і виконати її візуалізацію для побудови коректних буферів глибини за допомогою ГП. Для виконання обробки необхідно спочатку виконати попередню побудову бажаного зображення зі збереженням його в файл. Далі необхідно вказати програми для отримання файлу з бажаним зображенням та натиснути кнопку Download. Після цього за допомогою повзунка напроти Buffer number необхідно вибрати, до якого з 6 буферів глибини необхідно застосувати зміни. Наступним кроком є

визначення віддаленості бажаного зображення від точку огляду. Відповідно до принципу роботи буфера глибини дане віддалення встановлюється за допомогою повзунка напроти Intensive та лежить в межах від 0 до 1. Для завершення побудови необхідно натиснути кнопку Apply. Після цього буде виконано розрахунок інших 5 трафаретів та додавання їх до буферів глибини.

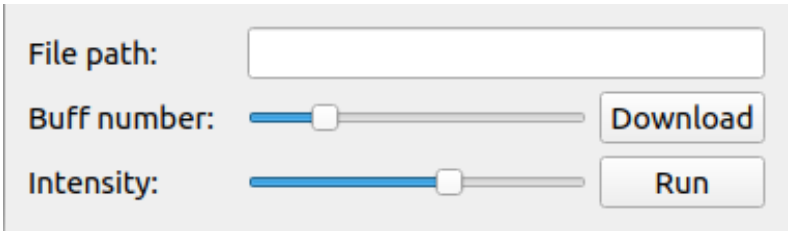


Рисунок 3.3 – Вигляд вікна редактору

3.2 Опис алгоритмів побудови та відображення розрідженого воксельного дерева.

За основу для представлення воксельних моделей в роботі концепція розрідженого воксельного дерева. Для побудови та візуалізації дерева були розроблені алгоритми схематичне представлення яких показано в додатку Д1.

При розробці алгоритмів було прийнято рішення використання концепції рекурсивних викликів для обходу дерева та виключення інформації про координати вокселів для максимального зменшення розміру пам’яті, що використовується.

Побудова дерева в програмі здійснюється за допомогою функції buildTree. Функція приймає в якості параметрів посилання на поточний вузол дерева, координати вокселу, що необхідно видалити з дерева та координати поточного вузла, поточну глибину дерева та розмір вокселу, що відповідає поточному вузлу.

Першим кроком роботи функції є перевірка чи поточна глибина дерева не перевищує необхідного рівня деталізації. У разі перевищення функція повертає управління.

Наступним кроком є перерахунок поточного рівня деталізації об’єкту та розміру вокселу вузла. Далі відбувається визначення октету в якому знаходиться необхідний воксел та номеру дочірнього вузла, що відповідає цьому октету.

Після визначення номеру дочірнього вузла відбувається перевірка чи існує цей вузол. Існування вузла еквівалентно наявності в поточному октеті видаленої частини. Відповідно якщо необхідний дочірній вузол, ще не створений відбувається його створення, та позначення біту відповідного номеру у спеціальній змінній, що фіксує факт наявності порожнини у відповідному октеті.

Останнім кроком функціонування є рекурсивний виклик функції buildTree для визначеного дочірнього вузла з відповідними параметрами. Після повернення з рекурсивного виклику відбувається перерахунок значень рівня глибини та розміру вокселу до початкового батьківського рівня. Реалізація описаної функції мовою C++ приведена нижче.

```
void glScene::buildTree(struct SVO *root, GLuint x, GLuint y, GLuint z, GLuint rx,
GLuint ry, GLuint rz){
   static unsigned lvl = 0;
   static unsigned delta = width;

   GLuint px, py, pz, k = 0;

   if (lvl < detalLevel){
      lvl++;
      delta /= 2;
      if (x < rx + delta) {
         px = rx;
      } else {
         px = (rx + delta);
         k += 1;
      }
      if (y < ry + delta){
         py = ry;
      } else {
         py = (ry + delta);
```

```
            k += 2;
        }
        if (z < rz + delta){
            pz = rz;
        } else {
            pz = (rz + delta);
            k += 4;
        }
        if (root->child[k] == NULL){
            root->child[k] = addNode();
            root->isSet = root->isSet | CH_SET[k];
        }
        buildTree(root->child[k], x, y, z, px, py, pz);
        delta *= 2;
        lvl--;
    }

    return;
}
```

Візуалізація дерева здійснюється за допомогою функції drawTree. Функція приймає в якості параметрів посилання на поточний вузол дерева, координати центру поточного вокселу та його розмір.

Першим кроком є перевірка чи наявні дочірні вузли з порожнинами за допомогою біт спеціальної змінної у вузлі. Якщо такі вузли відсутні функція завершується.

Наступним кроком є перерахунок розміру вокселу вузла. Далі для кожного з дочірних вузлів визначаються координати центру вокселу, що їм відповідає. Після визначення координат відбувається перевірка чи існує дочірній вузол. Якщо вузол існує відбувається рекурсивний виклик функції drawTree з відповідними параметрами. У разі відсутності вузла відбувається перевірка відповідного біту спеціальної змінної вузла, якщо біт не встановлений відбувається побудова поточного вокселу.

Після завершення обробки дочірніх вузлів необхідно повернути розмір вокселу до початкового значення, після чого функція завершиться. Реалізація описаної функції а мові C++ приведена нижче.

```
void glScene::drawTree(struct SVO *root, GLfloat x, GLfloat y, GLfloat z){
    static unsigned delta = width;

    GLfloat px = x, py = y, pz = z, k = 0, d;

    if (root->isSet != 0){
        delta /= 2;

        d = ((GLfloat) delta) / width;
        for (int i = 0; i < 8; i++){
            k = i;
            if (k >= 4) {
                k -= 4;
                pz = z + d / 2.0;
            } else {
                pz = z - d / 2.0;
            }
            if (k >= 2) {
                k -= 2;
                py = y + d / 2.0;
            } else {
                py = y - d / 2.0;
            }
            if (k >= 1) {
                k -= 1;
                px = x + d / 2.0;
            } else {
                px = x - d / 2.0;
            }
            if (root->child[i] != NULL){
                drawTree(root->child[i], px, py, pz);
            } else {
                if ((root->isSet & CH_SET[i]) == 0){
                    voxel.model = glm::mat4(1.0);
                    voxel.position = glm::vec3(px, py, pz);
                    voxel.model = glm::translate(voxel.model, voxel.position);
                    voxel.model = glm::scale(voxel.model, glm::vec3(d, d, d));
                    drawVoxel();
                }
            }
        }

        delta *= 2;
    }
```

```
    return;
}
```

Використання спеціальної змінної у вузлі дерева для збереження інформації про наявність видалень у вузлі дозволяє після завершення генерації дерева об’єднати заповнені частини дерева у більші вокселі шляхом видалення суцільних вузлів дерева зберігаючи при цьому інформацію про їх заповнення. Таким чином до описаних функцій побудови та візуалізації необхідна ще одно проміжна функція, що буде проводити очищення дерева від непотрібних вузлів. Фактично побудова моделі здійснюється в два етапи і реалізується за допомогою пари функцій buildTree та removeNode.

Видалення зайвих вузлів дерева здійснюється за допомогою функції removeNode. Вона приймає єдиний аргумент – посилання на поточний вузол. Ця функція також повертає значення 1, якщо в результаті її роботи було видалено поточний вузол, або 0 в іншому випадку.

Першим кроком роботи функції є рекурсивний виклик цієї функцію для кожного з існуючих дочірніх вузлів. Якщо після виклику для дочірнього вузла було повернуто значення 1, він видаляється з дерева.

Наступним кроком спеціальної змінної. Якщо ні один з дочірніх вузлів не відмічений, як з порожниною, то поточний вузол видаляється і функція повертає значення 1. В іншому випадку відбувається перевірка чи всі з вузлів містять порожнини та вказівники на них були видалені на попередньому етапі. В разі позитивної відповіді вузол видаляється та повертається значення 1, у будь якому іншому разі просто повертається 0. Реалізація описаної функції а мові C++ приведена нижче.

```
int glScene::removeNode(struct SVO *node){
    int a;

    for (int i = 0; i < 8; i++){
        if (node->child[i] != NULL){
            a = removeNode(node->child[i]);
```

```
            if (a) node->child[i] = NULL;
        }
    }
    if (node->isSet == 0 && node != root){
        free(node);
        return 1;
    } else {
        if (node->isSet == 255){
            a = 0;
            for (int i = 0; i < 8; i++){
                if (node->child[i] != NULL) a++;
            }
            if (a != 0){
                return 0;
            } else {
                    free(node);
                return 1;
            }
        } else {
            return 0;
        }
    }
}
```

# 4. ПОБУДОВА МОДЕЛІ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

## 4.1 Вокселізація поверхонь

Для початку дослідження можливостей розробленого графічного редактору необхідно проаналізувати візуалізацію простої симетричної поверхні на кшталт сфери. Побудова початкової моделі здійснюється за допомогою хмари точок і представлена на рисунку 4.1. Як видно з даних, представлених в буферах глибини, фігура є повністю симетричною.

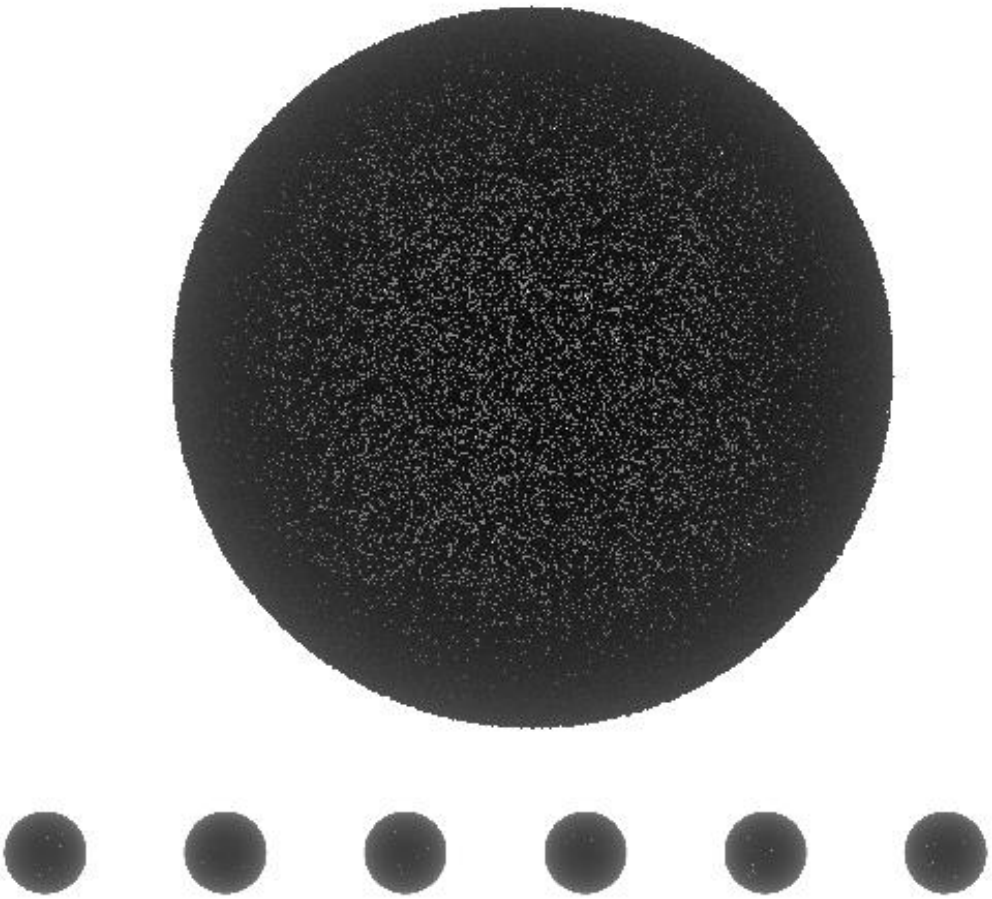

Рисунок 4.1 – Візуалізація поверхні сфери.

На рисунку 4.2 представлена воксельна реалізація сферичної поверхні, що відповідає попередній реалізації, як можна бачити форма модель доволі точно описує сферу, невелика шорсткість поверхні обумовлена використанням для візуалізації початкової моделі хмари точок. З іншого боку алгоритм не прив’язаний до методу візуалізації початкової моделі, і можна було б використати, наприклад, стандартні полігони, проте вони, в свою чергу, також внесли б відхилення в кінцеву побудову через тупі кути між полігонами.

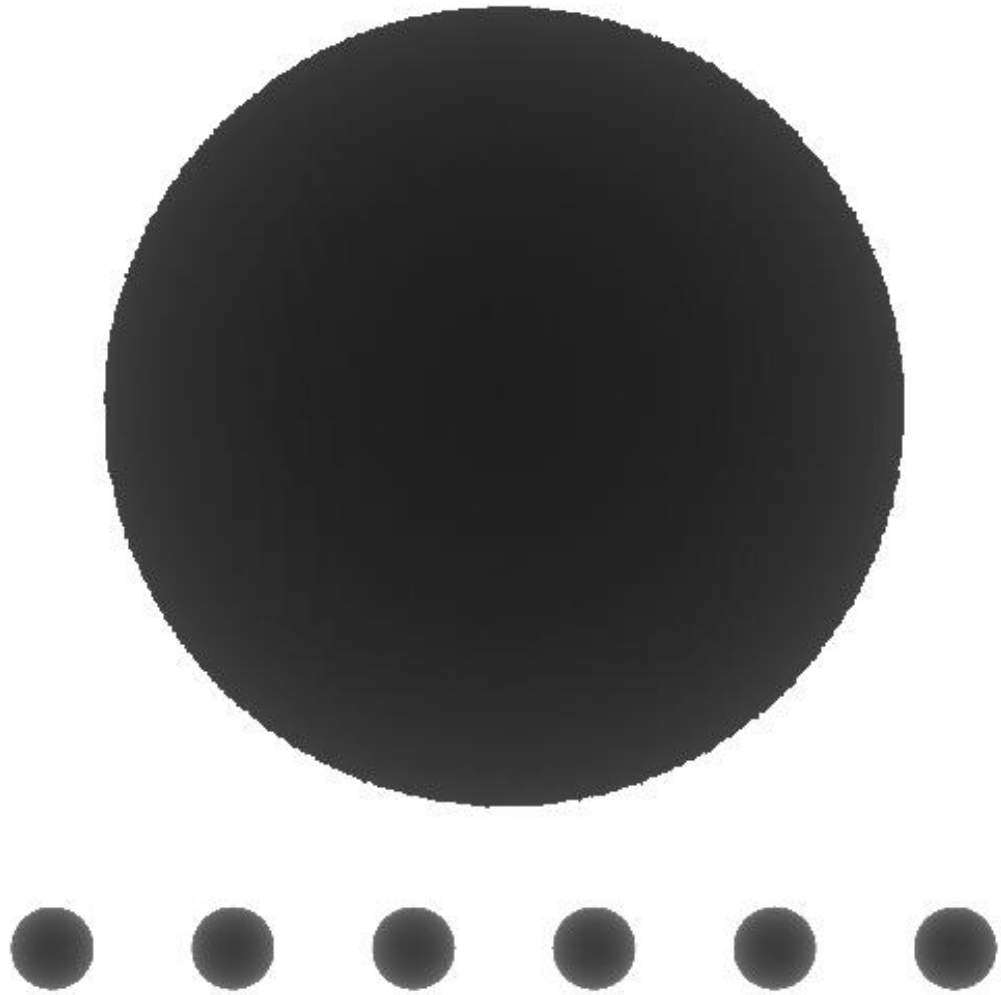

Рисунок 4.2 – Воксельна візуалізація сферичної поверхні.

Перейдемо до побудови більш складних моделей. Для початку варто здійснити побудову простої не повністю симетричної моделі. В якості прикладу

візьмемо модель тора. Для більшої несиметричності модель була розвернута на 45 градусів навколо осі, що в свою чергу дозволило збільшити кількість інформації про лінійні розміри тіла в буферах глибини. Візуалізація моделі представлена на рисунку 4.3.

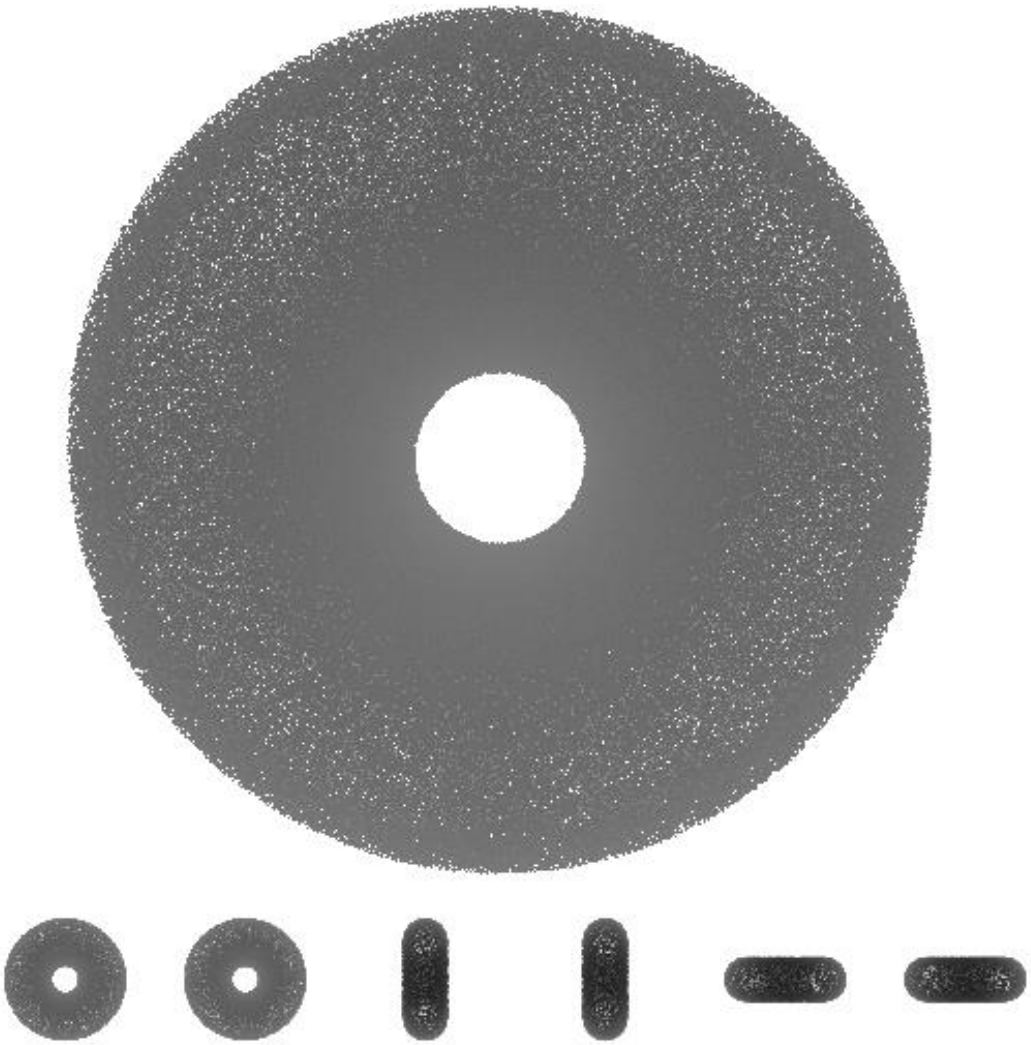

Рисунок 4.3 – Візуалізація поверхні тора

На рисунку 4.4 зображена вже вокселізований результат. Як і у випадку зі сферичною поверхнею, для розробленого редактора не стало проблемою доволі точно візуалізувати поверхню тора. Дійсно, хоча і здійсннено поворот фігури, вона все ще являється симетричною, а також повністю опуклою, тож насправді її візуалізацію можна було виконати, використовуючи лише одну пару буферів.

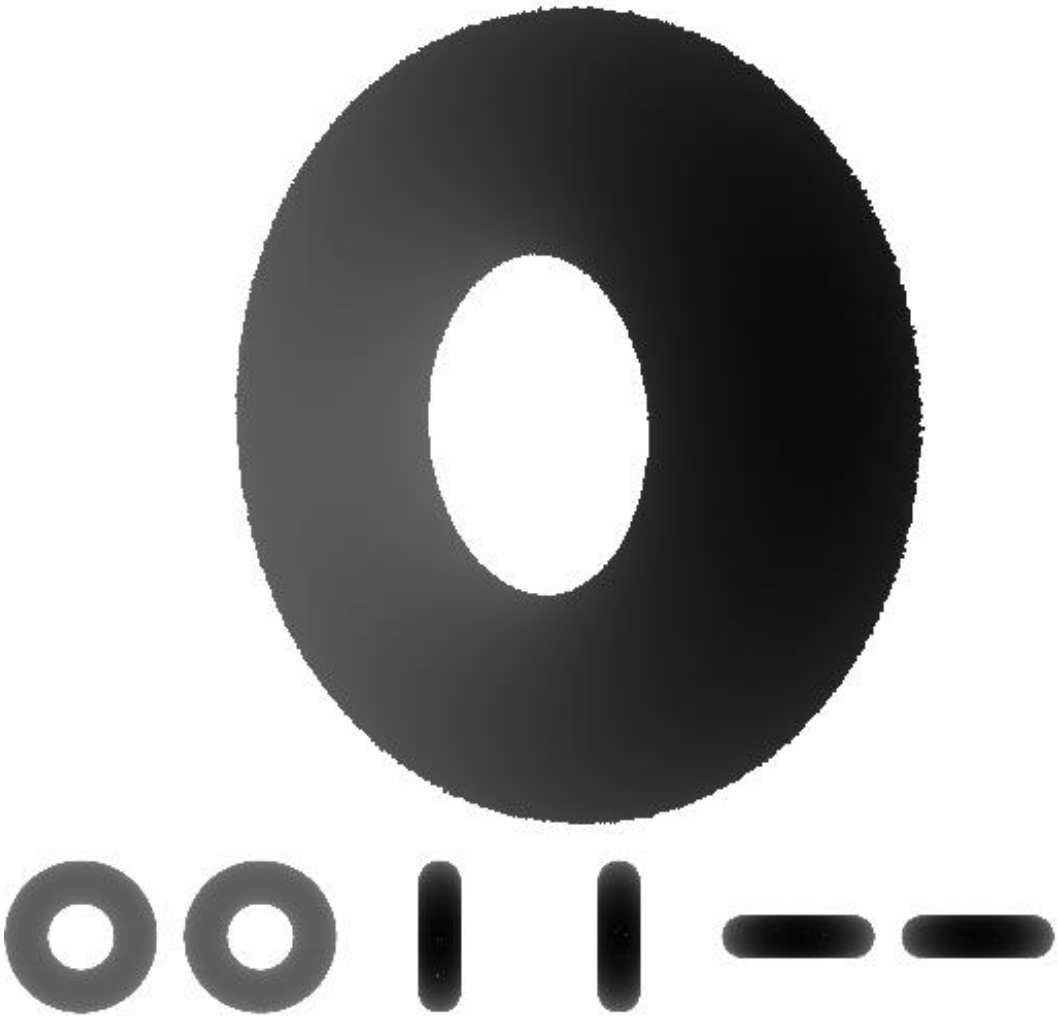

Рисунок 4.4 – Воксельна візуалізація поверхні тора

Перейдемо до повністю несиметричних фігур. Для прикладу було прийнято рішення взяти модель каркасу куба, проте каркас не є простою поверхнею і не може бути описаний простою аналітичною функцією, тому для його побудови були використані інші прості геометричні поверхні.

На рисунку 4.5 представлена візуалізація моделі каркасу куба. Для візуалізації були використані рівняння бокової поверхні циліндра та поверхні сфери. Побудова ребер куба відповідає побудові нахилених та повернутих поверхонь циліндру зміщених у потрібні координати, а сферичні поверхні використовуються як вершини куба для прикриття порожнистості трубчатої конструкції.

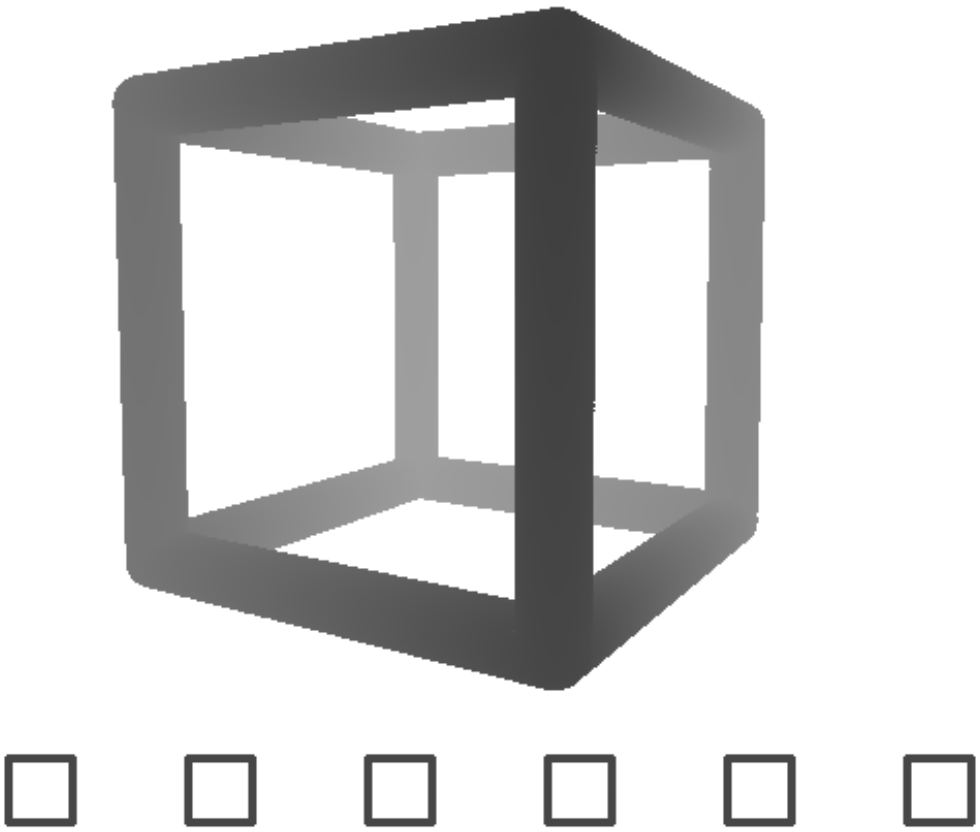

Рисунок 4.5 – Візуалізація трубчатого каркасу куба.

На рисунку 4.6 зображено результат вокселізації описаної конструкції. Як можна бачити, незважаючи на несиметричність конструкції і навіть той факт, що з певних ракурсів можна спостерігати перекриття граней та окремих точок конструкції на кшталт внутрішніх точок стику ребер, побудована фігура доволі точно повторює початкове зображення. Таким чином, можна говорити, що описаний алгоритм не обмежується візуалізацією лише симетричних моделей і для нього можлива побудова деяких типів скритих поверхонь. Даний результат обумовлений використанням відразу 6 буферів глибини, тобто фактично візуалізацією відразу з 6 ракурсів. Якщо хоча б на одному з 6 буферів

відображено коректне положення точки, то можливе її точне перетворення в вершину дерева.

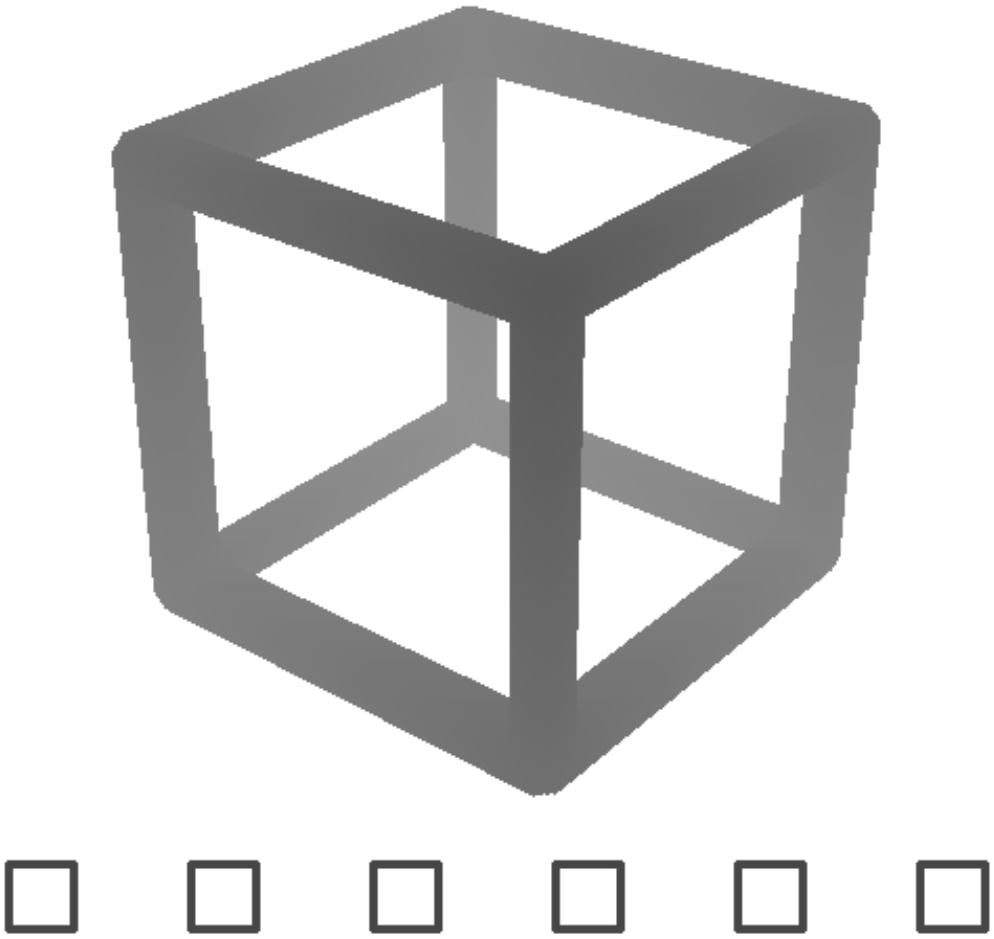

Рисунок 4.6 – Воксельна модель трубчатого каркасу куба.

Для перевірки описаного твердження побудуємо фігуру, що містить ділянки, видимі лише з одного ракурсу. В якості прикладу побудови, зважаючи на поставлені вимоги, а також як данину одному з традиційних примітив КГ, що носить назву «чайник з Юти», було прийняте рішення побудови простої моделі чашки. Як і в попередньому прикладі, чашка не може бути описана простою аналітичною функцією, тому для її побудови використані поверхні сфери та тору. З двох півсфер різного розміру, але зі спільним центром складається основне тіло чашки. Один з торів використовується для прикриття порожнини

між сферами. Другий тор реалізує компонент ручки, більша його частина з отвором виступає з чашки, а та, що залишилась, скрита між поверхнями сфер. Описана модель зображена на рисунку 4.7.

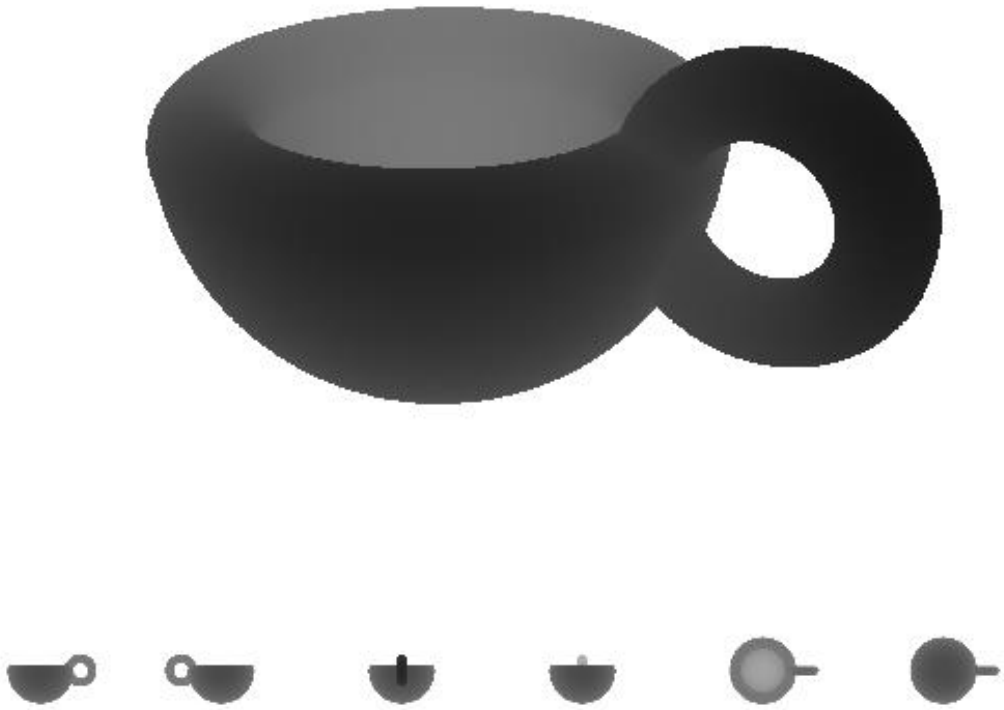

Рисунок 4.7 – Візуалізація описаної моделі чашки.

Як і очікувалось, алгоритм чудово справився з поставленою задачею. Підсумовуючи результати, можна сказати, що використаний простий алгоритм насправді являється досить дієвим і цілком життєздатним. Так, алгоритм має обмеження і в простому вигляді не здатен будувати скриті порожнини, проте при реальному використані виявляється, що описані випадки скоріше є винятками,

ніж правилом. При звичайному використані користувачам рідко потрібна візуалізація того, що не можна побачити.

Простота розуміння роботи алгоритму та його швидкодія, що не залежить від складності побудованої фігури, робить його конкурентоспроможним серед своїх родичів в задачах, що вимагають лише вокселізації тіл для подальшої візуалізації. В крайньому випадку при гострій необхідності реалізації порожнини в оброблюваній моделі, можна просто цю порожнину додатково побудувати, використовуючи той же алгоритм, але не видаляючи з дерева порожнини, а додаючи їх. Результат обробки описаної моделі відображено на рисунку 4.8.

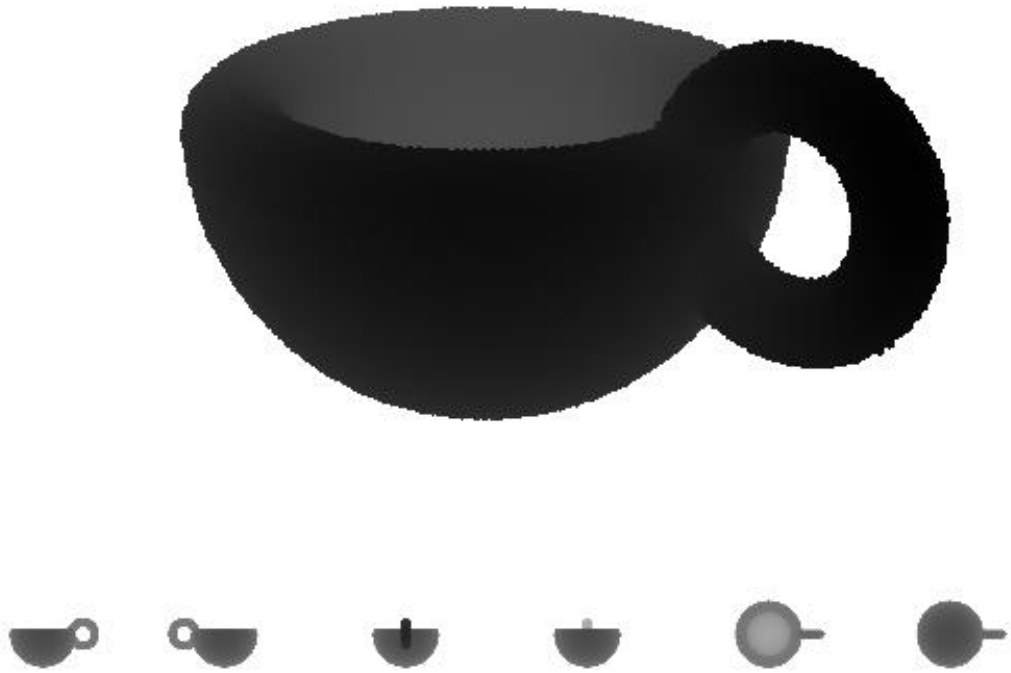

Рисунок 4.8 – Воксельна модель чашки.

Для дослідження можливостей редагування воксельних моделей в роботі був реалізований алгоритм, описаний в пункті 2.3 розділу 2. Відповідно для роботи з описаним методом необхідні початкові шаблони. В якості шаблонів були підготовлені зображення українських літер. Виготовлені шаблони зображені на рисунку 4.9.

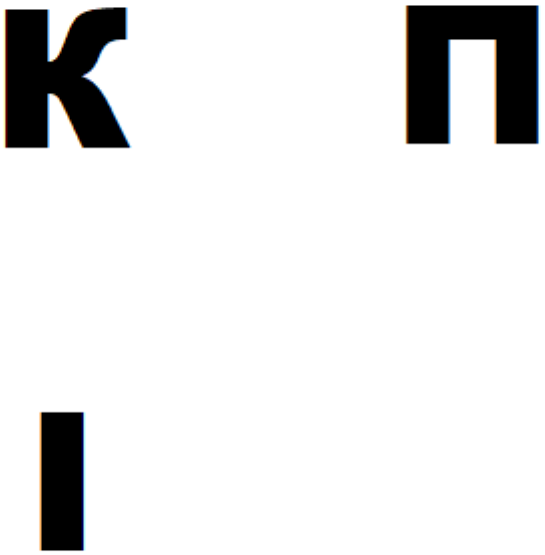

Рисунок 4.9 – Шаблони для редагування воксельного зображення

В якості початкової функції був вибраний куб одиничного розміру. В результаті підбору оптимальної комбінації параметрів був отриманий результат, представлений на рисунку 4.10.

Використаний метод, використовуючи отриманий шаблон, спочатку модифікується відповідно до встановлених параметрів. Далі на основі активних

буферів глибини добудовує додаткові шаблоні. На наступному кроці побудовані шаблони змішуються з існуючими буферами глибини, утворюючи проекції зовсім іншої фігури. Останнім кроком є побудова новоспеченої фігури.

Описаний підхід, попри свою простоту обчислень та розуміння, являється мало придатним для прямого використання. Для імітації ефектів, складніших ніж відтиск, потрібні додаткові розрахунки.

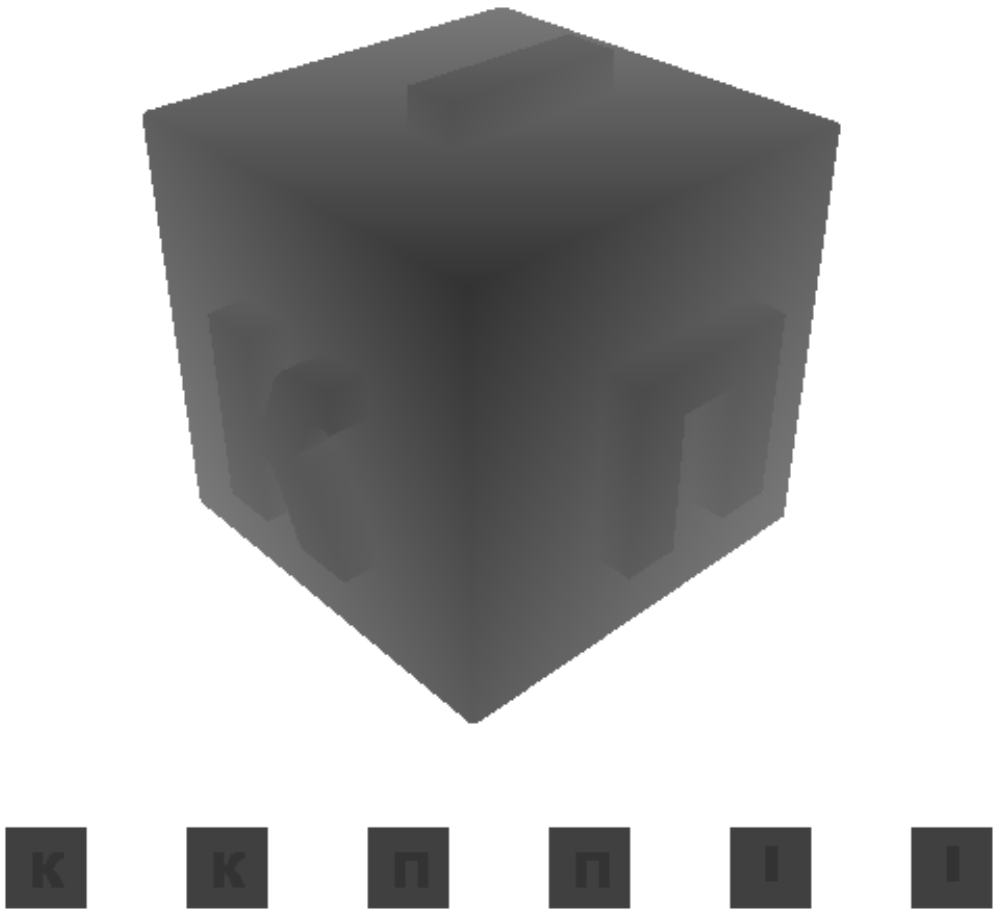

Рисунок 4.10 – Відредагована воксельна модель.

## 4.2 Аналіз результатів

Метою розробки було дослідження можливості використання воксельної технології на сучасних комп'ютерах, а також розробка простого графічного редактора з використанням 3D моделі Voxel, тому варто підсумувати отримані результати.

Для збору результатів були використані тіла, побудова яких була показана раніше. Отримані результати через особливості побудови початкової моделі виявились дуже чутливими до якості цієї побудови, тому оптимальні параметри сильно корелюють для різних тіл. Проте для збору результатів та загальної статистики, це не є надзвичайно суттєвим.

В таблиці 1 представлені результати, отримані для побудови моделі з розширенням 256*256*256 пікселів. Звичайна матрична модель з розрахунку 12 байт на координати та 4 байт на колір потребує 256 Мбайт пам'яті, в свою чергу здається марнуванням апаратних ресурсів на фоні полігональних чи сплайнових моделей, що займають в десятки чи сотні разів менше пам'яті. Як видно з таблиці, використання воксельних дерев дозволяє зменшити обсяг витрат пам'яті на порядки, що загалом наближає їх по обсягу використаних ресурсів до інших типів представлення. Степінь стиснення сильно відрізняється від форми поверхні, що здебільшого обумовлено збільшенням площі поверхні.

Таблиця 1 – Коефіцієнти стиснення поверхонь

| Поверхня | Кількість вузлів у дереві | Необхідний обсяг пам'яті, байт | Коефіцієнт стиснення |
|---|---|---|---|
| Сфера | 15847 | 1267760 | 211,74 |
| Тор | 55418 | 4433440 | 60,55 |
| Трубчатий каркас | 13337 | 1066960 | 251,59 |
| Чашка | 10244 | 819520 | 327,55 |

Чим ближчий воксель до поверхні об’єкту, тим більш деталізованим він має бути. Збільшення площі поверхні, призводить до збільшення кількості векселів максимального рівня деталізації, а отже й кількості вузлів дерева. Отримані результати дещо спотворені використанням однакової кількості точок для генерації поверхонь з різною площею. Таким чином поверхня тора, що має найбільшу площу виявилась рихлою, тим самим ще більше збільшивши площу і відповідно кількість вузлів.

Більш наглядною демонстрацією користі від використання розріджених дерев є менший коефіцієнт зростання необхідного розміру зі зростанням рівня деталізації. При використані звичайної побудови, зі збільшенням деталізації в 2 рази необхідний розмір збільшувався в 8 разів, а при використані дерева цей коефіцієнт залежить від складності поверхні і для тестових поверхонь лежить в межах від 4 до 8, оцінити цей коефіцієнт можна з результатів в таблиці 2.

Таблиця 2 – Залежність кількості вузлів дерева від рівня деталізації

| Рівень деталізації | 6 | 7 | 8 |
|---|---|---|---|
| Сфера | 926 | 3853 | 15847 |
| Тор | 2888 | 12620 | 55418 |
| Трубчатий каркас | 137 | 2080 | 13337 |
| Чашка | 602 | 2508 | 10244 |

З першого погляду може бути не очевидним звідки береться такий значний виграш у розмірі використаних ресурсів, якщо коефіцієнт поверхні і для звичайної побудови, і для використання дерева близький до 8. Відповідь на це питання полягає в тому, що дерево кодує не кожен окремий воксель простору, а лише заповнені вокселі. Таким чином, каркасна поверхня, що обмежує тіло з малим об’ємом і фактично представлена лише найдрібнішими векселями поверхні, має коефіцієнт, близький до звичайного, проте сама кількість векселів,

що необхідно кодувати через малий об’єм, дуже незначна, як наслідок отримане стиснення одне з найкращих. З іншого боку при кодуванні значних об’ємів, наприклад, при обмежені об’єму сферою, необхідне збереження інформації про велику кількість вокселей, але за рахунок об’єднання заповнених частин простору спостерігається значне зниження коефіцієнту і, відповідно, обсягів використаної пам’яті. Ці дві властивості є повністю незалежними і чудово гармонують між з собою, як наприклад, у випадку чашки. Тіло складної форми, що має помірну площу поверхні і помірний об’єм, чудово гармонує користуючись обома властивостями дерева, досягаючи найкращого результату стиснення загалом.

# ВИСНОВКИ

В роботі показано, що розробка графічного редактору для побудови 3D моделі Voxel представляє собою довгий та складний процес, що потребує гарного рівня знань відразу в кількох різних сферах.

Метою даної роботи було дослідження воксельної технології як можливою альтернативи актуальній моделі побудови 3D зображення та розробка простого графічного редактора для цього.

В ході розробки були дослідженні переваги та недоліки воксельної технології, а також розглянуті існуючі підходи до організації представлення воксельних моделей.

Для реалізації описаних технологій проведена розробка структури даних представлення воксельного дерева.

Для роботи з воксельним деревом проведено розробку алгоритмів обробки буферів, побудови та візуалізації дерева.

Для генерації початкових моделей розроблено алгоритми побудови аналітичної поверхні та генерації буферів.

Редагування моделей реалізовано за допомогою додавання буферів глибини з використанням бажаних шаблонів.

Реалізовано інтерфейс графічного редактору для визначення аналітичної поверхні та бажаних параметрів відображення.

Реалізовано інтерфейс графічного редактору для редагування готових моделей з використання вказаних шаблонів.

Візуалізація отриманої моделі виконується з використанням розроблених шейдерів та підтримки розробленої рухомої камери.

Аналізуючи результати дослідження, представленні в даній роботі, можна чітко сказати, що на даному етапі розвитку графічних процесорів, представлених у користувацькому сегменті ринку, використання вокселів цілком можливо. Обсяги відеопам’яті сучасних комп’ютерів дозволяють зберігати воксельні

моделі навіть десятки мільярдів вокселів. Проте, в реаліях відсутності апаратної підтримки даної технології, інкапсуляція вокселів полігонами значно скорочує перелік можливих сфер використання технології. Необхідність накладних обчислень полігонів ставить під сумнів можливість використання розглянутої технології для генерації високоякісного зображення реального часу.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Ветлугина К.И. Применение компьютерной графики в начальной школе // Современные научные исследования и инновации. 2017. № 4.: URL: http://web.snauka.ru/issues/2017/04/80747 (дата звернення: 02.02.2020).

2. Проузис Д. Как работает компьютерная графика: Москва. -2007. – 654 с.

3. Simulation of a two-giro gravity attitude control system - Edward Zajac – YouTube : URL: https://www.youtube.com/watch?v=m8Rbl7JG4Ng (дата звернення: 02.03.2020).

4. Gordon E. Moore. Cramming more components onto integrated circuits. April 19, 1965.

5. Станут ли воксели новой прорывной технологией? / Хабр : URL: https://habr.com/ru/post/371751/

6. Atomontage Inc.: URL:https://www.atomontage.com/ (дата звернення: 30.03.2020).

7. Вольхин К.А. Основы компьютерной графики : URL: http://ng.sibstrin.ru/wolchin/umm/l_kg/kg/r002/008_3.htm (дата звернення: 30.03.2020).

8. Berger, M., Tagliasacchi, A., Seversky, L. M., Alliez, P., Guennebaud, G., Levine, J. A., Sharf, A. and Silva, C. T. (2016), A Survey of Surface Reconstruction from Point Clouds. Computer Graphics Forum : URL: https://web.archive.org/web/20161221090040/http://gfx.uvic.ca/pubs/2016/reconstar/paper.pdf (дата звернення: 02.04.2020).

9. Bernardini, Fausto; Mittleman, Joshua; Rushmeier, Holly; Silva, Cláudio; Taubin, Gabriel (1999). The Ball-Pivoting Algorithm for Surface Reconstruction. IEEE Transactions on Visualization and Computer Graphics 5: 349–359.

10. Рендеринг с помощью вокселей: новый уровень графики в играх? - THG.RU : URL: http://www.thg.ru/graphic/voxel_ray_casting/onepage.html (дата звернення: 04.04.2020).

11. This demonstration of real-time sparse voxel octree technology was debut at Siggraph 2008. : URL: https://www.youtube.com/watch?v=VpEpAFGplnI (дата звернення: 05.04.2020).

12. S.Wang and A.Kaufman. “Volume Sampled Voxelization of Geometric Primitives.” In Proceedings of the Visualization '93 Conference, pp 78-85. IEEE Computer Society Press, 1993

13. D.Cohen, A.Kaufman and Y.Wang. “Generating a Smooth Voxel-Based Model from an Irregular Polygon Mesh.” The Visual Computer, 10:295-305 (1994).

14. M.Jones. “The Production of Volume Data from Triangular Meshes Using Voxelization.” Computer Graphics Forum, 15:311-318 (1996).

15. E.A. Karabassi, G. Papaioannou, Th. Theoharis, A Fast Depth Buffer Based Voxelization Algorithm, Journal of Graphics Tools, ACM, 4(4), pp.5-10, 1999.