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

**НН Інститут прикладного системного аналізу**

**Кафедра математичних методів системного аналізу**

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2023 р.

# Дипломна робота

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Системний аналіз і управління»**

**спеціальності 124 «Системний аналіз»**

**на тему: «Відновлення тривимірних об'єктів за фотографією»**

Виконав:
студент IV курсу, групи КА-94
Андрушко Назар Дмитрович _____

Керівник:
професор, д.т.н., Данилов Валерій Якович _____

Консультант з економічного розділу:
доцент, д.е.н., Рощина Надія Василівна _____

Консультант з нормоконтролю:
к.ф.-м.н. Статкевич Віталій Михайлович _____

Рецензент:
директор НН ФТІ, д.т.н., професор
Новіков Олексій Михайлович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**НН Інститут прикладного системного аналізу**
**Кафедра математичних методів системного аналізу**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«__» травня 2023 р.

# ЗАВДАННЯ

**на дипломну роботу студенту**
Андрушку Назару Дмитровичу

1. Тема роботи «Відновлення тривимірних об’єктів за фотографією», керівник роботи Данилов Валерій Якович, професор, д.т.н., затверджені наказом по університету від «__» травня 2023 р. № _____

2. Термін подання студентом роботи __.06.2023.

3. Вихідні дані до роботи

4. Зміст роботи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи[1]*

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Економічний | Рощина Надія Василівна, доцент, д.е.н. | | |

7. Дата видачі завдання _____

[1]* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

Календарний план

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Огляд літератури за темою роботи | 01.02.2023-28.02.2023 | Виконано |
| 2 | Ознайомлення з ДСТУ 3008:2015 | 20.02.2023-28.02.2023 | Виконано |
| 3 | Формулювання задачі дипломної роботи після теоретичного ознайомлення | 01.03.2023-10.03.2023 | Виконано |
| 4 | Проведення попередніх експериментів | 11.03.2023-28.04.2023 | Виконано |
| 5 | Підготовка розділів під час переддипломної практики | 28.04.2023-12.05.2023 | Виконано |
| 6 | Проведення остаточних експериментів, розробка програмного продукту | 13.05.2023-20.05.2023 | Виконано |
| 7 | Оформлення записок та розділів в дипломній роботі відповідно до норм контролю | 21.05.2023-29.05.2023 | Виконано |
| 8 | Попередній захист дипломної роботи | 30.05.2023-09.06.2023 | Виконано |
| 9 | Захист дипломної роботи | 19.06.2023-23.06.2023 | |

Студент Назар АНДРУШКО

Керівник Валерій ДАНИЛОВ

# РЕФЕРАТ

Дипломна робота містить - сторінок, 27 рисунків, 10 таблиць, 2 додатки. Було використано 31 джерело.

ШТУЧНІ НЕЙРОННІ МЕРЕЖІ, ГЛИБОКЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, МЕТОД ЗВОРОТНОГО ПОШИРЕННЯ ПОМИЛКИ, ТРИВИМІРНІ ОБ’ЄКТИ.

Об’єкт дослідження полягає у вивченні різних методів реконструкції тривимірних моделей.

Предмет дослідження - використання штучних нейронних мереж для вирішення проблем, пов’язаних із тривимірною реконструкцією за допомогою зображень.

Мета роботи - вивчення сучасних підходів до реконструкції об’єктів у тривимірному середовищі.

Цей проект представляє нову архітектуру нейронної мережі для представлення даних шляхом поєднання гіпермереж з мережами зайнятості (occupancy networks). Запропонована архітектура має на меті підвищити точність реконструкції тривимірних моделей з використанням представлення неявної функції. Робота привела до створення архітектури штучної нейронної мережі, яка була розроблена в результаті дослідження тривимірної реконструкції об'єктів. Отримані результати включають показники ефективності методу, а також надають напрямки для подальших досліджень. Експерименти, проведені на наборі даних ShapeNet, демонструють ефективність архітектури в досягненні якості представлення.

# ABSTRACT

Thesis contains - pages, 27 figures, 10 tables, 2 appendices, 31 sources.

ARTIFICIAL NEURAL NETWORKS, DEEP LEARNING, ARTIFICIAL INTELLIGENCE, ERROR BACKPROPAGATION METHOD, THREE-DIMENSIONAL OBJECTS, HYPERNETWORKS, EMPLOYMENT NETWORKS, THREE-DIMENSIONAL RECONSTRUCTION.

The object of the research is to study different methods of reconstruction of three-dimensional models.

The purpose of the research is to use artificial neural networks to solve problems related to three-dimensional reconstruction using an image.

The purpose of the work is to study modern approaches to the reconstruction of objects in a three-dimensional environment.

This project presents a new neural network architecture for data representation by connecting to a hyper-network with occupancy networks. The proposed architecture aims to improve the accuracy of reconstruction of three-dimensional models using implicit function representation. The work led to the creation of an artificial neural network architecture, which was developed as a result of research into three-dimensional reconstruction of objects. The obtained results include indicators of the effectiveness of the method, and also provide directions for further research. Experiments performed on the ShapeNet dataset demonstrate the effectiveness of the architecture in achieving representation quality.

ЗМІСТ

# ВСТУП

Відновлення тривимірних об'єктів з фотографії - захоплююча і складна проблема в області комп'ютерного зору. Він передбачає реконструкцію тривимірної моделі об’єкта або сцени з однієї чи кількох фотографій. Ця сфера досліджень привернула значну увагу завдяки широкому спектру застосувань у різних сферах, включаючи збереження культурної спадщини, віртуальну реальність, археологію та дизайн продуктів.

Здатність відновлювати тривимірну структуру об’єктів із фотографії має численні практичні наслідки. Наприклад, у сфері збереження культурної спадщини це дозволяє реставрувати та зберігати цінні артефакти, які можуть бути пошкоджені або втрачені з часом. Завдяки цифровій реконструкції цих об’єктів ми можемо захопити їхні складні деталі та забезпечити їх збереження для майбутніх поколінь.

У сфері віртуальної реальності реставрація тривимірних об’єктів із фотографії відіграє життєво важливу роль у створенні захоплюючого та реалістичного віртуального середовища. За допомогою точної реконструкції об’єктів і сцен можна покращити досвід віртуальної реальності, забезпечуючи користувачам більш захоплюючий та інтерактивний досвід.

Реставрація тривимірних об'єктів по фотографії знаходить застосування і в археології. Археологи часто мають доступ до фотографій історичних місць або артефактів, але їм бракує тривимірної інформації, необхідної для комплексного аналізу. Використовуючи передові технології комп’ютерного зору, ці фотографії можна перетворити на тривимірні моделі, що дозволяє археологам вивчати та аналізувати їх більш детально.

Крім того, такі галузі, як дизайн і виробництво продуктів, можуть отримати вигоду від реставрації тривимірних об’єктів за фотографіями. Це дозволяє дизайнерам створювати точні 3D-моделі фізичних об’єктів, полегшуючи процеси

проектування та створення прототипів. Це, у свою чергу, може призвести до більш ефективного виробництва та покращення якості продукції.

За останні роки було досягнуто значних успіхів у галузі комп’ютерного зору, що відкриває шлях до більш точних і надійних методів відновлення тривимірних об’єктів із фотографії. Ці досягнення включають розробку складних алгоритмів, покращення можливостей обробки зображень і доступність потужних обчислювальних ресурсів.

Підсумовуючи, відновлення тривимірних об’єктів із фотографії є переконливою сферою досліджень із значним наслідком у різних галузях. Використовуючи методи комп’ютерного бачення, ми можемо реконструювати тривимірні моделі з фотографій, що дозволяє нам зберігати культурну спадщину, покращувати досвід віртуальної реальності, допомагати археологічним дослідженням і підтримувати дизайн і виробничі процеси продукту.

# 1 СИСТЕМИ ВІДТВОРЕННЯ ТРИВИМІРНИХ МОДЕЛЕЙ

## 1.1 Огляд предметної області

Завдання 3D-реконструкції на основі зображень полягає у визначенні тривимірної геометрії та структури об'єктів або сцен за допомогою одного або кількох двовимірних зображень. Це важлива задача, яка має широкі застосування, такі як навігація роботів, розпізнавання об'єктів і розуміння сцени, 3D моделювання і анімація, промисловий контроль і медична діагностика.

Одним з основних напрямків досліджень було відновлення третього виміру за допомогою лише двовимірних зображень. Класичні методи, такі як багаторакурсне стерео і методи "shape-from-X", були присвячені цій проблемі протягом десятиліть. Перше покоління таких методів зосереджувалося на геометричних аспектах, намагаючись розробити математичні або алгоритмічні рішення для вирішення оберненої проблеми перетворення 3D в 2D.

Ефективні підходи зазвичай вимагають використання декількох зображень, знятих за допомогою добре налаштованих камер. Наприклад, стерео-методи базуються на порівнянні ознак між зображеннями, знятими з різних кутів, та використанні принципу триангуляції для відновлення тривимірних координат пікселів зображення. Методи "shape-from-silhouette" або "shape-by-space-carving" вимагають точної сегментації двовимірних силуетів.

Цікаво, що люди мають вміння ефективно вирішувати такі обернені задачі, використовуючи свої попередні знання. Вони можуть приблизно визначити розмір і геометрію об'єктів, користуючись лише одним оком. Навіть можуть уявити, як виглядатимуть об'єкти з іншої точки зору. Ми здатні до цього, оскільки маємо попередні знання та ментальні моделі того, як об'єкти зазвичай виглядають, засновані на тому, що ми бачили раніше.

Нові методи тривимірної реконструкції, що розвиваються, намагаються використовувати ці попередні знання, формулюючи проблему як завдання розпізнавання. Завдяки розвитку глибокого навчання та збільшенню доступності

великих наборів навчальних даних, з'явилося нове покоління методів, які здатні відновлювати тривимірну геометрію та структуру об'єктів з одного або кількох RGB-зображень, без необхідності в складному процесі калібрування камери. Незважаючи на те, що ці методи є новими, вони вже продемонстрували захоплюючі та обіцяючі результати в різних задачах комп'ютерного зору і комп'ютерної графіки.

Тому однією з перспектив розвитку в цій області є використання новітніх технологій, таких як глибоке навчання, машинне навчання та штучний інтелект, для поліпшення точності та швидкості процесу реконструкції. Також можуть розвиватись методи використання додаткової інформації, такої як додаткові датчики або використання контексту зйомки для поліпшення реконструкції.

Додатковими перспективами розвитку є використання реконструкції тривимірних об'єктів за фотографіями в різних доменах, таких як медицина (для створення тривимірних моделей органів або пухлин для планування лікування), археологія (для реконструкції античних споруд), культурна спадщина (для реставрації та збереження цінних артефактів) та інші.

Окрім того, розвиток предметної області реконструкції тривимірних об'єктів за фотографіями також може включати вдосконалення алгоритмів для реконструкції складних об'єктів, таких як людське тіло, транспортні засоби або будівлі. Важливим викликом є розробка методів, які б дозволяли реконструювати об'єкти в режимі реального часу, що має великий потенціал для використання у віртуальній реальності, аугментованій реальності та інших застосунках, де важлива взаємодія з тривимірними об'єктами у реальному часі.

## 1.2 Проблеми методів тривимірної реконструкції

На сьогодні є два фундаментально різні підходи до задачі тривимірної реконструкції: методи, що використовують глибоке навчання з вивченням

попередніх зназнь про структруку об’єкта, та класичні методи, що зазвичай використовують декілька зображень для реконструкції об’єкта, використовуючи триангуляцію та інші прийоми властиві фотограмметрії.

Як вже було згадано, у сфері 3D-реконструкції класичні методи широко використовуються протягом кількох десятиліть. Ці підходи спираються на традиційні методи комп’ютерного зору та алгоритми для реконструкції тривимірних моделей із двовимірних зображень. Однак ці класичні методи страждають від різноманітних обмежень і проблем, які перешкоджають їх ефективності та точності.

### 1.2.1 Обмежена точність і чутливість до шуму

Однією з ключових проблем, з якими стикаються класичні методи, є їх обмежена точність і чутливість до шуму. Оскільки ці методи часто базуються на виділенні ознак і зіставленні, вони значною мірою покладаються на якість функцій зображення. За наявності шуму, оклюзій або ділянок з низькою текстурою точність реконструйованих 3D-моделей має тенденцію до значного погіршення [1]. Крім того, класичні методи важко справляються зі складними умовами освітлення, що додатково впливає на точність реконструкції.

### 1.2.2 Залежність від ручного втручання

Класичні підходи до 3D-реконструкції часто вимагають ручного втручання на різних етапах процесу реконструкції. Наприклад, вибір точок функції, калібрування камери та налаштування параметрів зазвичай потребують досвіду та втручання людини. Ця залежність від ручного втручання робить процес тривалим, трудомістким і схильним до суб’єктивних помилок [2].

### 1.2.3 Обмежена стійкість до складності сцени

Класичні методи 3D-реконструкції стикаються з труднощами при роботі зі сценами, які виявляють високу складність, такими як сцени з повторюваними візерунками, безтекстурними областями або неоднозначними геометричними струдок, реконструйовані моделі можуть містити геометричні помилки або не мактурами. Ці проблеми виникають через обмежену здатність класичних методів отримувати та використовувати семантичну інформацію високого рівня з зображень. Як насліти деталей у складних областях [3].

## 1.3 Підходи на основі глибокого навчання

В останні роки підходи, засновані на глибокому навчанні, привернули значну увагу в області 3D реконструкції. Ці методи використовують потужність нейронних мереж, щоб вивчати складні взаємозв’язки між зображеннями та 3D-геометрією, забезпечуючи більш точні та детальні реконструкції. Однак вони не позбавлені власного набору проблем.

### 1.3.1 Залежність даних і анотовані дані навчання

Підходи, засновані на глибокому навчанні, вимагають великої кількості навчальних даних, щоб навчитися точним зображенням 3D-геометрії. Отримання анотованих навчальних даних для 3D-реконструкції є складним і трудомістким завданням. Це часто включає в себе ручне маркування базових істинних 3D-моделей, що вимагає досвіду в області та значних зусиль [4]. Крім того, якість і різноманітність навчальних даних значною мірою впливають на продуктивність моделей глибокого навчання, тому дуже важливо мати добре підібраний набір даних.

### 1.3.2 Узагальнення та адаптація до нових сцен

Методи, засновані на глибокому навчанні, мають тенденцію до труднощів з узагальненням до нових сцен або сценаріїв, які суттєво відрізняються від навчальних даних. Оскільки моделі навчаються на певному наборі даних, вони можуть мати обмежену адаптивність до нових середовищ, точок зору або категорій об’єктів. Цей недолік узагальнення може призвести до неоптимальних або неточних реконструкцій при застосуванні до сценаріїв реального світу [5].

### 1.3.3 Відсутність інтерпретації та пояснення

Хоча методи, засновані на глибокому навчанні, часто досягають вражаючої продуктивності з точки зору точності реконструкції, їм зазвичай бракує інтерпретації та пояснення. Природа «чорної скриньки» глибоких нейронних мереж ускладнює розуміння причин, що лежать в основі рішень моделі, а також виявлення та виправлення потенційних помилок або упереджень. Цей недолік

інтерпретації може обмежити практичне застосування та прийняття підходів, заснованих на глибокому навчанні, у критичних областях [6].

## 1.4 Висновки до розділу

На завершення розділ «Системи відтворення тривимірних моделей» містить огляд предметної області та обговорює проблеми, пов'язані з методами тривимірної реконструкції. Обмеження поточних підходів, включаючи обмежену точність і чутливість до шуму, обмежену стійкість до складності сцени, залежність від даних і потребу в анотованих навчальних даних, узагальнення та адаптацію до нових сцен, а також відсутність інтерпретації та пояснення, були підкреслені.

Дослідження в цій галузі засвідчили значний прогрес, зокрема у застосуванні методів глибокого навчання. Підходи, засновані на глибокому навчанні, пропонують багатообіцяючі рішення для підвищення точності та надійності тривимірної реконструкції. Проте все ще існують проблеми, такі як залежність від великої кількості анотованих навчальних даних, здатність узагальнювати та адаптуватися до нових сцен, а також потреба в інтерпретації та поясненні.

На завершення в розділі висвітлюється поточний стан систем відтворення тривимірних моделей і визначаються ключові проблеми, які необхідно вирішити. Вирішуючи ці проблеми шляхом подальших досліджень та інновацій, ми можемо підвищити точність, надійність і інтерпретацію методів тривимірної реконструкції, відкриваючи шлях для їх більш широкого застосування в різних сферах, таких як комп’ютерне бачення, робототехніка, віртуальна реальність і доповнена реальність.

# 2 МЕТОДИ ВІДНОВЛЕННЯ ТРИВИМІРНИХ МОДЕЛЕЙ

## 2.1 Огляд основних класичних методів для відновлення моделей

Класичні методи відновлення тривимірних моделей за фотографіями включають в себе підходи, які базуються на геометричних принципах та використовують методи стереозору, структуру з руху, триангуляцію та інші. Один з таких методів - Structure from Motion (SfM).

### 2.1.1 SFM

SfM включає в себе кілька математичних концепцій і методів для реконструкції тривимірної структури та руху об’єктів з кількох зображень. Давайте заглибимося в ключові математичні аспекти SfM:

Модель проекції камери.

Модель проекції камери описує зв’язок між 3D-точкою світу та її 2D-проекцією на площину зображення. Широко використовуваною моделлю є перспективна проекція, яку можна представити наступним рівнянням [7]:

$s * x = P * X$

Тут «x» представляє координати зображення 2D-точки, «X» позначає однорідні 3D-координати відповідної точки світу, «P» — матрицю проекції камери, а «s» — масштабний коефіцієнт.

Оцінка пози камери.

Перший крок у SfM передбачає оцінку параметрів камери, які включають внутрішні параметри (такі як фокусна відстань і головна точка) і зовнішні параметри (обертання та трансляція), які описують положення та орієнтацію

камери. Поширені методи оцінки пози камери включають методи на основі функцій, як-от алгоритм Perspective-n-Point (PnP), який оцінює пози камери на основі відповідності між 3D-точками та їхніми 2D-проекціями [8].

Відповідність функцій і тріангуляція.

Щоб реконструювати 3D-структуру, SfM покладається на пошук відповідностей між точками зображення в кількох переглядах. Ці відповідності можна встановити за допомогою методів виділення ознак, зазвичай методом SiFT [17], і зіставлення. Після отримання відповідностей виконується тріангуляція для оцінки 3D-координат кожної точки. Процес тріангуляції включає розв’язання проблеми перетину, де 3D-точку знаходять шляхом мінімізації помилки повторного проектування між спостережуваними точками 2D-зображення та їхніми відповідними 3D-проекціями [7].

Коригування.

Після отримання початкової оцінки поз камери та 3D-структури, для уточнення реконструкції використовується коригування пучка. Налаштування пакету — це проблема оптимізації, яка спільно покращує параметри камери та 3D-точки шляхом мінімізації помилки повторного проектування. Він ітеративно коригує параметри, щоб мінімізувати розбіжності між спостережуваними точками зображення та їхніми проекціями з оціненої 3D-структури [8].

Підводячи підсумок, SfM використовує моделі проекції камери, оцінку пози камери, відповідність характеристик, тріангуляцію та коригування пакетів для реконструкції тривимірної структури та руху об’єктів із кількох зображень. Постійно вдосконалюючи параметри камери та 3D-структуру, SfM прагне досягти точного представлення сцени. Результат роботи SfM можна побачити на рисунку 2.1.

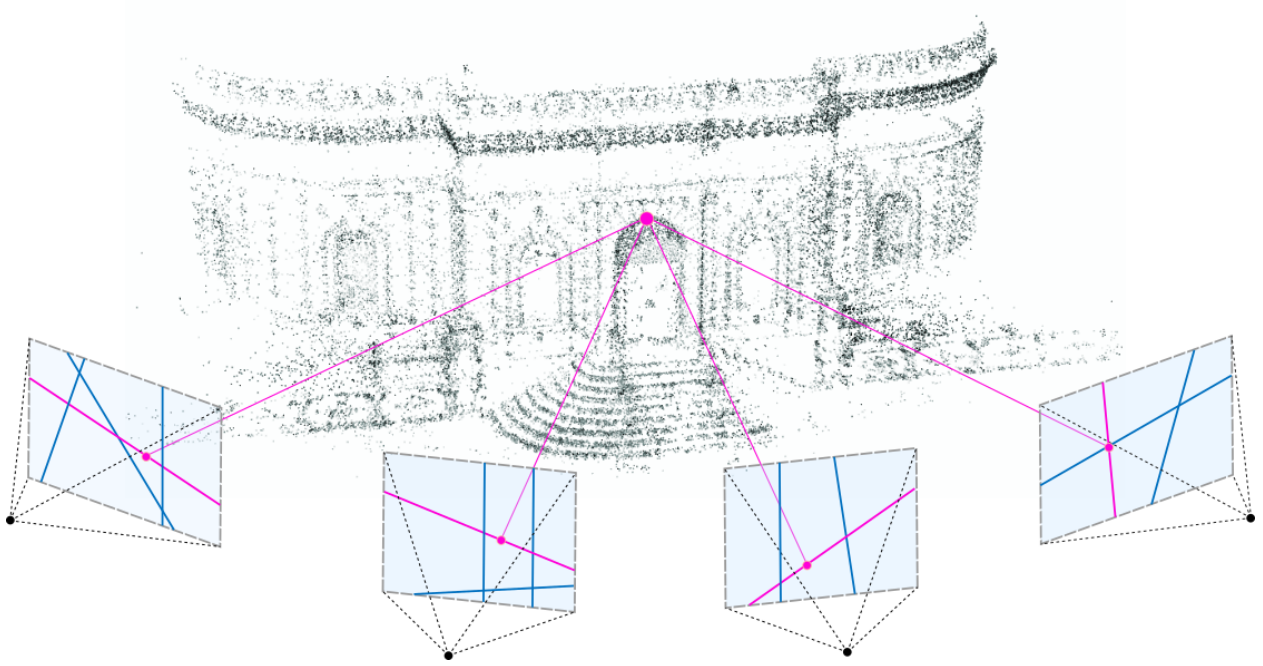


Рис. 2.1 Приклад роботи методу SfM

### 2.2.2 Stereo Reconstruction

Інший метод - Stereo Reconstruction, Stereo Reconstruction є одним з класичних методів відновлення тривимірних моделей за зображеннями, який використовує дві або більше фотографії з різних ракурсів. Метод заснований на принципі стереозору, тобто на сприйнятті глибини об'єктів за допомогою обробки зображень, знятих з двох різних точок. Процес реконструкції, рисунок 2.2, складається з кількох етапів, кожен з яких сприяє поступовому відновленню повної тривимірної моделі:

Калібрування камери.

На початку виконується калібрування камери, щоб визначити внутрішні параметри кожної камери, такі як фокусна відстань і положення оптичного центру. Цей процес калібрування забезпечує точні вимірювання та забезпечує точне узгодження відповідності на наступних етапах [2].

Зіставлення відповідності.

Наступний важливий крок передбачає встановлення відповідності між точками на зображеннях. Для кожної точки на одному зображенні необхідно знайти відповідник на іншому зображенні. Для виконання цього завдання використовуються різні алгоритми зіставлення зображень, такі як кореляційний аналіз або алгоритми оптичного потоку [3].

Розрахунок невідповідності.

Після встановлення відповідності обчислюється невідповідність між відповідними точками на двох зображеннях. Розбіжність являє собою різницю в координатах, надаючи цінну інформацію про просторове розташування об'єкта в тривимірному просторі [4].

Реконструкція тривимірної моделі.

Озброївшись обчисленими диспропорціями, тривимірну модель об’єкта можна реконструювати за допомогою геометричних принципів. Традиційні методи реконструкції, такі як тріангуляція, можуть бути використані для реконструкції геометрії об'єкта [4].

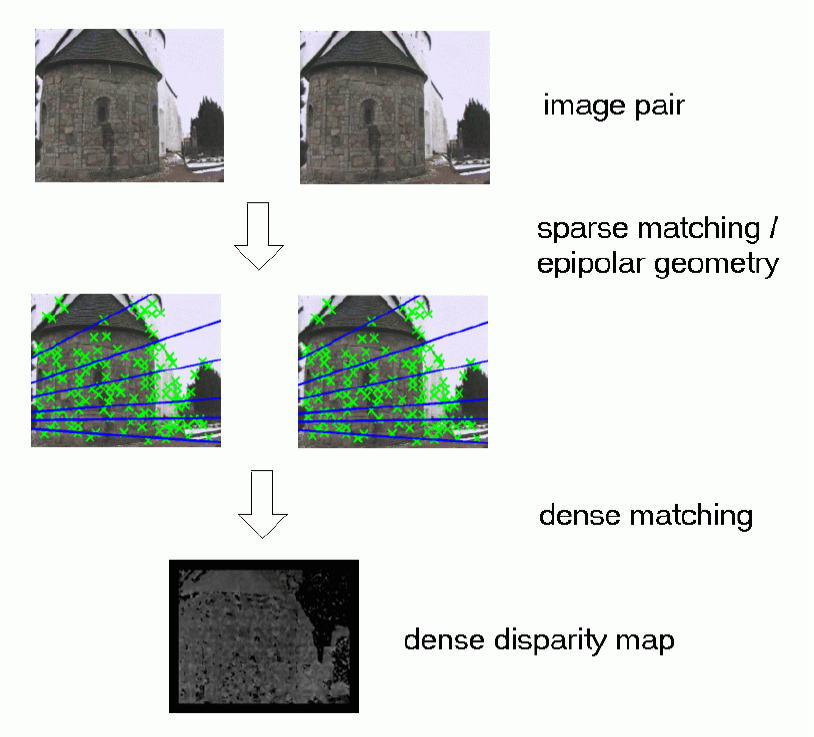



Рис. 2.2 Принцип роботи стереореконструкції.

Ключова відмінність між стереореконструкцією та структурою з руху (SfM) полягає в кількості зображень, які використовуються для реконструкції, і основних принципах, які вони використовують.

По-перше, кількість зображень: стереореконструкція зазвичай використовує два зображення (стереопару), зняті з різних точок зору. Ці зображення робляться одночасно або майже одночасно за допомогою стереокамер або пари звичайних камер [9]. На відміну від цього, SfM вимагає двох або більше зображень, знятих з різних точок зору, що дозволяє реконструювати тривимірну структуру та рух [8].

Іншою відмінністю є принцип реконструкції: стереореконструкція базується на принципі стереобачення, яке використовує невідповідність між відповідними точками в стереопарі зображень для отримання глибинної інформації та реконструкції тривимірної сцени [9]. В першу чергу він зосереджений на визначенні глибини статичної сцени за допомогою двох зображень. SfM, з іншого боку, охоплює ширший обсяг і спрямований на реконструкцію як тривимірної структури, так і руху об’єктів. Він використовує інформацію про рухи камери та відносне розташування об’єктів на кількох зображеннях для реконструкції

детальної тривимірної сцени. SfM використовує такі методи, як оцінка пози камери, відповідність функцій і коригування пакетів для відновлення параметрів камери, 3D-структури та руху з кількох зображень [8].

## 2.2 Відновлення моделей з використанням нейронних мереж

При використанні нейронних мереж для моделювання дані можуть бути представлені різними способами, що призводить до різних підходів у процесі моделювання. Також при використанні нейроних мереж потрібен набір даних, наразі більшість методів, які використовують нейронні мережі, були натреновані на датасеті ShapeNet[26]. Тому розглянуті методи будуть оцінюватись на цих даних. Щодо представлень, то серед найбільш відомих і широко використовуваних представлень — воксельне представлення, точкове представлення, сіткове представлення та представлення неявною функцією.

Ці представлення служать основою для моделювання тривимірних об’єктів і сцен за допомогою нейронних мереж. Розглянемо кожне представлення та відповідні до нього методи детальніше:

### 2.2.1 Воксельне представлення (Voxel Representation)

У воксельному представленні тривимірний простір дискретизується в сітчасту структуру, відому як воксельна сітка. Кожен воксель у сітці представляє невеликий елемент об’єму, і йому можна призначити значення, яке вказує на наявність чи відсутність об’єкта чи певних властивостей у цьому об’ємі .

Існує два поширених типи представлення вокселів. Двійкове представлення вокселів: цей підхід позначає кожен воксель як присутній (1) або як відсутній (0), представляючи об’єкти як двійкові маски або об’ємні булеві масиви, як на рисунку

2.4. Він забезпечує простий і зрозумілий спосіб представлення присутності об’єкта в сітці вокселів. «Воксельна сітка» є популярним методом у воксельному представленні. Він розділяє тривимірний простір на правильну сітку вокселів, що дозволяє легко візуалізувати об’єкти всередині воксельної сітки. Цей підхід широко використовувався в ранніх методах реконструкції або створення тривимірних моделей через його простоту та візуальну інтерпретацію. Різницю між воксельним представленням та класичним – меш-представленням можна побачити на рисунках 2.3-2.4.

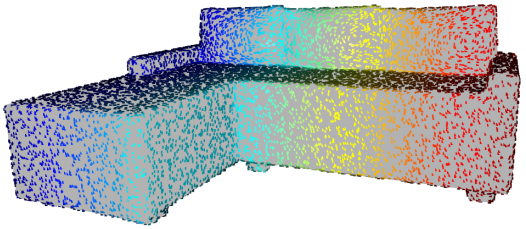


Рис. 2.3 Модель до вокселізації

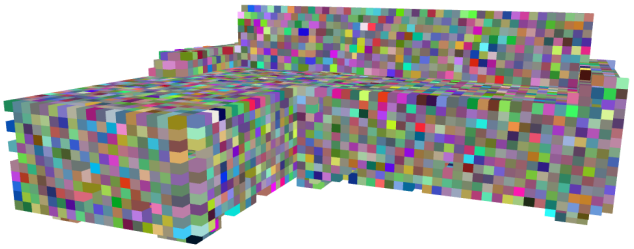


Рис. 2.4 Вокселізована модель у двійковому представленні

Числове представлення вокселів: у цьому варіанті кожному вокселю присвоюється числове значення, яке представляє різні атрибути об’єкта. Наприклад, значення яскравості або інтенсивності можуть фіксувати властивості світла, тоді як числові значення можуть кодувати такі фізичні характеристики, як щільність, прозорість або матеріали.

Однак воксельне представлення має свої обмеження.

1. Пам’ять і обчислювальні ресурси: об’єкти зі складною геометрією або складними деталями можуть потребувати великої кількості вокселів, що призводить до значного використання пам’яті та обчислювальних ресурсів. Це може створити проблеми під час роботи з великими обсягами даних або програмами реального часу.
2. Компроміс роздільної здатності: розмір вокселя безпосередньо впливає на роздільну здатність представлення. Великі вокселі можуть призвести до втрати дрібних деталей об’єкта, тоді як маленькі вокселі можуть внести значну кількість даних і ускладнити обчислення.
3. Неефективність розріджених даних: представлення вокселів може бути неефективним для розріджених об’єктів або сцен, де більшість вокселів залишаються порожніми. Ця надмірність у представленні порожнього простору може збільшити вимоги до зберігання даних і обчислювальне навантаження.

Внаслідок потреб в пам’яті методи побудовані з використанням цієї репрезентації даних є обмеженими в розмірі воксельної сітки, наприклад, роботи такі як [16,17] використовували сітку з $32^3$ вокселів, що зменшувало їх репрезентаційну потужністю, хоча деякі методм, наприклад [18], успішно збільшували розмір сітки до $128^3$ вокселів. Також при використанні такої репрезентації через велике використання пам’яті розмір батчу та нейронної мережі є дещо обмежені, а тому саме тренування триває довше.

### 2.2.2 Алгоритми для реконструкції у воксельній репрезентації

Одним з піонерських методів, для реконструкції тривимірних моделей у воксельній репрезентації є 3D-R2N2 [15] (Three-Dimensional Recurrent

Reconstruction Neural Network) — це модель глибокого навчання, яка вирішує проблему реконструкції тривимірної форми з одного двовимірного зображення.

Основною метою 3D-R2N2 є створення тривимірного воксельного представлення об’єкта на основі його двовимірного зображення. Модель складається з двох основних компонентів: 2D згорткової нейронної мережі (CNN), 3D рекурентної нейронної мережі (RNN).

2D CNN відповідає за обробку вхідного зображення та вилучення значущих ознак. Зазвичай він складається з кількох згорткових шарів, за якими слідують шари субдискретизації. Згорткові шари навчаються виявляти різні візуальні шаблони та фіксувати ієрархічні представлення вхідного зображення. Результатом 2D CNN є високорівневе представлення ознак, яке кодує візуальну інформацію.

3D RNN розроблено, щоб взяти ознаки високого рівня з 2D CNN і створити 3D воксельне представлення об’єкта. Він використовує мережу довгострокової короткочасної пам’яті (LSTM), типу RNN, яка може фіксувати довготривалі залежності в послідовних даних. Мережа LSTM розгортається по виміру глибини сітки вокселів і обробляє характеристики просторово. Вона передбачає ймовірність заповнення кожного вокселя, вказуючи, чи належить воксель об’єкту чи фону.

Під час навчання моделі 3D-R2N2 потрібен набір даних 2D-зображень у поєднанні з їхніми відповідними правдивими 3D-воксельними представленнями. Функція втрат, яка використовується для навчання моделі, як правило, є двійковою крос-ентропійною втратою рисунок 2.5, яка порівнює прогнозовану ймовірність зайнятості вокселя з базовими значеннями істинності.

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

Рис.2.5 Формула двійкової крос-ентропійної втрати

Більш новітній метод [16] використовує архітектуру автокодувальної нейронної мережі V3DOR - це метод, який використовує автокодер та варіаційний автокодер для генерації 3D-моделей об'єктів. Автоенкодер є двомодульним, складається з кодера і декодера рисунок 2.6. Енкодер використовується для вилучення ознак з вхідного зображення, а декодер генерує вихідні дані у вигляді 3D-моделі об'єкта.

Загальний опис мережі можна зробити наступним чином: кодувальник вивчає складні геометричні особливості об'єкта та створює інваріантне щодо розміру та кута приховане представлення. Для цього використовуються сім шарів згортки, а потім додається повністю пов'язаний рівень.

Декодер отримує це приховане представлення і створює 3D-структуру об'єкта з деталізацією. Декодер складається з п'яти рівнів тривимірного транспонування. Результатом роботи нейромережі є 3D-модель об'єкта у вокселізованому вигляді.

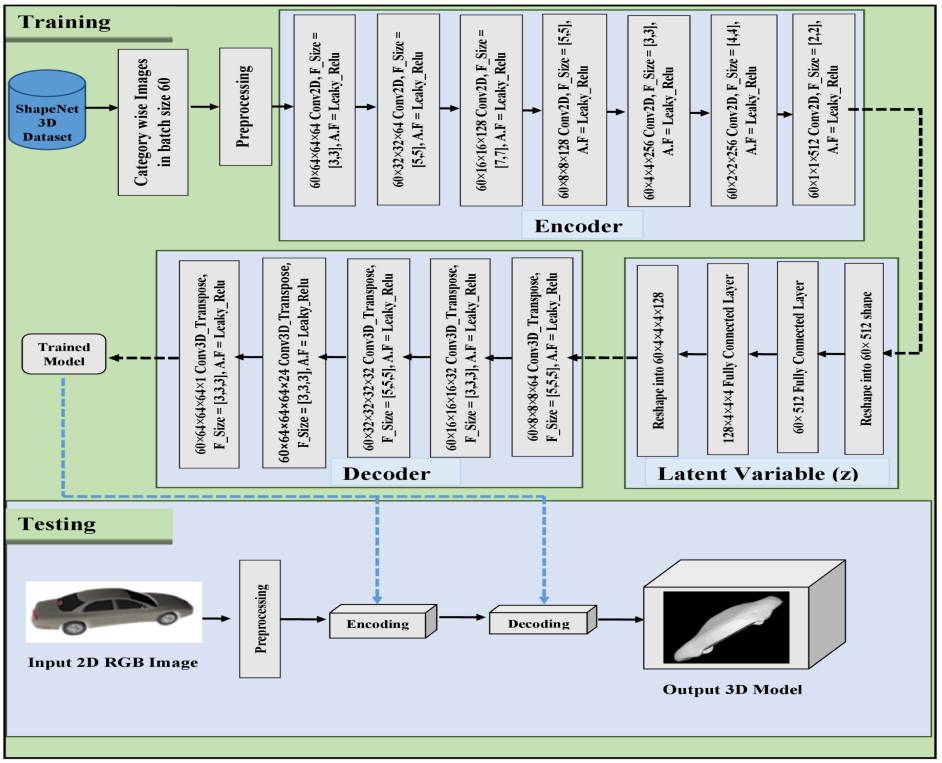



Рис.2.6 Архітектура V3DOR

В даному методі, навідміну від 3D-R2N2, автори обрали іншу функцію похибки, яку вони мінімізували, а саме Mean Squared False Cross-Entropy Loss (MSFEL), яка дозволяє досягнути кращих результатів на відміну від бінарної кросентропії.

MSFEL=FPCE+FNCE

$$FPCE = -\frac{1}{N}\sum_{n=1}^{N}[V_n log V_n' + (1 - V_n)\, log\,(1 - V_n')]$$

$$FNCE = -\frac{1}{P}\sum_{p=1}^{P}[V_p log V_p' + (1 - V_p)\, log\,(1 - V_p')]$$

Рис.2.7 Визначення FPCE і FNCE

де P – кількість зайнятих вокселів; Vn  це n-й незайнятий воксель, а N - загальна кількість незайнятих вокселів V, а Vp  p-й зайнятий воксель; V′n і V′p є передбаченням Vn  і Vp, відповідно.

### 2.2.3 Точкове представлення (Point Representation)

Точкове представлення (рис. 2.8) (Point Representation) є одним із способів представлення тривимірних об'єктів, в якому об'єкт розглядається як набір точок в тривимірному просторі, кожна з яких має свої координати та може мати додаткові атрибути, такі як нормалі, кольори, текстури тощо. Таке представлення використовується в комп'ютерній графіці, комп'ютерному зорі, робототехніці та інших областях.

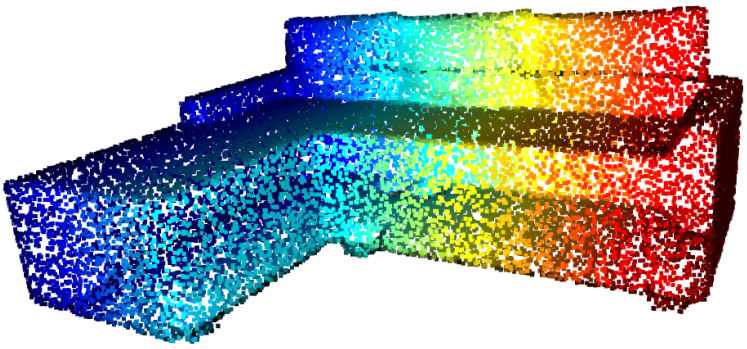


Рис. 2.8 Приклад точкового представлення

Одним з ключових переваг точкового представлення є його компактність. Оскільки представлення базується на точках, воно може бути ефективним з точки зору використання пам'яті та обчислювальних ресурсів. Воно також дозволяє представляти складні геометрії та деталізовані об'єкти, такі як хмари точок, що може бути важко або неможливо здійснити за допомогою воксельного представлення.

Ще однією перевагою точкового представлення є його незалежність від топології об'єкту. Точки можуть бути розташовані на поверхні об'єкта в будь-якому порядку, без необхідності визначення структури об’єкта або сітки, як це робиться в інших методах представлення, наприклад у воксельному представленні.

Однак, точкове представлення також має свої недоліки. Зокрема, воно може бути більш вимогливим до обробки, оскільки операції з точками можуть бути менш ефективними, ніж операції з вокселями. Також, точкове представлення може бути вразливим до шуму або неточностей в даних, оскільки точки можуть бути розташовані дискретно та бути чутливими до похибок у вимірюваннях.

### 2.2.4 Метод реконструкції PointOutNet для точкової репрезентації

Стаття [24] описує метод реконструкції тривимірних об'єктів з одного зображення за допомогою нейронних мереж.

Основна ідея методу полягає в тому, щоб згенерувати точковий набір з 1024 точок, який описує поверхню об'єкта, а потім використати цей набір для побудови тривимірної моделі об'єкта. Для досягнення цього було розроблено PointOutNet, яка використовується для генерації точок на поверхні об'єкта.

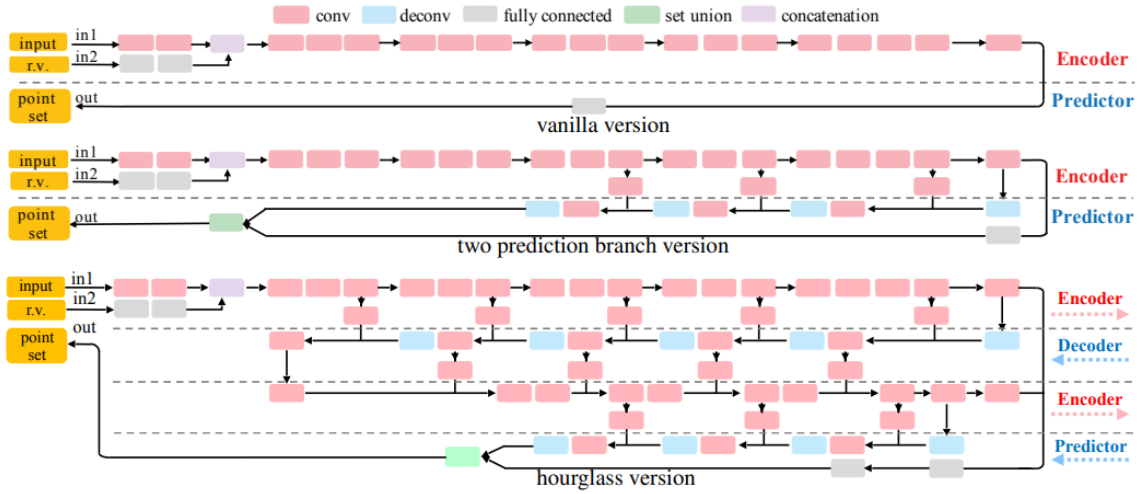



Рис. 2.9 Архітектура PointOutNet (PSGN)

Було розроблено 3 архітектури моделі (рис. 2.9), від простішої до складнішої, де найскладніша давала найкращі результати і використовувала багатошаровий підхід, як в , що дозволяло їй краще звертати увагу як на малі, так і на великі деталі зображення.

Архітектура мережі складається з етапу кодування та етапу прогнозування. Кодер приймає як вхідні дані пару, що складається із зображення (I) і випадкового вектора (r), і відображає їх у просторі векторів. Мета випадкового вектора полягає в тому, щоб внести варіативність у передбачення, дозволяючи мережі досліджувати можливі розподіли базових форм істинності.

Етап прогнозування відповідає за створення набору точок, що представляє форму 3D-об’єкта. Спочатку проста версія предиктора використовує повністю зв’язану мережу для виведення координат N точок. Однак цій версії не вистачає здатності фіксувати статистичні властивості та плавність природних геометричних структур.

Щоб усунути це обмеження, представлено вдосконалену версію предиктора. Він включає дві паралельні гілки: повністю зв’язану (fc) і деконволюційну (deconv). Гілка fc передбачає N1 точок, як і раніше, тоді як гілка deconv генерує 3-канальне зображення, що представляє координати додаткових точок. Потім ці передбачення з обох гілок об’єднуються, щоб сформувати повний набір точок у формі тривимірного об’єкта. Мережа також включає з’єднання пропуску для покращення потоку інформації між кодувальником і предиктором, сприяючи кращому навчанню представлення.

Крім того, стаття представляє версію мережі «пісочний годинник», натхненну роботою [10]. Ця глибока мережева архітектура періодично виконує операції кодування та декодування, що дозволяє мережі більш ефективно отримувати глобальну та локальну інформацію. Такий підхід підвищує потужність представлення мережі та покращує злиття різних рівнів інформації.

Підводячи підсумок, мережева архітектура в статті вирішує проблему реконструкції структур 3D-об’єктів з одного зображення. Включаючи випадковий вектор і використовуючи паралельні гілки на етапі прогнозування, мережа може генерувати різноманітні прогнози, вловлюючи притаманну невизначеність і неоднозначність завдання. Архітектура розроблена для використання статистичних властивостей геометричних даних і покращення потужності представлення складних структур.

Для тренування мережі було запропоновано дві наступні функції похибки: Chamfer distance та Earth Mover’s distance. Визначення chamfer distance можна побачити на рисунку 2.10.

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

Рис. 2.10 Визначення Chamfer distance

У строгому розумінні функція відстані d_CD не задовольняє нерівності трикутника, таким чином вона не є справжньою метрикою відстані. Однак для цілей цієї статті термін «відстань» використовується для позначення будь-якої невід'ємної функції, визначеної на парах множин точок. Алгоритм CD ефективно ідентифікує найближчого сусіда в альтернативному наборі для кожної точки та обчислює суму квадратів відстаней. З точки зору розташування точок в S1 і S2, CD можна розглядати як неперервну і кусково гладку функцію. Примітно, що пошук діапазону для кожної точки є незалежним, що забезпечує пряме розпаралелювання. Незважаючи на свою простоту, CD забезпечує задовільні та високоякісні результати в практичних застосуваннях. Визначення Earth Mover's distance можна побачити на рисунку 2.11.

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2$$

Рис. 2.11 Визначення Earth Mover's distance, тут ϕ - бієкція.

Відстань EMD є рішенням проблеми оптимізації, відомої як проблема призначення, яка є центром цього дослідження. Було помічено, що, за винятком нескінченно малої підмножини пар точкових множин, оптимальна бієкція φ однозначно визначається і залишається інваріантною навіть за найменших збурень точок. Отже, показник EMD можна диференціювати майже всюди, що робить його цінним інструментом для аналізу. Однак у практичних застосуваннях точне

обчислення EMD є непомірно дорогим, особливо в контексті глибокого навчання, навіть якщо використовується високопродуктивне графічне обладнання. Щоб вирішити цю проблему, була реалізована схема апроксимації (1 + ε), як описано в [12], яка пропонує обчислювально ефективну альтернативу без істотної шкоди для точності. Ця схема наближення служить практичним рішенням для включення EMD в різні додатки, включаючи ті, що включають методології глибокого навчання.

Завдяки іншій постановці задачі – прогнозування точок, які лежать на об’єктів, а не прогнозування належності вокселя до об’єкту, цей метод може отримувати кращі результати в порівнянні з 3D-R2N2 (рис. 2.12), але з іншого боку, результат прогнозу такої мережі не дає остаточного тривимірного об’єкту, а значить при його використанні потрібно використати пост-обробку, наприклад, [11] яка ймовірно зменшить точність моделі.

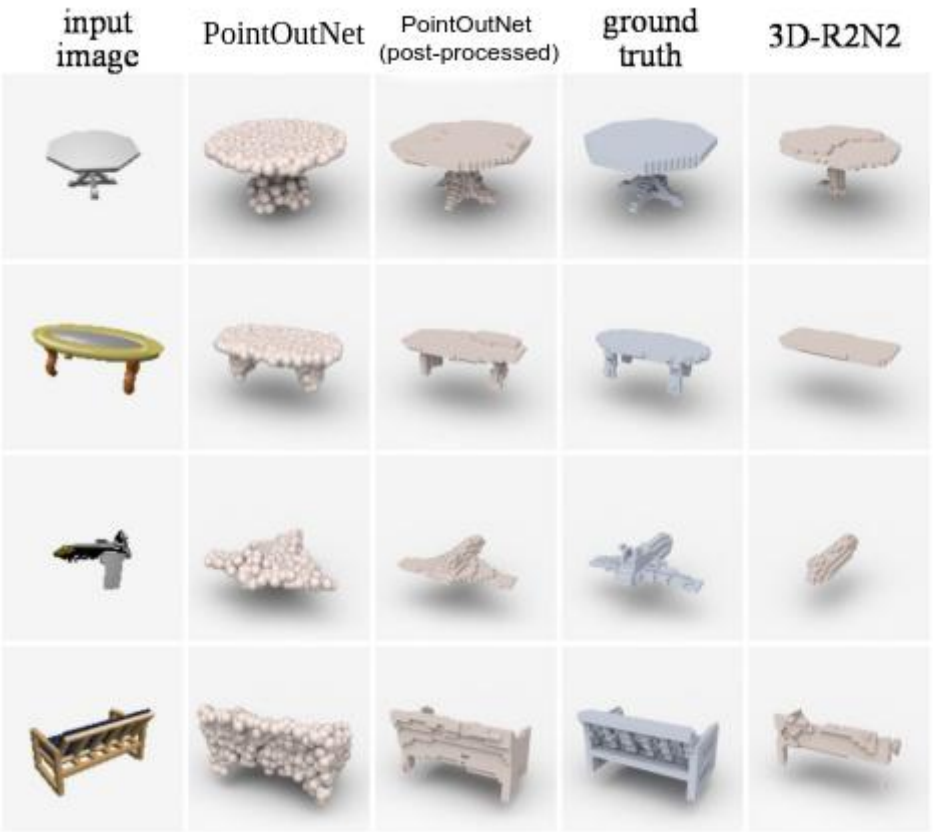



Рис. 2.12 Візуальне порівняння 3D-R2N2 та PointOutNet

### 2.2.5 Меш-представлення (Mesh Representation)

Меш-представлення (рис. 2.13) використовується для відображення 3D об'єктів у вигляді мережі полігонів (трикутників, чотирикутників або багатокутників), які утворюють поверхню об'єкта. Меш-представлення є одним з найпоширеніших способів візуалізації та обробки 3D об'єктів у комп'ютерній графіці, комп'ютерному зорі та інших додатках, таких як віртуальна реальність, комп'ютерне моделювання, медична візуалізація та багато інших.

Меш-представлення включає набір вершин, ребер та полігонів, які з'єднані між собою. Вершини є точками у 3D просторі, ребра визначають зв'язки між вершинами, а полігони утворюють поверхню об'єкта. Полігони можуть бути різних форм, таких як трикутники, чотирикутники, п'ятикутники, багатокутники та інші, в залежності від потреб конкретної задачі.

Нещодавно, меш-представлення стало розглядатися як вихідне представлення для відновлення 3D об'єктів [19,20,21]. Незважаючи на це, більшість таких підходів мають свої обмеження. Вони часто генерують самоперетинаючі меші, можуть створювати меші з простою топологією [21], вимагають наявності посилання на шаблон з того ж класу об'єкта [20], або не можуть гарантувати закритих поверхонь [19]. Liao та співавтори [22] запропонували власну версію алгоритму маршируючих кубів (marching cubes) [23], яка може навчатися виконувати реконструкцію спочатку і до кінця. Однак, їхній підхід все ще обмежений вимогами до оперативної пам'яті основної 3D сітки, тому він також обмежений розширенням до роздільної здатності 32x32x32 вокселів [22].

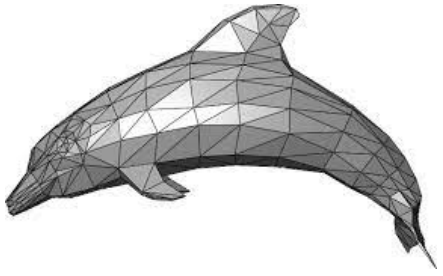


Рис. 2.13 Приклад меш-представлення

### 2.2.6 Метод реконструкції Pixel2Mesh для меш-представлення

Один з найпопулярніших методів для реконструкції був представлений в [21]. У статті пропонується наскрізна структура глибокого навчання для генерації 3D-моделей сітки з RGB зображень. Фреймворк складається з мережі ознак зображення та каскадної мережі деформації сітки (рис. 2.14).

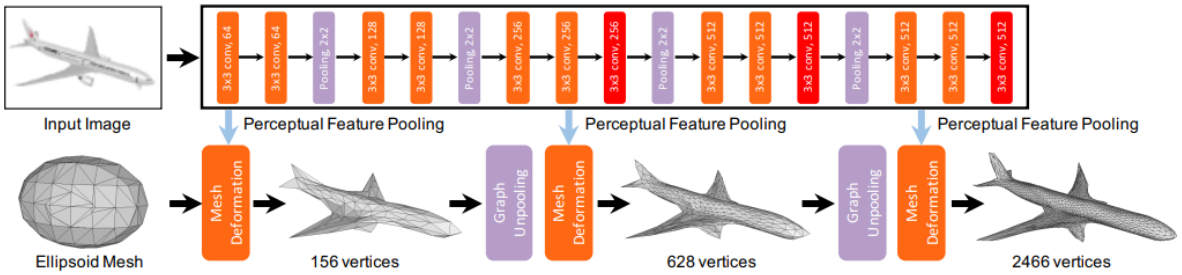


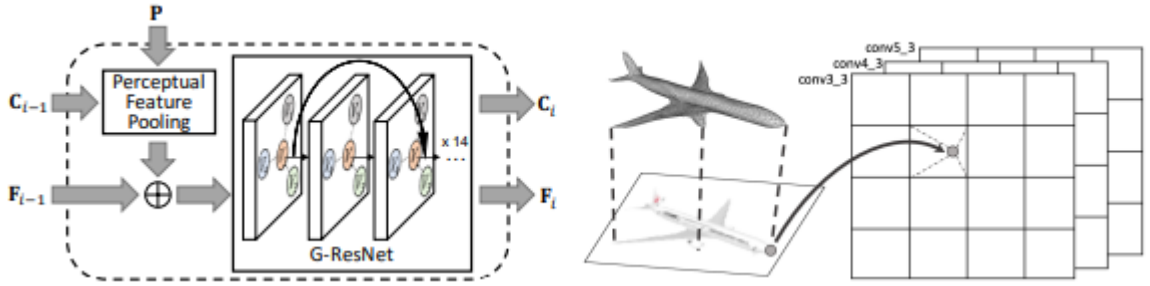

Рис. 2.14 Архітектура мережі Pixel2Mesh

Мережа ознак зображення — це двовимірна згортка нейронної мережі (CNN), яка витягує характеристики з вхідного зображення. Потім ці ознаки використовуються мережею деформації сітки для поступової деформації еліпсоїдної сітки в бажану 3D-модель.

Мережа деформації каскадної сітки базується на згортковій мережі на основі графів (GCN) і складається з трьох блоків деформації з двома шарами роз'єднання графів, що перетинаються між ними. Кожен блок деформації приймає вхідний граф, що представляє поточну модель сітки, і створює нові розташування вершин і функції. Шари роз'єднання графів збільшують кількість вершин для обробки деталей, зберігаючи топологію трикутної сітки.

Щоб навчити мережу, автори вводять кілька функцій втрат. Вони розширюють Chamfer Distance loss за допомогою трьох додаткових втрат, специфічних для сітки: втрату нормаль поверхні, регуляційна втрата Лапласа та втрат довжини краю. Ці втрати допомагають забезпечити стабільну деформацію та створити точні сітки.

Архітектура блоку деформації сітки включає рівень об’єднання ознак, який об’єднує ознаки вхідного зображення на основі 2D-проекції кожної вершини. Потім ці об’єднані перцептивні функції об’єднуються з тривимірними функціями форми, прикріпленими до вершин із вхідного графа. Об’єднані об’єкти вводяться в мережу ResNet (G-ResNet) на основі графів для створення нових координат і об’ємних об’єктів форми для кожної вершини.




(a) Блок деформації сітки (b) Об'єднання характеристик сприйняття

Рис. 2.15 Блок деформації сітки та блок об'єднання характеристик

Блок деформації сітки, як показано на малюнку 2.15 (a), відіграє вирішальну роль у створенні узгодженої тривимірної моделі сітки, яка вирівнюється з об’єктом у вхідному зображенні. Щоб досягти цього, блок деформації використовує мережу ознак зображення та рівень пулу перцептивних ознак, враховуючи розташування вершини (Ci-1) у поточній моделі сітки. Рівень об’єднання збирає відповідні функції (P) із вхідного зображення та об’єднує їх із об’єктом тривимірної форми (Fi-1), прикріпленим до вершини з вхідного графа. Потім ці об’єднані функції подаються в серію шарів ResNet (G-ResNet) на основі графів. G-ResNet виводить нові координати (Ci) і 3D-форму (Fi) для кожної вершини, слугуючи кінцевим результатом блоку деформації сітки.

G-ResNet — це глибока мережа з короткими з’єднаннями, призначена для ефективного обміну інформацією між вершинами. Вона складається з 14 графових залишкових згорткових шарів і створює нову 128-вимірну 3D функцію. Крім того, є гілка, яка застосовує додатковий згортковий шар графа до об’єктів останнього шару та виводить тривимірні координати вершини.

Щоб збільшити кількість вершин у згортковій нейронній мережі на основі графа, у статті введено шар роз'єднання графів. Шар роз'єднання додає нові вершини, з'єднуючи їх або в центрі кожного трикутника (на основі граней), або в центрі кожного ребра (на основі ребер). Тривимірні функції для щойно доданих вершин визначаються на основі середнього значення їхніх сусідніх вершин.

Також визначено чотири функції втрат для обмеження вихідної форми та процедури деформації: втрати фаски, нормальні втрати, регуляризація Лапласа та регуляризація довжини краю. Ці втрати застосовуються з рівною вагою як до проміжної, так і до кінцевої сітки, щоб забезпечити бажані результати.

Загалом запропонована структура поєднує в собі 2D CNN для виділення ознак зображення, каскадну мережу деформації сітки з використанням GCN і різні функції втрат для створення 3D-моделей сітки з RGB зображень.

### 2.2.7 Представлення за допомогою неявної функції (Implicit Function Representation)

В цьому методі об'єкт описується математичною функцією, яка визначає наявність або відсутність об'єкта в кожній точці 3D простору. Функція може приймати будь-яке значення в певному діапазоні, наприклад, від -1 до 1, де значення близьке до 1 вказує на наявність об'єкта, а значення близьке до -1 вказує на відсутність об'єкта, і значення функції на відстані між цими значеннями відображає неоднозначність або невизначеність наявності об'єкта.

Одним з варіантів неявних функцій є мережі глибокого навчання, такі як варіації глибоких нейронних мереж (Variational Autoencoders, VAEs) або генеративно-суперечливі мережі (Generative Adversarial Networks, GANs). У цих підходах, неявна функція може бути навчена шляхом тренування мережі на великому наборі даних 3D моделей. Після тренування, мережа може генерувати

значення функції для будь-якої точки в 3D просторі, дозволяючи відтворити поверхню об'єкта з використанням цієї функції.

Перевагою представлення за допомогою неявних функцій полягає в тому, що воно може бути більш гнучким, ніж інші представлення, такі як воксельне або меш-представлення, оскільки не вимагає фіксованого розділення або дискретизації простору. Крім того, воно може бути більш компактим, оскільки не вимагає зберігання великої кількості точок або мешів. Оскільки неявна функція описує поверхню об'єкта аналітично, вона може бути більш точною та забезпечувати більш високу якість реконструкції.

Однак, використання неявних функцій також має свої виклики. Наприклад, обчислення значення неявної функції може бути обчасно складним, особливо для великих моделей, оскільки вимагає оцінки функції в кожній точці 3D простору. Крім того, навчання неявної функції може бути вимогливим за ресурсами, оскільки вимагає великого набору даних та обчислювальних ресурсів для тренування складних моделей.

### 2.2.8 Метод реконструкції Occupancy Networks для представлення за допомогою неявних функцій

В методі Occupancy Networks [25], неявна функція використовується для представлення поверхні 3D об'єкта, де значення функції вказує на наявність об'єкта в кожній точці простору. Це дозволяє моделі генерувати реалістичні 3D моделі об'єктів з високою якістю поверхні, уникнувши проблем, таких як самоперетинання мешів або обмеження на топологію мешів. Використання неявних функцій також дозволяє моделі генерувати об'єкти зі складною геометрією та забезпечує більшу гнучкість у відтворенні різних форм об'єктів.

Ядром моделі є глибока нейронна мережа, яка приймає як вхідні 3D точку та зображення, після чого прогнозує ймовірність того, що точка належить об’єкту.

Архітектура мережі складається з кількох повністю пов’язаних шарів, за якими зазвичай йдуть нелінійні функції активації ReLU (Rectified Linear Unit). Ці рівні допомагають мережі вивчати складні моделі та залежності у вхідних даних.

Автори також вводять пропускні з’єднання [14] в архітектурі моделі, щоб полегшити потік інформації в різних масштабах. Ці зв’язки дозволяють мережі поєднувати локальні функції низького рівня з глобальними функціями високого рівня, покращуючи загальну якість реконструкції.

Під час навчання модель оптимізується за допомогою функції втрат, яка вимірює розбіжність між прогнозованою ймовірністю зайнятості та значеннями фактичної зайнятості. Автори використовують двійкову ентропію, щоб заохотити мережу точно передбачити, чи зайняте кожне місце в 3D-просторі чи ні.

Реконструкція.

Першим кроком у отриманні тривимірного об’єкту є виділення поверхні реконструйованої форми з ймовірностей заповнення. Це робиться шляхом визначення місць, де ймовірність зайнятості перетинає попередньо визначений поріг, який зазвичай становить 0.5, місця в свою чергу є сіткою з точок в тривимірному просторі, яка може бути будь-якого розміру 32x32x32, чи наприклад, 128x128x128. Після прогнозування зайнятості для точок використовується алгоритм маршируючих кубів [13], який створює елементи трикутної сітки на основі визначених місць розташування поверхні. Він перебирає зайняті та незайняті вокселі в 3D-сітці та створює трикутники, які приблизно відповідають формі поверхні. Отримана сітка складається з набору вершин і трикутників, які представляють реконструйовану форму.

Щоб удосконалити якість сітки, у статті пропонується застосувати методи згладжування сітки, такі як згладжування Лапласа або згладжування Таубіна. Ці методи регулюють положення вершин сітки на основі локальної геометрії, що призводить до більш візуально приємної та геометрично точної сітки.

На додаток до основного процесу генерації сітки, у документі також представлено розширення під назвою «Топологічна оптимізація форми» для

покращення якості кінцевої сітки. Це розширення використовує метод оптимізації графічного розрізу для застосування топологічних обмежень до реконструйованої форми, таких як зв’язність і рід. Включаючи топологічну інформацію, метод створює сітки, які більш узгоджуються з основною структурою форми.

### 2.2.9 Оцінка якості реконструкції

У попередніх розділах описано процес відновлення тривимірних (3D) об’єктів із фотографій за допомогою нейронних мереж. Однак успіх будь-якого методу реконструкції в кінцевому рахунку залежить від якості реконструйованого результату. У цьому розділі зосереджено увагу на оцінці ефективності та точності процесу реконструкції, досягнутого за допомогою нейронних мереж. Використовуючи різні метрики та методології оцінки, ми можемо оцінити точність і надійність відновлених 3D-об’єктів. Процес оцінки відіграє вирішальну роль у перевірці ефективності алгоритму відновлення та надає цінну інформацію для подальшого вдосконалення.

Щоб об’єктивно оцінити якість реконструйованих 3D-об’єктів, необхідно визначити відповідні метрики оцінки. Двома найбільш часто використовуваними показниками для оцінки якості реконструкції, досягнутої нейронними мережами, є перетин через з’єднання (IoU) і Chamfler L1 distance. Ці показники надають кількісні показники подібності між реконструйованим 3D-об’єктом і базовою правдою, кожен з яких зосереджується на різних аспектах якості реконструкції.

Перетин через об’єднання (IoU) — це популярна метрика оцінки, яка використовується в задачах комп’ютерного зору, включаючи виявлення та сегментацію об’єктів. У контексті оцінки якості реконструкції для нейронних мереж IoU можна використовувати для вимірювання точності реконструйованого 3D-об’єкта шляхом порівняння його з базовою правдою. IoU обчислюється шляхом ділення об’єму перекриття між реконструйованим об’єктом і наземною

правдою на об’єм їх об’єднання. Більше значення IoU вказує на вищий рівень узгодженості між реконструйованим і наземним об’єктами істинності. Порівняння метрики IoU для згаданих методів на датасеті ShapeNet можна побачити в таблиці 2.1.

Таблиця 2.1 Порівняння метрики IoU для згаданих методів на датасеті ShapeNet

| IoU | | | | |
|---|---|---|---|---|
| category | 3D-R2N2 | PSGN | Pix2Mesh | Occupancy Network |
| airplane | 0.426 | - | 0.420 | **0.571** |
| bench | 0.373 | - | 0.323 | **0.485** |
| cabinaet | 0.667 | - | 0.664 | **0.733** |
| car | 0.661 | - | 0.552 | **0..737** |
| chair | 0.439 | - | 0.396 | **0.501** |
| display | 0.440 | - | **0.490** | 0.471 |
| lamp | 0.281 | - | 0.323 | **0.371** |
| loudspeaker | 0.611 | - | 0.599 | **0.647** |
| rifle | 0.375 | - | 0.402 | **0.474** |
| sofa | 0.626 | - | 0.613 | **0.680** |
| table | 0.420 | - | 0.395 | **0.506** |
| telephone | 0.611 | - | 0.661 | **0.720** |
| vessel | 0.482 | - | 0.397 | **0.530** |
| mean | 0.493 | - | 0.480 | **0.571** |

Відстань Chamfer-L1 є ще одним широко використовуваним показником оцінки якості реконструйованих 3D-об’єктів. Він вимірює середню евклідову відстань між кожною точкою на реконструйованому об’єкті та її найближчою точкою на земній правді, а також середню відстань між кожною точкою на

наземній правді та її найближчою точкою на реконструйованому об’єкті. Відстань Chamfer-L1 є сумою цих двох середніх значень, що забезпечує міру відмінності між реконструйованим і відшліфованим об’єктами правдивості. Менша відстань Chamfer-L1 вказує на вищий рівень подібності та точності реконструкції. Порівняння метрики Chamfer-L1 для згаданих методів на датасеті ShapeNet можна побачити в таблиці 2.2.

Таблиця 2.2 Порівняння метрики Chamfer-L1 для згаданих методів на датасеті ShapeNet

| Chamfer-L1 | | | | |
|---|---|---|---|---|
| category | 3D-R2N2 | PSGN | Pix2Mesh | Occupancy Network |
| airplane | 0.227 | **0.137** | 0.187 | 0.147 |
| bench | 0.194 | 0.181 | 0.201 | **0.155** |
| cabinaet | 0.217 | 0.215 | 0.196 | **0.167** |
| car | 0.213 | 0.169 | 0.180 | **0.159** |
| chair | 0.270 | 0.247 | 0.265 | **0.228** |
| display | 0.314 | 0.284 | **0.239** | 0.278 |
| lamp | 0.778 | 0.314 | **0.308** | 0.479 |
| loudspeaker | 0.318 | 0.316 | **0.285** | 0.300 |
| rifle | 0.183 | **0.134** | 0.164 | 0.141 |
| sofa | 0.229 | 0.224 | 0.212 | **0.194** |
| table | 0.239 | 0.222 | 0.218 | **0.189** |
| telephone | 0.195 | 0.161 | 0.149 | **0.140** |
| vessel | 0.238 | **0.188** | 0.212 | 0.218 |
| mean | 0.278 | **0.215** | 0.216 | **0.215** |

## 2.3 Висновки до розділу

На завершення в цій главі представлений вичерпний огляд різних методів 3D-реконструкції, починаючи від класичних методів і закінчуючи новітніми підходами на основі нейронних мереж. Обговорювані класичні методи, такі як структура з руху (SFM) і стереореконструкція, давно використовуються в цій галузі та продемонстрували свою ефективність у реконструкції 3D-моделей.

Однак із появою нейронних мереж з'явилися нові шляхи відновлення моделі. У цій главі досліджено різні схеми представлення, включаючи воксельне, точкове та сітчасте представлення, представлення за допомогою невизначених функцій, усі з яких використовують можливості нейронних мереж для покращення процесу реконструкції. Кожне представлення має свої сильні сторони та обмеження, пропонуючи унікальні способи захоплення та представлення геометрії об'єктів.

Серед цих представлень неявне представлення функції привернуло значну увагу завдяки своїм перевагам і низьким вимогам до пам'яті. Методи, засновані на неявних функціях, використовують потужність глибокого навчання для отримання інформації про тривимірну форму з розріджених або неповних вхідних даних. Вивчаючи базову неявну функцію, ці методи можуть генерувати детальні та високоякісні 3D-моделі.

На основі ретельної оцінки обговорюваних методів було вибрано представлення з використанням неявної функції для продовження подальших досліджень та експериментів. Це рішення в першу чергу обумовлено перевагами, які він пропонує, зокрема можливістю генерувати складні та деталізовані форми при мінімізації потреб у пам'яті. Очікується, що за допомогою методів, заснованих на неявних функціях, можна реконструювати більш точні та реалістичні 3D-моделі, розширюючи межі поля та сприяючи його розвитку.

Таким чином, цей розділ заклав основу для подальших досліджень шляхом вивчення різних методів 3D-реконструкції. Рішення продовжити представлення за

допомогою неявної функції відображає перспективний підхід, який охоплює можливості нейронних мереж і їхній потенціал революціонізувати сферу. Використовуючи переваги представлення неявних функцій, майбутня робота може бути спрямована на досягнення більш точних і реалістичних 3D реконструкцій, що в кінцевому підсумку сприятиме ширшому розвитку комп’ютерного зору та обчислювальної геометрії.

# 3 РОЗРОБЛЕНИЙ МЕТОД

У цьому розділі представлено розроблений метод відновлення тривимірних об’єктів із фотографії. Підхід використовує неявне представлення функції в поєднанні з підходом гіпермереж[27]. Використовуючи ці методи, підхід прагне подолати обмеження традиційних методів і досягти точних і детальних реконструкцій.

## 3.1 Гіпермережі

Гіпермережі (рис 3.1) використовують концепцію метанавчання для динамічного генерування ваг первинної мережі. Це дозволяє основній мережі адаптувати свої параметри до різних завдань або вхідних даних. Використовуючи гіпермережі, основна мережа отримує підвищену гнучкість і можливості узагальнення, оскільки вона може навчитися регулювати свої ваги на основі конкретних вимог кожного завдання чи екземпляра. Ця адаптивна природа гіпермереж робить їх особливо придатними для складних і різноманітних завдань, таких як реконструкція тривимірних об’єктів із фотографії, де форми та характеристики об’єктів можуть значно відрізнятися.

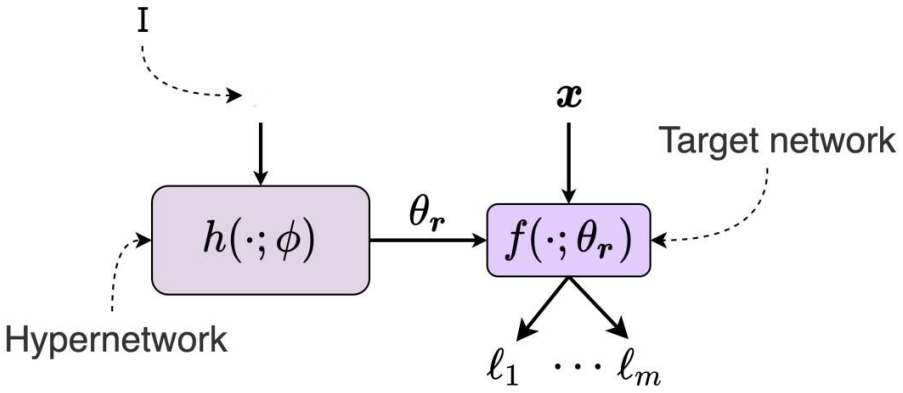



Рис. 3.1 Концепт гіпермереж.

В нашому випадку, буде дві мережі – перша мережа h, яка приймає на вхід зображення I, і прогнозує ваги $\theta$ для мережі f – цільової мережі. Далі мережа f маючи ваги $\theta$, які були прогнозовані мережею h для зображення I, приймає на вхід вектор x, який є тривимірним вектором, що репрезентує собою точку в просторі. Після чого мережа f робить прогноз, який може бути або відстанню точки до об’єкта, що тоді буде задачею регресії, чи належністю точки до об’єкта, що буде бінарною класифікацією на класи : точка всередині об’єкта, або точка ззовні об’єкта.

## 3.2 Попередні експерименти

В ході попередніх експериментів, було вирішено зробити спрощені експерименти, дослідити, яку архітектуру потрібно обрати для мережі f без використання мережі h. Для всіх експериментів, та подальшої роботи було обрано датасет ShapeNet, який попередньо був оброблений як в [25].

Без мережі h, ми фактично намагаємось навчити мережу f, так щоб вона розуміла форму об’єкта. Оскільки для інтерполяції точок зачасту використовують РБФ мережу, до того ж схожу задачу розв’язували в [28], за допомогою РБФ мережі, то вирішено, що основою для мережі f буде РБФ мережа з подальшими модифікаціями.

Задача, для якої було проведено попередні експерименти є спрощеною, але є суттєво важливою, оскільки зменшує кількість експериментів в подальшому з повною системою. Саму задачу можна описати наступним чином : маючи N тривимірних точок всередині обмежувальної рамки, в якій знаходиться об’єкт натренувати мережу так, щоб вона достатньо точно прогнозувала відстань від об’єкта до точки, тоді саму реконструкцію можна зробити обравши точки, які лежать на досить малій відстані від об’єкта і застосувавши [11]. При проведенні експериментів було вибрано 20 тривимірних моделей, для кожної з яких було

згенеровано 120 тисяч точок, з яких 75% це набір для тренування, а 25% набір для тетсу. Результуюча метрика є середньою метрикою по 20 моделям. Для тренування мережі використовувався оптимізатор Adam з кроком навчання 0.01, та кількість епох – 100.

При проведенні експерименту досліджувались наступні елементи:

1. Кількість повнозв'язних шарів нейронної мережі, кількість нейронів в них.
2. Вибір точок для навчання мережі, кількість цих точок.
3. Кількість РБФ центрів, РБФ функція.
4. Вибір функції втрат.
5. Тип регуляризації.

Архітектура моделі виглядає наступним чином, спочатку йде РБФ модуль, який приймає тривимірну точку і переводить її в простір більшої розмірності, який має розмірність як і кількість РБФ центрів. Центри в свою чергу вибрані так, щоб вони формували сітку з точок в обмежувальній рамці об'єкта. Далі йдуть від одного до 5 повнозв'язних шарів, між повнозв'язними шарами – fc шарами використовується функція активації mish [29]. Останній повнозв'язний шар завжди має тільки один вихідний нейрон, який використовується для прогнозування відстані.

Для оцінки якості було використано метрику Root Mean Squared Error (RMSE): $\sum_{i=1}^{n} \sqrt{(y_i - y_{i\,predicted})^2/n}$ , де $y_i$ - справжня відстань від точки до об'єкту, $y_{i\,predicted}$ – прогнозована відстань від точки до об'єкту, $n$ – кількість точок.

В таблиці 3.1 $fc_i$ - кількість нейронів на виході в і-того повнозв'язного шару, на вхід в $fc_i$ подається вихід з минулого шару $fc_{i-1}$ , або ж якщо це перший шар, то вихід з РБФ модулю.

Таблиця 3.1 Результати експериментів з вибором моделі

| Кількість РБФ центрів | РБФ функція | $fc_1$ | $fc_2$ | $fc_3$ | $fc_4$ | $fc_5$ | Test RMSE | Train RMSE |
|---|---|---|---|---|---|---|---|---|
| 3375 | Пуассона | 128 | 64 | 64 | 32 | 1 | 0.0101 | 0.0101 |
| 3375 | Гаусса | 128 | 64 | 64 | 32 | 1 | 0.0194 | 0.0196 |
| 1000 | Пуассона | 256 | 128 | 64 | 32 | 1 | 0.009 | 0.009 |
| 1000 | Пуассона | 128 | 64 | 64 | 32 | 1 | **0.0046** | **0.0046** |
| 1000 | Гаусса | 128 | 64 | 64 | 32 | 1 | 0.018 | 0.018 |
| 1000 | Гаусса | 256 | 128 | 64 | 32 | 1 | 0.0178 | 0.0176 |
| 125 | Гаусса | 256 | 128 | 64 | 32 | 1 | 0.0184 | 0.0184 |
| 125 | Пуассона | 128 | 64 | 64 | 32 | 1 | 0.0096 | 0.0097 |
| 1000 | Пуассона | 128 | 64 | 64 | 1 | - | 0.014 | 0.014 |
| 1000 | Пуассона | 512 | 256 | 64 | 1 | - | 0.0136 | 0.0137 |

Результати експерименту.

1. Максимальна кількість шарів, яка була в експериментах – 5. Збільшення кількості шарів призводить до позитивного покращення метрики, з кількістю нейронів так не завжди.
2. Щодо вибору точок, то було перевірено декілька методів вибору точок. Вибір частини точок на поверхні об’єкта та частини точок випадково всередині обмежувальної рамки показав кращі результати порівняно з випадковим вибором всередині обмежувальної рамки об’єкта. Щодо

кількості точок, то кількість точок в тренувальному наборі, до 20 тисяч показує сильно гірші результати, більше за 20-30 тисяч не так сильно впливає на результат моделі, а після приблизно 150 тисяч взагалі нічого не змінює.

3. Вид РБФ функції відіграє надзвичайно важливу роль в остаточному результаті, у всіх випадках фунція РБФ Пуасона, показувало значно кращі результати порівняно з функцією Гауса.
4. Функцію втрат, основною функцією втрат для задач регресії є Mean Squared Error (MSE) або Root Mean Squared Error(RMSE), саме тому як одну з основних було обрано MSE = $\sum_{i=1}^{n}(y_i - y_{i\,predicted})^2 /n$,, але була спроба модифікувати її так, щоб коли $|y_i - y_{i\,predicted}| < \varepsilon$, то $|y_i - y_{i\,predicted}| = 0$, де ε – деяка мала дозволена похибка, така функція втрат є досить схожою на MSFEL в методі V3DOR[16]. Таким чином, очікувалось, що мережа перестане звертати увагу на незначні похибки, яких досить багато, і перемістить увагу на ті точки, де похибка більша за ε, але в цьому випадку мережа повільно тренувалась, а якщо спробувати її тренувати швидше, то відбувався вибух градієнтів, внаслідок чого модель розбігалась.
5. Було виявлено, що регуляризація, а саме Dropout і L2 регуляризація негативно впливають на тренування, оскільки мережа не перенавчається, але при використанні регуляризації мережа повільніше збігається та не досягає тих же результатів навіть при більшій кількості епох.

Приклад отриманих точок найкращою моделлю в таблиці можна спостерігати на рис 3.2 та модель з отриманими точками на рис 3.3.

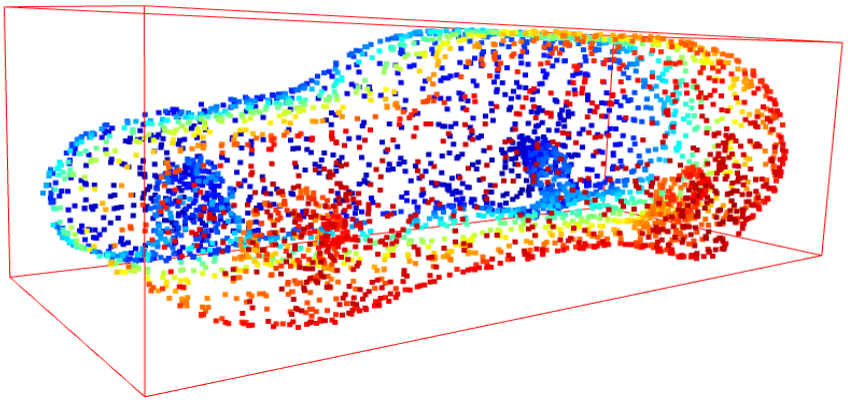


Рис. 3.2 Приклад точок прогнозованих моделлю

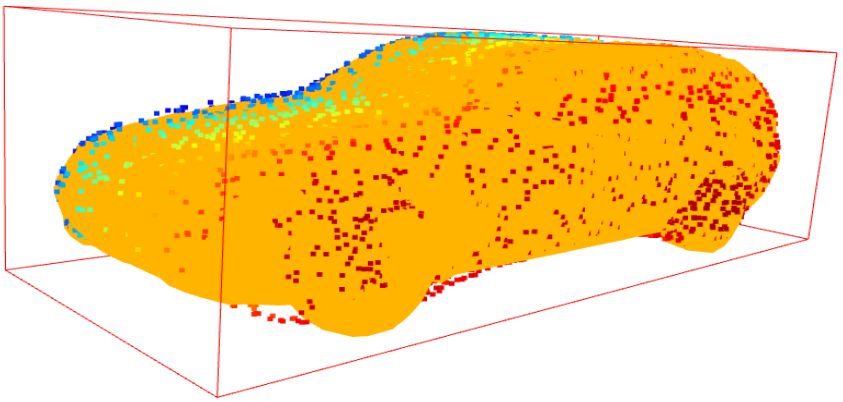


Рис. 3.3 Приклад тривимірної моделі з прогнозованими точками моделлю

## 3.3 Розроблена архітектура

В попередніх експериментах задача формулювалась як прогнозування відстані, але під час розробки повноцінної мережі як з мережею h, так і мережею f було виявлено, що така мережа занадто погано узагальнюється для випадку з прогнозуванням відстані, тому все ж таки було вирішено змінити підхід, до прогнозування належності точки об’єкту, сама архітектура моделі f не змінилась.

Запропонована нейронна мережа h складається з кількох компонентів, які працюють разом для виконання тривимірної реконструкції. Мережа приймає вхідне зображення розміром 224x224x3 і виконує серію операцій для вилучення ознак і генерування ваг для мережі f. Повну архітектуру моделі можна побачити на рис. 3.4.

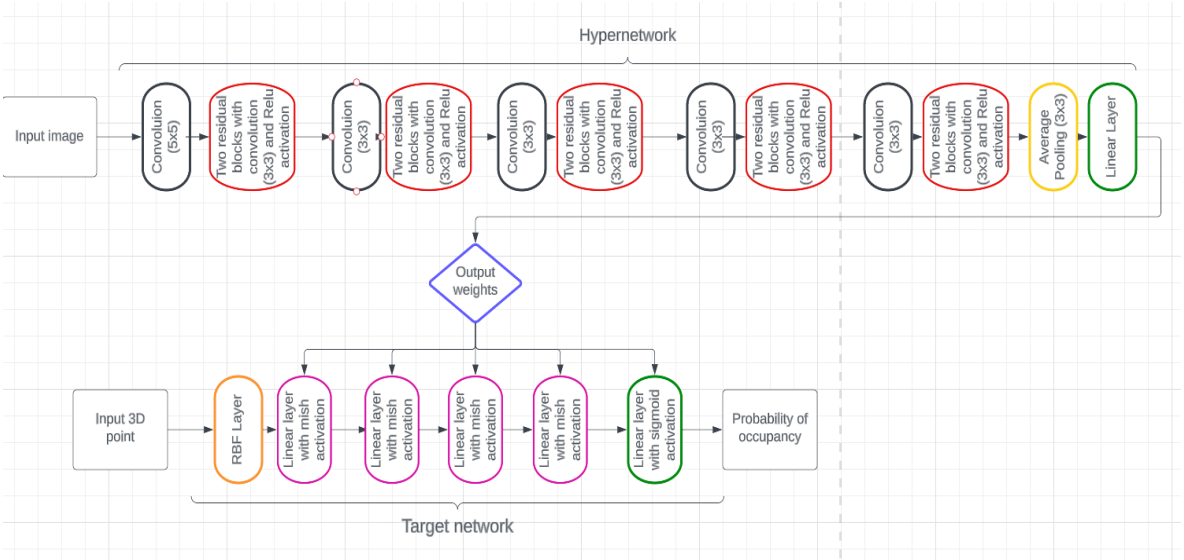



Рис. 3.4 Повна архітектура моделі

Початковий згортковий шар.

Мережа починається зі згорткового шару, який приймає вхідне зображення та застосовує операцію двовимірної згортки. Цей шар має три вхідні канали та генерує карти характеристик із вісьмома вихідними каналами.

Блоки 1-5.

Мережа включає п'ять блоків, кожен з яких містить згортковий шар, за яким слідує послідовність залишкових блоків. Ці блоки поступово зменшують дискретизацію карт функцій і збільшують кількість вихідних каналів для захоплення складніших моделей і функцій. Кількість вихідних каналів для кожного блоку різна, і були проведені експерименти для оцінки різних конфігурацій.

Об’єднання середніх.

Після блоку 5 застосовується шар усередненого об’єднання (mean pooling) для подальшого зниження дискретизації карт об’єктів і узагальнення інформації. Цей шар зменшує просторові розміри карт об’єктів, зберігаючи важливі об’єкти.

Зведення та лінійне перетворення.

Вихід шару об’єднання вирівнюється в одновимірний тензор. Потім цей сплощений тензор пропускається через лінійний шар, який виконує лінійне перетворення для створення вагових коефіцієнтів для мережі f. Кількість ваг, які генерує мережа h на цьому лінійному рівні залежить від бажаної кількості центрів RBF і додаткових компонентів, специфічних для мережі f.

Функціональність мережі.

Прямий перехід моделі гіпермережі включає в себе проходження вхідного зображення через визначені блоки та шари послідовним чином. Вхідне зображення проходить через початковий згортковий шар, а потім блоки 1-5. Потім карти об’єктів обробляються середнім шаром об’єднання. Отримані карти об'єктів зводяться в одновимірний тензор і перетворюються лінійним шаром для створення тензора ваг для мережі f.

Експериментальні конфігурації.

Щоб дослідити вплив різних конфігурацій вихідних каналів, було проведено експерименти, змінюючи кількість вихідних каналів для згорткових шарів у Блоках 1-5. Ці експерименти спрямовані на оцінку продуктивності та точності реконструкції моделі Гіпермережі за різних налаштувань.

## 3.4 Тренування

Процес навчання включає кілька ключових компонентів, включаючи градієнтний масштабувальник, оптимізатор, функцію втрат і планувальник швидкості навчання. Під час тренування використовувалась аугментація даних , а саме вертикальне та горизонтальне відображення вхідного зображення, що збільшує стабільність моделі. Також під час навчання замірялась метрика точності, яку можна побачити на рисунку 3.5.

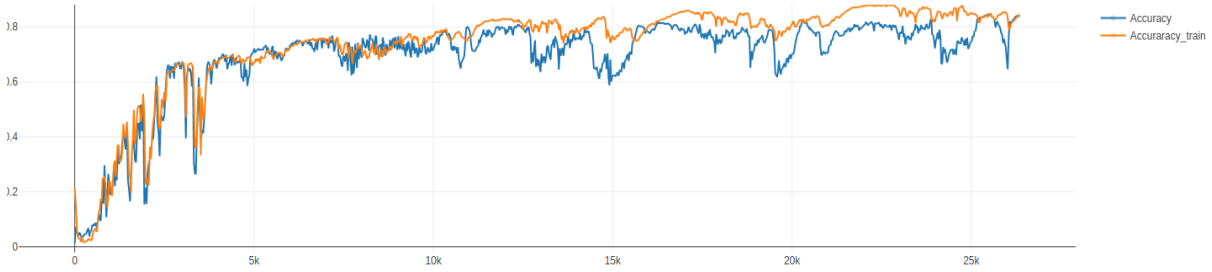



Рис. 3.5 Графік метрики точності під час навчання.

Оптимізатор і швидкість навчання.

Стохастичний градієнтний спуск (SGD) використовується як оптимізатор для навчання гіпермережі. Швидкість навчання встановлено на 1e-5, що визначає розмір кроку для кожного оновлення параметра. Вибір оптимізатора та швидкості навчання ґрунтувався на емпіричних оцінках та експериментах.

Функція втрати.

Функція втрат, яка використовується для навчання, — це двійкова перехресна ентропія з логітами (BCEWithLogitsLoss). Ця функція втрат підходить для завдань двійкової класифікації та забезпечує плавний процес оптимізації. Під час розробки моделі також досліджувалися альтернативні функції втрат, такі як втрати L1 або користувальницькі функції втрат.

Планувальник темпів навчання.

Для додаткової оптимізації навчального процесу використовується планувальник темпів навчання. Обраним планувальником є CyclicLR [30], який циклічно регулює швидкість навчання між базовою швидкістю навчання 1e-5 і максимальною швидкістю навчання 1e-4. Циклічний шаблон передбачає розмір кроку як для збільшення, так і для зменшення швидкості навчання.

## 3.5 Результати

Для оцінки результатів було підраховано метрику IoU, двома способами. Перший спосіб - оцінка метрики IoU на сітці з 50^3 точок в кубі -1,1. Другий спосіб - оцінка метрики IoU на 100 тисяч випадково обраних точок з обмежувальної рамки, такий спосіб використовували в [25]. Також через обмеження в потужності графічного процесору, метрика була виміряна лише на декількох класах. Отримані метрики можна побачити в таблиці 3.2.

Таблиця 3.2 Порівняння метрики IoU між розробленим методом та одним з вищезгаданих на датасеті ShapeNet

| IoU | | | |
|---|---|---|---|
| category | Our, in cube with 50^3 grid | Our, in bounding box with 100k random points | Occupancy Network |
| airplane | - | - | 0.571 |
| bench | 0.401 | 0.412 | **0.485** |
| cabinet | 0.650 | **0.751** | 0.733 |
| car | 0.583 | 0.702 | **0.737** |
| chair | - | - | 0.501 |

Кінець таблиці 3.2

| | | | |
|---|---|---|---|
| display | - | - | 0.471 |
| lamp | - | - | 0.371 |
| loudspeaker | - | - | 0.647 |
| rifle | - | - | 0.474 |
| sofa | 0.565 | **0.784** | 0.680 |
| table | 0.406 | 0.453 | **0.506** |
| telephone | - | - | 0.720 |
| vessel | - | - | 0.530 |
| mean | 0.520 | **0.584** | 0.571 |

Метрики, отримані за допомогою поточного методу, демонструють покращену продуктивність для певних класів порівняно з [25]. Однак важливо зазначити, що модель не була навчена на всіх класах, що слід враховувати при інтерпретації результатів. Незважаючи на це обмеження, є обнадійливі ознаки того, що поточний метод може перевершити [25]. Щоб підтвердити ці висновки, потрібні подальші експерименти з кращими графічним процесором і більше часу для підвищення точності. Слід зазначити, що фаза розробки була обмежена обмеженою доступністю GPU, що вплинуло на масштаб експерименту, оскільки лише невелика підмножина моделей використовувалася для навчання та оцінки. Таким чином, незважаючи на те, що поточні результати дають обнадійливі результати, для повної оцінки переваг поточного методу необхідні подальші дослідження з більш широкими наборами тренування та оцінювання.

## 3.6 Реконструкція

При реконструкції спочатку випадковим чином обираються N точок, для яких потім обчислюється належність до тривимірної моделі (рис 3.4). Після чого, використовується метод реконструкції з точок, а саме [31] Після чого додатково використовується фільтр для усереднення, котрий зміщує поточу вершину тривимірної моделі до її середнього арифметичного з прилеглими вершинами, що дозволяє отримати більш гладкий об’єкт. Візуально, реконструкцію можна побачити на рисунках 3.6 - 3.8.

Рис. 3.6 Зображення, з якого відбувається реконструкція

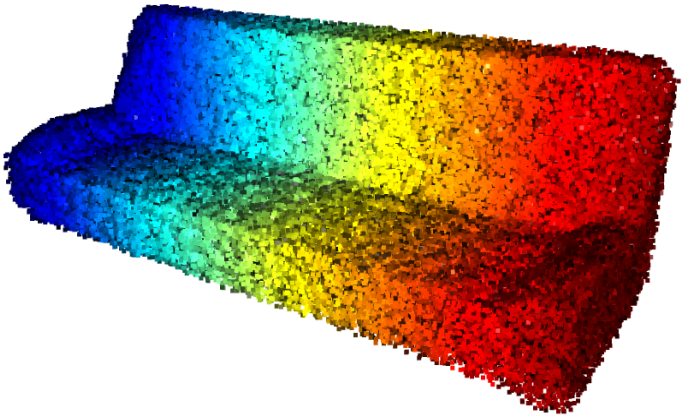


Рис. 3.7 Прогнозовані точки, які належать об’єкту

Рис. 3.8 Відновлена тривимірна модель

## 3.7 Висновки до розділу

Підводячи підсумок, у цьому розділі представлено вичерпний огляд розробленого методу, який використовує гіпермережі та базується на попередніх експериментах із цільовими мережами. Розділ охоплював концепцію гіпермереж, попередні експерименти, структуру розробленої архітектури, методологію навчання, аналіз результатів і застосування 3D-реконструкції. Розроблена методика продемонструвала кращу продуктивність порівняно з існуючими методами в окремих класах, однак слід зазначити, що модель не була навчена на всіх класах, охоплених порівняльним методом. Також, було охоплено навчальний процес і приклади зображень, що демонструють успішну 3D-реконструкцію.

# 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

## 4.1 Постановка задачі проектування

У даній роботі метою є розробка програмного продукту для тривимірної реконструкції об'єктів. Тривимірна реконструкція є процесом відтворення тривимірної моделі об'єкта або сцени на основі набору двовимірних зображень або інших даних. Основною задачею проекту є розробка програми з оптимальним набором функцій та простим і зрозумілим інтерфейсом, яка забезпечує точну та достовірну тривимірну реконструкцію об'єктів.

Технічні вимоги до програмного продукту наступні:

- можливість функціонування на комп’ютерах зі стандартним набором компонентів;
- наявність зручного та зрозумілого користувацького інтерфейсу;
- швидке функціонування та можливість використання ПП у реальному часі;
- мінімальна кількість часу для навчання користувача;
- можливість зменшення витрат на впровадження програмного продукту;

## 4.2 Обґрунтування функцій програмного продукту

Головна функція $F_0$ – розробка програмного продукту, який дає найменшу похибку при реконструкції тривимірної моделі. Виходячи з конкретних цілей, реалізованих програмним засобом, виділимо її основні функції:

$F_1$ – вибір мови програмування;

$F_2$ – вибір репрезентації об’єктів;

$F_3$ – інтерфейс користувача;

Кожна з описаних функцій має декілька варіантів реалізації:

Функція $F_1$:

а) мова програмування Python;

б) мова програмування C++;

Функція $F_1$:

а) Воксельна репрезентація;

б) Точкова репрезентація;

в) Меш-представлення;

г) Представлення невизначеною функцією;

Функція $F_3$:

а) інтерфейс користувача, створений на базі бібліотеки Tkinter;

б) інтерфейс користувача, створений на базі бібліотеки Qt.

Варіанти реалізації основних функцій наведено у морфологічній карті системи (Рис. 4.1).

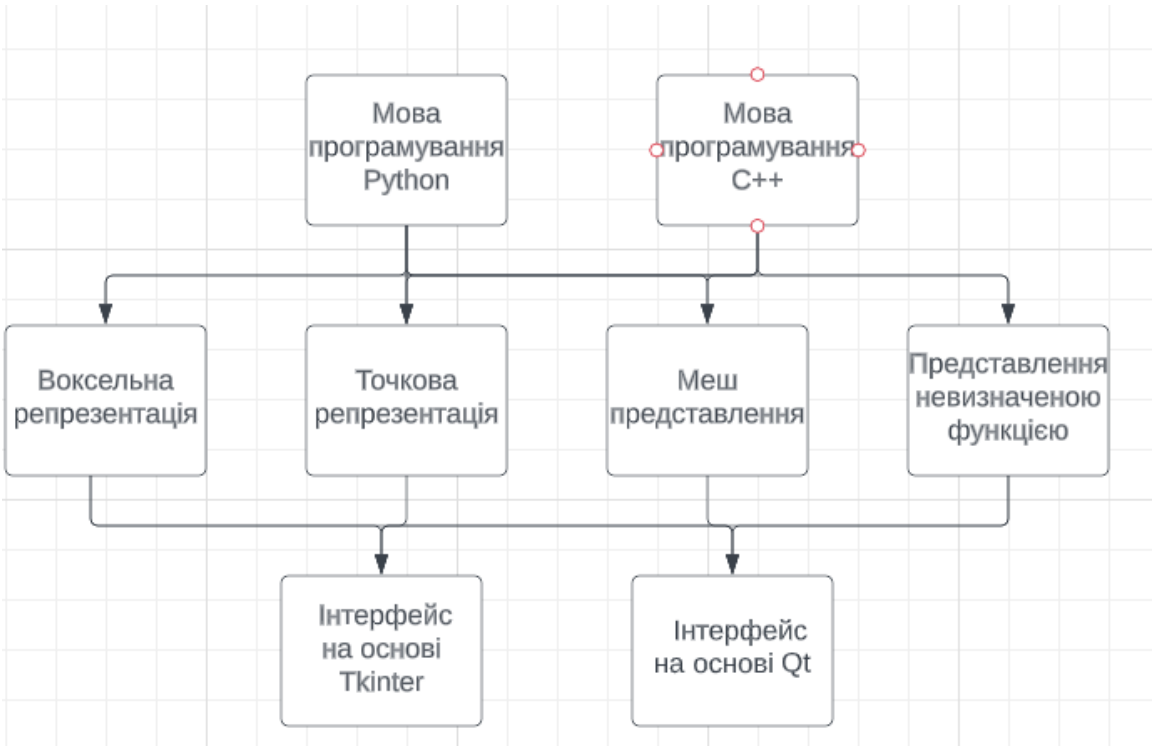



Рис. 4.1 Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. На її основі було побудовано позитивно-негативну матрицю варіантів основних функцій (табл. 4.1).

Таблиця 4.1 Позитивно-негативна матриця

| Функції | Варіанти реалізації | Переваги | Недоліки |
|---|---|---|---|
| $F_1$ | а | Простота використання | Дещо менша швидкість роботи методів. |
| | б | Швидкість виконання програм; кросплатформеність | Відсутність багатьох необхідних методів; повільна розробка |
| $F_2$ | а | Простота реалізації; інтуїтивність; проста пост-обробка | Обчислювальна складність; обмежена ефективність |
| | б | Незалежність від топології об’єкту; відсутність великих вимог до обчислювальних потужностей | Складність пост-обробки; великий вплив шуму |
| | в | Відсутність пост-обробки; відсутність великих вимог до обчислювальних потужностей | Створення моделей, що самоперетинаються; складність навчання моделей |

Кінець таблиці 4.1.

| | | | |
|---|---|---|---|
| $F_2$ | г | Відсутність великих вимог до обчислювальних потужностей; простота реалізації | Поступовий процес виконання пост-обробки |
| $F_3$ | а | Простота використання; кросплатформеність | Обмежені можливості дизайну |
| | б | Кросплатформеність; багатофункціональність | Складність установки та використання |

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція $F_1$:

Акцентуємо увагу на простоті використання та кросплатформеності як перевагах. Наша мета - спростити процес написання коду та знизити витрати на розробку, тому варіант б) буде відкинутий.

Функція $F_2$:

Варто відкинути варіант а), оскільки він вимагає багато обчислювальних потужностей, варіант, а варіант в) не виключає створення самоперетинаючихся моделей, тому обираємо варіанти б) і г)

Функція $F_3$:

Перша варіація реалізації є значно простішою та більш зрозумілою. Оскільки інтерфейс користувача не має великого впливу на функціонал програмного продукту, то варіант б) буде відкинутий.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

1. $F_1$а – $F_2$б – $F_3$а
2. $F_1$а – $F_2$г – $F_3$а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

## 4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для характеристики програмного продукту будемо використовувати наступні параметри:

- *X1* – час навчання мережі;
- *X2* – об’єм необхідної пам’яті для навчання мережі;
- *X3* – час генерації тривимірної моделі;

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

| Назва Параметра | Умовні позначення | Одиниці виміру | Значення параметра | | |
|---|---|---|---|---|---|
| | | | гірші | середні | кращі |
| Час навчання мережі | X1 | дні | 20 | 10 | 5 |
| Об’єм необхідної пам’яті для навчання мережі | X2 | Гб | 8 | 4 | 1 |
| Час генерації тривимірної моделі | X3 | Хв | 20 | 5 | 1 |

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

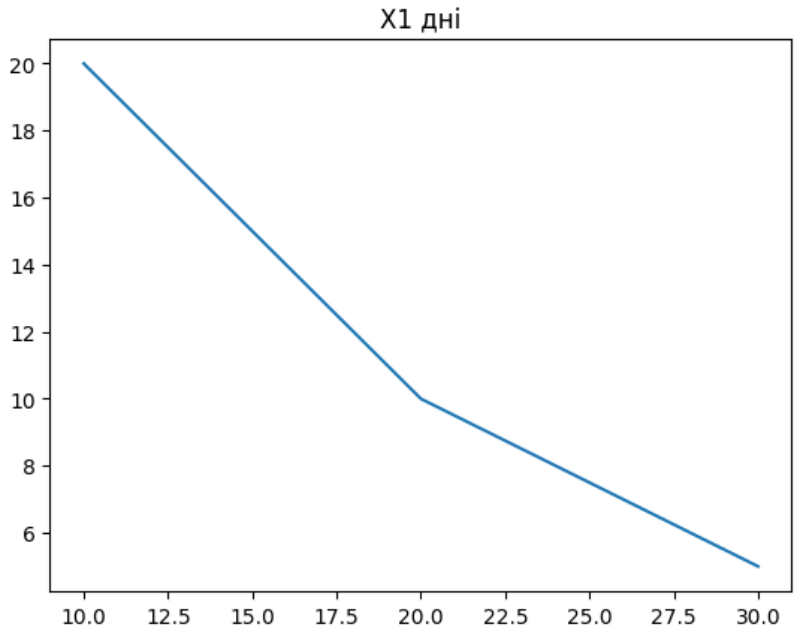



Рис. 4.2 X1 – Час навчання мережі

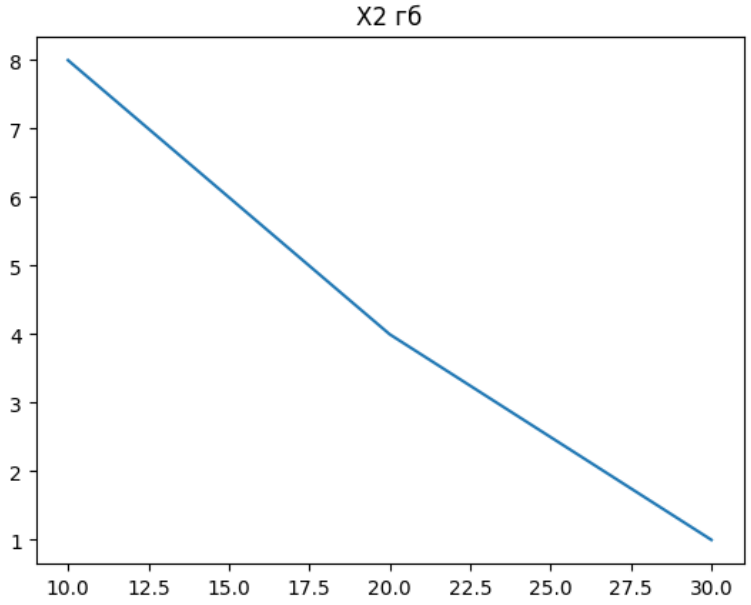



Рис. 4.3 X2 – Об’єм необхідної пам’яті для навчання мережі

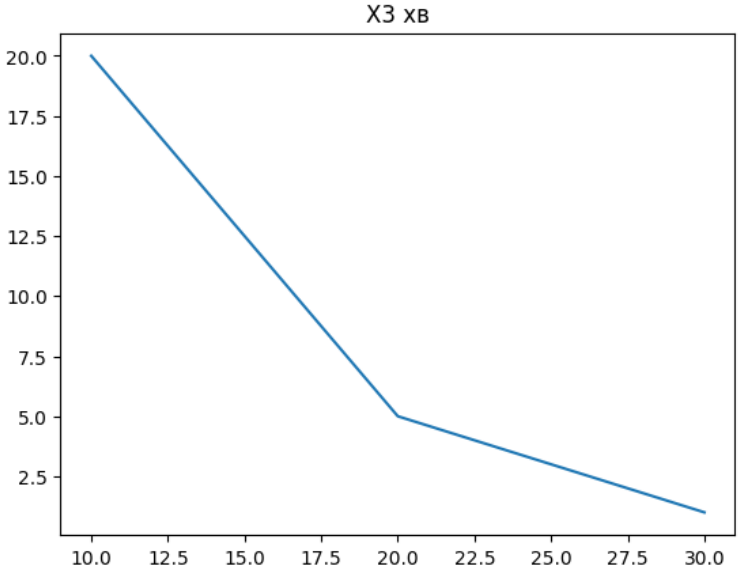



Рис. 4.4 X3 – Час генерації тривимірної моделі

## 4.4 Аналіз експертного оцінювання параметрів

Після проведення детального обговорення та аналізу, кожен експерт оцінює ступінь важливості кожного параметра з метою розробки програмного продукту, який забезпечує генерацію найкращих тривимірних моделей та оцінку їх якості. Оцінка значимості кожного параметра здійснюється за допомогою методу попарного порівняння. Експертна комісія, складена з 5 осіб, проводить оцінку згідно наступних кроків:

1. Визначення рівня значимості параметра шляхом присвоєння різних рангів.
2. Перевірка придатності експертних оцінок для подальшого використання.
3. Визначення оцінки попарного пріоритету параметрів.
4. Обробка результатів та визначення коефіцієнта значимості.

Цей процес дозволяє встановити вагомість кожного параметра та забезпечити об'єктивну оцінку їх впливу на якість генерованих тривимірних моделей. Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

| Параметр | Найменування параметра | Од. вим. | Ранг показника по оцінці експерта | | | | | Сума рангів $R_i$ | Відхилення $\Delta_i$ | $\Delta_i^2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | | | |
| X1 | Час навчання мережі | дні | 4 | 3 | 2 | 4 | 2 | 16 | 1.34 | 1.79 |
| X2 | Об’єм необхідної пам’яті для навчання мережі | Гб | 5 | 4 | 4 | 2 | 5 | 20 | 5.34 | 28.51 |
| X3 | Час генерації тривимірної моделі | Хв | 1 | 1 | 2 | 1 | 1 | 6 | -8.66 | 74.99 |
| | | | 10 | 8 | 8 | 7 | 11 | 44 | -1.97 | 105.29 |

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 30 \ (4.1)$$

де $N$ – число експертів,

$n$ – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 14.66 \ (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \ (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 105.29 \ (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2\left(n^3 - n\right)} = \frac{12 \cdot 105.29}{5^2\left(3^3 - 3\right)} = 2.10 \ (4.5)$$

Дане ранжирування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний коефіцієнт $W_k = 0,67$.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати запишемо у таблицю 4.4.

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, $a_{ij}$ визначається по формулі:

$$a_{ij} = \begin{cases} 1.5, & X_i > X_j \\ 1.0, & X_i = X_j \\ 0.5, & X_i < X_j \end{cases} \qquad (4.6)$$

Таблиця 4.4 - Попарне порівняння параметрів.

| Параметри | Експерти | | | | | Кінцева оцінка | Числове значення |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | | |
| X1 і X2 | < | < | < | > | < | < | 0.5 |
| X1 і X3 | > | > | = | > | > | > | 1.5 |
| X2 і X3 | > | > | > | > | > | > | 1.5 |

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}}, \quad (4.9)$$

$$b_i^{'} = \sum_{i=1}^{N} a_{ij} b_j \quad (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

| Параметри $X_i$ | Параметри $X_j$ | | | Перша ітер. | | Друга ітер. | |
|---|---|---|---|---|---|---|---|
| | X1 | X2 | X3 | $b_i$ | $K_{\text{ві}}$ | $b_i^1$ | $K_{\text{ві}}^1$ |
| X1 | 1 | 0.5 | 1.5 | 3 | 0.33 | 8 | 0.32 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| X2 | 1.5 | 1 | 1.5 | 4 | 0,44 | 11.5 | 0.46 |
| X3 | 0.5 | 0.5 | 1 | 2 | 0,22 | 5.5 | 0,22 |
| Всього: | | | | 9 | 1 | 25 | 1 |

4.5 Аналіз рівня якості варіантів реалізації функції

Визначимо рівень якості кожного з обраних варіантів виконання основних функцій окремо.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \quad (4.11)$$

де n – кількість параметрів;

$K_{\text{в}i}$ – коефіцієнт вагомості і–го параметра;

Ві – оцінка і–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

| Основна функція | Варіант реалізації | Абсолютне значення параметра | Бальна оцінка параметра | Коефіцієнт вагомості параметра | Коефіцієнт рівня якості |
|---|---|---|---|---|---|
| F1(X1) | а | 7 | 26 | 0.32 | 8.32 |
| F2(X2) | б | 2.8 | 23 | 0.46 | 10.58 |
| | г | 2.5 | 25 | 0.46 | 11.50 |
| F3(X3) | а | 7 | 18 | 0.22 | 3.96 |

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}\left[F_{1k}\right] + K_{\text{ТУ}}\left[F_{2k}\right] + \ldots + K_{\text{ТУ}}\left[F_{zk}\right], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 8.32 + 10.58 + 3.96 = 22.86 ;$$

$$K_{K2} = 8.32 + 11.50 + 3.96 = 23.78.$$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

## 4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи В, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 2; а в завданні 2 – до групи 1.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$\text{Т}_{\text{О}} = \text{Т}_{\text{Р}} \cdot \text{К}_{\text{П}} \cdot \text{К}_{\text{СК}} \cdot \text{К}_{\text{М}} \cdot \text{К}_{\text{СТ}} \cdot \text{К}_{\text{СТ.М}}, \quad (4.13)$$

де $T_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни В та групи складності алгоритму 2, трудомісткість дорівнює: $T_Р$ =19 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.2. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 19 \cdot 1.2 \cdot 0.9 = 20.52 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм першої групи складності, степінь новизни Б), тобто $T_Р$ = 64 людино-днів, $К_П$ =1.1, $К_{СК}$ = 1, $К_{СТ}$ =0.9:

$$T_2 = 15 \cdot 1.1 \cdot 0.9 = 63.36 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (20.52 + 63.36 + 6.91 + 63.36) \cdot 8 = 1233.20 \text{ людино-годин.}$$

$$T_{II} = (20.52 + 63.36 + 4.8 + 63.36) \cdot 8 = 1216.32 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 20000 грн. Визначимо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} грн., \ (4.14)$$

де М – місячний оклад працівників;

$T_m$ – кількість робочих днів в місяць;

$t$ – кількість робочих годин в день.

$$С_ч = \frac{20000 * 2}{2 \cdot 21 \cdot 8} = 119.04 \ грн. \ (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$СЗП = С_ч \cdot Т_i \cdot КД, (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 119.04 \cdot 1233.20 \cdot 1.2 = 176160.15$ грн

II. $С_{ЗП} = 119.04 \cdot 1216.32 \cdot 1.2 = 173748.87$ грн

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 176160.15 \cdot 0.22 = 38755.23$ грн.

$$\text{II. } С_{ВІД} = С_{ЗП} \cdot 0.22 = 173748.87 \cdot 0.22 = 38224.75 \text{ грн.}$$

Обчислимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 20000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 20000 \cdot 0,2 = 48000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 48000 \cdot (1 + 0.2) = 57600 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 57600 \cdot 0,22 = 12672 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 12% та вартості ЕОМ – 30000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.1 \cdot 0.12 \cdot 30000 = 3960 \text{ грн.,}$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.1 \cdot 30000 \cdot 0.08 = 2640 \text{ грн.,}$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.9 = \\ = 1677.6 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677.6 \cdot 0.2 \cdot 0.85 \cdot 4.87 = 1388.88 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 30000 \cdot 0,67 = 20100 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, (4.17)$$

$$С_{ЕКС} = 57600 + 12672 + 3960 + 1120 + 1388.88 + 20100 = 96840.88 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 96840.88 \ / \ 1677.6 = 57.72 \ \text{грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot T, (4.18)$$

I. $С_М = 57.72 \cdot 1233.20 = 71180.30$ грн.

II. $С_М = 57.72 \cdot 1216.32 = 70205.99$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0.67, (4.19)$$

I. $С_Н = 176160.15 \cdot 0.67 = 118027.3005$ грн.

II. $С_Н = 173748.87 \cdot 0.67 = 116411.7429$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \#(4.20)$$

I. $С_{ПП} = 176160.15 + 38755.23 + 71180.30 + 118027.30 = 404122.98$ грн.
II. $С_{ПП} = 173748.87 + 38224.75 + 70205.99 + 116411.74 = 398591.35$ грн.

## 4.7 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{К j} / С_{Ф j}, \quad (4.21)$$

$$К_{ТЕР1} = 22.86 \ / 404122.98 = 5{,}656 \cdot 10^{-5},$$
$$К_{ТЕР2} = 23.78 \ / \ 398591.35 = 5{,}966 \cdot 10^{-5}.$$

Отже, найбільш ефективним є другий варіант реалізації програми.

## 4.8 Висновки до розділу

Шляхом виконання економічного розділу були систематизовані та закріплені теоретичні знання в сфері економіки та організації виробництва. Ці знання використовувалися для техніко-економічного обґрунтування розробки за допомогою методу функціонально-вартісного аналізу.

На основі інформації про основні функції, які має виконувати програмний продукт, було визначено два найбільш перспективні варіанти його реалізації. З них виявився найбільш ефективним другий варіант, який володіє вищою ступенем простоти реалізації і водночас досягає достатньої рівноваги.

Обраний варіант реалізації програмного продукту забезпечує високу рівень універсальності та легкість його модифікації, що робить його особливо цінним.

# ВИСНОВКИ

Під час проведення дослідження був впроваджений новий метод реконструкції тривимірних об'єктів, який показав вищу продуктивність у порівнянні з наявними методами в певних класах. Варто зазначити, що модель не була навчена на всіх класах, які розглядалися у порівняльному методі. Однак, існує ймовірність, що цей метод зможе досягти або навіть перевершити порівняльний метод з точки зору відповідних навчальних метрик.

Дипломна робота складається з чотирьох розділів.

Перший розділ присвячений огляду та аналізу проблем, що існують у вибраній предметній області, і в ньому описуються поточні проблеми цієї області.

У другому розділі розглядаються наявні методи та способи представлення даних, які мають суттєвий вплив на функціонування обраного методу, і також згадуються переваги та недоліки цих підходів.

Третій розділ містить опис самого розробленого методу, його аналіз та опис проведених експериментів. В цьому розділі розглядаються різні сценарії, характеристики та налаштування штучної нейронної мережі з метою досягнення більш точної реконструкції. Також проводиться порівняльний аналіз з методом схожого формату, і представляються основні коефіцієнти та значення, які допомагають визначити, який з методів працює краще.

У четвертому розділі наведено опис функціонально-вимогового аналізу для розробленого програмного продукту.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Furukawa, Y., & Ponce, J. Accurate, dense, and robust multiview stereopsis. IEEE transactions on pattern analysis and machine intelligence, 32(8), 1362-1376, 2010.

2. Snavely, N., Seitz, S. M., & Szeliski, R. Photo tourism: exploring photo collections in 3D. ACM Transactions on Graphics (TOG), 25(3), 835-846, 2006.

3. Li, H., Snavely, N., & Huttenlocher, D. P. Location recognition using prioritized feature matching. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1-8, 2008.

4. Yao, R., Urtasun, R., & Fidler, S. 3D reconstruction in the wild. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 4088-4096, 2018.

5. Dai, A., Qi, C. R., & Nießner, M. Shape completion using 3D-encoder-predictor CNNs and shape synthesis. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 6545-6554, 2017.

6. Zhu, Z., Xu, L., & Wang, X. Multiview consistency as supervisory signal for learning shape and pose prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1746-1754, 2019.

7. Hartley, R., & Zisserman, A. Multiple view geometry in computer vision. Cambridge University Press, 2004.

8. Szeliski, R. Computer vision: algorithms and applications. Springer Science & Business Media, 2010.

9. Scharstein, D., & Szeliski, R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1-3), 7-42, 2002.

10. Newell, A., Yang, K., Deng, J. "Stacked Hourglass Networks for Human Pose Estimation." In Proceedings of the European Conference on Computer Vision (ECCV), 2016.

11. Bernardini, F., Mittleman, J., Rushmeier, H.E., Silva, C.T., Taubin, G.: The ball-pivoting algorithm for surface reconstruction. IEEE Trans. Vis. Comput. Graph. 5(4), 349–359, 1999.

12. D. P. Bertsekas. A distributed asynchronous relaxation algorithm for the assignment problem. In Decision and Control, 1985 24th IEEE Conference on, pages 1703–1704. IEEE, 1985.

13. W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In ACM Trans. on Graphics (SIGGRAPH), 1987.

14. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

15. Choy, C.B., Xu, D., Gwak, J., Chen, K., Savarese, S. "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction." In Proceedings of the European Conference on Computer Vision (ECCV), 2016.

16. Tahir, R., Sargano, A.B., Habib, Z. "Voxel-Based 3D Object Reconstruction from Single 2D Image Using Variational Autoencoders." Mathematics, 2021, 9(18), 2288.

17. Lowe, D.G. "Distinctive Image Features from Scale-Invariant Keypoints." International Journal of Computer Vision (IJCV), 2004, 60(2): 91-110.

18. J. Wu, Y. Wang, T. Xue, X. Sun, B. Freeman, and J. Tenenbaum. MarrNet: 3D shape reconstruction via 2.5D sketches. In Advances in Neural Information Processing Systems (NIPS), 2017.

19. T. Groueix, M. Fisher, V. G. Kim, B. Russell, and M. Aubry. AtlasNet: A papier-mach ˆ e approach to learning 3d surface ´ generation. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2018.

20. A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik. Endto-end recovery of human shape and pose. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2018.

21. N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. Pixel2Mesh: Generating 3D mesh models from single RGB images. In Proc. of the European Conf. on Computer Vision (ECCV), 2018.

22. Y. Liao, S. Donne, and A. Geiger. Deep marching cubes: Learning explicit surface representations. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2018.

23. W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In ACM Trans. on Graphics (SIGGRAPH), 1987.

24. Fan, H., Su, H., Guibas, L. "A Point Set Generation Network for 3D Object Reconstruction from a Single Image." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

25. Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A. "Occupancy Networks: Learning 3D Reconstruction in Function Space." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

26. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. "ShapeNet: An Information-Rich 3D Model Repository." arXiv preprint arXiv:1512.03012, 2015.

27. Ha, D., Dai, A., & Le, Q. V. . Hypernetworks. arXiv preprint arXiv:1609.09106, 2016.

28. Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., & Evans, T. R. Reconstruction and representation of 3D objects with radial basis functions. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (pp. 67-76), 2006.

29. Misra, D. Mish: A self-regularized non-monotonic activation function. arXiv preprint arXiv:1908.08681, 2020.

30. Smith, L. N. "Cyclical Learning Rates for Training Neural Networks." arXiv preprint arXiv:1506.01186, 2017.

31. Edelsbrunner and D. G. Kirkpatrick and R. Seidel: On the shape of a set of points in the plane, IEEE Transactions on Information Theory, 29 (4): 551–559, 1983.

# ДОДАТОК А

## Лістинг програми

main.py

```
import torch
from torchvision import transforms
import numpy as np
from torch import nn
from PIL import Image
import open3d as o3d
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import itertools
import torchvision
from tqdm import tqdm
from mlflow import log_metric, log_param, log_artifacts
from sklearn.metrics import f1_score

torch.set_printoptions(8)

k =
pd.read_csv('/run/media/eingird/6300c1fd-9c08-4f0f-8682-8f90e1432e75/3d_reconstructi
on/sofa.csv')
k.imgs_path = k.imgs_path.apply(lambda x :
x.replace('/mnt/Dev/STUDY/','/run/media/eingird/ec26c78b-20bc-47f1-b2d5-33a92d92c9b6
/STUDY/'))
k.obj_path = k.obj_path.apply(lambda x :
x.replace('/mnt/Dev/WORK/','/run/media/eingird/ec26c78b-20bc-47f1-b2d5-33a92d92c9b6/
WORK/'))


def check_watertight(obj_path):
    mesh = o3d.io.read_triangle_mesh(obj_path)
    return mesh.is_watertight()

k['is_watertight'] = k['obj_path'].apply(lambda x : check_watertight(x))
data = k

data['obj_path'] = data['imgs_path'].apply(lambda x :  'mesh-fusion/output/3_out/' +
x.split('/')[-2] + '.obj' )
data['obj_path'] = data['obj_path'].apply(lambda x :
x.replace('output/3_out','scaled_open3d_meshes'))
data['is_watertight'] = data['obj_path'].apply(lambda x : check_watertight(x))

data = []
for i, row in df.iterrows():
    imgs_path = row['imgs_path']
    obj_path = row['obj_path']
    class_name = row['class']
    img_files = os.listdir(imgs_path)
    for img_file in img_files:
        if 'png' in img_file:
            img_path = os.path.join(imgs_path, img_file)
            data.append((img_path, obj_path, class_name))

df = pd.DataFrame(data, columns=['img_path', 'obj_path', 'class'])
```

```
def compute_occupancy(mesh,random_points):
    legacy_mesh = o3d.t.geometry.TriangleMesh.from_legacy(mesh)
    scene = o3d.t.geometry.RaycastingScene()
    _ = scene.add_triangles(legacy_mesh)
    occupancy = scene.compute_occupancy(random_points).numpy()
    return occupancy

def create_uniform_points_for_mesh_in_bounding_volume(df, n):

    # Loop through unique mesh paths in the DataFrame
    for mesh_path in df['obj_path'].unique():
        # Load the mesh from file
        mesh = o3d.io.read_triangle_mesh(mesh_path)
        legacy_mesh = o3d.t.geometry.TriangleMesh.from_legacy(mesh)

        scene = o3d.t.geometry.RaycastingScene()
        _ = scene.add_triangles(legacy_mesh)
        # Get the bounding box of the mesh
        bbox = mesh.get_axis_aligned_bounding_box()

        # Generate n random points uniformly within the bounding box
        # random_points = np.random.uniform(-1,1,(n,3)).astype(np.float32)
        random_points = np.random.uniform(bbox.min_bound, bbox.max_bound, (n,
3)).astype(np.float32)
        occupancy = scene.compute_occupancy(random_points).numpy()

        print(f'inside_points = {(occupancy == 1).sum()}, outside_points =
{(occupancy == 0).sum()}')
        print('GENERATING NEW outside_points')
        inside_points = random_points[occupancy==1]
        outside_points = random_points[occupancy==0]
        outside_points = outside_points[:len(outside_points)//2]
        new_random_points = np.random.uniform(-1, 1, (len(outside_points)//2,
3)).astype(np.float32)
        occupancy_new = scene.compute_occupancy(new_random_points).numpy()
        new_outside_points = new_random_points[occupancy_new == 0]
        final_outside_points =
np.concatenate((new_outside_points,outside_points),axis=0)
        final_points = np.concatenate((inside_points,final_outside_points),axis=0)
        final_occupancy = np.array([1]* len(inside_points) + [0] *
len(final_outside_points))

        np.savez(f"points_folder/{mesh_path.split('/')[-1].split('.')[0]}.npz",
points=final_points, distances=final_occupancy)
        # return  random_points,occupancy

create_uniform_points_for_mesh_in_bounding_volume(df,n=1_000_000)


def get_centers(per_axis = 5,on_surface = False):
    if on_surface:
        centers =
df[df['distance']==0].sample(frac=1)[:per_axis**3].drop(columns=['distance']).values
    else:
        x = np.linspace(-1,1,per_axis)
        y = np.linspace(-1,1,per_axis)
        z = np.linspace(-1,1,per_axis)

        centers = np.array(list(itertools.product(*[x,y,z])))
    return centers

train_augmentation =  torchvision.transforms.Compose([
```

```
torchvision.transforms.RandomHorizontalFlip(),

torchvision.transforms.RandomVerticalFlip()]
                                                    )

class MyDataset():
    def __init__(self,df,augmentation = None):
        self.df = df
        self.augmentation = augmentation

    def __len__(self):
        return self.df.shape[0]

    def __getitem__(self,idx):
        img_path, mesh_path, _ = df.iloc[idx]

        loaded_data =
np.load(f"points_folder/{mesh_path.split('/')[-1].split('.')[0]}.npz")
        points_loaded = loaded_data['points']
        distances_loaded = loaded_data['distances']

        M = 16384
        random_indices = np.random.choice(points_loaded.shape[0], size=M,
replace=False)
        points_random = points_loaded[random_indices]
        distances_random = distances_loaded[random_indices]


        img = Image.open(img_path)
        img_tensor = transforms.PILToTensor()(img)
        img_tensor = img_tensor[:3, :, :]
        img_tensor = img_tensor/255.0

        if self.augmentation is not None:
            img_tensor = self.augmentation(img_tensor)

        tensor_points = torch.Tensor(points_random)
        tensor_dist = torch.Tensor(distances_random)


        return img_tensor, tensor_points, tensor_dist

dataset = MyDataset(df)
df = df.sample(frac=1)
unique_obj_paths = df.obj_path.unique()

# Calculate the number of obj_path values for the train set
train_obj_paths = unique_obj_paths[:int(0.8*len(unique_obj_paths))]#

# Split the dataframe into train and test sets
train_df = df[df.obj_path.isin(train_obj_paths)]
test_df = df[~df.obj_path.isin(train_obj_paths)]

train_dataset, test_dataset = MyDataset(train_df,train_augmentation),
MyDataset(test_df)#


class ResidualBlock(torch.nn.Module):
    def __init__(self,module):
        super().__init__()

        self.module = module
```

```
    def forward(self,x):
        return self.module(x) + x


class HyperNetwork(torch.nn.Module):
    def __init__(self,rbf_centers=1000):
        super().__init__()
        self.conv = torch.nn.Conv2d(3,8,5)

        self.block1_conv = nn.Conv2d(8,16,3,2)
        self.block1 = nn.Sequential(
                                    ResidualBlock(nn.Sequential(
                                                  nn.BatchNorm2d(16),
                                                  torch.nn.ReLU(),

nn.Conv2d(16,16,3,padding='same'))),
                                    ResidualBlock(nn.Sequential(
                                                  nn.BatchNorm2d(16),
                                                  torch.nn.ReLU(),

nn.Conv2d(16,16,3,padding='same')))
                                    )
        self.block2_conv = nn.Conv2d(16,32,3,2)
        self.block2 = nn.Sequential(
                                    ResidualBlock(nn.Sequential(
                                                  nn.BatchNorm2d(32),
                                                  torch.nn.ReLU(),

nn.Conv2d(32,32,3,padding='same'))),
                                    ResidualBlock(nn.Sequential(
                                                  nn.BatchNorm2d(32),
                                                  torch.nn.ReLU(),

nn.Conv2d(32,32,3,padding='same'))),
                                    )
        self.block3_conv = nn.Conv2d(32,128,3,2)
        self.block3 = nn.Sequential(ResidualBlock(nn.Sequential(
                                                  nn.BatchNorm2d(128),
                                                  torch.nn.ReLU(),

nn.Conv2d(128,128,3,padding='same'))),
                                    ResidualBlock(nn.Sequential(
                                                  nn.BatchNorm2d(128),
                                                  torch.nn.ReLU(),

nn.Conv2d(128,128,3,padding='same'))),
                                    )
        self.block4_conv = nn.Conv2d(128,512,3,2)
        self.block4 = nn.Sequential(ResidualBlock(nn.Sequential(
                                                  nn.BatchNorm2d(512),
                                                  torch.nn.ReLU(),

nn.Conv2d(512,512,3,padding='same'))),
                                    ResidualBlock(nn.Sequential(
                                                  nn.BatchNorm2d(512),
                                                  torch.nn.ReLU(),

nn.Conv2d(512,512,3,padding='same'))),
                                    )

        self.block5_conv = nn.Conv2d(512,1024,3,2)
        self.block5 = nn.Sequential(ResidualBlock(nn.Sequential(
```

```
                                                    nn.BatchNorm2d(1024),
                                                    torch.nn.ReLU(),

nn.Conv2d(1024,1024,3,padding='same'))),
                                    ResidualBlock(nn.Sequential(
                                                    nn.BatchNorm2d(1024),
                                                    torch.nn.ReLU(),

nn.Conv2d(1024,1024,3,padding='same'))),
                                    )

        # self.block6 = nn.Sequential(ResidualBlock(nn.Sequential(
        #                                             nn.BatchNorm2d(1024),
        #                                             torch.nn.ReLU(),
        #
nn.Conv2d(1024,1024,3,padding='same'))),
        #                               ResidualBlock(nn.Sequential(
        #                                             nn.BatchNorm2d(1024),
        #                                             torch.nn.ReLU(),
        #
nn.Conv2d(1024,1024,3,padding='same'))),
        #                               )


        self.avg_pooling = nn.AvgPool2d(3)

        self.blocks = nn.Sequential(self.conv,
                                    self.block1_conv,
                                    self.block1,
                                    self.block2_conv,
                                    self.block2,
                                    self.block3_conv,
                                    self.block3,
                                    self.block4_conv,
                                    self.block4,
                                    self.block5_conv,
                                    self.block5,
                                    # self.block6,
                                    self.avg_pooling
        )

        self.flatten = torch.nn.Flatten()
        self.linear_weigts = torch.nn.Linear(1024,rbf_centers * 128 + 14368 + 5)
        # self.fc1_bias = torch.nn.Linear(2803712,1)

    def forward(self,x):
        """x - image"""

        x = self.blocks(x)
        x = self.flatten(x)
        weights_tensor = self.linear_weigts(x)
        return weights_tensor

hp = HyperNetwork(1000)
W = hp(train_dataset[0][0].unsqueeze(0))


class RBF(nn.Module):
    """
    Transforms incoming data using a given radial basis function:
    u_{i} = rbf(||x - c_{i}|| / s_{i})
    Arguments:
        in_features: size of each input sample
```

```
        out_features: size of each output sample
    Shape:
        - Input: (N, in_features) where N is an arbitrary batch size
        - Output: (N, out_features) where N is an arbitrary batch size
    Attributes:
        centres: the learnable centres of shape (out_features, in_features).
            The values are initialised from a standard normal distribution.
            Normalising inputs to have mean 0 and standard deviation 1 is
            recommended.

        log_sigmas: logarithm of the learnable scaling factors of shape
(out_features).

        basis_func: the radial basis function used to transform the scaled
            distances.
    """

    def __init__(self, in_features, out_features, basis_func,centers=None):
        super(RBF, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        if centers is None:
            self.centres = nn.Parameter(torch.Tensor(out_features, in_features))
        else:
            self.centres = torch.Tensor(centers).to('cuda')
        self.log_sigmas = torch.Tensor([0.5]*out_features).to('cuda')
        self.basis_func = basis_func
        self.reset_parameters()

    def reset_parameters(self):
        nn.init.normal_(self.centres, 0, 1)
        nn.init.constant_(self.log_sigmas, 0)

    # def forward(self, input):
    #     size = (input.size(0), self.out_features, self.in_features)
    #     x = input.unsqueeze(1).expand(size)
    #     c = self.centres.unsqueeze(0).expand(size)
    #     print(x.shape, c.shape)
    #     distances = (x - c).pow(2).sum(-1).pow(0.5) /
torch.exp(self.log_sigmas).unsqueeze(0)
    #     return self.basis_func(distances)

    def forward(self, x):
        # x is the input batch of points of size (B,K,3)
        B, K, _ = x.size()
        C, _ = self.centres.size()

        # Repeat the input batch K times along the first dimension
        # and repeat the centers batch B times along the first dimension
        x = x.unsqueeze(2).repeat(1, 1, C, 1)
        centers = self.centres.unsqueeze(0).repeat(B, K, 1, 1)

        # Compute the Euclidean distances between each point in the input batch
        # and each center in self.centers using PyTorch's broadcasting feature
        distances = torch.sqrt(torch.sum((x - centers)**2, dim=-1))

        # Divide each distance by the corresponding sigma
        sigmas = torch.exp(self.log_sigmas).unsqueeze(0).unsqueeze(1).repeat(B, K,
1)
        distances = distances / sigmas

        # Return the distances with shape (B,K,C)
        return distances
```

```
# RBFs

def gaussian(alpha):
    phi = torch.exp(-1*alpha.pow(2))
    return phi

def linear(alpha):
    phi = alpha
    return phi

def quadratic(alpha):
    phi = alpha.pow(2)
    return phi

def inverse_quadratic(alpha):
    phi = torch.ones_like(alpha) / (torch.ones_like(alpha) + alpha.pow(2))
    return phi

def multiquadric(alpha):
    phi = (torch.ones_like(alpha) + alpha.pow(2)).pow(0.5)
    return phi

def inverse_multiquadric(alpha):
    phi = torch.ones_like(alpha) / (torch.ones_like(alpha) + alpha.pow(2)).pow(0.5)
    return phi

def spline(alpha):
    phi = (alpha.pow(2) * torch.log(alpha + torch.ones_like(alpha)))
    return phi

def poisson_one(alpha):
    phi = (alpha - torch.ones_like(alpha)) * torch.exp(-alpha)
    return phi

def poisson_two(alpha):
    phi = ((alpha - 2*torch.ones_like(alpha)) / 2*torch.ones_like(alpha)) \
    * alpha * torch.exp(-alpha)
    return phi

def matern32(alpha):
    phi = (torch.ones_like(alpha) + 3**0.5*alpha)*torch.exp(-3**0.5*alpha)
    return phi

def matern52(alpha):
    phi = (torch.ones_like(alpha) + 5**0.5*alpha + (5/3) \
    * alpha.pow(2))*torch.exp(-5**0.5*alpha)
    return phi

def triharmonic_spline(alpha):
    return alpha**3

def basis_func_dict():
    """
    A helper function that returns a dictionary containing each RBF
    """

    bases = {'gaussian': gaussian,
             'linear': linear,
             'quadratic': quadratic,
             'inverse quadratic': inverse_quadratic,
             'multiquadric': multiquadric,
```

```
            'inverse multiquadric': inverse_multiquadric,
            'spline': spline,
            'poisson one': poisson_one,
            'poisson two': poisson_two,
            'matern32': matern32,
            'matern52': matern52,
            'thriharmonic_spline':triharmonic_spline}
    return bases


class BoundaryPredictor(torch.nn.Module):
    def __init__(self, in_features, out_features, basis_func, centers):
        super().__init__()
        self.hypernet = HyperNetwork(centers.shape[0])
        self.out_features = out_features
        self.rbf = RBF(in_features, out_features, basis_func, centers)
        # self.dropout = torch.nn.Dropout(0.5)

    def forward(self, img, xyz):
        batch_size = xyz.shape[0]
        rbf_points = self.rbf(xyz)  # Shape: (B, N, C)

        W = self.hypernet(img)  # Shape: (B, W)
        W = W.view(batch_size, -1)
        W1 = W[:, :self.out_features * 128].view(batch_size, self.out_features, 128)
        W2 = W[:, self.out_features * 128:self.out_features * 128 + 128 *
64].view(batch_size, 128, 64)
        W3 = W[:, self.out_features * 128 + 128 * 64:self.out_features * 128 + 128 *
64 + 64 * 64].view(batch_size, 64, 64)
        W4 = W[:, self.out_features * 128 + 128 * 64 + 64 * 64:self.out_features *
128 + 128 * 64 + 64 * 64 + 64 * 32].view(batch_size, 64, 32)
        W5 = W[:, self.out_features * 128 + 128 * 64 + 64 * 64 + 64 *
32:self.out_features * 128 + 128 * 64 + 64 * 64 + 64 * 32 + 32].view(batch_size, 32,
1)
        b = W[:, self.out_features * 128 + 128 * 64 + 64 * 64 + 64 * 32 +
32:].view(batch_size, 5)

        b1 = b[:, 0]
        b2 = b[:, 1]
        b3 = b[:, 2]
        b4 = b[:, 3]
        b5 = b[:, 4]

        fc1 = torch.matmul(rbf_points, W1) + b1.view(batch_size, 1, -1)
        fc1 = torch.nn.functional.mish(fc1)
        # fc1 = self.dropout(fc1)

        fc2 = torch.matmul(fc1, W2) + b2.view(batch_size, 1, -1)
        fc2 = torch.nn.functional.mish(fc2)

        fc3 = torch.matmul(fc2, W3) + b3.view(batch_size, 1, -1)
        fc3 = torch.nn.functional.mish(fc3)

        fc4 = torch.matmul(fc3, W4) + b4.view(batch_size, 1, -1)
        fc4 = torch.nn.functional.mish(fc4)

        fc5 = torch.matmul(fc4, W5) + b5.view(batch_size, 1, -1)

        return fc5.squeeze(-1)

    def predict_using_multiple_images(self,imgs, xyz):
        batch_size = xyz.shape[0]
        rbf_points = self.rbf(xyz)  # Shape: (B, N, C)
```

```
        W = []
        for img in imgs:
            W.append(self.hypernet(img))  # Shape: (B, W)
        W = torch.mean(torch.stack(W,axis=0),axis=0)
        W = W.view(batch_size, -1)
        W1 = W[:, :self.out_features * 128].view(batch_size, self.out_features, 128)
        W2 = W[:, self.out_features * 128:self.out_features * 128 + 128 *
64].view(batch_size, 128, 64)
        W3 = W[:, self.out_features * 128 + 128 * 64:self.out_features * 128 + 128 *
64 + 64 * 64].view(batch_size, 64, 64)
        W4 = W[:, self.out_features * 128 + 128 * 64 + 64 * 64:self.out_features *
128 + 128 * 64 + 64 * 64 + 64 * 32].view(batch_size, 64, 32)
        W5 = W[:, self.out_features * 128 + 128 * 64 + 64 * 64 + 64 *
32:self.out_features * 128 + 128 * 64 + 64 * 64 + 64 * 32 + 32].view(batch_size, 32,
1)
        b = W[:, self.out_features * 128 + 128 * 64 + 64 * 64 + 64 * 32 +
32:].view(batch_size, 5)

        b1 = b[:, 0]
        b2 = b[:, 1]
        b3 = b[:, 2]
        b4 = b[:, 3]
        b5 = b[:, 4]

        fc1 = torch.matmul(rbf_points, W1) + b1.view(batch_size, 1, -1)
        fc1 = torch.nn.functional.mish(fc1)
        # fc1 = self.dropout(fc1)

        fc2 = torch.matmul(fc1, W2) + b2.view(batch_size, 1, -1)
        fc2 = torch.nn.functional.mish(fc2)

        fc3 = torch.matmul(fc2, W3) + b3.view(batch_size, 1, -1)
        fc3 = torch.nn.functional.mish(fc3)

        fc4 = torch.matmul(fc3, W4) + b4.view(batch_size, 1, -1)
        fc4 = torch.nn.functional.mish(fc4)

        fc5 = torch.matmul(fc4, W5) + b5.view(batch_size, 1, -1)

        return fc5.squeeze(-1)

def get_image_tensor(img_path):
    img = Image.open(img_path)
    img_tensor = transforms.PILToTensor()(img)
    img_tensor = img_tensor[:3, :, :]
    img_tensor = img_tensor/255.0
    img_tensor = img_tensor.unsqueeze(0)
    return img_tensor

def eval_uniform(df,num_points=32,use_bounding_box =
False,threshold=0.5,sampling=None):
    # Create a 1D array of points along each axis
    x = np.linspace(-1, 1, num_points).astype(np.float32)
    y = np.linspace(-1, 1, num_points).astype(np.float32)
    z = np.linspace(-1, 1, num_points).astype(np.float32)
    torch.cuda.empty_cache()
    # Create a 3D meshgrid from the 1D arrays
    X, Y, Z = np.meshgrid(x, y, z, indexing='ij')


    # Flatten the meshgrid into a 3xN array of points, where N is the total number
of points
    points = np.vstack((X.flatten(), Y.flatten(), Z.flatten())).T
```

```
    if sampling == 'random':
        points = np.random.uniform(-1, 1, (100_000, 3)).astype(np.float32)
    mean_iou = 0

    for i,data in df.iterrows():
        obj = data['obj_path']
        img = data['img_path']
        mesh = o3d.io.read_triangle_mesh(obj)
        if use_bounding_box:
            bbox = mesh.get_axis_aligned_bounding_box()
            # Generate n random points uniformly within the bounding box
            # random_points = np.random.uniform(-1,1,(n,3)).astype(np.float32)
            points = np.random.uniform(bbox.min_bound, bbox.max_bound, (100_000,
3)).astype(np.float32)


        y_true = compute_occupancy(mesh,points)
        with torch.no_grad():
            img_tensor = get_image_tensor(img).to('cuda')
            points_tensor = torch.tensor(points).unsqueeze(0).to('cuda')
            predicted = model(img_tensor, points_tensor)
            iou = compute_iou(torch.tensor(y_true), predicted.cpu(),threshold)
            mean_iou += iou/len(df)
    torch.cuda.empty_cache()

    print(f'mean_iou = {mean_iou}')


centers  = get_centers(6)

model = BoundaryPredictor(3,centers.shape[0],basis_func_dict()['poisson
two'],centers).to('cuda')

train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=8)
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=8)

scaler = torch.cuda.amp.GradScaler()
optim = torch.optim.SGD(model.parameters(),lr=1e-5)
# optim = torch.optim.SGD(net.parameters(),lr=1e-2)
criterion = nn.BCEWithLogitsLoss()##MyLoss()#torch.nn.L1Loss() #MyLoss()##
# scheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(optim,patience=350,threshold=0.005,
factor=0.2,verbose=True)
scheduler = torch.optim.lr_scheduler.CyclicLR(optim,base_lr =
1e-5,max_lr=1e-4,step_size_up=100,step_size_down=100)

MSE_METRIC = nn.BCEWithLogitsLoss()

def compute_accuracy(y_true, y_pred):
    same = (y_true == y_pred).sum()
    total = y_true.shape[0] * y_pred.shape[1]
    return same / total

def compute_f1(y_true,y_pred):
    f1 = 0
    y_true = y_true.cpu().numpy()
    y_pred = y_pred.cpu().numpy()
    for i in range(y_true.shape[0]):
        y_t = y_true[i]
        y_p = y_pred[i]
        f1 += f1_score(y_t,y_p)
    return f1/y_true.shape[0]
```

```
def compute_iou(y_true, y_pred, threshold=0.5):
    """
    Computes the Intersection over Union (IoU) for binary segmentation masks.

    Args:
        y_true (torch.Tensor): Ground truth tensor with values of 0 or 1,
representing points inside the mesh.
        y_pred (torch.Tensor): Prediction tensor with predicted probabilities.
        threshold (float): Threshold value for prediction to be considered as inside
the mesh. Defaults to 0.5.

    Returns:
        torch.Tensor: IoU score.
    """
    # Convert y_pred to binary mask based on threshold
    y_pred_binary = (y_pred > threshold).float()

    # Compute the intersection
    intersection = torch.sum(y_true * y_pred_binary)

    # Compute the union
    union = torch.sum(y_true) + torch.sum(y_pred_binary) - intersection

    # Compute the IoU
    iou = intersection / (union + 1e-6)  # Add a small epsilon to avoid division by
zero
    return iou

def test(dataloader,criterion,optim,model):
    model.eval()
    mse_score = 0
    accuracy_score = 0
    with torch.no_grad():
        for i,(img,X_test,y_test) in enumerate(dataloader):
            with torch.cuda.amp.autocast():
                pred = model(img.to('cuda'),X_test.to('cuda'))
            # loss = torch.sqrt(criterion(pred,y_test.to('cuda')))
                y_test = y_test.to('cuda')
                mse_score = mse_score +
MSE_METRIC(pred,y_test).item()/len(dataloader)
            accuracy = compute_iou(y_test,pred)
            accuracy_score = accuracy_score + accuracy/len(dataloader)
    log_metric("BCE_test", mse_score)
    log_metric("Accuracy", accuracy_score)
    return mse_score,accuracy_score


def train(train_dataloader,test_dataloader,criterion,optim,model,log_time,epoch):
    mse_running = 0
    mse_test = 0
    accuracy_score = 0
    accuracy_test = 0
    with tqdm(enumerate(train_dataloader)) as tepoch:
        for i,(img,X_train,y_train) in tepoch:
            model.train()
            tepoch.set_description(f"Epoch {epoch}")
            optim.zero_grad()
            with torch.cuda.amp.autocast():
                pred = model(img.to('cuda'),X_train.to('cuda'))
                y_train = y_train.to('cuda')
                loss = torch.sqrt(criterion(pred,y_train))
```

```
            scaler.scale(loss).backward()
            # loss.backward()
            scaler.step(optim)
            # optim.step()
            scaler.update()
            mse_running = mse_running
+MSE_METRIC(pred,y_train.to('cuda')).item()/len(train_dataloader)
            accuracy = compute_iou(y_train,pred)
            accuracy_score = accuracy_score + accuracy/len(train_dataloader)
            if (i % (log_time-1))==0 and i !=0:
                mse_test,accuracy_test = test(test_dataloader,criterion,optim,model)
                log_metric("BCE_train", mse_running)
                log_metric("Accuraracy_train", accuracy_score)

            tepoch.set_postfix(BCE_train =mse_running,BCE_test = mse_test, iou_train
= accuracy_score, iou_test = accuracy_test)

        scheduler.step()


epochs = 1000
print(epochs)
for epoch in range(epochs):

train(train_dataloader,test_dataloader,criterion,optim,model,len(train_dataloader),e
poch=epoch)


model.eval()


for threshold in range(1,10):
    print(f'threshold={threshold/10}')
    eval_uniform(test_df,num_points=50,use_bounding_box=
True,threshold=threshold/10)


def predict_labels_multiple_images(random_points,imgs_tensors):
    b_s = 1024*8
    print(random_points.shape)
    pred_label = []
    for i in range(0, random_points.shape[1], b_s):
        batch = random_points[:,i:i+b_s]
        with torch.no_grad():
            cuda_imgs_tensors = []
            for img_tensor in imgs_tensors:
                cuda_imgs_tensors.append(img_tensor.to('cuda'))
            batch = batch.to('cuda')
            predicted = model.predict_using_multiple_images(cuda_imgs_tensors,batch)
            predicted = torch.sigmoid(predicted)
            print(predicted.cpu().numpy()[0])
            pred_label.extend(predicted.cpu().numpy()[0])
    return pred_label


def predict_labels(random_points,img_tensor):
    b_s = 1024*8
    print(random_points.shape)
    pred_label = []
    for i in range(0, random_points.shape[1], b_s):
        batch = random_points[:,i:i+b_s]
        with torch.no_grad():
            img_tensor = img_tensor.to('cuda')
```

```
            batch = batch.to('cuda')
            predicted = model(img_tensor,batch)
            predicted = torch.sigmoid(predicted)
            print(predicted.cpu().numpy()[0])
            pred_label.extend(predicted.cpu().numpy()[0])
    return pred_label


img,obj,label =train_df.iloc[60]

img = Image.open(img)
img_tensor = transforms.PILToTensor()(img)
img_tensor = img_tensor[:3, :, :]
img_tensor = img_tensor/255.0
img_tensor = img_tensor.unsqueeze(0)
loaded_data = np.load(f"points_folder/{obj.split('/')[-1].split('.')[0]}.npz")

mesh = o3d.io.read_triangle_mesh(obj)
bbox = mesh.get_axis_aligned_bounding_box()

random_points = np.random.uniform(bbox.min_bound, bbox.max_bound, (500_000,
3)).astype(np.float32)
random_points = torch.from_numpy(random_points).unsqueeze(0)

pred_label =  predict_labels(random_points,img_tensor)
compute_iou(torch.tensor(compute_occupancy(mesh,random_points)),torch.tensor(pred_la
bel),0.5)


num_points = 100

# Create a 1D array of points along each axis
x = np.linspace(-1, 1, num_points).astype(np.float32)
y = np.linspace(-1, 1, num_points).astype(np.float32)
z = np.linspace(-1, 1, num_points).astype(np.float32)

# Create a 3D meshgrid from the 1D arrays
X, Y, Z = np.meshgrid(x, y, z, indexing='ij')

points = np.random.uniform(bbox.min_bound, bbox.max_bound, (500_000,
3)).astype(np.float32)
pred_labels = predict_labels(torch.tensor(points).unsqueeze(0),img_tensor)
# Flatten the meshgrid into a 3xN array of points, where N is the total number of
points
points = np.vstack((X.flatten(), Y.flatten(), Z.flatten())).T

points2 = points[np.array(pred_labels) > 0.9]

# Create Open3D PointCloud object
pcd = o3d.geometry.PointCloud()

# Set point cloud points and colors from NumPy arrays
pcd.points = o3d.utility.Vector3dVector(points2)
pcd.colors = o3d.utility.Vector3dVector(np.random.rand(num_points, 3))  # Random
colors

# Assign colors to points based on labels
label_colors = np.array([[1, 1, 1],  # Red
                         [0, 1, 0],  # Green
                         [0, 0, 1],  # Blue
                         [1, 1, 0],  # Yellow
                         [1, 0, 1]]) # Magenta
```

```
# pcd.colors =
o3d.utility.Vector3dVector(label_colors[(np.array(pred_labels)>0.5)*1])

pcd.estimate_normals()


tetra_mesh, pt_map = o3d.geometry.TetraMesh.create_from_point_cloud(pcd)
# for alpha in np.logspace(np.log10(0.5), np.log10(0.01), num=4):
alpha = 0.016
print(f"alpha={alpha:.3f}")
mesh_rec = o3d.geometry.TriangleMesh.create_from_point_cloud_alpha_shape(
    pcd, alpha, tetra_mesh, pt_map)
mesh_rec.compute_vertex_normals()
o3d.visualization.draw_geometries([mesh_rec], mesh_show_back_face=True)


mesh_out = mesh_rec.filter_smooth_simple(number_of_iterations=2)
mesh_out.compute_vertex_normals()
o3d.visualization.draw_geometries([mesh_out])

o3d.visualization.draw_geometries([mesh_out])



previous_experiments.py

import open3d as o3d
import numpy as np
import pandas as pd
import torch
from torch import nn
import itertools
from torch.nn import functional as F

from tqdm import tqdm
from mlflow import log_metric, log_param, log_artifacts


textured_mesh =
o3d.io.read_triangle_mesh("mesh-fusion/output/3_out/1c5a350ea0f55f793fbce9ec40e1f047
.obj")

aabb = textured_mesh.get_axis_aligned_bounding_box()
aabb.color = np.array([1,0,0])
object_points = np.asarray(aabb.get_box_points())

legacy_mesh = o3d.t.geometry.TriangleMesh.from_legacy(textured_mesh)
scene = o3d.t.geometry.RaycastingScene()
_ = scene.add_triangles(legacy_mesh)

c = textured_mesh.sample_points_poisson_disk(120_000)

bbox = textured_mesh.get_axis_aligned_bounding_box()
min_bound = bbox.min_bound
max_bound = bbox.max_bound
new_points = np.random.uniform(min_bound, max_bound, size=(120_000,
3)).astype(np.float32)

signed_distance = scene.compute_signed_distance(new_points).numpy()

object_pcd = o3d.geometry.PointCloud()
object_pcd.points = o3d.utility.Vector3dVector(new_points)
o3d.visualization.draw_geometries([textured_mesh,aabb,object_pcd])
```

```
df = pd.DataFrame(np.vstack([np.asarray(c.points)]),columns=['x1','x2','x3'])
df['distance'] = np.hstack([np.zeros(120_000)])


d = pd.DataFrame(new_points,columns=['x1','x2','x3'])
d['distance'] = signed_distance

df = pd.concat([df,d])

class RBF(nn.Module):
    """
    Transforms incoming data using a given radial basis function:
    u_{i} = rbf(||x - c_{i}|| / s_{i})
    Arguments:
        in_features: size of each input sample
        out_features: size of each output sample
    Shape:
        - Input: (N, in_features) where N is an arbitrary batch size
        - Output: (N, out_features) where N is an arbitrary batch size
    Attributes:
        centres: the learnable centres of shape (out_features, in_features).
            The values are initialised from a standard normal distribution.
            Normalising inputs to have mean 0 and standard deviation 1 is
            recommended.

        log_sigmas: logarithm of the learnable scaling factors of shape
(out_features).

        basis_func: the radial basis function used to transform the scaled
            distances.
    """

    def __init__(self, in_features, out_features, basis_func,centers=None):
        super(RBF, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        if centers is None:
            self.centres = nn.Parameter(torch.Tensor(out_features, in_features))
        else:
            self.centres = torch.Tensor(centers).to('cuda')
        self.log_sigmas =
torch.Tensor([0.5]*out_features).to('cuda')#nn.Parameter(torch.Tensor(out_features))
        self.basis_func = basis_func
        self.reset_parameters()

    def reset_parameters(self):
        # nn.init.normal_(self.centres, 0, 1)
        # nn.init.constant_(self.log_sigmas, 0.5)
        pass

    def forward(self, input):
        size = (input.size(0), self.out_features, self.in_features)
        x = input.unsqueeze(1).expand(size)
        c = self.centres.unsqueeze(0).expand(size)
        distances = (x - c).pow(2).sum(-1).pow(0.5) /
torch.exp(self.log_sigmas).unsqueeze(0)
        return self.basis_func(distances)

# RBFs

def gaussian(alpha):
    phi = torch.exp(-1*alpha.pow(2))
    return phi
```

```
def linear(alpha):
    phi = alpha
    return phi

def quadratic(alpha):
    phi = alpha.pow(2)
    return phi

def inverse_quadratic(alpha):
    phi = torch.ones_like(alpha) / (torch.ones_like(alpha) + alpha.pow(2))
    return phi

def multiquadric(alpha):
    phi = (torch.ones_like(alpha) + alpha.pow(2)).pow(0.5)
    return phi

def inverse_multiquadric(alpha):
    phi = torch.ones_like(alpha) / (torch.ones_like(alpha) + alpha.pow(2)).pow(0.5)
    return phi

def spline(alpha):
    phi = (alpha.pow(2) * torch.log(alpha + torch.ones_like(alpha)))
    return phi

def poisson_one(alpha):
    phi = (alpha - torch.ones_like(alpha)) * torch.exp(-alpha)
    return phi

def poisson_two(alpha):
    phi = ((alpha - 2*torch.ones_like(alpha)) / 2*torch.ones_like(alpha)) \
    * alpha * torch.exp(-alpha)
    return phi

def matern32(alpha):
    phi = (torch.ones_like(alpha) + 3**0.5*alpha)*torch.exp(-3**0.5*alpha)
    return phi

def matern52(alpha):
    phi = (torch.ones_like(alpha) + 5**0.5*alpha + (5/3) \
    * alpha.pow(2))*torch.exp(-5**0.5*alpha)
    return phi

def triharmonic_spline(alpha):
    return alpha**3

def basis_func_dict():
    """
    A helper function that returns a dictionary containing each RBF
    """

    bases = {'gaussian': gaussian,
            'linear': linear,
            'quadratic': quadratic,
            'inverse quadratic': inverse_quadratic,
            'multiquadric': multiquadric,
            'inverse multiquadric': inverse_multiquadric,
            'spline': spline,
            'poisson one': poisson_one,
            'poisson two': poisson_two,
            'matern32': matern32,
            'matern52': matern52,
            'thriharmonic_spline':triharmonic_spline}
```

```
    return bases


class ResidualBlock(torch.nn.Module):
    def __init__(self,module):
        super().__init__()

        self.module = module

    def forward(self,x):
        return self.module(x) + x



class Network(nn.Module):
    def __init__(self, in_features, out_features, basis_func,centers):
        super(Network, self).__init__()
        self.RBF = RBF(in_features, out_features, basis_func,centers)
        # self.RBF = nn.Linear(3,out_features)
        # self.dropout = torch.nn.Dropout()
        if centers is not None:
            out_features = centers.shape[0]
        self.fc = nn.Linear(out_features,512)
        self.fc2 = nn.Linear(512,256)
        self.fc3 = nn.Linear(256,64)
        self.bn =nn.BatchNorm1d(64)
        self.fc4 = nn.Linear(64,1)
        # self.fc5 = nn.Linear(32,1)
#
    def forward(self,x):
        x = self.RBF(x)
        # x = torch.nn.functional.mish(x)
        # x = self.dropout(x)
        # x = torch.nn.functional.mish(x)
        x = self.fc(x)
        x = torch.nn.functional.mish(x)
        x = self.fc2(x)
        x = torch.nn.functional.mish(x)
        x = self.fc3(x)
        x = self.bn(x)
        x = torch.nn.functional.mish(x)
        x = self.fc4(x)
        # x = torch.nn.functional.mish(x)
        # x = self.fc5(x)
        return x



def get_centers(per_axis = 5,on_surface = True):
    if on_surface:
        centers =
df[df['distance']==0].sample(frac=1)[:per_axis**3].drop(columns=['distance']).values
    else:
        x = np.linspace(object_points[:,0].min(),object_points[:,0].max(),per_axis)
        y = np.linspace(object_points[:,1].min(),object_points[:,1].max(),per_axis)
        z = np.linspace(object_points[:,2].min(),object_points[:,2].max(),per_axis)

        centers = np.array(list(itertools.product(*[x,y,z])))
    return centers

centers = get_centers(per_axis=10,on_surface=False)

net = Network(3,centers.shape[0],basis_func_dict()['poisson two'],centers)
```

```
net.to('cuda')

class Dataset(torch.utils.data.Dataset):
    def __init__(self,df):
        self.df = df

    def __len__(self):
        return self.df.shape[0]

    def __getitem__(self,idx):
        row = self.df.iloc[idx].values
        x = torch.Tensor(row[:-1])
        y = torch.Tensor([row[-1]])
        return x,y

df = df.sample(frac=1)
train_dataset = Dataset(df[:int(df.shape[0]*0.7)])
test_dataset = Dataset(df[int(df.shape[0]*0.7):])


train_dataloader = torch.utils.data.DataLoader(train_dataset,batch_size=512*32)
test_dataloader = torch.utils.data.DataLoader(test_dataset,batch_size=512*32)

class MyLoss(torch.nn.Module):
    def __init__(self,zero_weights=0.7):
        super(MyLoss, self).__init__()
        self.zero_weights = zero_weights

    def forward(self,output, target):
        zeros_idx = torch.logical_or(torch.abs(target) < 0.005,torch.abs(output) <
0.005)
        not_zero_idx = torch.logical_not(zeros_idx)
        not_zeros_mse = 0
        zeros_mse = 0
        # print(output.shape,zeros_idx.shape,not_zero_idx.shape)
        # print(target[not_zero_idx].shape)
        # print(torch.sub(output[zeros_idx],target[zeros_idx]))
        if target[zeros_idx].shape[0] != 0:
            zeros_mse = F.mse_loss(output[zeros_idx],target[zeros_idx])#
torch.mean(torch.square(torch.sub(output[zeros_idx],target[zeros_idx]))) + 1e-7
        if target[not_zero_idx].shape[0] != 0:
            not_zeros_mse =
F.mse_loss(output[not_zero_idx],target[not_zero_idx])#torch.mean(torch.square(torch.
sub(output[not_zero_idx],target[not_zero_idx]))) + 1e-7

        if torch.isnan(zeros_mse):
            raise Exception(f'zeros_mse is Nan, {output[zeros_idx]} ,
{target[zeros_idx]}')


        if torch.isnan(zeros_mse):
            raise Exception(f'not_zeros_mse is Nan, {output[not_zeros_mse]} ,
{target[not_zeros_mse]}')

        # close_mse =
        return zeros_mse * self.zero_weights *zeros_idx.sum()/(output.shape[0]+1) \
            +
not_zeros_mse*(1-self.zero_weights)*not_zero_idx.sum()/(output.shape[0]+1)


optim = torch.optim.Adam(net.parameters(),lr=1e-2)
# optim = torch.optim.SGD(net.parameters(),lr=1e-2)
criterion = torch.nn.MSELoss()##MyLoss()#torch.nn.L1Loss() #MyLoss()##
```

```
scheduler =
torch.optim.lr_scheduler.ReduceLROnPlateau(optim,patience=5,threshold=0.001,
factor=0.2,verbose=True)
# scheduler = torch.optim.lr_scheduler.CyclicLR(optim,base_lr =
1e-2,max_lr=1e-1,step_size_up=100,step_size_down=300)

MSE_METRIC = torch.nn.MSELoss()

def test(dataloader,criterion,optim,model):
    model.eval()
    mse_score = 0
    with torch.no_grad():
        for i,(X_test,y_test) in enumerate(dataloader):
            pred = model(X_test.to('cuda'))
            # loss = torch.sqrt(criterion(pred,y_test.to('cuda')))
            mse_score = mse_score +
torch.sqrt(MSE_METRIC(pred,y_test.to('cuda'))).item()/len(dataloader)
    return mse_score

def train(train_dataloader,test_dataloader,criterion,optim,model,log_time,epoch):
    mse_running = 0
    mse_test = 0
    with tqdm(enumerate(train_dataloader)) as tepoch:
        for i,(X_train,y_train) in tepoch:
            model.train()
            tepoch.set_description(f"Epoch {epoch}")
            optim.zero_grad()
            pred = model(X_train.to('cuda'))
            loss = torch.sqrt(criterion(pred,y_train.to('cuda')))
            loss.backward()
            optim.step()
            mse_running = mse_running +
torch.sqrt(MSE_METRIC(pred,y_train.to('cuda'))).item()/len(train_dataloader)

            if (i % (log_time-1))==0 and i !=0:
                mse_test = test(test_dataloader,criterion,optim,model)
                log_metric("Mean batch RMSE Test", mse_running)

            tepoch.set_postfix(RMSE_train =mse_running,RMSE_test = mse_test)
        log_metric("Mean batch RMSE Train", mse_running)
        scheduler.step(mse_test)


for epoch in range(100):

train(train_dataloader,test_dataloader,criterion,optim,net,len(train_dataloader),epo
ch=epoch)

net.eval()

points = []
while len(points) < 20000:
    x =
np.random.uniform(low=object_points[:,0].min()-0.05,high=object_points[:,0].max()+0.
05,size=1).astype(np.float32)
    y =
np.random.uniform(low=object_points[:,1].min()-0.05,high=object_points[:,1].max()+0.
05,size=1).astype(np.float32)
    z =
np.random.uniform(low=object_points[:,2].min()-0.05,high=object_points[:,2].max()+0.
05,size=1).astype(np.float32)

    random_points = torch.Tensor(np.stack([x,y,z],axis=1)).to('cuda')
```

```
    if (torch.abs(net(random_points)) < 0.001):
        points.append(random_points.cpu().numpy())
        print(len(points))

object_pcd = o3d.geometry.PointCloud()
object_pcd.points = o3d.utility.Vector3dVector(np.vstack(points))
o3d.visualization.draw_geometries([aabb,object_pcd,textured_mesh])
```

ДОДАТОК Б

Презентаційні матеріали

# Дипломна робота на здобуття ступеня бакалавра на тему: «Відновлення тривимірних об’єктів за фотографією»

Виконав: Андрушко Н.Д., КА-94, НН “ІПСА”

Дипломний керівник: професор д.т.н Данилов Валерій Якович

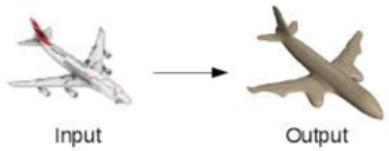
## Вступ

Об’єкт дослідження полягає у вивченні різних методів реконструкції тривимірних моделей.

Предмет дослідження - використання штучних нейронних мереж для вирішення проблем, пов’язаних із тривимірною реконструкцією за допомогою зображень.

Мета роботи - вивчення сучасних підходів до реконструкції об’єктів у тривимірному середовищі.

# Постановка завдання

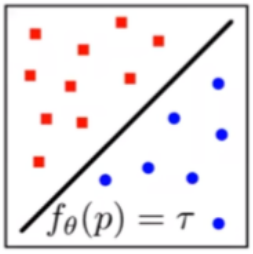



1. Дослідити існуючі методи для тривимірної реконструкції
2. Розробити метод на для тривимірної реконструкції
3. Проаналізувати отримані результати

## Актуальність роботи

1. Візуалізація реального простору: Тривимірна реконструкція дозволяє точно відтворити об'єкти та їх поведінку у тривимірному просторі. Це дозволяє нам отримати більш реалістичне уявлення про предмети та їхні взаємовідношення. Наприклад, в архітектурі, це дозволяє візуалізувати будівлі з усіх кутів та розуміти їх розташування в контексті оточуючого середовища.

2. Показ деталей та особливостей: Тривимірна реконструкція дозволяє передати поверхневі деталі об'єктів, такі як текстури, кольори та форми. Це надає можливість краще розглянути та розуміти дрібні деталі, які важко сприймати на двовимірних зображеннях. Наприклад, у медицині, тривимірна реконструкція допомагає лікарям більш докладно досліджувати внутрішні органи та знаходити можливі проблеми.

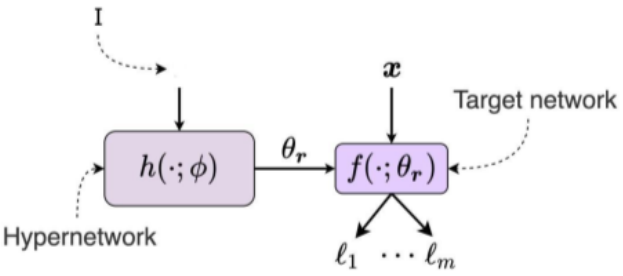



## Представлення за допомогою неявної функції

В цьому методі об'єкт описується математичною функцією, яка визначає наявність або відсутність об'єкта в кожній точці 3D простору. Функція може приймати будь-яке значення в певному діапазоні, наприклад, від -1 до 1, де значення близьке до 1 вказує на наявність об'єкта, а значення близьке до -1 вказує на відсутність об'єкта, і значення функції відстані між цими значеннями відображає неоднозначність або невизначеність наявності об'єкта.

## Використаний підхід

Для вирішення задачі було вирішено взяти за основу метод Occupancy Networks з використанням гіпермереж. Таким чином задача нейронної мережі – прогнозування належності точок до об’єкту (бінарна класифікація).



# Попередні експерименти, тренування мережі

Спочатку було вирішено провести попередні експерименти для вибору архітектури однієї з можливих нейронних мереж, а саме target мережі, яка приймає точку, та прогнозує її належність до об'єкту, або відстань до нього.

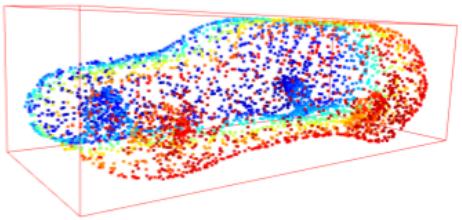
Під час проведення експериментів використовувався набір даних ShapeNet. Саму задачу в попередніх експериментах можна описати наступним чином: маючи N тривимірних точок всередині обмежувальної рамки, в якій знаходиться об'єкт натренувати мережу так, щоб вона достатньо точно прогнозувала відстань від об'єкта до точки, тоді саму реконструкцію можна зробити обравши точки, які лежать на досить малій відстані від об'єкта і застосувавши пост-обробку.

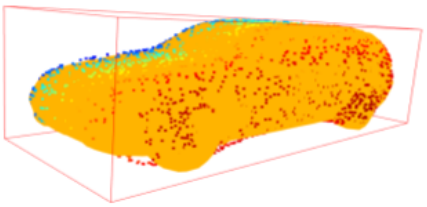

# Попередні експерименти

В попередніх експериментах таргет мережа прогнозувала відстань. Було отримано наступні результати


| Кількість РБФ центрів | РБФ функція | $fc_1$ | $fc_2$ | $fc_3$ | $fc_4$ | $fc_5$ | Test RMSE | Train RMSE |
|---|---|---|---|---|---|---|---|---|
| 3375 | Пуассона | 128 | 64 | 64 | 32 | 1 | 0.0101 | 0.0101 |
| 3375 | Гаусса | 128 | 64 | 64 | 32 | 1 | 0.0194 | 0.0196 |
| 1000 | Пуассона | 256 | 128 | 64 | 32 | 1 | 0.009 | 0.009 |
| 1000 | Пуассона | 128 | 64 | 64 | 32 | 1 | **0.0046** | **0.0046** |
| 1000 | Гаусса | 128 | 64 | 64 | 32 | 1 | 0.018 | 0.018 |
| 1000 | Гаусса | 256 | 128 | 64 | 32 | 1 | 0.0178 | 0.0176 |
| 125 | Гаусса | 256 | 128 | 64 | 32 | 1 | 0.0184 | 0.0184 |
| 125 | Пуассона | 128 | 64 | 64 | 32 | 1 | 0.0096 | 0.0097 |
| 1000 | Пуассона | 128 | 64 | 64 | 1 | - | 0.014 | 0.014 |
| 1000 | Пуассона | 512 | 256 | 64 | 1 | - | 0.0136 | 0.0137 |

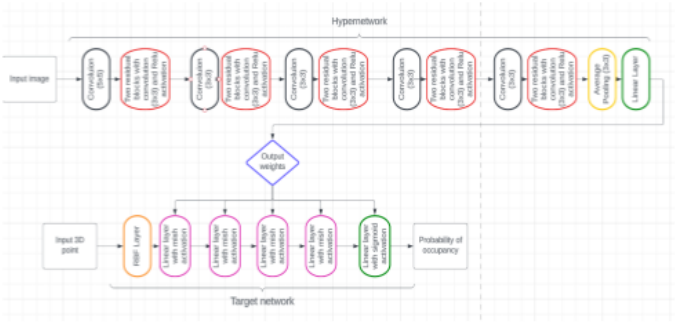
## Приклад використання лише target мережі





## Розроблена архітектура

В попередніх експериментах задача формулювалась як прогнозування відстані, але під час розробки повноцінної мережі як з мережею h, так і мережею f було виявлено, що така мережа занадто погано узагальнюється для випадку з прогнозуванням відстані, тому все ж таки було вирішено змінити підхід, до прогнозування належності точки об’єкту, сама архітектура моделі f не змінилась.



# Навчання мережі

Процес навчання включає кілька ключових компонентів, включаючи градієнтний масштабувальник, оптимізатор, функцію втрат і планувальник швидкості навчання. Під час тренування використовувалась аугментація даних , а саме вертикальне та горизонтальне відображення вхідного зображення, що збільшує стабільність моделі.

Функція втрати - BCEWithLogitsLoss.

Планувальник темпів навчання - CyclicLR.

Оптимізатор і швидкість навчання - SGD(lr=1e-5)

# Результати

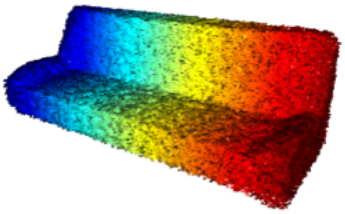
Для оцінки результатів було підраховано метрику IoU, двома способами. Перший спосіб - оцінка метрики IoU на сітці з 50^3 точок в кубі -1,1. Другий спосіб - оцінка метрики IoU на 100 тисяч випадково обраних точок з обмежувальної рамки, такий спосіб використовували в Occupancy Networks. Також через обмеження в потужності графічного процесору, метрика була виміряна лише на декількох класах.

| IoU | | | |
|---|---|---|---|
| category | Our, in cube with 50^3 grid | Our, in bounding box with 100k random points | Occupancy Network |
| airplane | - | - | 0.571 |
| bench | 0.401 | 0.412 | **0.485** |
| cabinet | 0.650 | **0.751** | 0.733 |
| car | 0.583 | 0.702 | **0.737** |
| chair | - | - | 0.501 |
| display | - | - | 0.471 |
| lamp | - | - | 0.371 |
| loudspeaker | - | - | 0.647 |
| rifle | - | - | 0.474 |
| sofa | 0.565 | **0.784** | 0.680 |
| table | 0.406 | 0.453 | **0.506** |
| telephone | - | - | 0.720 |
| vessel | - | - | 0.530 |
| mean | 0.520 | **0.584** | 0.571 |

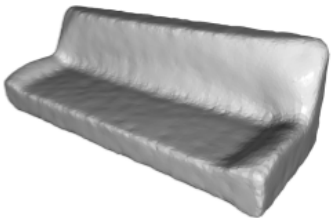
# Приклад реконструкції

При реконструкції спочатку випадковим чином обираються N точок, для яких потім обчислюється належність до тривимірної моделі. Після чого, використовується метод реконструкції з точок (Alpha Shapes). Після чого додатково використовується фільтр для усереднення, котрий зміщує поточу вершину тривимірної моделі до її середнього арифметичного з прилеглими вершинами, що дозволяє отримати більш гладкий об’єкт.

Вхідне зображення



Вихід нейронної мережі



Пост-оброка виходу мережі

# Висновок

Підводячи підсумок:

- Було проаналізовано наявні методи, та обрано метод, який в подальшому був модифікований.
- Реалізовано алгоритм-метод та показано результат його роботи.
- Розроблена методика продемонструвала кращу продуктивність порівняно з існуючими методами в окремих класах, однак слід зазначити, що модель не була навчена на всіх класах, охоплених порівняльним методом.

# Дякую за увагу!