**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**Факультет комп’ютерних наук та кібернетики**

**Кафедра інтелектуальних програмних систем**

Звіт до лабораторної роботи

З предмету Математичні основи обчислювальної геометрії

на тему:

**Реконструкція поверхні методом Пуассона**

Виконали студенти 4-го курсу

Баликов Антон Олексійович,

Волчецький Руслан Васильович,

Саворона Назарій Вікторович

**Київ – 2022**

**Зміст**

[Постановка задачі 3](#_Toc120838979)

[Теоретичні відомості 3](#_Toc120838980)

[Неявна поверхня на основі вокселів 3](#_Toc120838981)

[Характеристичні функції твердих тіл 5](#_Toc120838982)

[Пуассонівська реконструкція поверхні 6](#_Toc120838983)

[Градієнти на регулярній сітці 8](#_Toc120838984)

[Індексація 3D масивів](#_Toc120838985)

[Вхідні нормалі можуть не знаходитися у вузлах сітки 12](#_Toc120838986)

[Вибір хорошого ізо-значення 14](#_Toc120838987)

[Наскільки це завдання спрощує [Kazhdan et al. 2006]? 15](#_Toc120838988)

[Скріншоти результатів роботи програми 16](#_Toc120838989)

[Література](#_Toc120838990)

# Постановка задачі

З поданої множини точок та нормалей у 3D отримати апроксимацію поверхні, заданої цією хмарою точок, методом реконструкції Пуассона.

# Теоретичні відомості

У цьому завданні ми будемо реалізовувати “спрощену” версію методу, описаного в роботі ["Poisson Surface Reconstruction" by Kazhdan et al. 2006](https://hhoppe.com/poissonrecon.pdf).

При дослідженні певного об’єкту найчастіше різноманітні технології сканування отримують на виході набір з точок на поверхні цього ж об’єкта. З цих точок і, можливо, розташування камери, можна також оцінити нормалі до поверхні для кожної точки .

Для аналізу форми, візуалізації та інших подальших етапів геометричної обробки ми хотіли б перетворити дані хмари точок (Point cloud) з кінцевою вибіркою в явне безперервне представлення поверхні: тобто в сітку трикутників (окремий випадок сітки багатокутників).

## Неявна поверхня на основі вокселів

Перетворення хмари точок безпосередньо в трикутну сітку робить дуже важким забезпечення відповідності сітки певним топологічним постулатам: тобто, щоб вона була різноманітною, замкненою, і мала малу кількість “дірок”, або “[родів](https://uk.wikipedia.org/wiki/%D0%A0%D1%96%D0%B4_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))” у термінах топології.

Замість цього ми спочатку перетворимо представлення вибірки хмари точок в неявне представлення поверхні: тут невідома поверхня визначається як [множина рівня](https://uk.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD%D0%B0_%D1%80%D1%96%D0%B2%D0%BD%D1%8F) деякої функції, тобто відображає всі точки простору в скалярну величину. Наприклад, ми можемо задати поверхню певної щільної (суцільної), об'ємної форми , представленою у вигляді всіх точок таких, що , де ми можемо довільно встановлювати .

.

Програмно дискретизувати неявну функцію нескладно. Визначимо регулярну 3D сітку вокселів, що містить принаймні [обмежувальну область](https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D0%BD%D1%96%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D1%83%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BA%D0%BE%D1%80%D0%BE%D0%B1%D0%BA%D0%B0) S. Воксель (від англ. Volume та англ. pixel) — елемент простору, містить значення певної величини в клітинках рівномірної просторової сітки. У кожному вузлі сітки (вокселі) ми зберігаємо значення неявної функції .

A picture containing graphical user interface

Description automatically generatedТаким чином ми визначаємо g скрізь на сітці через [трилінійну інтерполяцію](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D0%BB%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D1%96%D1%8F).

A picture containing indoor, computer

Description automatically generated*Рис.1 Вісім кутових точок на кубі навколо точки інтерполяції C.*

*Рис.2 Зображення 3D інтерполяції.*

Наприклад, скажімо є точка **,** що лежить усередині крайньої нижньої, передньої, лівої клітинки. Значення у восьми кутах відомі. Трилінійну інтерполяцію можна розуміти як лінійну інтерполяцію у напрямку на по кожному -напрямленому ребру, що дасть нам 4 можливих значення в одній площині. Потім вони можуть бути лінійно інтерпольовані в -напрямку на, що дасть 2 точки на одній прямій, і, нарешті, у напрямку на , щоб отримати нашу бажану точку **.**

Неявна поверхня, що зберігається як множина рівня трилінійно інтерпольованої сітки, може бути перетворена в трикутну сітку за допомогою алгоритму [Marching Cubes](https://uk.wikipedia.org/wiki/Marching_cubes).

## Характеристичні функції твердих тіл

Ми припускаємо, що наш набір точок лежить на поверхні деякого фізичного твердого об’єкту . Цей твердий об'єкт повинен мати деякий нетривіальний об'єм, який ми можемо абстрактно обчислити як інтеграл одиничної густини по твердому тілу:

**.**

Ми можемо переписати цей визначений інтеграл як невизначений інтеграл по всіх :

**,**

, попередньо ввівши характеристичну функцію від **S**, яка дорівнює одиниці для точок всередині фігури і нулю для точок поза нею:

На відміну від типових неявних поверхневих функцій, ця функція представляє поверхню фігури як розрив між одиничними та нульовими значеннями. Цікаво, що градієнт характеристичної функції не визначена вздовж .

Одне з ключових спостережень, зроблених в роботі [Kazhdan et al. 2006], полягає в тому, що градієнт нескінченно гладкої характеристичної функції:

* напрямлений в напрямку нормалі біля поверхні,
* 0 в усіх інших точках.

Нашою метою є використання точок і нормалей для оптимізації неявної функції на регулярній сітці так, щоб її градієнт задовольняв цим двом критеріям. Таким чином, є апроксимацією гладкої характеристичної функції.

# Пуассонівська реконструкція поверхні

Для початку зробимо два припущення:

1. ми знаємо, як обчислити в кожному вузлі ,
2. усі вхідні точки ідеально лежать у вузлах сітки: **.**

Ми з'ясуємо, що ці припущення не є реалістичними і нам доведеться їх послабити. Однак, це полегшить наступний алгоритмічний опис.

Якщо наші точки лежать в точках сітки, то наші відповідні нормалі також лежать в точках сітки. Це призводить до дуже простого набору лінійних рівнянь для визначення функції з градієнтом, що дорівнює поверхневій нормалі на поверхні і нульовим градієнтом поза поверхнею:

Це рівняння з векторними величинами. Градієнти, нормалі та нульові вектори є тривимірними (наприклад, ). Фактично, це три рівняння для кожного вузла сітки.

Оскільки нам потрібно лише одне число в кожному вузлі сітки (значення ), ми маємо занадто багато рівнянь.

Як і багато алгоритмів геометричної обробки, що стикаються з такою надмірною визначеністю, ми будемо оптимізовувати розв'язок, який найкращим чином мінімізує похибку рівняння:

**.**

Ми будемо розглядати похибку кожної вузла сітки однаково, мінімізуючи суму по всіх вузлах сітки:

де **,** за яким ми мінімізуємо, це вектор невідомих значень у вузлах сітки, де **.**

Частково зручність роботи на регулярній сітці полягає в тому, що ми можемо використовувати метод скінченних різниць для апроксимації градієнта на сітці.

Переглянувши наші припущення, ми зможемо обчислити наближення -, - і -компонент через розріджене матричне множення "градієнтної матриці" і нашого вектора невідомих значень сітки . Ми зможемо записати поставлену вище задачу мінімізації в матричній формі:

Або, після розкриття норми:

Це так звана квадратична "функція енергії" від змінних **g**, її мінімум має місце тоді, коли нескінченно мала зміна в **g** не призводить до зміни енергії:

Застосування цієї похідної дає нам розріджену систему лінійних рівнянь

Тепер повернемося до наших припущень.

# Градієнти на регулярній сітці

Градієнт функції в 3D - це ніщо інше, як вектор, що містить часткові похідні в кожному напрямку координат:

Будемо апроксимувати кожну частинну похідну окремо. Розглянемо частинну похідну в напрямку , , і будемо вважати без втрати загальності, що отримані результати симетрично справедливі для і .

Часткова похідна в напрямку **x** є одновимірною похідною. Найпростіше це зробити за допомогою скінченних різниць. Наближено похідна функції за напрямком є різницею між функцією, обчисленою в одному вузлі сітки та вузлі сітки перед ним в напрямку **x**, поділеною на просторову відстань між сусідніми вузлами (тобто, на розмір кроку сітки):

де ми використовуємо індекс для того, щоб вказати, що ця похідна в напрямку живе на “східчастій сітці” ([staggered grid](https://docs.juliahub.com/CartesianGrids/jCgtm/0.1.7/manual/overview/)) в проміжках між вузлами сітки, де знаходяться значення функції .

A picture containing light, wire, colorful

Description automatically generated*Рис.3 Сітка у 2D, значення* ***g*** *відомі у вузлах сітки 5x5.*

A picture containing text, map, wire, light

Description automatically generated

*Рис.4 Значення частинних похідних* ***g*** *у напрямку* ***x*** *лежатимуть у вузлах* ***зеленої*** *сітки 4x5.*

Diagram

Description automatically generated *Рис.5 Значення частинних похідних* ***g*** *у напрямку* ***y*** *лежатимуть у вузлах* ***жовтої*** *сітки 5x4*

Поклавши як вектор-стовпець значень функції на первинній сітці (синій колір на рисунках прикладу), можна побудувати розріджену матрицю таку, що кожен рядок обчислює часткову похідну у відповідному вузлі східчастої сітки. -те входження у цьому рядку отримує значення тільки для сусідніх вузлів первинної сітки:

## Індексація 3D масивів

Певна річ, в нашому коді ми не можемо індексувати колонки вектора **g** трійками чисел або рядки трійками. Ми будемо вважати, що розкривається як . Схожим чином для індексів у східчастій сітці ми будемо вважати, що позначає елемент матриці , де округлено донизу.

Аналогічно можна побудувати матриці і та скласти ці матриці вертикально, щоб отримати градієнтну матрицю :

Наступна частина коду реалізує описаний алгоритм:

def fd\_partial\_derivative(nx, ny, nz, h, direction):  
 primary\_grid\_idx = np.arange(nx \* ny \* nz).reshape((nx, ny, nz))   
  
 if direction == "x":   
 num\_staggered\_grid = (nx - 1) \* ny \* nz  
 col\_idx = np.concatenate((primary\_grid\_idx[1:, :, :].flatten(), primary\_grid\_idx[:-1, :, :].flatten()))  
 elif direction == "y":   
 num\_staggered\_grid = nx \* (ny - 1) \* nz  
 col\_idx = np.concatenate((primary\_grid\_idx[:, 1:, :].flatten(), primary\_grid\_idx[:, :-1, :].flatten()))  
 elif direction == "z":   
 num\_staggered\_grid = nx \* ny \* (nz - 1)  
 col\_idx = np.concatenate((primary\_grid\_idx[:, :, 1:].flatten(), primary\_grid\_idx[:, :, :-1].flatten()))  
  
 row\_idx = np.arange(num\_staggered\_grid)  
 row\_idx = np.tile(row\_idx, 2)  
 data\_term = [1 / h] \* num\_staggered\_grid + [-1 / h] \* num\_staggered\_grid  
 D = coo\_matrix((data\_term, (row\_idx, col\_idx)), shape=(num\_staggered\_grid, nx \* ny \* nz))  
 return D  
  
  
def fd\_grad(nx, ny, nz, hx, hy, hz):  
 Dx = fd\_partial\_derivative(nx, ny, nz, hx, "x")  
 Dy = fd\_partial\_derivative(nx, ny, nz, hy, "y")  
 Dz = fd\_partial\_derivative(nx, ny, nz, hz, "z")  
 return vstack((Dx, Dy, Dz))

Це означає, що наш вектор **v** нулів і нормалей в нашій задачі мінімізації не повинен лежати на первинній сітці, а навпаки, він теж повинен бути розбитий на **x**-, **y**- і **z**- компоненти, які лежать на своїх відповідних східчастих сітках:

Це призводить до розгляду нашого другого припущення.

# Вхідні нормалі можуть не знаходитися у вузлах сітки

На цьому етапі ми б хотіли, щоб наші вхідні нормалі були подані покомпонентно на східчастій сітці. Тоді ми могли б одразу вставити їх у . Але це не має особливого сенсу, оскільки кожна нормаль знаходиться у відповідній точці , незалежно від будь-яких сіток.

Щоб виправити це, ми розподілимо кожну компоненту кожної вхідної нормалі до у відповідному вузлі східчастої сітки.

Наприклад, розглянемо нормаль у деякій точці Концептуально, ми будемо думати про *x*-компоненту нормалі як про плаваючу в східчастій сітці що відповідає , між вісьмома вузлами східчастої сітки:

Кожен із цих вузлів східчастої сітки має відповідне значення у векторі .

Ми розподілимо між цими входженнями у , додавши частину до кожного. Тобто,

Де вага трилінійної інтерполяції, пов’язана з вузлом східчастої сітки для інтерполяції значення в точці **p**. Трилінійна інтерполяція має таку вагу, що:

Оскільки нам потрібно зробити це для *x*-компоненти кожної вхідної нормалі, ми зберемо розріджену матрицю , яка інтерполює  у кожній точці :

транспонування є не зовсім його оберненням, натомість його можна інтерпретувати як розподіл значень на вузли східчастої сітки, де лежить :

Використовуючи це визначення і аналогічно для і , ми можемо побудувати вектор у нашій задачі мінімізації енергії вище.

Наступна частина коду реалізує описаний алгоритм:

Wx = trilinear\_interpolation\_weights(nx, ny, nz, bottom\_left\_front\_corner, P, hx, hy, hz, direction="x")  
Wy = trilinear\_interpolation\_weights(nx, ny, nz, bottom\_left\_front\_corner, P, hx, hy, hz, direction="y")  
Wz = trilinear\_interpolation\_weights(nx, ny, nz, bottom\_left\_front\_corner, P, hx, hy, hz, direction="z")  
W = trilinear\_interpolation\_weights(nx, ny, nz, bottom\_left\_front\_corner, P, hx, hy, hz)

# distribute the normal vectors to staggered grids  
vx = Wx.T @ N[:, 0]  
vy = Wy.T @ N[:, 1]  
vz = Wz.T @ N[:, 2]  
v = np.concatenate([vx, vy, vz])

# Вибір хорошого ізо-значення

Константні функції не мають градієнта. Це означає, що ми можемо додати константну функцію до нашої неявної функції , не змінюючи її градієнт:

Те ж саме вірно для нашої дискретної градієнтної матриці : якщо вектор значень сітки постійний, то буде вектором нулів.

Це потенційно проблематично для нашого розв’язання методом найменших квадратів: існує багато рішень, оскільки ми можемо просто додати константу. На щастя, для нас це має не велике значення. Досить елегантно сказати, що наша поверхня визначена як , але ми були б не менш щасливі, заявивши, що наша поверхня визначена як .

Для цього нам просто потрібно знайти рішення, а потім вибрати хороше ізо-значення .

Як запропоновано в [Kazhdan et al. 2006], ми можемо вибрати гарне ізо-значення, інтерполюючи наше рішення **g** у кожній із вхідних точок (оскільки ми знаємо, що вони на поверхні) та усереднюючи їхні значення. Для відповідної інтерполяційної матриці **W** на первинній (не-східчастій) сітці це можна записати так:

,

де це вектор одиниць.

Наступна частина коду реалізує описаний алгоритм:

# align the zero iso surface with the input point cloud  
sigma = np.mean(W @ g)  
g -= sigma  
  
# extract the mesh using marching cubes  
g\_field = g.reshape(nx, ny, nz)  
vertices, triangles = mcubes.marching\_cubes(g\_field, 0)

# Наскільки це завдання спрощує [Kazhdan et al. 2006]?

Окрім наведених вище думок, великий внесок [Kazhdan et al. 2006] було налаштувати та вирішити цю проблему на адаптивній сітці, а не на звичайній сітці. Вони також відстежують «достовірність» своїх вхідних даних, впливаючи на те, як вони згладжують та інтерполюють значення. Як результат, їхній метод є одним із найбільш використовуваних методів реконструкції поверхні донині.

# Скріншоти результатів роботи програми

A statue of a bird

Description automatically generated with low confidenceA close-up of a stone

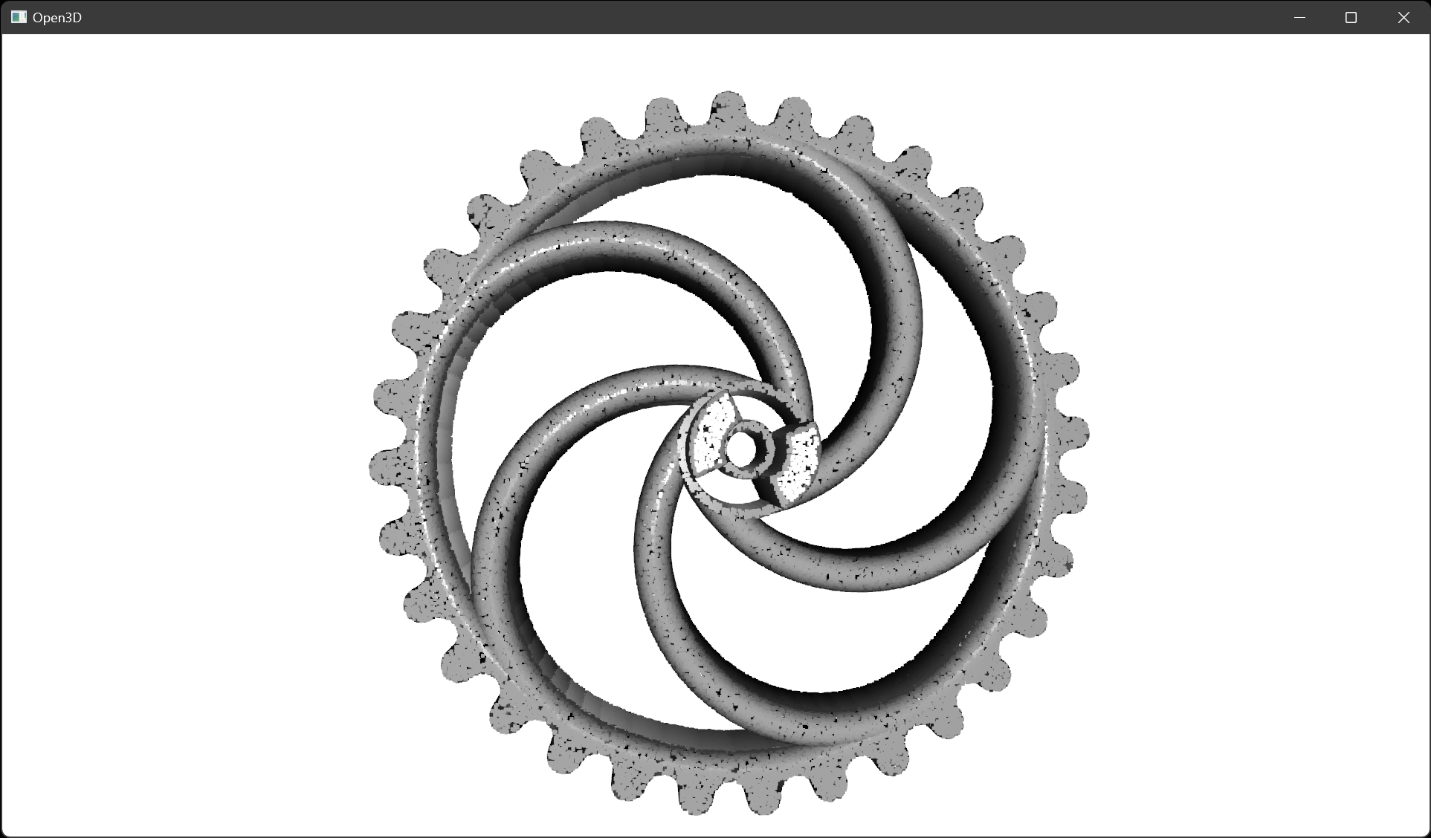
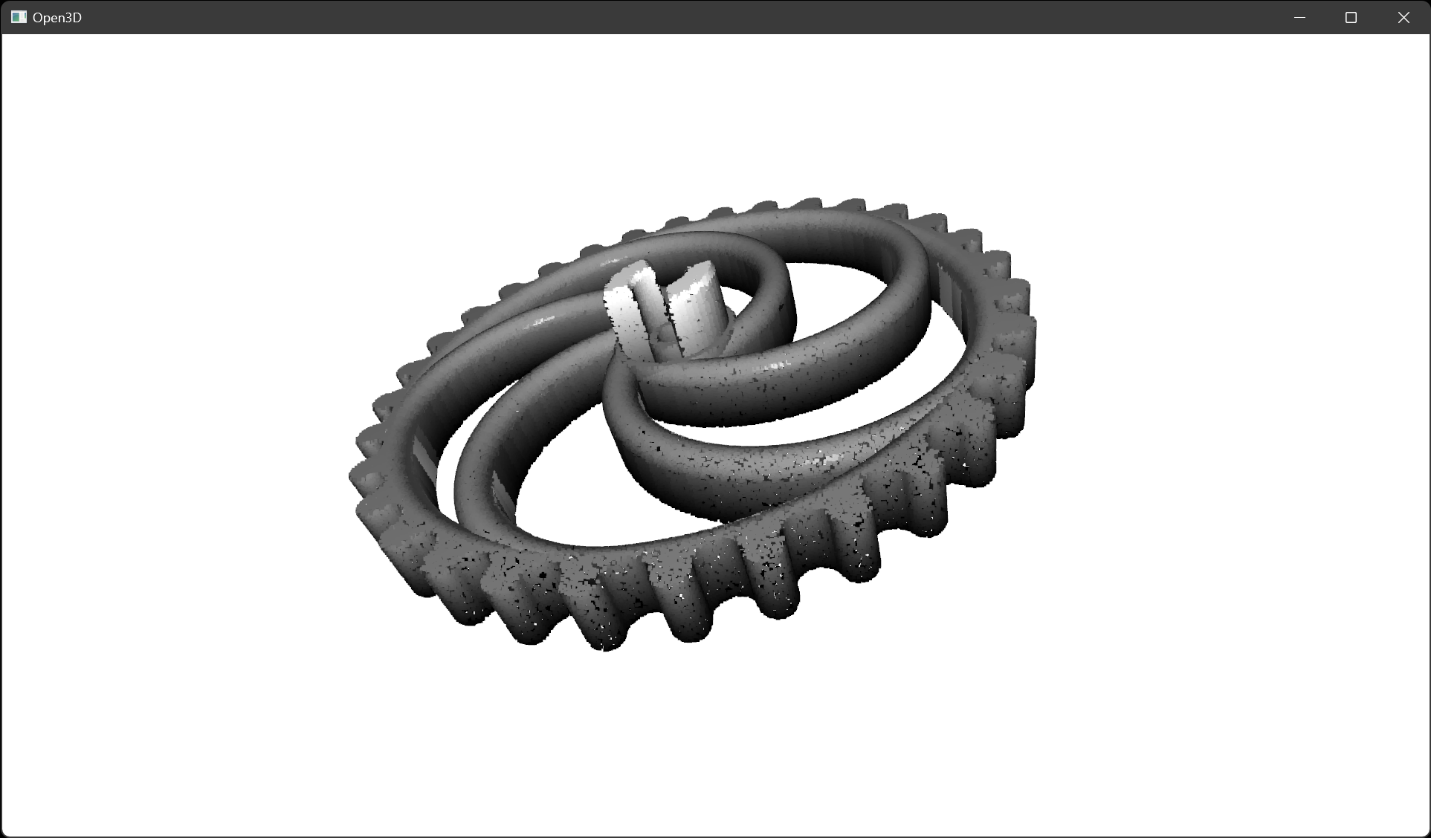
Description automatically generated with low confidence

*Рис.6, 7 хмара точок «Орел»*

*A picture containing text

Description automatically generatedA close-up of a feather

Description automatically generated with low confidenceРис. 8, 9 триангульована поверхня «Орел»*

*Рис.10,11 хмара точок «Шестерня»*

A picture containing wheel, gear

Description automatically generatedA picture containing text

Description automatically generated*Рис.12, 13 триангульована поверхня «Шестерня»*

# Література

1. Робота Michael Kazhdan , Matthew Bolitho і Hugues Hoppe, «Poisson Surface Reconstruction» (eng): <https://hhoppe.com/poissonrecon.pdf>
2. Принцип роботи методу (eng): <http://vr.tu-freiberg.de/scivi/?page_id=365>
3. GitHub Michael Kazhdan (eng): <https://github.com/mkazhdan/PoissonRecon/tree/master/Src>
4. 5-крокова інструкція як згенерувати 3D меші (meshes) з хмари точок на мові програмування Python (eng): <https://towardsdatascience.com/5-step-guide-to-generate-3d-meshes-from-point-clouds-with-python-36bad397d8ba>
5. Опис алгоритму-спрощення [[1]](https://hhoppe.com/poissonrecon.pdf) (eng): <https://github.com/alecjacobson/geometry-processing-mesh-reconstruction>