Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики

3-D ВІЗУАЛІЗАЦІЯ ОБ'ЄКТІВ НА ОСНОВІ МЕДИЧНИХ ЗОБРАЖЕНЬ

Текстова частина до курсової роботи

за спеціальністю 121 «Інженерія програмного забезпечення»

Керівник курсової роботи

Бучко О. А.
(*прізвище та ініціали*)

_____

(*підпис*)

"____"_____ 2022 р.

Виконав студент БП ІПЗ-4

Першута П.В.

(*прізвище та ініціали*)

_____

(*підпис*)

"____" _____ 2022 р.

Київ-2022

Тема: 3D-візуалізація об’єктів на основі медичних зображень

Календарний план виконання роботи:

№	Назва етапу курсової роботи	Термін виконання	Примітка
1.	Отримання завдання на курсову роботу	листопад	
2.	Огляд літератури за темою роботи	листопад-грудень	
3.	Створення практичної частини роботи	січень-березень	
4.	Написання текстової частини роботи	березень-квітень	
6.	Надання роботи керівнику для перевірки	кінець квітня	
8.	Коригування роботи за результатами перевірки	кінець квітня-початок травня	
9	Подання роботи на кафедру для перевірки на плагіат	тиждень до захисту	
10.	Захист курсової роботи	кінець травня	

Студент Першута П. В.

Керівник Бучко О. А.

“____” _____

Зміст

Анотація

Метою даної курсової роботи є дослідження роботи алгоритмів побудови 3D-моделей на основі зрізів, отриманих в результаті томографії голови. Проведено аналіз актуальності теми та проаналізовано два алгоритми побудови 3D-зображень, а саме “крокуючі куби” та “крокуючі тетраедри”. Також було розглянуто алгоритми сегментації для покращення вихідної моделі. Реалізовано алгоритм “крокуючі куби” для отримання 3D-моделі на мові C++ з використанням бібліотеки OpenCV.

Ключові слова: 3D-модель, томографія, зріз, сегментація, крокуючі куби

Вступ

Сьогодні неможливо уявити сучасне життя без використання комп'ютерних технологій, вони проникли в усі сфери нашого життя. Одна з них – медицина. Комп’ютери – це незамінний інструмент у сучасній медичній практиці. У всьому світі відбувається масштабне впровадження інноваційних технологій у різні галузі медицини, а також і в процес навчання нових медиків.

Комп’ютерна томографія – це метод діагностики, заснований на використанні рентгенівських променів та відіграє важливу роль у житті людини, особливо онкологів та онкохворих. Багатофазне сканування на мультиспіральних томографах дозволяє виявити ракові пухлини на ранніх стадіях. Також використовується для виконання певних хірургічних втручань (дренаж, абляція).

3D-технології зробили справжню революцію в медицині. Завдяки 3D-моделюванню, скануванню і друку стало можливим формування медичних виробів надзвичайної якості, адаптованих індивідуально для кожного пацієнта залежно від його особливостей. Сьогодні застосування 3D-технологій – це поширена практика в хірургії та стоматології.

Новітнє обладнання дає змогу виготовляти макети органів і кісток, створювати 3D-моделі та здійснювати високоточний 3D-друк протезів, імплантів та інших необхідних виробів. Завдяки цим нововведенням можна підвищити якість протезів, зменшити час їх виготовлення, знизити витрати на виробництво, а отже і ціну на кінцеву продукцію, що робить протези доступними для більшого кола пацієнтів. Таким чином, сучасні 3D-принтери можуть покращити життя не одній тисячі пацієнтів.

Метою даної роботи є дослідження алгоритмів побудови 3D-моделі на основі зрізів голови, отриманих після магнітно-резонансної томографії та алгоритмів, які дозволять покращити деталізацію моделі. Також розробити програмний застосунок, який формує 3D модель із зрізів за допомогою розглянутих алгоритмів.

Завданням роботи є створення застосунку на мові C++ з використанням бібліотеки з відкритим сирцевим кодом OpenCV. Вхідні дані - набір зрізів голови.

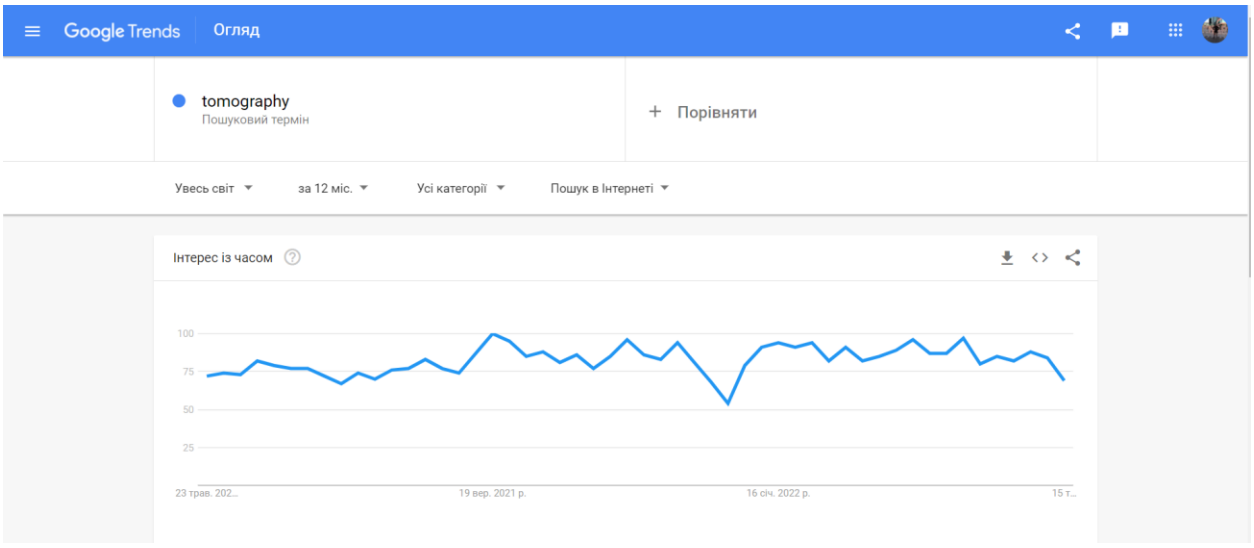
Об’єктом дослідження є реалізація побудови 3D-моделі зі зрізів, за допомогою вище перерахованого стеку технологій.

Предметом дослідження є аналіз існуючих алгоритмів побудови 3D-моделей та алгоритмів, які можуть покращити якість побудованої моделі.

Розділ 1: Аналіз предметної області

1.1 Дослідження актуальності проблеми

Для дослідження актуальності даної теми був проведений аналіз популярності пошукових запитів на тему “Tomography” за допомогою системи “Google Trends”. На наступному скріншоті відображено результат за останні 12 місяців і ми бачимо, що популярність запиту майже постійно тримається в діапазоні від 70% до 90%, а іноді сягає й 100%. Це говорить про те, що люди регулярно шукають щось пов'язане з томографією, а значить користуватимуться цією технологією. Для зручного користування, швидшої обробки даних, побудови 3D-моделей потрібно працювати над розробкою застосунків призначених для цієї сфери. Зважаючи на це, можна впевнено сказати, що тема є актуальною.



1.2 Томографія

Томографія - метод отримання пошарових зображень досліджуваних органів та тканин. Відповідно до способу проведення томографії її поділяють на три види: лінійна, комп’ютерна та магнітно-резонансна[1].

1.2.1 Лінійна томографія

Лінійна томографія - метод рентгенологічного дослідження з отриманням пошарових зображень (зрізів) досліджуваних органів і тканин пацієнта на рентгенівській плівці[1]. Цей тип томографії використовують для отримання зображення тканин та органів, розташованих в одній площині, на рівні певного шару без ефекту їх накладання один на одного. Такий ефект досягається за допомогою особливого технічного підходу: відбувається безперервний рух під час зйомки в різних напрямках рентгенівської трубки, що випромінює пучок променів, та касети з плівкою щодо досліджуваного об'єкта[1].

1.2.2 Комп’ютерна томографія

Комп’ютерна томографія - метод дослідження, який ґрунтується на обробці безлічі фотографій для отримання пошарових зрізів досліджуваних органів і тканин[3]. Отже, КТ - це метод візуалізації, що створює 2D-зображення поперечного перетину із тривимірної структури тіла. Для отримання зображення використовується математичний метод реконструкція (reconstruction). КТ-зображення є результатом «розбиття» тривимірної структури на частини, а потім її математичного поєднання і відображенні у вигляді 2D-зображення. Основне завдання томографії – точне відтворення внутрішніх структур тіла у вигляді 2D-зображень поперечного перетину.

Комп’ютерна томографія найкраще підходить для дослідження, внутрішніх органів (серця, легенів, нирок, головного мозку та інших), хребта, кісток скелета і судин (КТ з контрастуванням).

У комп'ютерній томографії використовують рентгенівські промені, тому в її основі лежить здатність різних органів та тканин людини нерівномірно послаблювати рентгенівське випромінювання. У процесі проходження крізь тканини рентгенівські промені ослабляються через поглинання та розсіювання енергії. Це ослаблення можна описати наступним рівнянням: I=$I_o c^{-\mu d}$. Тобто інтенсивність випромінювання, що було пропущено дорівнює добутку інтенсивності випромінювання на вході в тканину та коефіцієнту $c^{-\mu d}$, де μ - коефіцієнт повного лінійного ослаблення для тканини, d - товщина тканини. Коефіцієнт ослаблення μ обумовлений атомним номером та електронною щільністю тканини. Чим вище атомне число та щільність електронів, тим вище коефіцієнт ослаблення[3].

Основою комп'ютерного томографа є рентгенівська трубка. Вона випускає віялоподібний пучок рентгенівського випромінювання, спрямований перпендикулярно довгій осі тіла пацієнта. Рентгенівська трубка обертається навколо нього і повертається до тіла хворого під різними кутами, у загальній складності проходячи 360°[3].

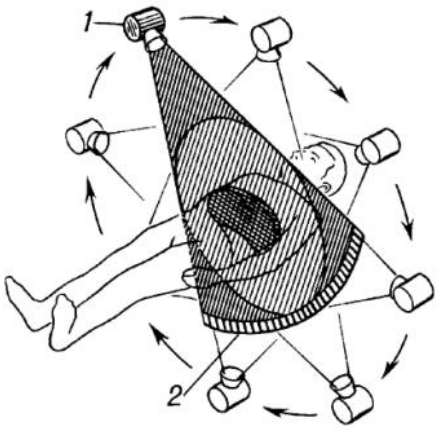


Рисунок 1 Схема роботи комп'ютерного томографа

Переваги комп’ютерної томографії:

- Висока роздільна здатність - можна розрізняти більшу кількість деталей в зображенні досліджуваного об'єкта якщо порівняти з рентгенографією. Завдяки своїй чутливості КТ дозволяє відділити окремі органи і тканини один від одного залежно від їх щільності [3].
- Можливість кількісно визначати рентгенівську щільність досліджуваного об'єкта: це дозволяє доповнювати візуальну оцінку комп'ютерно-томографічної картини аналізом щільності структур. Обробка сигналів від комп'ютерного томографа дозволяє точно виміряти послаблення рентгенівського випромінювання різними ділянками тканини в числовому значенні за умовною лінійною шкалою від -1000 до +3000. Воно вимірюється в одиницях Хаунсфілда. Значенням «0», за шкалою Хаунсфілда (од.Н), приймається послаблення рентгенівського випромінювання водою, а за -1000 – повітрям[3].

Tissue	CT Number (HU)
Bone	+1000
Liver	40 - 60
Whiter mater	−20 to −30
Grey mater	−37 to −45
Blood	40
Muscle	10 - 40
Kidney	30
CSF	15
Water	0
Fat	−50 to −100
Air	−1000

Рисунок 2 Шкала Хаунсфілда.

- КТ надає можливість реконструювати первинні зображення, тобто отримати зрізи у фронтальній, сагітальній та інших площинах. Також формувати 3D-зображення, що в свою чергу дозволяє визначати точну топографію, взаєморозташування органів і патологічних структур.

1.2.3 Магнітно-резонансна томографія

Магнітно-резонансна томографія (МРТ) – це ще одна процедура, в результаті якої можна отримати рентгенівські знімки частин тіла, але на відміну від КТ процедура проводиться довше, а також діє на основі електромагнітних хвиль. З цієї причини МРТ є абсолютно безпечним методом діагностики навіть для вагітних, маленьких дітей і людей із захворюваннями щитовидної залози, так як відсутнє опромінення людини. Крім того, магнітно-резонансну томографію можна проводити будь-яку кількість разів без побоювання за своє здоров’я. Однак, МРТ заборонено проходити людям з вживленими кардіостимуляторами, деякими видами стентів, розміщених всередині судин, кохлеарними імплантантами , кліпсами, які застосовуються при аневризмах головного мозку[4].

Розділ 2: Аналіз алгоритмів для сегментації зображення та побудови 3D-моделі та

2.1 Попередня обробка зображень

Найчастіше складність обробки зображень полягає у неоднорідності освітлення та наявності занадто освітлених або навпаки затемнених областей. На етапі попередньої обробки часто застосовують коригування рівня яскравості, контрастності та морфологічні операції. Це також дозволяє прибрати шуми на зображенні або об’єкти, що заважають для його подальшої обробки. Основні морфологічні операції: ерозія – “роз’їдання”, дилатація – “роздування”, розмиття зображення з використанням фільтра (ядра) один або кілька разів.

2.1.1 Фільтрація шуму

На зображення можуть виникати шуми внаслідок будь-яких маніпуляцій з ним: передачею, отриманням, стисненням. Також на дані зображення впливає канал передачі та навколишнє середовище, що призводить до їх втрати, а через це до спотворення початкового зображення. Наявність шуму негативно впливає на можливість наступних етапів обробки зображень, такі як знаходження контурів, виділення об’єктів та інші. Тому в сучасних системах обробки зображень прибирання шумів відіграє дуже важливу роль. Принцип фільтрації шумів полягає в тому, щоб підрівняти зображення по рівню яскравості. Однак, оскільки шум, границя і текстура є високочастотними компонентами, їх важко розрізнити в процесі знешумлення, тому зображення з шумом неминуче може втратити деякі деталі.

Оскільки фільтрація є основним засобом обробки зображень для зменшення шуму, було створено велику кількість фільтрів, але в загальному вони поділяються на два типи: лінійні і нелінійні.

Розглянемо два фільтри: медіанний та гаусовий.

Медіанний фільтр є нелінійним методом придушення шуму. Він не використовує згортку для обробки зображення з ядром коефіцієнтів. Значення кожного пікселя розраховується як медіанне значення серед яскравостей його сусідів, що попадають в матрицю. Тобто в нас є якась матриця (3x3/5x5) і для кожного пікселя беремо його сусідні елементи відповідно до розміру матриці та встановлюємо значення пікселя рівним медіанному значенню з всіх елементів-сусідів[5]. Наприклад візьмемо значення нашої матриці:

$$\begin{bmatrix} 100 & 150 & 152 \\ 0 & 120 & 160 \\ 180 & 200 & 50 \end{bmatrix}$$

Тоді для обрахунку значення пікселя необхідно відсортувати значення з матриці у порядку зростання: [0, 50, 100, 120, 150, 152, 160, 180, 200], а медіанне значення вибірки 150, що і буде новим значенням яскравості пікселя. Такий метод фільтрації чудово підходить для зображень, де шуми дуже різкі, схожі на раптові спалахи.

Фільтрація методом Гауса базується на функції Гауса і розмиває зображення за наступним принципом: чим ближче до центрального пікселя є його сусід, тим більшим є його коефіцієнт. Коефіцієнти в матриці розраховуються за наступною формулою:

$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$, де x та y координати елементу відносно центрального, а σ – стандартне відхилення. Отже для розрахунку, ми повинні

спочатку встановити значення σ. Нехай σ = 1.5, далі ми повинні порахувати коефіцієнти для кожного елементу в матриці коефіцієнтів. Для нашого σ вона має наступний вигляд[6]:

$$\begin{matrix} 0.0947416 & 0.118318 & 0.0947416 \\ 0.118318 & 0.147761 & 0.118318 \\ 0.0947416 & 0.118318 & 0.0947416 \end{matrix}$$

Далі потрібно перемножити відповідний коефіцієнт з матриці на елемент з матриці для конкретного пікселя. Використаймо матрицю значень із попереднього прикладу і отримаємо:

$$\begin{bmatrix} 100 * 0.0947416 & 150 * 0.118318 & 152 * 0.0947416 \\ 0 * 0.118318 & 120 * 0.147761 & 160 * 0.118318 \\ 180 * 0.0947416 & 200 * 0.118318 & 50 * 0.0947416 \end{bmatrix}$$

Наступний крок це порахувати суму всіх значень для цієї матриці, що і буде новим значенням яскравості пікселя. (100 + 152 + 180 + 50) * 0. 0947416+ (150 + 160 + 200) * 0. 118318+ $120 * 0.147761$= 123.738951, значення пікселя встановиться 124, оскільки повинне бути ціле число в діапазоні [0, 255]. Далі цей крок потрібно повторити з кожним пікселем зображення і в результати отримаємо зображення розмите фільтром Гауса, в якому шуми будуть мінімізовані.

2.1.2 Морфологічні операції

У цьому розділі варто розглянути операції, про які я вже згадував раніше, а саме: ерозія і дилатація[8]. Це прості операції які використовуються для прибирання шумів на бінарних зображеннях (пікселі зображення мають значення тільки 0 або 255) , які використовуються для того, щоб прибрати точковий шум та закрити можливі дірки всередині об’єкту.

Ерозія – це процес “роз’їдання” об’єктів, тобто на вихідному зображенні будуть зафарбовані лише пікселі, всі сусіди якого зафарбовані на вхідному. Таким чином крайні елементи об’єкту та точки-шуми будуть видалені із зображення.

Дилатація – це зворотній процес до ерозії, можна назвати його “набуханням” об’єктів, тобто на вихідному зображенні будуть зафарбовані пікселі, серед сусідів яких є хоча б один зафарбований піксель. Таким чином будуть закриті шуми-дірки всередині об’єкту.

Комбінуючи ці операції можна досягти значного покращення у якості вихідного зображення, адже це дозволить прибрати значну частину шумів та закрити значну частину небажаних дірок в об’єктах.

2.1.3 Просторові перетворення

У тривимірному просторі ми можемо виконувати наступні операції: переміщення, повороти та масштабування[9].

Важливо знати як можна обертати отриману 3D-модель зберігаючи пропорції. Для повороту об'єкта у просторі, нам необхідно змістити його центр на початок координат. Для цього спочатку потрібно знайти мінімальні x,y,z – координати та максимальні. Від всіх вершин моделі відняти, значення мінімальної координати та серединне зміщення для кожної осі. Серединне зміщення це – половина довжини відрізка між мінімальною осевою координатою та максимальною. Наприклад мінімальні координати (100,150,200) , а максимальні (200,200,260), значить відстані відповідно будуть (100, 50, 60). Можемо побачити, що додатково потрібно буде відняти 50 по x-координаті, 25 по y та 30 по z.

Наступний крок – це перемноження всіх вершин на матрицю повороту. Поворот відбувається одночасно лише по одній осі та для кожної осі є своя матриця. Нехай у нас є вершина A(x, y, z), щоб знайти координату вершини A’, поверненої відносно осі X на кут α, потрібно перемножити координати точки A на матрицю повороту відосно осі X. Отримаємо наступне рівняння, результатом якого буде точка з необхідними нам координатами:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Для поворотів відносно осей Y та Z також є матриці повороту:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Рисунок 3 Матриця повороту відносно осі Y

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Рисунок 4 Матриця повороту відносно осі Z

Окрім поворотів також є переміщення та масштабування. Аналогічно до поворотів для цих операцій є свої матриці: матриця переміщення та матриця масштабування.

Для масштабування потрібно знати коефіцієнти масштабування по кожній з осей (scaling factor), а сама матриця виглядає наступним чином:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Рисунок 5 Матриця масштабування

S_x – коефіцієнт масштабування відносно X, S_y – коефіцієнт масштабування відносно Y, S_z – коефіцієнт масштабування відносно Z.

Для переміщення необхідно знати зміщення по осях X, Y, Z, а сама матриця виглядає так:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Рисунок 6 Матриця переміщення

Де T_x, T_y, T_z – зміщення по осям X, Y та Z відповідно.

2.2 Побудова 3D-моделі

3D-модель це об’єкт представлений за допомогою набору точок у тривимірному просторі, з’єднаних між собою різноманітними геометричними об’єктами: трикутниками, лініями, криволінійними поверхнями тощо. Оскільки моделі це сукупність даних (точки та інша інформація), то їх можна

створювати вручну, алгоритмічно (процедурне моделювання), або відскановувати.

Майже всі 3D-моделі можна розділити на дві категорії за способом створення моделі.

• Твердотільне моделювання – ці моделі визначають об’єм об’єкта, який вони представляють (наприклад, камінь). Вони є більш реалістичні, але складніші в побудові. Твердотільні моделі в основному використовуються для невізуального моделювання наприклад медичне та інженерне моделювання, для систем автоматичного проектування, трасування променів та інших.

• Граничне моделювання – ці моделі представляють поверхню, наприклад межі об'єкта, а не його об’єм (як нескінченно тонка яєчна шкаралупа). З ними легше працювати, ніж з твердими моделями. Майже всі візуальні моделі, що використовуються в іграх і фільмах, є граничними моделями.

Існує три популярних способи представлення моделі:

1. Полігональне моделювання: точки в тривимірному просторі, які називаються вершинами, з'єднуються відрізками і утворюють полігональну сітку. Переважна більшість 3D-моделей сьогодні побудовані як текстуровані полігональні моделі, тому що вони гнучкі і комп’ютери можуть швидко їх відтворювати. Однак багатокутники можуть лише наблизитись до відображення криволінійної поверхні, використовуючи багато багатокутників, але зі збільшенням їх кількості зростає час обробки моделі та об’єм пам’яті який вона займає.

2. Моделювання кривих: поверхні визначаються кривими, на які впливає зважений контроль точок. Крива слідує за (не обов’язково інтерполює)

точками. Збільшення ваги для певної точки потягне криву ближче до неї. Типи кривих включають нерівномірні раціональні B-сплайни (NURBS), сплайни, латки та геометричні примітиви.

3. 3D-ліплення (також зване цифровим ліпленням) — це процес, коли художник створює 3D-об’єкт на комп’ютері з матеріалу, схожого на оцифровану глину. Програмне забезпечення з пензлями та інструментами, які штовхають, тягнуть, щипають і згладжують, дозволяють легко створювати детальні скульптури, які імітують реальні текстури та об’єкти.

Далі я розглядатиму полігональне моделювання. Одною з головних проблем моделювання є отримання ізоповерхонь з точок у просторі. Якщо перебирати всі можливі варіанти, то отримаємо настільки велику кількість трикутників, що комп’ютер не зможе їх нормально оброблювати. Тому не дивно, що існують алгоритми для реконструкції поверхні за точками у просторі. Далі розглянемо два алгоритми призначених для цього, а саме “крокуючі куби” та “крокуючі тетраедри”

2.2.1 Воксельна структура

В основі наступних алгоритмів лежить воксельна структура, тому почнемо з цього. Воксель – це такий собі піксель у тривимірному просторі, що являє собою куб. Воксельну структуру можна уявити як сітку, ідеально розграфлену на куби:

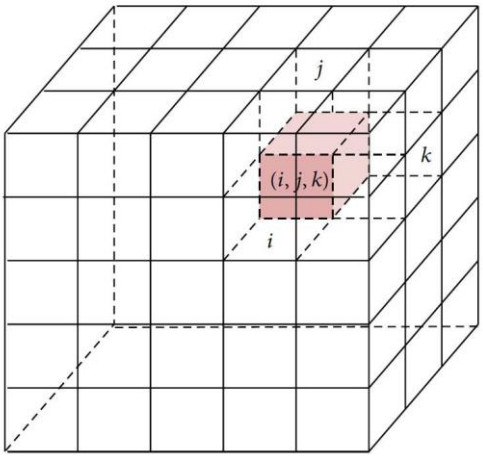


Рисунок 7 Приклад воксельної структури

2.2.2 Алгоритм “крокуючі куби”

Marching Cubes — це алгоритм візуалізації ізоповерхень, запропонований у 1987-му році Вільямом Лоренсом та Харві Клайном.

Основний принцип полягає в тому, що ми можемо визначити воксель (куб) за значеннями пікселів у восьми вершинах куба. Якщо один або кілька пікселів куба мають значення, менші від заданого користувачем ізоповерхні, а один або більше мають значення, більші за це значення, ми знаємо, що воксель повинен додати певний компонент ізоповерхні. Визначивши, які ребра куба перетинає ізоповерхня, ми можемо створити трикутні ділянки, які розділяють куб між областями всередині ізоповерхні та областями за її межами. З’єднавши латки з усіх кубів на межі ізоповерхні, ми отримаємо представлення поверхні.

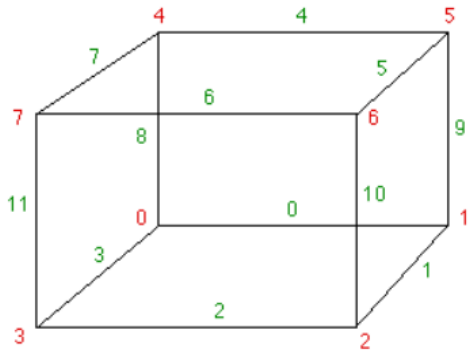
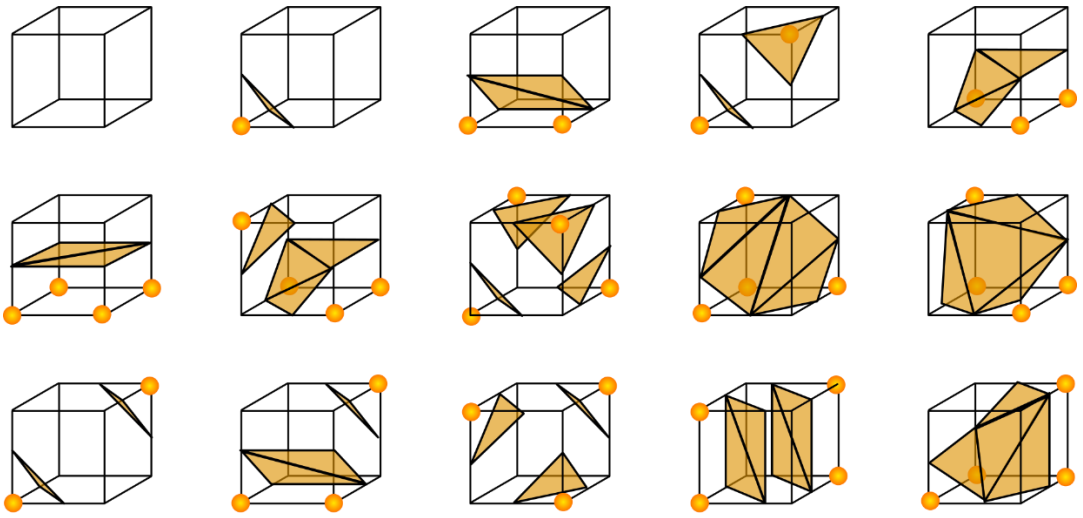


Рисунок 8 Нумерація вершин куба та його ребер

Залежно від того, які вершини куба лежать всередині та назовні об'єкта, є 256 варіантів (у куба 8 вершин, відповідно $2^8 = 256$) проведення площин, однак лише 15 з них є унікальними, решта це просто симетричне відображення унікальних варіантів[7].



Отримана модель не буде ідеальною, адже краї моделі будуть мати рублені грані, а кількість отриманих трикутників буде дуже великою. Для покращення відображення моделі варто скористатись лінійною інтерполяцією. Точки перетину P обчислюються за допомогою формули:

$$P = P_1 + \frac{(c - v_1)(P_2 - P_1)}{v_2 - v_1} \quad ,$$

де P1 і P2 вершини ребра, V1 і V2 є скалярні значення в кожній вершині, c – обране ізозначення. Таким чином поверхні будуть тісніше прилягати до кубів(вокселів) та утворюватимуть модель ближчу до реальної.

Втім алгоритм має ряд недоліків:

1) Великий розмір моделі, адже кожен не порожній воксель може містити кілька трикутників, що сильно роздуває модель, а збільшення розміру сітки зменшить деталізацію моделі.
2) Неоднозначність визначення площин. Є випадки коли потрібно проводити додаткові перевірки, щоб визначити як має бути проведена площина.

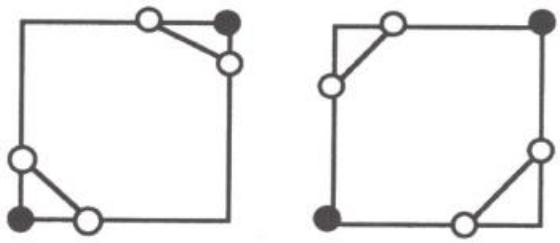
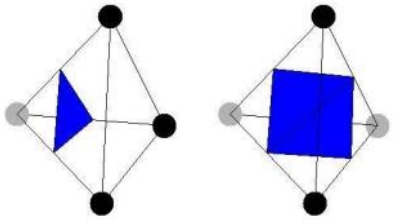


Рисунок 9 Приклад неоднозначності у 2D форматі

2.2.3 Алгоритм крокуючі тетраедри

Цей алгоритм майже ідентичний з алгоритмом крокуючих кубиків. Відмінністю є те, що куб розбивається на 6 тетраедрів і вже по їхнім вершинам відбувається перевірка на те, чи поверхня перетинає ребра тетраедра. Цей алгоритм запропонували Г.М. Трис, Р. У. Прагер та А. Х. Джі для кластеризації вершин, яка допомагає спрощувати сітку та витягувати ізоповерхню. Вершиною кластеризації є метод, який можна використовувати для створення трикутника на основі крокуючого тетраедра, і він дозволяє зберегти вихідну топологічну структуру за допомогою кластеризації тетраедричної решітки. Результатом крокуючих тетраедрів є поверхня, що складається з трикутників.

У зв'язку зі зменшенням кількості вершин, зменшується кількість варіантів перетину тетраедра площиною до 16, а усунувши дублі залишиться лише 3 унікальні варіанти[10] (два не порожні та один порожній)



Переваги алгоритму крокуючих тетраедрів:

1) Цей алгоритм не підпадає під патент на “крокуючі кубики”, з яким можуть виникнути проблеми при імплементації його в комерційному проекті.
2) Він може працювати як з неструктурованими сітками, так і зі структурованими (які можна розділити на шість тетраедрів). Фактично, було доведено, що будь-яку геометричну комірку можна розкласти на серію тетраедрів, що робить крокуючі тетраедри загальним рішенням для отримання ізоповерхні на всіх типах сіток.
3) Уникає невизначених ситуацій, що виникають в алгоритмі крокуючих кубиків.

Розділ 3: Відображення 3D-моделі

Наступним важливим кроком, після того як ми отримали набір поверхонь, є їх відображення на екрані. Можна виділити три основні кроки при відображенні: перехід від світових координат до екранних, освітлення об’єкта, та приховування невидимих площин, для більш реалістичного рендерингу об’єкта.

3.1 Світові координати та екранні координати

Для зображення тривимірного об’єкту на пристрої відображення, потрібно задати перетворення координат зі світової системи в екранну[11].

Світова система координат - це система, в якій розміщуються об’єкти, іншими словами, це наша сцена. Є окремо системи координат спостерігача(камери) і об’єкта. Кожна точка представлена тривимірними координатами.

Екранна система координат - система координат, на яку ми проектуємо. X,Y - координати пікселя, Z - показує глибину зображення.

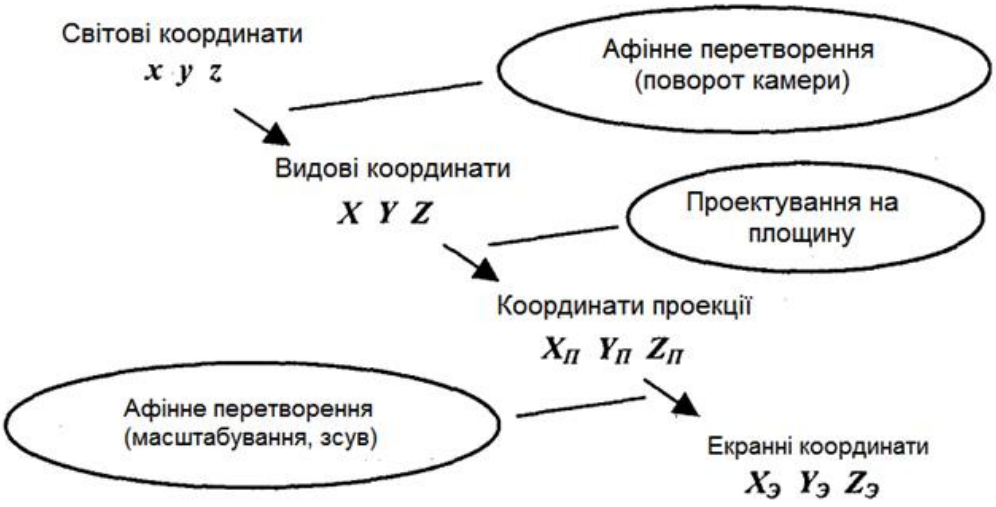


Рисунок 10 Етапи переходу від світових координат до екранних

На зображенні чітко показано порядок виконання операцій переходу до екранних координат. Спочатку потрібно застосувати матриці повороту, які розглядались раніше і розділі “Просторові операції”, далі спроектувати на площину. Тобто має бути матриця розміром 3x3 на основній діагоналі якої розміщені коефіцієнти з A, B, C з рівняння цієї площини і перемноживши координати точки на цю матрицю, ми отримаємо координати спроектовані на площину.

Після проектування потрібно виконати масштабування, оскільки координати можуть бути нормалізовані і побудована модель буде дуже маленькою. Та виконати зсув з початку координат, бо частина вершин матиме від'ємні координати. Таким чином ви отримаємо спроектовані екранні координати, де X та Y це координати на площині, а Z визначає глибину вершини.

3.2 Освітлення моделі

Наступним кроком є освітлення моделі. Тобто у нас є вектор, який визначає напрямок світла, а значить ми можемо порахувати як має бути освітлений кожен трикутник [12]. Для цього потрібно знайти нормаль до кожної площини. Нормаль — це технічний термін, який використовується в комп’ютерній графіці для опису орієнтації поверхні геометричного об’єкта в точці на цій поверхні. Поверхню, нормальну до другої поверхні, можна розглядати як вектор, перпендикулярний до площини, дотичної до поверхні в певній точці.

Знайти нормаль можна за допомогою операції векторного добутку:

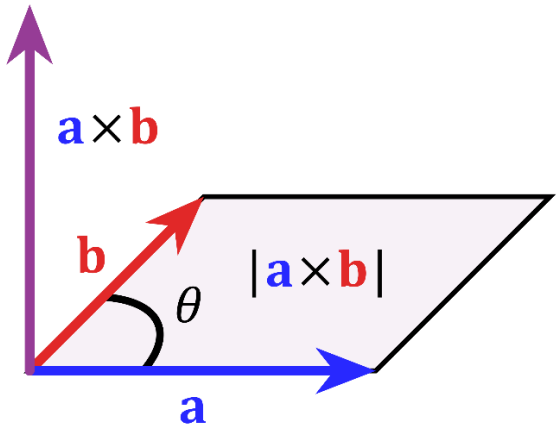


Рисунок 11 Вектор c - результат векторного добутку векторів a і b

Отриманий вектор потрібно нормалізувати (кожну координату поділити на довжину вектора, щоб довжина вектора стала рівна 1). Рівень яскравості трикутника визначається скалярним добутком нормалі та напрямку світла. Якщо помножити рівень яскравості на максимальне значення яскравості, то отримаємо конкретну яскравість трикутника.

3.3 Приховування невидимих площин.

Приховування невидимих площин – це один з найважливіших етапів рендерингу 3D-моделі. Адже трикутники не є впорядковані, тому можуть накладатись один на одного, що призведе до зіпсування картинки і неправильної візуалізації моделі. Для того щоб цього уникнути існує багато алгоритмів: Робертса, Художника, Z-буфер[13].

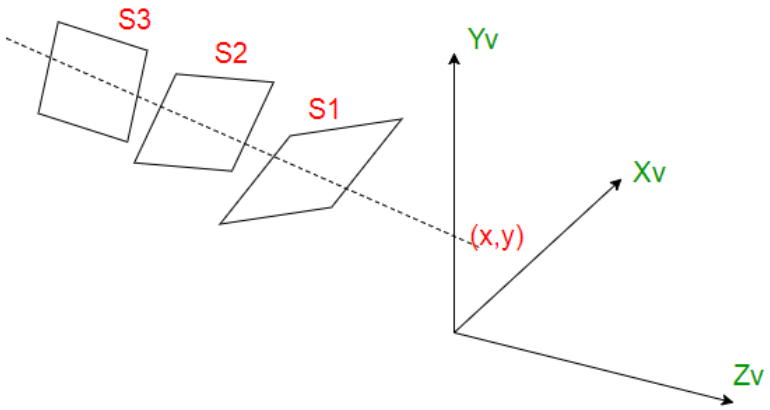


Рисунок 12 Приклад накладання площин

Алгоритм художника можна розділити на наступні кроки:

- Сортування трикутників по глибині (z - координата) від найглибшого.
- Зафарбовування трикутників у посортованому порядку.

Порядок, який використовує алгоритмом називається «порядок глибини», який використовує числову відстань до частин сцени: істотна властивість цього впорядкування, полягає в тому, якщо один об'єкт заступає частину іншого, тому наступний об'єкт буде зафарбований поверх попереднього об'єкта.

Z – буфер, це по піксельна реалізація алгоритму художника. Спочатку потрібно створити матрицю значень глибини для кожного пікселя. Далі при зафарбовуванні трикутника, потрібно перевіряти значення глибини конкретного пікселя і якщо поточне число є меншим, то змінити значення яскравості та оновити дані про глибину. Таким чином ми отримаємо реалістичне зображення в якому невидимі площини будуть заховані.

Розділ 4: Створення програми для побудови 3D-моделі

Програму для побудови 3D-моделі, було вирішено реалізувати за допомогою бібліотеки OpenCV мовою програмування C++. Алгоритм для реалізації – крокуючі куби. Вхідні дані для створення моделі – набір МРТ зрізів голови у форматі “.tif” .

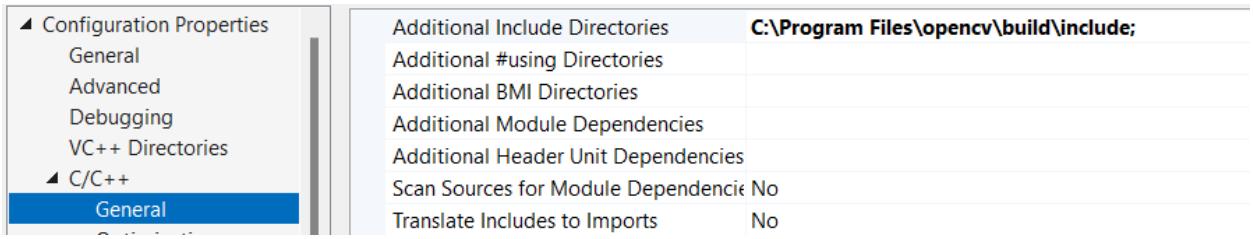
4.1 Налаштування проекту

Спочатку потрібно завантажити OpenCV та встановити на ПК. У змінних оточення в змінну PATH потрібно додати шляхи до папок bin та lib під необхідну мову програмування:

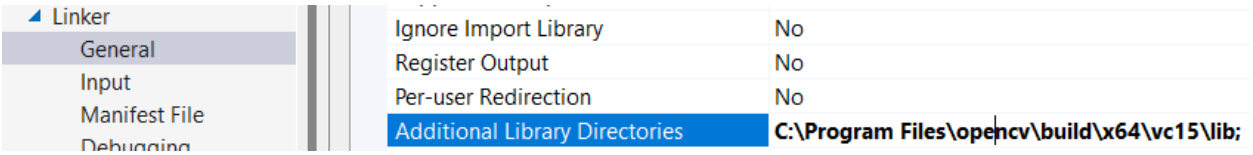
```
C:\Program Files\opencv\build\x64\vc15\bin
C:\Program Files\opencv\build\x64\vc15\lib
```

Перед початком роботи необхідно підключити OpenCV до C++ проекту в середовищі розробки, в моєму випадку це Visual Studio 2022.

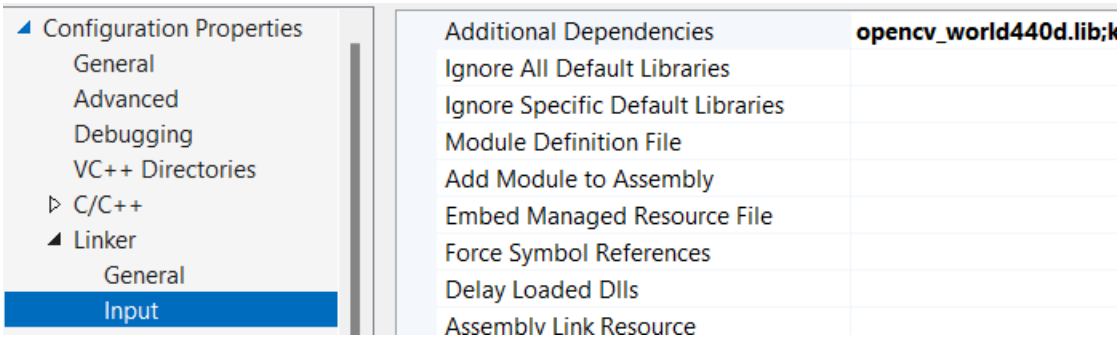
1) C/C++ -> General -> Additional Include Directories додати шлях до директорії include:



2) Linker -> General -> Additional Library Directories та додати шлях до директорії bin:



3) Linker -> Input -> Additional Dependencies та додати назву .lib файлу:



Тепер в проекті можна приєднувати заголовні файли бібліотеки та використовувати її функціонал.

4.2 Опис необхідних структур

Проаналізувавши алгоритми, які мають бути реалізовані, було створено наступні структури для групування даних і роботи з ними як з об’єктами.

1) Point – структура, яка являє собою екранну вершину та має поля типу int для збереження x, y та z координат. Вона використовується для визначення екранних координат
2) Vertex - структура для роботи з нормалізованими вершинами має поля поля типу float для збереження x, y та z координат.
3) Triangle – клас, що є реалізацією трикутника. Містить масив, що складається із трьох вершин. Має метод draw, що на вхід примає наступні параметри: відсилку на зображення в якому треба намалювати трикутник, відсилку на z-буфер, розмір буферу, колір трикутника.
4) GridCell – структура яка складається з двох масивів: масиву вершин так масиву значень вершин. Вона являє собою куб(воксель) для якого будуть рахуватись площини.

4.3 Реалізація крокуючих кубиків

Перш за все потрібно провести попередню обробку вхідних зображень, а саме прибрати шум використовуючи фільтр Гауса, принцип роботи якого був описаний у другому розділі роботи. Я використав реалізацію цього фільтру із бібліотеки OpenCV. Після обробки я створюю тривимірний масив, який є представленням об'ємного світу та двовимірний масив – Z-буфер. Далі всі точки з вхідних зображень, записуються у цей світ з такими самими X та Y координатами. Z - координата це номер зрізу помножений на розмір воксельної сітки. Таким чином вершини вокселів будуть потрапляти в точки із вхідних зображень.

```
const int size = 800;
const int grid_size = 10;
auto world = new float[size][size][size];
float ** zbuff = new float*[size];
for (int i = 0; i < size; i++) {
    zbuff[i] = new float[size];
}

for (int x = 0; x < size; x += 10) {
    for (int y = 0; y < size; y += 10) {
        zbuff[x][y] = -1000;
        for (int z = 0; z < size; z += 10) {
            world[x][y][z] = 0;
        }
    }
}
for (int i = 0; i < 24; i++) {
    cv::Mat Image = cv::imread(images[i], cv::IMREAD_GRAYSCALE);
    cv::blur(Image, Image, cv::Size(5, 5));

    for (int x = 0; x < Image.cols; x++) {
        for (int y = 0; y < Image.rows; y++) {
            int val = Image.at<uchar>(x, y);
            world[x][y][i * grid_size] = val;
        }
    }
}
```

Наступним кроком є створення класу MarchingCube в якому визначається індекс кубу за значенням вершин та зберігаються всі можливі варіанти проведення площини, що зберігаються у статичному векторі triangulationTable. У векторі edges зберігаються індекси ребер, що мають бути з'єднані відповідно до індексу кубу.

```
class MarchingCube {
    static std::vector<std::vector<int>> triangulationTable;
    static std::vector<int> edges;
    int getCubeIndex(GridCell cell, int background);
    VVertex VertexInterp(int background, VVertex vert1, VVertex vert2, float value1, float value2);

public:
    MarchingCube();
    void addTriangle(GridCell gridcell, std::vector<VertTriangle>& out, int background);
};;
```

Метод getCubeIndex() рахує індекс кубу, відповідно до значення вершин та ізозначення, яке визначає чи знаходиться вершина в об’єктів чи поза ним. Якщо вершина входить в об’єкт, то в біт, що відповідає за цю вершину вписуємо 1 використовуючи бітові операції.

```
int MarchingCube::getCubeIndex(GridCell cell, int background)
{
    int index = 0;
    for (int i = 0; i < 8; i++) {
        if (cell.value[i] > background) {
            index |= 1 << i;
        }
    }
    return index;
}
```

Коли було визначено індекс кубу та координати точок через які проходить площина, створюється трикутник, який додається у вихідний вектор.

```
for (i = 0; triangulationTable[cubeindex][i] != -1; i += 3) {
    VertTriangle t;
    t.vert[0] = vertlist[triangulationTable[cubeindex][i]];
    t.vert[1] = vertlist[triangulationTable[cubeindex][i + 1]];
    t.vert[2] = vertlist[triangulationTable[cubeindex][i + 2]];
    out.push_back(t);
}
```

Після того як буде перевірений кожен воксель зі світу, буде отримано вектор трикутників, які формують поверхню. Для того щоб мати можливість обертати отриману модель потрібно нормалізувати координати всіх отриманих трикутників. Для цього в файлі 3DVisualizationOfObjects.cpp є метод normalizeTriangles (). Нормалізація була виконана так як описано в другому розділі цієї роботи.

```
void normalizeTriangles(const std::vector<VertTriangle>& triangles, std::vector<VertTriangle>& normTriangles) {
    int minX = 0;
    int maxX = 0;
    int minY = 0;
    int maxY = 0;
    int minZ = 0;
    int maxZ = 0;
    findMinMaxInTriangles(minX, maxX, minY, maxY, minZ, maxZ, triangles);
    double diagLength = sqrt((maxX - minX) * (maxX - minX) + (maxY - minY) * (maxY - minY) + (maxZ - minZ) * (maxZ - minZ));
    for (int i = 0; i < triangles.size(); i++) {
        VertTriangle vt;
        for (int j = 0; j < 3; j++) {
            vt.vert[j].x = (triangles[i].vert[j].x - minX) - (maxX - minX)/2;
            vt.vert[j].y = (triangles[i].vert[j].y - minY) - (maxY - minY) / 2;
            vt.vert[j].z = (triangles[i].vert[j].z - minZ) - (maxZ - minZ) / 2;
        }
        normTriangles.push_back(vt);
    }
}
```

Після отримання нормалізованих координат, можна переходити до рендерингу моделі. Для кожного трикутника нам необхідно порахувати його екранні координати. Як було розглянуто в третьому розділі роботи, спочатку ми повинні виконати необхідні повороти об'єкту, масштабувати його та змістити із початку координат. Так ми отримаємо координати трикутників.

```
VVertex v1 = rotateY(t.vert[j], 1.57);
VVertex v2 = rotateZ(v1, -1.57);
VVertex v = rotateY(v2, angle);
screen_coords[j] = cv::Vec2i(v.x * scale + shift, v.y * scale + shift);
world_coords[j] = { v.x  + shift, v.y  + shift , v.z  };
```

Наступним кроком є освітлення моделі. Для цього були визначені оператори векторного та скалярного добутків векторів, а також оператор множення вектору на число та метод нормалізації вектору.

```
cv::Vec3f operator ^(const cv::Vec3f& v1, const cv::Vec3f& v2) {
    return cv::Vec3f(v1[1] * v2[2] - v1[2] * v2[1], v1[2] * v2[0] - v1[0] * v2[2], v1[0] * v2[1] - v1[1] * v2[0]);
}

cv::Vec3f normalize(cv::Vec3f v, float l = 1)
{
    v = v * (l / (std::sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2])));
    return v;
}

inline cv::Vec3f operator *(float f, cv::Vec3f v)
{
    return cv::Vec3f(v[0] * f, v[1] * f, v[2] * f);
}
inline float operator *(const  cv::Vec3f& v, const  cv::Vec3f& v1)
{
    return v[0] * v1[0] + v[1] * v1[1] + v[2] * v1[2];
}
```

Отримавши нормаль до площини, ми можемо порахувати необхідну яскравість трикутника.

```
cv::Vec3f n = (world_coords[2] - world_coords[0]) ^ (world_coords[1] - world_coords[0]);
cv::Vec3f norm = normalize(n);
float intensity = norm * light_dir;
if (intensity > 0) {
    Triangle tr(world_coords[0], world_coords[1], world_coords[2]);
    tr.draw(Image3D, zbuff, size, cv::Vec3f(intensity * 196, intensity * 223, intensity * 255), true);
}
```

Для створення ефекту анімації потрібно запустити в циклі поворот та рендеринг об'єкту, не забуваючи очищати Z-буфер після кожної ітерації.

Після запуску програми отримаємо модель, зображення якої ви можете побачити в додатку.

Висновки

Під час роботи було досліджено етапи обробки зображень: фільтрація шумів, ерозія, дилатація; та етапи побудови 3D-моделей: отримання точок, знаходження ізоповерхні, операції з моделлю в просторі, освітлення моделі та приховування невидимих площин. Проаналізовано предметну область, принцип отримання рентгенівських знімків. Розглянуто алгоритми “Крокуючі куби” та “Крокуючі тетраедри”

Також було розроблено програму, що із сету зрізів МТР будує 3D-модель. Під час розробки було використано бібліотеку OpenCV, а реалізований алгоритм - “Крокуючі куби”. Також програма надає можливість обертати модель відносно осей X, Y, Z.

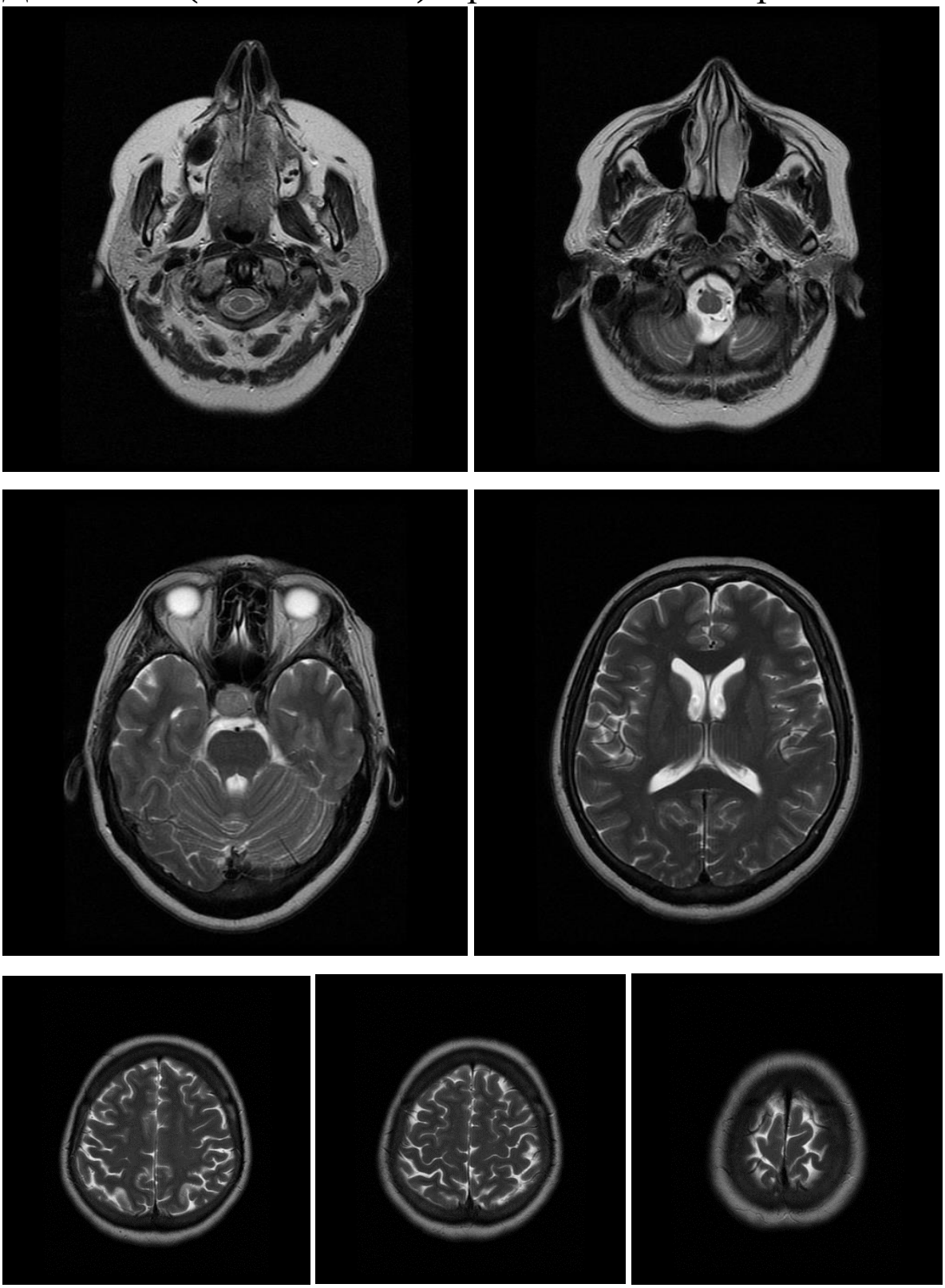
Завдяки цій роботі я краще зрозумів як і для чого відбувається попередня обробка зображень, як працює рендеринг та як формуються 3D-моделі. Впевнений, що мені це знадобиться у моїй подальшій професійній діяльності.

Розроблену програму можна модифікувати, а саме змінити алгоритм побудови моделі, для кращого рендерингу, на Dual Contouring наприклад. Також можна додати підтримку виділення/виокремлення певних тканин або застосувати технологію напівпрозорих вокселів, що дозволить зазирнути всередину об'єкта.

Джерела

1. “Рентгенологічні методи дослідження” - Н.В.Туманська, К.С.Барська, С.В. Скринченко, Т.М. Кічангіна, 2017, 36-43
2. Стаття Amy Tikkanen в https://www.britannica.com/science/tomography
3. “Рентгенологічні методи дослідження” - Н.В.Туманська, К.С.Барська, С.В. Скринченко, Т.М. Кічангіна, 2017, 54-81
4. Інтернет ресурс : https://euromed.kr.ua/statti
5. Інтернет ресурс : http://ai-tern.in.ua/Filtration.html
6. Інтернет ресурс : https://www.pixelstech.net/article/1353768112-Gaussian-Blur-Algorithm
7. Інтернет ресурс : http://paulbourke.net/geometry/polygonise
8. K A M Said and A B Jambek 2021 Journal of Physics, Conference Series: Analysis of Image Processing Using Morphological Erosion and Dilation
9. Інтернет ресурс: https://www.geeksforgeeks.org/computer-graphics-3d-rotation-transformations/
10. “Comparative Study of Marching Cubes Algorithms for the Conversion of 2D image to 3D” - Sreeparna Roy and Peter Augustine, 2017
11. Інтернет ресурс: https://learnopengl.com/Getting-started/Coordinate-Systems
12. Інтернет ресурс: https://learnopengl.com/PBR/Lighting
13. “THE MAGIC OF THE Z-BUFFER: A SURVEY” - Theoharis Theoharis, Georgios Papaioannou, Evaggelia-Aggeliki Karabassi

Додаток А (обов’язковий). Зразок вхідних зображень



Додаток Б (обов’язковий) Результат роботи програми