**Міністерство освіти і науки України**

**Національний технічний університет України «КПІ» імені Ігоря Сікорського**

**Кафедра обчислювальної техніки ФІОТ**

**ЗВІТ**

**з лабораторної роботи №3**

**з навчальної дисципліни «Computer Vision»**

**Тема:**

**ДОСЛІДЖЕННЯ АЛГОРИТМІВ ФОРМУВАННЯ ТА ОБРОБКИ ВЕКТОРНИХ ЦИФРОВИХ ЗОБРАЖЕНЬ**

**Виконав:**

Студент 3 курсу кафедри ІПІ ФІОТ,

Навчальної групи ІП-11

Головня О.Р.

**Перевірив:**

Професор кафедри ОТ ФІОТ

Писарчук О.О.

**Київ 2024**

**І. Мета роботи:**

Виявити дослідити та узагальнити особливості реалізації алгоритмів формування та обробки векторних цифрових зображень на прикладі застосування алгоритмів інтерполяції, апроксимації та згладжування складних 3D растрових об’єктів та застосування технологій видалення невидимих граней та ребер.

**ІІ. Завдання**

Завдання ІІІ рівня – максимально 9 балів.

Здійснити виконання завдання ІІ рівня складності – програмний скрипт No1.

Реалізувати розробку програмного скрипта No2, що реалізує виділення контору обраного об’єкту на цифровому растровому зображенні. За необхідності передбачити корекцію кольору цифрового растрового зображення для покращення якості виділення контору обраного об’єкту.

|  |  |  |
| --- | --- | --- |
| 3 | Відображення 3D фігури  реалізується з використанням аксонометричної проекції будь-якого типу.  Обрати самостійно: бібліотеку, розмір графічного вікна, розмір фігури, динаміку зміни положення фігури, кольорову гамму графічного об’єкту.  Всі операції перетворень мають  здійснюватися у межах графічного вікна. | Піраміда з чотирикутною основою.  Метод інтерполяції: сплайн інтерполяція.  Метод видалення невидимих ліній та поверхонь:  алгоритм z-буфера. |

**ІІІ. Результати виконання лабораторної роботи.**

**3.1. Синтезована математична модель перетворень графічних об’єктів відповідно до індивідуального завдання.**

Відповідно до завдання, синтезовано математичну модель операцій над пірамідою з чотирикутною основою

Використати метод інтерполяції: сплайн інтерполяція

Сплайн-інтерполяція - це метод апроксимації функції, який використовується для створення гладкої кривої (сплайну), що проходить через задані точки даних. Основна ідея полягає в тому, щоб розділити вхідні точки на інтервали і наблизити кожен інтервал поліномом невисокого ступеня.

Один з найпоширеніших видів сплайнів - це кубічна сплайн-інтерполяція, яка використовує кубічні поліноми для наближення даних. Рівняння кубічного сплайну для кожного інтервалу між точками (x[i], y[i]) і (x[i+1], y[i+1]) виглядає так:

Використовуючи **splprep** функцію, ми визначаємо параметри сплайна для кожної бічної сторони піраміди. Потім ми генеруємо нову послідовність точок на кожній стороні, використовуючи ці параметри, і використовуємо **splev** для отримання координат цих точок. Нарешті, ми використовуємо ці точки для побудови гладкої кривої на кожній бічній стороні піраміди.

Параметр **k** вказує на ступінь полінома, у моєму випадку **k=2**

Параметр **s** вказує на згладжування сплайна. Згладжування використовується для управління ступенем гладкості сплайна. Зазвичай **s=0** вказує на строгу інтерполяцію, де сплайн проходить точно через всі вхідні точки.  
 Параметр **points.T** є транспонованим масивом точок, оскільки **splprep** очікує масив точок у вигляді **(x, y, z)**, де кожний рядок масиву представляє координати однієї точки.

З буфер (**Z-buffer**) - це метод приховування поверхонь, який використовується для визначення того, які об'єкти або частини об'єктів на зображенні будуть видимі, а які - приховані.

Цей метод базується на збереженні значення Z-координати (глибини) для кожної точки на екрані. Під час обробки кожної нової точки зображення порівнюються значення Z-координати цієї точки зі значеннями, які вже зберігаються в Z-буфері. Якщо нове значення Z-координати менше, ніж те, що вже є в Z-буфері, то нове значення замінює старе, і ця точка вважається видимою.

Під час обробки кожної поверхні піраміди визначається нормаль до цієї поверхні, і якщо скалярний добуток цієї нормалі на вектор, що з'єднує точку перегляду з будь-якою точкою поверхні, менше нуля, то поверхня вважається видимою і її точки додаються до списку видимих поверхонь. Далі для кожної точки на екрані порівнюється її Z-координата з відповідною значенням в Z-буфері. Якщо нова Z-координата менше, ніж та, що зберігається в Z-буфері, вона замінює стару, і ця точка вважається видимою.

**3.2. Блок схема алгоритму та її опис.**

Блок-схема розв’язку матиме вигляд на рис. 1.

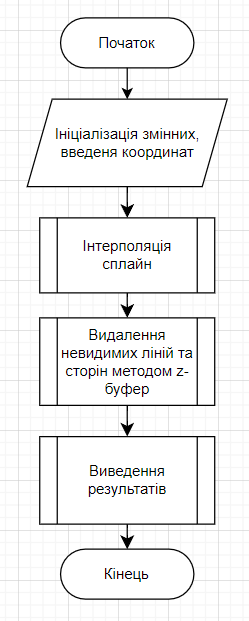


Рис. 1. Блок-схема програм first.py та second.py відповідно

**3.3. Опис структури проекту програми**

|  |
| --- |
|  |
| Рис.2. Структура проекту. |

file.jpg – зображення для зміни

Holovnia\_lr\_3.docx – звіт

first.py - файл програми для першого завдання

second.py - файл програми для другого(контур)

**3.4. Результати роботи програми відповідно до завдання.**

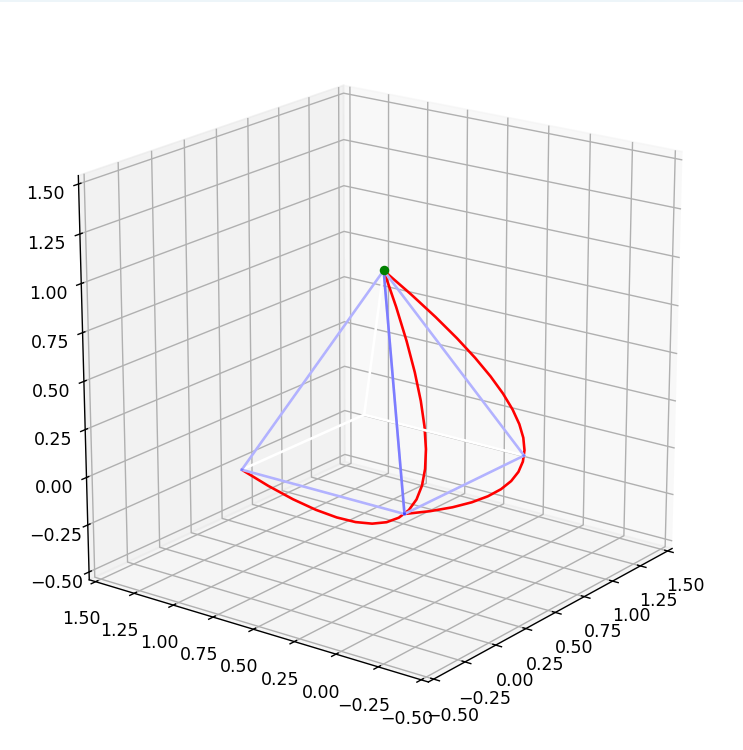
****

Рис. 3. Зображення піраміди(Кольори:  
червоний – гладкі криві(сплайн), синій – видимі лінії, білий - невидимі



Рис. 4. Виявлення контурів на зображенні(Зображення після змін)

**3.5. Програмний код.**

**First.py**

def create\_pyramid(base\_points, apex, num\_points):

pyramid = []

for i in range(len(base\_points)):

points = [base\_points[i], base\_points[(i+1)%len(base\_points)], apex]

for j in range(len(points)):

face = np.array([points[j], points[(j+1)%len(points)], apex])

pyramid.append(face)

return pyramid

def z\_buffer\_algorithm(faces, view\_point, num\_points):

visible\_faces = []

z\_buffer = np.zeros((num\_points, num\_points))

for face in faces:

normal = np.cross(face[1]-face[0], face[-1]-face[0])

if np.dot(normal, face[0]-view\_point) < 0:

visible\_faces.append(face)

for i in range(num\_points):

for j in range(num\_points):

z = np.dot(normal, face[0] - view\_point) / np.dot(normal, np.array([i, j, 0]) - view\_point)

if z > z\_buffer[i, j]:

z\_buffer[i, j] = z

return visible\_faces

def plot\_spline(ax, points, num\_points, color='r'):

tck, u = splprep(points.T, k=2, s=0)

u\_new = np.linspace(u.min(), u.max(), num\_points)

spline\_points = splev(u\_new, tck)

ax.plot3D(\*spline\_points, color=color)

**Second.py**

# Застосування фільтра для виявлення контурів (Canny)

edges = cv2.Canny(image, 100, 200)

contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

max\_contour = max(contours, key=cv2.contourArea)

cv2.drawContours(image, [max\_contour], 0, (0, 255, 0), 2)

cv2.imshow('Contour', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**3.6. Аналіз результатів відлагодження та верифікації результатів роботи програми.**

Результати відладки та тестування підтвердили працездатність розробленого коду. Порівняння отриманих результатів з технічними умовами завдання на лабораторну роботу та верифікація функціоналу програмного коду показали, що всі вимоги були виконані у повному обсязі.

**IV. Висновки.**

Під час виконання лабораторної роботи було розроблено два програмних скрипта для реалізації векторних алгоритмів. Використано метод інтерполяції Сплайн для побудови сторін піраміди та алгоритм z-buffer для видалення невидимих ліній. Також реалізовано виділення контуру об’єкту на зображенні.