**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

Трикош І. В.

**Перевірив:**

Професор кафедри ОТ ФІОТ

Писарчук О.О.

**Київ 2024**

## **І. Мета:**

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**

Здійснити виконання завдання лабораторної роботи із застосуванням алгоритму інтерполяції для побудови векторного зображення 2D, 3D графічного об’єкту та алгоритму видалення невидимих ліній та поверхонь – програмний скрипт №1.

Реалізувати розробку програмного скрипта №2, 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.

## **ІІІ. Результати виконання лабораторної роботи.**

* 1. **Синтезована математична модель**

Відповідно до умов задачі синтезовано математичні моделі операцій над структурами вхідних графічних об’єктів.

Для програмного скрипту №1 модель є реалізує алгоритм інтерполяції та видалення невидимих граней за допомогою алгоритму плаваючого обрію.

Інтерполяція за двома точками знаходиться за допомогою поліному Лагранжа:

де (x1, y1) – перша точка, (x2, y2) – друга точка, x – точка, для якої визначається інтерпольоване значення y.

Видалення невидимих граней виконується за допомогою алгоритму плаваючого обрію. Цей алгоритм працює наступним чином: задаються максимальні значення x, y, z. І точки граней порівнюються між собою. Для визначення передньої та задньої граней використовується координата z. Для визначення лівої та правої грані використовується координати x. Для визначення, чи відображати нижню грань, використовується координата y.

Для програмного скрипту №2 модель реалізує виділення контору обраного об’єкту на цифровому растровому зображенні. При цьому відбувається попередня обробка зображення, а саме корекція кольору та розмиття шуму для кращого розпізнавання контурів об’єктів (трикутників).

* 1. **Результати архітектурного проектування та їх опис**

На рис. 1 зображена блок-схема до моделі до програмного скрипту №1:



Рисунок 1 – Блок схема алгоритму №1

Робота алгоритму розпочинається із задання початкових параметрів. Сюди входить ширина та висота вікна, розмір сторони основи піраміди, висота піраміди, точка переміщення, кут проєкції та формування точок піраміди (вершина та 4 точки основи), кількість точок інтерполяції. Ці дії виконуються у блоці 1 блок-схеми алгоритму рис.1.

У блоці 2 виконується аксонометрія піраміди на кут θ по осях Y та X.

У блоці 3 виконується зміщення фігури на точку, задану в блоці 1.

У блоці 4 виконується проєкція фігури на площину xy.

У блоці 5 відображається фігура перед застосуванням інтерполяції та видаленням невидимих граней.

У блоці 6 виконується відображення інтерпольованої фігури за допомогою поліному Лагранжа. Між точками піраміди виконується інтерполяція. Кількість точок задається в блоці 1.

У блоці 7 виконується відображення фігури з видаленими невидимим гранями за допомогою алгоритму плаваючого обрію. На цьому виконання алгоритму завершено.

На рис. 2 зображена блок-схема до моделі до програмного скрипту №2:



Рисунок 2 – Блок схема алгоритму №2

Робота алгоритму розпочинається із зчитування зображення з допомогою бібліотеки OpenCV. Ці дії виконуються у блоці 1 блок-схеми алгоритму рис.1.

У блоці 2 виконується попередня обробка зображень. Тут зображення перетворюється на градацію сірого, застосовується Гаусівське розмиття, виявлення країв за допомогою Canny і виконується морфологічне закриття.

У блоці 3 виконується виділення зовнішніх контурів об’єктів.

У блоці 4 виконується визначення трикутників. Це відбувається за допомогою фільтрації по 3 вершинах у контурі.

У блоці 5 відображається результат виділення контурів.

У блоці 6 виконується збереження результату. На цьому виконання алгоритму завершено.

* 1. **Опис структури проекту програми**

Для реалізації розробленого алгоритму мовою програмування Python з використанням можливостей інтегрованого середовища PyCharm сформовано проєкт.

Проєкт базується на лінійній бізнес-логіці функціонального програмування та має таку структуру.

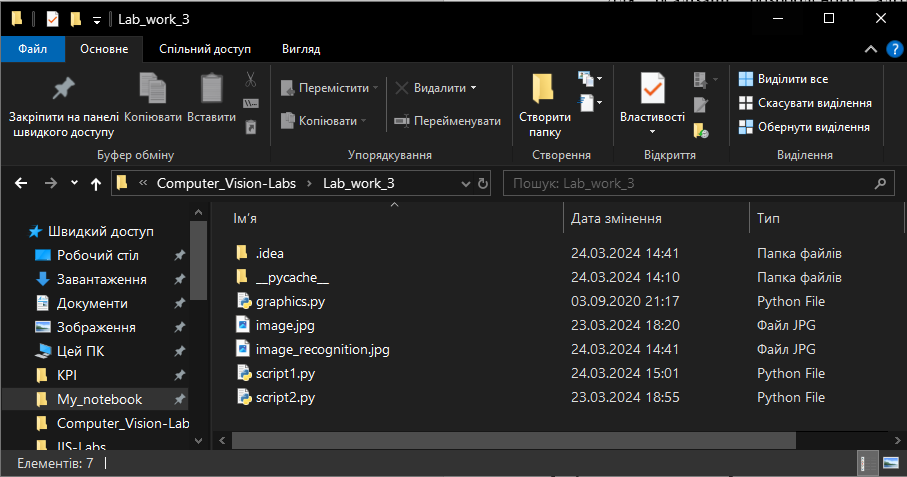


Рисунок 3 – Структура проєкту

Lab\_work\_3 – головний каталог проєкту

graphics.py – графічний модуль

script1.py – файл програмного коду №1

script2.py – файл програмного коду №2

image.jpg – зображення для тестування програми

image\_recognition.jpg – результат виділення контуру трикутників.

* 1. **Результати роботи програми відповідно до завдання (допускається у формі скриншотів)**

Результатом роботи програми є сукупність послідовності графічних вікон, що реалізують умови завдання лабораторної роботи.

1. Початкова фігура:

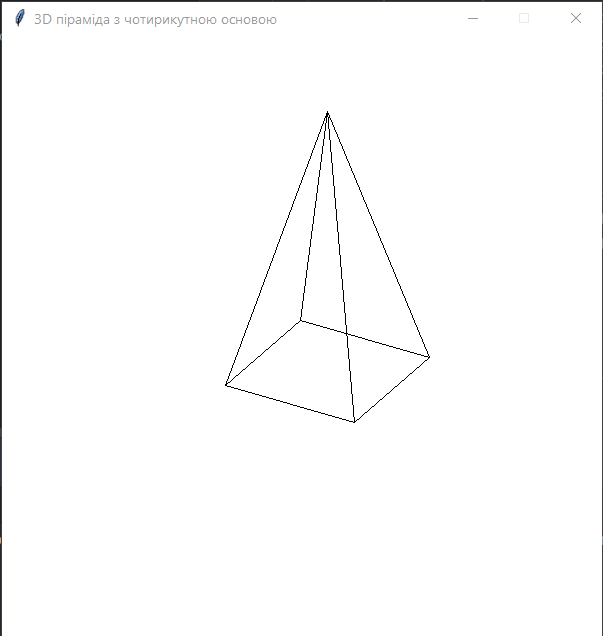


Рисунок 4 – Початкова фігура

1. Інтерполяція фігури:

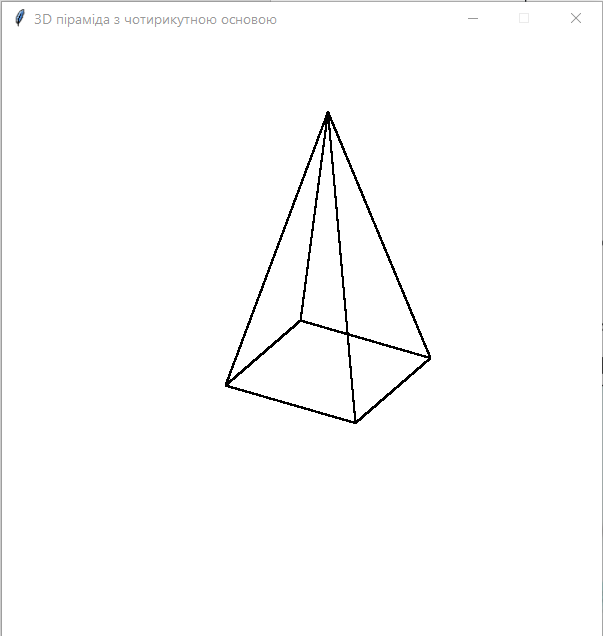


Рисунок 5 – Інтерполяція фігури

1. Видалення невидимих граней фігури:

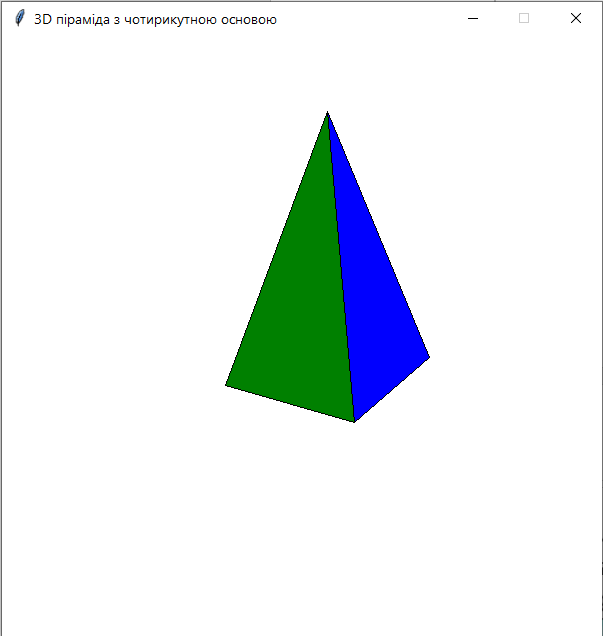


Рисунок 6 – Видалення невидимих граней фігури

1. Початкове зображення для виділення контурів трикутників:

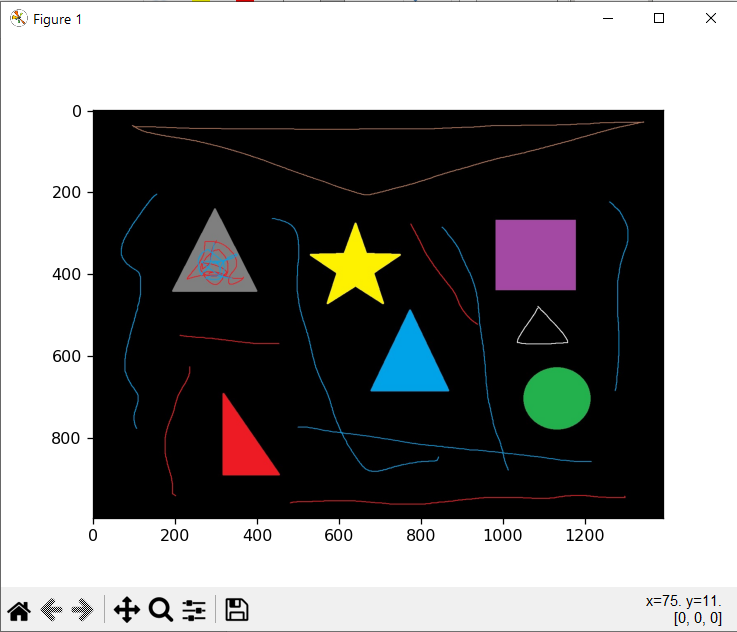


Рисунок 7 – Початкове зображення

1. Оброблене зображення:

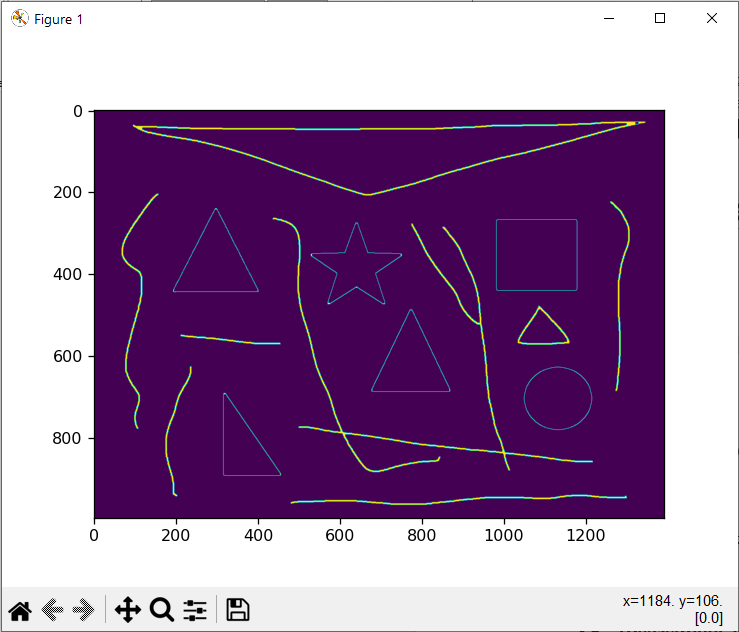


Рисунок 8 – Оброблене зображення

1. Виділення трикутників:

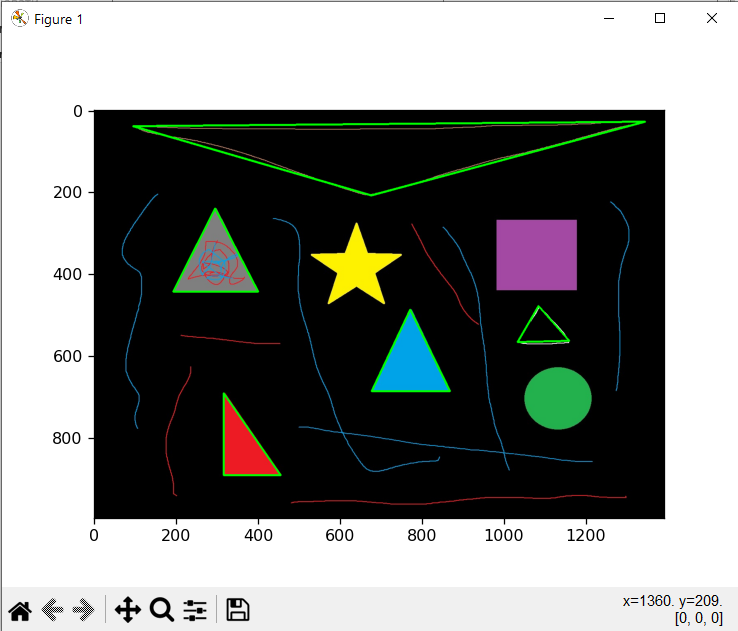


Рисунок 9 – Виділення трикутників

Таким чином, представлені результати у повному обсязі відповідають завданню лабораторної роботи.

* 1. **Програмний код, що забезпечує отримання результату (допускається у формі скриншотів)**

Програмний код послідовно реалізує алгоритми рис.1-2 та спрямовано на отримання результатів, поданих на рис.4-9.

Для спрощення програмного коду і раціоналізації обчислень застосовано функціональні механізми створення підпрограм.

Обчислення проводились з матрицею растра зображення.

При цьому використано можливості Python бібліотек: numpy, math, matplotlib, graphics, OpenCV.

Контексні коментарі пояснюють сутність окремих скриптів наведеного коду програми.

script1.py

*"""  
--------------------- Алгоритми над 3D фігурами ---------------------------  
Завдання:  
Синтез 3D об'єкту (піраміда з чотирикутною основою) та наступні дії над ним:  
1. Застосувати алгоритм інтерполяції методом Лагранжа;  
2. Застосувати алгоритм видалення невидимих ліній методом плаваючого обрію.  
"""  
  
from* graphics *import* \*  
*import* numpy *as* np  
*import* math  
  
*# Параметри вікна*WINDOW\_WIDTH = 600  
WINDOW\_HEIGHT = 600  
  
*# Параметри піраміди*PYRAMID\_WIDTH = 150  
PYRAMID\_HEIGHT = 300  
  
*# Координати піраміди. Перша - вершина*pyramid = np.array([[0, 0, 0, 1],  
 [-PYRAMID\_WIDTH//2, PYRAMID\_HEIGHT, -PYRAMID\_WIDTH//2, 1],  
 [+PYRAMID\_WIDTH//2, PYRAMID\_HEIGHT, -PYRAMID\_WIDTH//2, 1],  
 [-PYRAMID\_WIDTH//2, PYRAMID\_HEIGHT, +PYRAMID\_WIDTH//2, 1],  
 [+PYRAMID\_WIDTH//2, PYRAMID\_HEIGHT, +PYRAMID\_WIDTH//2, 1]])  
  
  
*def* project\_on\_x\_y(figure):  
 *""" Спроєктувати на площину XY """* t = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1]])  
 projection\_x\_y = figure.dot(t)  
 *return* projection\_x\_y  
  
  
*def* shift\_figure(figure, l, m, n):  
 *""" Змістити фігуру """* t = np.array([[1, 0, 0, l], [0, 1, 0, m], [0, 0, 1, n], [0, 0, 0, 1]])  
 tt = t.T  
 shifted\_figure = figure.dot(tt)  
 *return* shifted\_figure  
  
  
*def* degrees\_to\_radians(angle):  
 *""" Конвертувати градуси в радіани """  
 return* (math.pi \* angle) / 180  
  
  
*def* to\_axonometric\_view(figure, theta\_d\_1, theta\_d\_2):  
 *""" Зробити аксонометричну проєкцію """* theta\_r\_1 = degrees\_to\_radians(theta\_d\_1)  
 theta\_r\_2 = degrees\_to\_radians(theta\_d\_2)  
  
 t1 = np.array([[math.cos(theta\_r\_1), 0, -math.sin(theta\_r\_1), 0],  
 [0, 1, 0, 0],  
 [math.sin(theta\_r\_1), 0, math.cos(theta\_r\_1), 0],  
 [0, 0, 0, 1]])  
 view\_y = figure.dot(t1)  
  
 t2 = np.array([[1, 0, 0, 0],  
 [0, math.cos(theta\_r\_2), math.sin(theta\_r\_2), 0],  
 [0, -math.sin(theta\_r\_2), math.cos(theta\_r\_2), 0],  
 [0, 0, 0, 1]])  
 view\_y\_x = view\_y.dot(t2)  
 *return* view\_y\_x  
  
  
*def* lagrange\_interpolation(x, x1, y1, x2, y2):  
 *"""  
 Інтерполяція за допомогою методу Лагранжа.  
 Формула для обчислення за двома точками: y - y1 = (y2-y1) / (x2-x1) \* (x-x1)  
 """  
 return* (y2 - y1) / (x2 - x1) \* (x - x1) + y1  
  
  
*def* apply\_interpolation(x1, y1, x2, y2, num\_points):  
 *""" Застосувати інтерполяцію для відрізка між точками (x1, y1) та (x2, y2) """* interpolated\_points = []  
 *for* k *in* range(num\_points):  
 x = x1 + (x2 - x1) \* k / num\_points  
 y = lagrange\_interpolation(x, x1, y1, x2, y2)  
 interpolated\_points.append([x, y])  
 *return* interpolated\_points  
  
  
*class* Figure:  
 *def* \_\_init\_\_(*self*, window):  
 *self*.window = window  
  
 *self*.side1 = Polygon()  
 *self*.side2 = Polygon()  
 *self*.side3 = Polygon()  
 *self*.side4 = Polygon()  
 *self*.side5 = Polygon()  
  
 *def* \_\_clear\_figure(*self*):  
 *""" Очистити попередню фігуру """  
 self*.side1.undraw()  
 *self*.side2.undraw()  
 *self*.side3.undraw()  
 *self*.side4.undraw()  
 *self*.side5.undraw()  
  
 *def* \_\_draw\_figure(*self*):  
 *""" Намалювати фігуру """  
 self*.side1.draw(*self*.window)  
 *self*.side2.draw(*self*.window)  
 *self*.side3.draw(*self*.window)  
 *self*.side4.draw(*self*.window)  
 *self*.side5.draw(*self*.window)  
  
 *def* visualize(*self*, figure):  
 *""" Візуалізувати фігуру """* e\_x, e\_y = figure[0, 0], figure[0, 1]  
 a\_x, a\_y = figure[1, 0], figure[1, 1]  
 b\_x, b\_y = figure[2, 0], figure[2, 1]  
 c\_x, c\_y = figure[3, 0], figure[3, 1]  
 d\_x, d\_y = figure[4, 0], figure[4, 1]  
  
 *self*.\_\_clear\_figure()  
  
 *self*.side1 = Polygon(Point(a\_x, a\_y), Point(b\_x, b\_y), Point(d\_x, d\_y), Point(c\_x, c\_y))  
 *self*.side2 = Polygon(Point(a\_x, a\_y), Point(b\_x, b\_y), Point(e\_x, e\_y))  
 *self*.side3 = Polygon(Point(a\_x, a\_y), Point(c\_x, c\_y), Point(e\_x, e\_y))  
 *self*.side4 = Polygon(Point(b\_x, b\_y), Point(d\_x, d\_y), Point(e\_x, e\_y))  
 *self*.side5 = Polygon(Point(c\_x, c\_y), Point(d\_x, d\_y), Point(e\_x, e\_y))  
  
 *self*.\_\_draw\_figure()  
  
 *def* visualize\_interpolation(*self*, figure, num\_points=1000):  
 *""" Виконати інтерполяцію та візуалізувати фігуру """* e\_x, e\_y = figure[0, 0], figure[0, 1]  
 a\_x, a\_y = figure[1, 0], figure[1, 1]  
 b\_x, b\_y = figure[2, 0], figure[2, 1]  
 c\_x, c\_y = figure[3, 0], figure[3, 1]  
 d\_x, d\_y = figure[4, 0], figure[4, 1]  
  
 *# Отримати інтерпольовані точки між вершиною та точками основи* interpolated\_points = []  
 interpolated\_points.extend(apply\_interpolation(e\_x, e\_y, a\_x, a\_y, num\_points))  
 interpolated\_points.extend(apply\_interpolation(e\_x, e\_y, b\_x, b\_y, num\_points))  
 interpolated\_points.extend(apply\_interpolation(e\_x, e\_y, c\_x, c\_y, num\_points))  
 interpolated\_points.extend(apply\_interpolation(e\_x, e\_y, d\_x, d\_y, num\_points))  
 *# Отримати інтерпольовані точки між точками основи* interpolated\_points.extend(apply\_interpolation(a\_x, a\_y, b\_x, b\_y, num\_points))  
 interpolated\_points.extend(apply\_interpolation(a\_x, a\_y, c\_x, c\_y, num\_points))  
 interpolated\_points.extend(apply\_interpolation(d\_x, d\_y, b\_x, b\_y, num\_points))  
 interpolated\_points.extend(apply\_interpolation(d\_x, d\_y, c\_x, c\_y, num\_points))  
  
 *self*.\_\_clear\_figure()  
  
 *# Намалювати кожну інтерпольовану точку  
 for* i *in* range(len(interpolated\_points)):  
 point = Point(\*interpolated\_points[i])  
 point.draw(*self*.window)  
  
 *def* clear\_window(*self*):  
 clear = Rectangle(Point(0, 0), Point(WINDOW\_WIDTH, WINDOW\_HEIGHT))  
 clear.setFill('white')  
 clear.setOutline('white')  
 clear.draw(*self*.window)  
  
 *def* visualize\_removing(*self*, figure, projection, x\_max, y\_max, z\_max):  
 *""" Видалення невидимих граней та відображення фігури """* e\_x, e\_y, e\_z = figure[0, 0], figure[0, 1], figure[0, 2]  
 a\_x, a\_y, a\_z = figure[1, 0], figure[1, 1], figure[1, 2]  
 b\_x, b\_y, b\_z = figure[2, 0], figure[2, 1], figure[2, 2]  
 c\_x, c\_y, c\_z = figure[3, 0], figure[3, 1], figure[3, 2]  
 d\_x, d\_y, d\_z = figure[4, 0], figure[4, 1], figure[4, 2]  
  
 *# Визначення передньої/задньої грані по координаті z  
 if* (abs(a\_z - z\_max) > abs(c\_z - z\_max)) *and* (abs(b\_z - z\_max) > abs(d\_z - z\_max)):  
 is\_ab\_front = *True  
 else*:  
 is\_ab\_front = *False  
  
 # Визначення лівої/правої грані по координаті x  
 if* (abs(b\_x - x\_max) > abs(a\_x - x\_max)) *and* (abs(d\_x - x\_max) > abs(c\_x - x\_max)):  
 is\_bd\_left = *True  
 else*:  
 is\_bd\_left = *False  
  
 # Визначення нижньої грані по координаті y  
 if* abs(a\_y - y\_max) > abs(e\_y - y\_max):  
 is\_bottom = *True  
 else*:  
 is\_bottom = *False* e\_x, e\_y = projection[0, 0], projection[0, 1]  
 a\_x, a\_y = projection[1, 0], projection[1, 1]  
 b\_x, b\_y = projection[2, 0], projection[2, 1]  
 c\_x, c\_y = projection[3, 0], projection[3, 1]  
 d\_x, d\_y = projection[4, 0], projection[4, 1]  
  
 *self*.clear\_window()  
  
 *self*.side1 = Polygon(Point(a\_x, a\_y), Point(b\_x, b\_y), Point(d\_x, d\_y), Point(c\_x, c\_y))  
 *self*.side2 = Polygon(Point(a\_x, a\_y), Point(b\_x, b\_y), Point(e\_x, e\_y))  
 *self*.side3 = Polygon(Point(a\_x, a\_y), Point(c\_x, c\_y), Point(e\_x, e\_y))  
 *self*.side4 = Polygon(Point(b\_x, b\_y), Point(d\_x, d\_y), Point(e\_x, e\_y))  
 *self*.side5 = Polygon(Point(c\_x, c\_y), Point(d\_x, d\_y), Point(e\_x, e\_y))  
  
 *if* is\_bottom:  
 *self*.side1.setFill('black')  
 *self*.side1.draw(*self*.window)  
  
 *if* is\_ab\_front:  
 *self*.side2.setFill('green')  
 *self*.side2.draw(*self*.window)  
 *else*:  
 *self*.side5.setFill('yellow')  
 *self*.side5.draw(*self*.window)  
  
 *if* is\_bd\_left:  
 *self*.side3.setFill('red')  
 *self*.side3.draw(*self*.window)  
 *else*:  
 *self*.side4.setFill('blue')  
 *self*.side4.draw(*self*.window)  
  
  
win = GraphWin("3D піраміда з чотирикутною основою", WINDOW\_WIDTH, WINDOW\_HEIGHT)  
win.setBackground('white')  
  
pyramid\_figure = Figure(win)  
  
*# Точка зсуву фігури*shift\_point = [325, 75, 0]  
*# Кут проєкції*theta = 30  
  
axonometric\_pyramid = to\_axonometric\_view(pyramid, theta, theta)  
shifted\_pyramid = shift\_figure(axonometric\_pyramid, \*shift\_point)  
pyramid\_projection = project\_on\_x\_y(shifted\_pyramid)  
*# Звичайна піраміда*pyramid\_figure.visualize(pyramid\_projection)  
win.getMouse()  
*# Інтерполяція*pyramid\_figure.visualize\_interpolation(pyramid\_projection, 1000)  
win.getMouse()  
*# Видалення невидимих граней*pyramid\_figure.visualize\_removing(axonometric\_pyramid, pyramid\_projection, WINDOW\_WIDTH, WINDOW\_HEIGHT, WINDOW\_HEIGHT)  
win.getMouse()

script2.py

*"""  
Виділення контору трикутника на цифровому растровому зображенні  
"""  
  
import* cv2  
*from* matplotlib *import* pyplot *as* plt  
  
  
*def* show\_image(image):  
 *""" Показ зображення """* plt.imshow(image)  
 plt.show()  
  
  
*def* image\_read(file\_image):  
 *""" Зчитування зображення з файлу """* image = cv2.imread(file\_image)  
 image\_in\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
 *return* image\_in\_rgb  
  
  
*def* image\_processing(image):  
 *""" Обробка зображення """* gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)  
 blurred = cv2.GaussianBlur(gray, (5, 5), 0)  
 edged = cv2.Canny(blurred, 50, 150)  
 kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5))  
 closed = cv2.morphologyEx(edged, cv2.MORPH\_CLOSE, kernel)  
 *return* closed  
  
  
*def* image\_contours(image):  
 *""" Знаходження контурів """* contours, \_ = cv2.findContours(image.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 *return* contours  
  
  
*def* save\_result(file\_name, image\_in\_rgb):  
 *""" Збереження результату у файл """* image\_in\_bgr = cv2.cvtColor(image\_in\_rgb, cv2.COLOR\_RGB2BGR)  
 cv2.imwrite(file\_name, image\_in\_bgr)  
  
  
*def* image\_recognition(image, contours):  
 *""" Розпізнавання трикутника """* total\_triangles = 0  
 *for* contour *in* contours:  
 peri = cv2.arcLength(contour, *True*)  
 approx = cv2.approxPolyDP(contour, 0.05 \* peri, *True*)  
 *if* len(approx) == 3:  
 cv2.drawContours(image, [approx], -1, (0, 255, 0), 4)  
 total\_triangles += 1  
 *return* image, total\_triangles  
  
  
image\_entrance = image\_read("image.jpg")  
show\_image(image\_entrance)  
  
image\_exit = image\_processing(image\_entrance)  
show\_image(image\_exit)  
  
image\_contours = image\_contours(image\_exit)  
  
result, total\_figures = image\_recognition(image\_entrance, image\_contours)  
print(f"Знайдено {total\_figures} трикутних об'єктів")  
save\_result("image\_recognition.jpg", result)  
show\_image(result)

## **IV. Висновки.**

У ході виконання лабораторної роботи було проведено дослідження особливостей реалізації алгоритмів формування та обробки векторних цифрових зображень на прикладі застосування алгоритмів інтерполяції з допомогою поліномів Лагранжа складних 3D растрових об’єктів (піраміда з чотирикутною основою), застосування технологій видалення невидимих граней з допомогою алгоритму плаваючого обрію, розпізнавання об’єктів на цифровому растровому зображенні з використанням засобів мови програмування Python.

Виконав: студент Трикош І. В.