**Міністерство освіти і науки України**

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

**Кафедра обчислювальної техніки ФІОТ**

**ЗВІТ**

**з виконання завдань заліку**

**з навчальної дисципліни «Computer Vision»**

**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. Білет №9:**

## **ІІ. Завдання:**

1. Що таке растрові зображення, форми їх представлення, характеристики.
2. Алгоритм k-Means / Hard-c-means / k-середніх.
3. Розробити програмний скрипт, що забезпечує реалізацію алгоритмів зафарбовування та видалення невидимих ліній / граней над 3D графічними примітивами. Тип графічного об’єкту обрати самостійно.

## **ІІІ. Результати виконання модульної контрольної роботи.**

* 1. **Відповідь на теоретичне питання №1.**

Растрове зображення – це зображення, що складається з пікселів (найменша одиниця зображення, що має власний колір), які разом формують матрицю растра із заданими координатами та яскравістю (для монохрому) або кольором (для кольорового зображення).

До пікселів є певні вимоги: усі фігури пікселя повинні бути однаковими і покривати усю площину без пропусків та накладань.

Є такі типи растрів:

* Прямокутний
* Трикутний
* Шестикутний
* Круглий

Характеристики растрових зображень:

* Роздільна здатність – вимірюється у пікселях на дюйм або точках на дюйм. Вища роздільна здатність призводить до чіткішого та деталізованішого зображення, проте таке зображення має більший розмір.
* Глибина кольору – визначає кількість бітів, які використовуються для представлення кольору кожного пікселя. Для двокольорових зображень це 1 біт на піксель, для напівтону – це 8 бітів, для кольорового зображення – від 16 бітів на піксель.
* Частота сітки – визначає кількість ліній на дюйм та характеризує якість друку зображення.
* Інтенсивність тону – визначає рівень освітленості зображення. Зазвичай 256, мінімальний рівень – 150.

Растрові зображення можуть зберігатися у наступних форматах: JPG, PNG, BMP, TIFF, тощо.

* 1. **Відповідь на теоретичне питання №2.**

Алгоритм k-means (іноді називають Hard-c-means або k-середніх) – це один із алгоритмів кластеризації (впорядкування множини об’єктів у порівняно однорідні групи).

Мета алгоритму – розділити n спостережень на k кластерів так, щоб кожне спостереження належало до кластера з найближчим до нього середнім значенням. Метод базується на мінімізації суми квадратів відстаней між кожним спостереженням та центром його кластера. Тобто, відбувається мінімізації функції

де – метрика, – i-тий об’єкт даних, а – центр кластера, якому на -тій ітерації приписаний елемент .

Опис алгоритму:

1. Ініціалізація – обирається кількість кластерів k та випадкові точки з набору даних як початкові центроїди.
2. Призначення кластерів – для кожної точки знаходиться найближчий центроїд. Для цього обчислюється відстань від точки до кожного центроїда та призначення точки до кластера з найближчим центроїдом.
3. Оновлення центроїдів – обчислення нових центроїдів як середнє значення точок, що входять до кластера. Центроїд кожного кластера переміщується до середини точок, що належать до цього кластера.
4. Повторення кроків 2 та 3, допоки центроїди не перестануть змінюватися (тобто досягнуто збіжності) або до досягнення заданої кількості ітерацій.

Переваги алгоритму:

* Простота реалізації
* Швидкість виконання

Недоліки алгоритму:

* Необхідність знати кількість кластерів заздалегідь
* Чутливість до вибору початкових центроїдів та до викидів, які можуть викривлювати середнє
  1. **Відповідь на практичне питання №3.**
     1. **Синтезована математична модель**

Відповідно до умов задачі синтезовано математичну модель, що реалізує зафарбовування та видалення невидимих граней піраміди з чотирикутною основою.

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.

Окрім, цього, модель також виконує додаткові дії на 3D об’єктом, а саме переміщення, створення аксонометричної та ортогональної проєкції об’єкту.

Переміщення 3D об’єкту відбувається наступним чином:

де – координати точки переміщення.

Аксонометрична проєкція 3D об’єкту виконується наступним чином:

де – кут повороту (в радіанах) відносно осі Y, – кут повороту (в радіанах) відносно осі X.

Ортогональна проєкція 3D об’єкту на площину XY здійснюється наступним чином:

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

На рис. 1 зображена блок-схема до моделі до програмного скрипту:

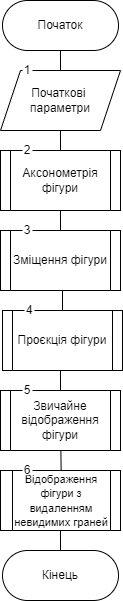


Рисунок 1 – Блок схема алгоритму

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). Ці дії виконуються у блоці 1 блок-схеми алгоритму рис.1.

У блоці 2 виконується аксонометрія піраміди на кути Φ та θ по осях Y та X відповідно.

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 видаленням невидимих граней.

У блоці 6 виконується відображення фігури з видаленими невидимими гранями за допомогою алгоритму плаваючого обрію. 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.

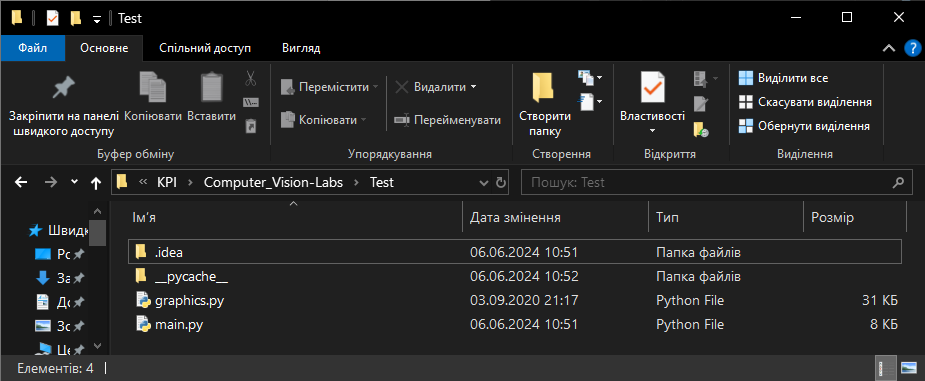


Рисунок 2 – Структура проєкту

Test – головний каталог проєкту

main.py – файл програмного коду

graphics.py – графічний модуль

* + 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:

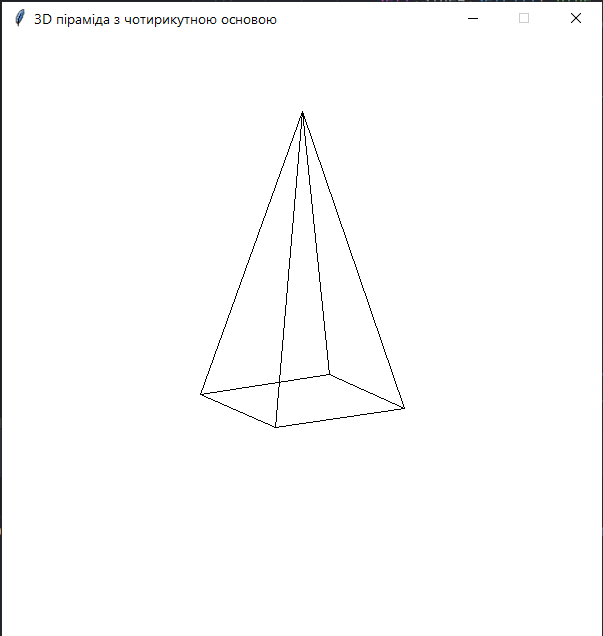


Рисунок 3 – Початкова фігура

1. Видалення невидимих граней фігури та її зафарбування:

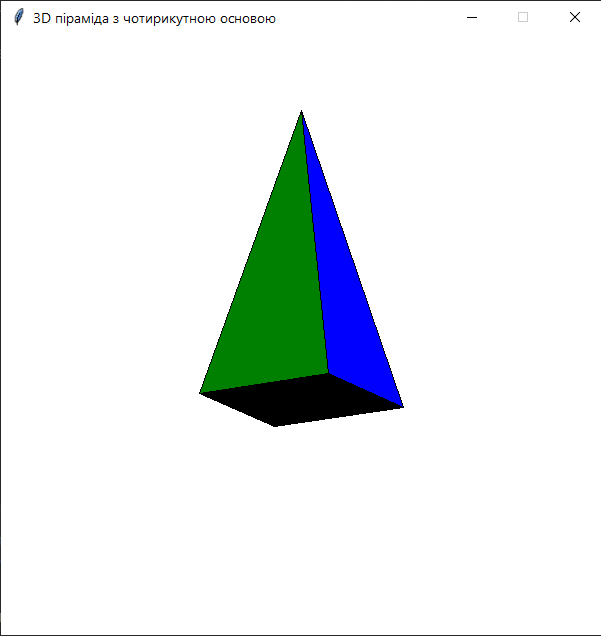


Рисунок 4 – Видалення невидимих граней фігури і її зафарбування

Бачимо, що алгоритм зафарбував передні грані, тому поставлену задача можна вважати виконаною.

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

Програмний код послідовно реалізує алгоритм на рис.1 та спрямований на отримання результатів, поданих на рис.3-4.

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

При цьому використано можливості Python бібліотек: graphics.

Контекстні коментарі пояснюють сутність окремих скриптів наведеного коду програми.

main.py

*"""  
--------------------- Алгоритми над 3D фігурами ---------------------------  
Завдання:  
Синтез 3D об'єкту (піраміда з чотирикутною основою) та наступні дії над ним:  
1. Застосувати алгоритм зафарбовування та видалення невидимих граней 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  
  
  
*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* 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(e\_y - y\_max) > abs(a\_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 = [300, 75, 0]  
*# Кути проєкції*theta1 = 30 *# По X*theta2 = -15 *# По Y*axonometric\_pyramid = to\_axonometric\_view(pyramid, theta1, theta2)  
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()  
*# Видалення невидимих граней*pyramid\_figure.visualize\_removing(axonometric\_pyramid, pyramid\_projection, WINDOW\_WIDTH, WINDOW\_HEIGHT, WINDOW\_HEIGHT)  
win.getMouse()

Виконав: студент Трикош І. В.