Обробка зображень
Лабораторна робота №3
Грищенко Юрій, ІПС-32

Опис завдання:

Розглянемо метод апроксимації яскравості функцією (поліномом другого порядку) для вікна 3×3

$$\begin{bmatrix} f(x-1, y-1) & f(x-1, y) & f(x-1, y+1) \\ f(x, y-1) & f(x, y) & f(x, y+1) \\ f(x+1, y-1) & f(x+1, y) & f(x+1, y+1) \end{bmatrix}. \quad (1)$$

Для елементів цього вікна побудувати апроксимуючу площину

$$\hat{f}(x, y) = ax^2 + by^2 + cxy + \alpha x + \beta y + \gamma. \quad (2)$$

Якщо площина побудована, тобто відомі $a, b, c, \alpha, \beta, \gamma$, то можна знайти частинні похідні

$$\frac{\partial \hat{f}(x, y)}{\partial x} = 2ax + cy + \alpha, \quad \frac{\partial \hat{f}(x, y)}{\partial y} = 2by + cx + \beta. \quad (3)$$

1) Обчислити модуль градієнта, це - ознака локального перепаду яскравості:

$$\left|\nabla \hat{f}(x, y)\right| = \sqrt{\left(\frac{\partial \hat{f}(x, y)}{\partial x}\right)^2 + \left(\frac{\partial \hat{f}(x, y)}{\partial y}\right)^2}. \quad (4)$$

2) Відобразити значення матриці системи лінійних алгебраїчних рівнянь, які отримуються з необхідних умов мінімуму квадратичної похибки при апроксимації функцією (2).

$Ax = b$, де вектор $x^T = (a, b, c, \alpha, \beta, \gamma)$, матриця A розмірності 6×6, вектор b розмірності 6.

Програмна реалізація:

Для побудови площини використовуємо метод найменших квадратів.

Мінімізуємо величину:

$$\epsilon^2 = (\hat{f}(x-1, y-1) - f(x-1, y-1))^2 + (\hat{f}(x, y-1) - f(x, y-1))^2 + \ldots + (\hat{f}(x+1, y+1) - f(x+1, y+1))^2$$

З умов мінімуму маємо 6 похідних:

$$\frac{\partial \epsilon^2}{\partial a} = 0, \frac{\partial \epsilon^2}{\partial b} = 0, \ldots \frac{\partial \epsilon^2}{\partial \gamma} = 0$$

Для простоти обчислень розглянемо область навколо f(1, 1).

Завдяки бібліотеці SymPy отримаємо таку систему лінійних рівнянь:

102*a + 50*b + 54*c + 54*α + 30*β + 30*γ - 2*f(1, 0) - 2*f(1, 1) - 2*f(1, 2) - 8*f(2, 0) - 8*f(2, 1) - 8*f(2, 2) = 0

50*a + 102*b + 54*c + 30*α + 54*β + 30*γ - 2*f(0, 1) - 8*f(0, 2) - 2*f(1, 1) - 8*f(1, 2) - 2*f(2, 1) - 8*f(2, 2) = 0

54*a + 54*b + 50*c + 30*α + 30*β + 18*γ - 2*f(1, 1) - 4*f(1, 2) - 4*f(2, 1) - 8*f(2, 2) = 0

54*a + 30*b + 30*c + 30*α + 18*β + 18*γ - 2*f(1, 0) - 2*f(1, 1) - 2*f(1, 2) - 4*f(2, 0) - 4*f(2, 1) - 4*f(2, 2) = 0

30*a + 54*b + 30*c + 18*α + 30*β + 18*γ - 2*f(0, 1) - 4*f(0, 2) - 2*f(1, 1) - 4*f(1, 2) - 2*f(2, 1) - 4*f(2, 2) = 0

30*a + 30*b + 18*c + 18*α + 18*β + 18*γ - 2*f(0, 0) - 2*f(0, 1) - 2*f(0, 2) - 2*f(1, 0) - 2*f(1, 1) - 2*f(1, 2) - 2*f(2, 0) - 2*f(2, 1) - 2*f(2, 2) = 0

Жирним виділена матриця A, все інше вважаємо за вектор b.

Також за допомогою SymPy знаходимо розв’язок СЛАР:

a = f(0, 0)/6 + f(0, 1)/6 + f(0, 2)/6 - f(1, 0)/3 - f(1, 1)/3 - f(1, 2)/3 + f(2, 0)/6 + f(2, 1)/6 + f(2, 2)/6

b = f(0, 0)/6 - f(0, 1)/3 + f(0, 2)/6 + f(1, 0)/6 - f(1, 1)/3 + f(1, 2)/6 + f(2, 0)/6 - f(2, 1)/3 + f(2, 2)/6

c = f(0, 0)/4 - f(0, 2)/4 - f(2, 0)/4 + f(2, 2)/4

α = -3*f(0, 0)/4 - f(0, 1)/2 - f(0, 2)/4 + 2*f(1, 0)/3 + 2*f(1, 1)/3 + 2*f(1, 2)/3 + f(2, 0)/12 - f(2, 1)/6 - 5*f(2, 2)/12

β = -3*f(0, 0)/4 + 2*f(0, 1)/3 + f(0, 2)/12 - f(1, 0)/2 + 2*f(1, 1)/3 - f(1, 2)/6 - f(2, 0)/4 + 2*f(2, 1)/3 - 5*f(2, 2)/12

γ = 29*f(0, 0)/36 + 2*f(0, 1)/9 - f(0, 2)/36 + 2*f(1, 0)/9 - f(1, 1)/9 - f(1, 2)/9 - f(2, 0)/36 - f(2, 1)/9 + 5*f(2, 2)/36

Завдяки функції lambdify ці розв’язки перетворюємо у функції, які досить швидко обчислюються мовою Python.

Загальний час виконання програми для зображення 256x256: 13 секунд (разом із запуском інтерпретатора Python, зчитуванням та записом зображення і т.д.)

```
from PIL import Image
import sys
import math
import numpy as np

from sympy import *
init_printing(use_unicode=True)


if len(sys.argv) <= 2:
    print("Usage: lab2.py [input] [output]")
    exit(1)


image = Image.open(sys.argv[1]).convert("L")
inputArr = np.asarray(image)
outputArr = np.zeros_like(inputArr)

print("Starting...")

x, y, a, b, c, alpha, beta, gamma = symbols('x y a b c \u03b1 \u03b2 \u03b3')
f = Function('f')

# Squared residual
f_a = a*x**2 + b*y**2 + c*x*y + alpha*x + beta*y + gamma
err_sq = 0
for i in [0, 1, 2]:
    for j in [0, 1, 2]:
        err_sq += (f_a.subs([(x, x + i), (y, y + j)]) - f(x + i, x + j)) ** 2


# Replace f(x, y) with f(0, 0)
err_minimization = []
for var in [a, b, c, alpha, beta, gamma]:
    err_minimization.append(diff(err_sq, var).subs([(x, 0), (y, 0)]))
    print(err_minimization[-1])
    print()

# Solve system of linear equations
solution = solve(err_minimization, [a, b, c, alpha, beta, gamma])

# Turn SymPy expressions into fast NumPy functions

offset_x = 0
offset_y = 0
def get_pixel_with_offset(x, y):
    return inputArr[offset_x + x, offset_y + y]

X = {}
```

```
for var in solution:
    X[var] = lambdify([], solution[var], [{'f': get_pixel_with_offset}, 'numpy'])


for x in range(inputArr.shape[0] - 2):
    for y in range(inputArr.shape[1] - 2):
        offset_x = x
        offset_y = y
        a_val = X[a]()
        b_val = X[b]()
        c_val = X[c]()
        alpha_val = X[alpha]()
        beta_val = X[beta]()
        outputArr[x, y] = math.sqrt((2*a_val + c_val + alpha_val) ** 2 + (2*b_val + c_val + beta_val) **
2)

Image.fromarray(outputArr, 'L').save(sys.argv[2])
```