|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**ЛАБОРАТОРНАЯ РАБОТА №2**

**«**Базовые операции обработки изображений**»**

**ДИСЦИПЛИНА:** «Программные системы распознавания и обработки информации»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-31М | |  |  | ( | Сафронов Н.С, | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Гагарин Ю.Е. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

Калуга, 2025

**Цель:**

Изучить базовые операции обработки изображений с использованием реализаций соответствующих функций библиотеки компьютерного зрения OpenCV.

**Задачи:**

1. Изучить принцип работы базовых операций обработки изображений: линейная фильтрация, сглаживание с различными ядрами, морфологические преобразования, применение оператора Собеля, применение оператора Лапласа, вычисление и выравнивание гистограммы.
2. Рассмотреть прототипы функций, реализующих перечисленные операции в библиотеке OpenCV.
3. Разработать простые примеры использования указанного набора функций.

**Задание**

Разработка консольного редактора изображений. На данном этапе предлагается разработать простейший консольный редактор изображений, предусматривающий возможность применения всех операций, перечисленных в предыдущих разделах. К приложению предъявляются следующие требования:

Организация диалога с пользователем. Предполагается вывод перечня пунктов меню (загрузка изображения и набор операций) и возможность выбора определенной операции. Отметим, что программа должна обеспечивать многократное выполнение различных операций.

• Отображение исходного изображения.

• Отображение результата применения операции.

• Контроль корректности вводимых пользователем данных.

**Вариант 6**

Добавьте возможность рисования геометрических примитивов в разработанный консольный редактор. Предусмотрите возможность удаления отрисованных примитивов.

**Результаты выполнения работы**

**Изображение выглядит как текст, программное обеспечение, снимок экрана, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**Рисунок 1 –** Пример загруженного изображения

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**Рисунок 2 –** Рисование примитивов

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 3 –** Использование ластика для стирания примитивов

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 4 –** Применение Гауссовского размытия к изображению

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 5 –** Применение фильтра Canny к изображению

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 6 –** Сохранение результирующего изображения

**Вывод:** в процессе выполнения лабораторной работы были изучены базовые операции обработки изображений с использованием реализаций соответствующих функций библиотеки компьютерного зрения OpenCV.

**Листинг программы**

#!/usr/bin/env python3

import cv2

import math

import numpy as np

import tkinter as tk

from tkinter import ttk, colorchooser, messagebox, filedialog

from datetime import datetime

from copy import deepcopy

import typing as t

class HistoryEntity(t.TypedDict):

"""Запись в истории."""

primitives: list[dict]

canvas: np.ndarray | None

class EditorState:

"""Состояние редактора."""

def \_\_init\_\_(self):

"""."""

self.canvas: np.ndarray | None = None

self.fixed\_canvas: np.ndarray | None = None

self.background: np.ndarray | None = None

self.history: list[HistoryEntity] = []

self.redo\_stack: list[HistoryEntity] = []

self.primitives: list[dict] = []

self.current\_tool: str = "line"

self.color: tuple[int, int, int] = (0, 255, 0) # BGR

self.thickness: int = 3

self.drawing: bool = False

self.start\_point: tuple[int, int] | None = None

self.temp\_canvas: np.ndarray | None = None

state = EditorState()

def create\_canvas(width: int = 1024, height: int = 768, bg\_color: tuple[int, int, int] = (255, 255, 255)) -> np.ndarray:

"""Создаёт холст."""

return np.full((height, width, 3), bg\_color, dtype=np.uint8)

def redraw\_canvas():

"""Перерисовывает холст на основе примитивов."""

if state.fixed\_canvas is None:

if state.background is None:

return

state.fixed\_canvas = state.background

state.canvas = state.fixed\_canvas.copy()

for prim in state.primitives:

if prim["type"] == "line":

cv2.line(state.canvas, prim["pt1"], prim["pt2"], prim["color"], prim["thickness"])

elif prim["type"] == "rectangle":

cv2.rectangle(state.canvas, prim["pt1"], prim["pt2"], prim["color"], prim["thickness"])

elif prim["type"] == "circle":

cv2.circle(state.canvas, prim["center"], prim["radius"], prim["color"], prim["thickness"])

cv2.imshow("Canvas", state.canvas)

def save\_state():

"""Сохраняет состояние для undo."""

if state.canvas is not None:

state.history.append(HistoryEntity(

primitives=deepcopy(state.primitives),

canvas=state.fixed\_canvas,

))

state.redo\_stack.clear()

def find\_primitive\_at(x: int, y: int) -> int | None:

"""Находит индекс примитива под точкой (x,y)."""

for i in reversed(range(len(state.primitives))):

prim = state.primitives[i]

if prim["type"] == "line":

pt1, pt2 = prim["pt1"], prim["pt2"]

dist = cv2.pointPolygonTest(np.array([pt1, pt2]), (x, y), True)

if abs(dist) < 10:

return i

elif prim["type"] == "rectangle":

x1, y1 = prim["pt1"]

x2, y2 = prim["pt2"]

if min(x1, x2) <= x <= max(x1, x2) and min(y1, y2) <= y <= max(y1, y2):

return i

elif prim["type"] == "circle":

cx, cy = prim["center"]

radius = prim["radius"]

if np.sqrt((x - cx)\*\*2 + (y - cy)\*\*2) <= radius + 5:

return i

return None

def mouse\_callback(event, x, y, flags, param) -> None:

"""Обработка мыши на холсте."""

global state

if event == cv2.EVENT\_LBUTTONDOWN:

state.drawing = True

state.start\_point = (x, y)

state.temp\_canvas = state.canvas.copy()

elif event == cv2.EVENT\_MOUSEMOVE and state.drawing:

if state.start\_point is None or state.temp\_canvas is None:

return

img = state.temp\_canvas.copy()

color = (0, 0, 0)

thickness = max(1, state.thickness)

if state.current\_tool == "line":

cv2.line(img, state.start\_point, (x, y), color, thickness)

elif state.current\_tool == "rectangle":

cv2.rectangle(img, state.start\_point, (x, y), color, thickness)

elif state.current\_tool == "circle":

radius = int(np.sqrt((x - state.start\_point[0])\*\*2 + (y - state.start\_point[1])\*\*2))

cv2.circle(img, state.start\_point, radius, color, thickness)

cv2.imshow("Canvas", img)

elif event == cv2.EVENT\_LBUTTONUP:

state.drawing = False

if state.start\_point is None:

return

if state.current\_tool == "eraser":

idx = find\_primitive\_at(x, y)

if idx is not None:

state.primitives.pop(idx)

redraw\_canvas()

save\_state()

print("Примитив удалён")

else:

print("Нет примитива под курсором")

return

prim = {

"type": state.current\_tool,

"color": state.color,

"thickness": state.thickness

}

if state.current\_tool == "line":

prim.update({"pt1": state.start\_point, "pt2": (x, y)})

elif state.current\_tool == "rectangle":

prim.update({"pt1": state.start\_point, "pt2": (x, y)})

elif state.current\_tool == "circle":

radius = int(np.sqrt((x - state.start\_point[0])\*\*2 + (y - state.start\_point[1])\*\*2))

prim.update({"center": state.start\_point, "radius": radius})

state.primitives.append(prim)

redraw\_canvas()

save\_state()

print(f"Добавлен {state.current\_tool}")

def set\_tool(tool\_name: str):

"""Устанавливает текущий инструмент."""

state.current\_tool = tool\_name

print(f"Инструмент: {tool\_name}")

def choose\_color():

"""Открывает диалог выбора цвета."""

rgb, hex\_color = colorchooser.askcolor(

title="Выберите цвет",

initialcolor=('#%02x%02x%02x' % (state.color[2], state.color[1], state.color[0]))

)

if rgb:

state.color = (int(rgb[2]), int(rgb[1]), int(rgb[0]))

color\_btn.config(bg=hex\_color)

print(f"Цвет установлен: BGR{state.color}")

def update\_thickness(val: str):

"""Обновляет толщину."""

state.thickness = math.floor(float(val))

print(f"Толщина: {state.thickness}")

def undo\_action():

"""Отмена."""

if len(state.history) > 1:

last = state.history.pop()

state.redo\_stack.append(last)

state.fixed\_canvas = state.history[-1]['canvas']

state.primitives = state.history[-1]['primitives']

redraw\_canvas()

print("Отменено")

else:

print("Нечего отменять")

def redo\_action():

"""Повтор."""

if state.redo\_stack:

last = state.redo\_stack.pop()

state.history.append(last)

state.fixed\_canvas = last['canvas']

state.primitives = last['primitives']

redraw\_canvas()

print("Повторено")

else:

print("Нечего повторять")

def clear\_canvas() -> None:

"""Очистка."""

state.canvas = state.background.copy()

state.primitives.clear()

state.history.clear()

state.redo\_stack.clear()

save\_state()

cv2.imshow("Canvas", state.canvas)

print("Холст очищен")

def save\_canvas() -> None:

"""Сохранение."""

if state.canvas is None:

messagebox.showerror("Ошибка", "Нет изображения для сохранения")

return

timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

default = f"primitive\_{timestamp}.png"

filename = tk.filedialog.asksaveasfilename(

initialfile=default,

defaultextension=".png",

filetypes=[("PNG", "\*.png"), ("JPEG", "\*.jpg"), ("Все файлы", "\*.\*")]

)

if filename:

success = cv2.imwrite(filename, state.canvas)

if success:

messagebox.showinfo("Сохранено", f"Файл сохранён:\n{filename}")

else:

messagebox.showerror("Ошибка", "Не удалось сохранить файл")

def get\_kernel\_size\_from\_user(title: str = "Размер ядра") -> int:

"""Запрашивает размер ядра у пользователя через диалоговое окно."""

def on\_ok():

try:

val = int(entry.get())

if val < 3 or val % 2 == 0:

messagebox.showwarning("Ошибка", "Размер должен быть нечётным и >= 3")

return

nonlocal result

result = val

dlg.destroy()

except ValueError:

messagebox.showwarning("Ошибка", "Введите целое число")

result = None

dlg = tk.Toplevel()

dlg.title(title)

dlg.geometry("300x120")

dlg.transient()

dlg.grab\_set()

tk.Label(dlg, text="Введите нечётный размер ядра (>=3):").pack(pady=10)

entry = tk.Entry(dlg)

entry.pack(pady=5)

entry.insert(0, "5")

tk.Button(dlg, text="OK", command=on\_ok).pack(pady=5)

dlg.wait\_window()

return result if result else 5

def apply\_filter2d() -> None:

"""Применяет линейный фильтр резкости."""

kernel = np.array([[0, -1, 0],

[-1, 5, -1],

[0, -1, 0]], dtype=np.float32)

state.fixed\_canvas = cv2.filter2D(state.canvas, -1, kernel)

state.primitives = []

save\_state()

redraw\_canvas()

print("Применён фильтр резкости")

def apply\_blur() -> None:

"""Применяет усредняющий фильтр."""

ksize = get\_kernel\_size\_from\_user("Размер ядра для blur")

if ksize:

state.fixed\_canvas = cv2.blur(state.canvas, (ksize, ksize))

state.primitives = []

save\_state()

redraw\_canvas()

print(f"Применён blur с ядром {ksize}")

def apply\_median\_blur() -> None:

"""Применяет медианный фильтр."""

ksize = get\_kernel\_size\_from\_user("Размер ядра для medianBlur")

if ksize:

state.fixed\_canvas = cv2.medianBlur(state.canvas, ksize)

state.primitives = []

save\_state()

redraw\_canvas()

print(f"Применён medianBlur с ядром {ksize}")

def apply\_gaussian\_blur() -> None:

"""Применяет гауссов фильтр."""

ksize = get\_kernel\_size\_from\_user("Размер ядра для GaussianBlur")

if ksize:

state.fixed\_canvas = cv2.GaussianBlur(state.canvas, (ksize, ksize), 0)

state.primitives = []

save\_state()

redraw\_canvas()

print(f"Применён GaussianBlur с ядром {ksize}")

def apply\_erode() -> None:

"""Применяет морфологическое сжатие."""

ksize = get\_kernel\_size\_from\_user("Размер ядра для erode")

if ksize:

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (ksize, ksize))

state.fixed\_canvas = cv2.erode(state.canvas, kernel)

state.primitives = []

save\_state()

redraw\_canvas()

print(f"Применён erode с ядром {ksize}")

def apply\_dilate() -> None:

"""Применяет морфологическое расширение."""

ksize = get\_kernel\_size\_from\_user("Размер ядра для dilate")

if ksize:

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (ksize, ksize))

state.fixed\_canvas = cv2.dilate(state.canvas, kernel)

state.primitives = []

save\_state()

redraw\_canvas()

print(f"Применён dilate с ядром {ksize}")

def apply\_sobel() -> None:

"""Применяет оператор Собеля."""

gray = cv2.cvtColor(state.canvas, cv2.COLOR\_BGR2GRAY)

dx = cv2.Sobel(gray, cv2.CV\_64F, 1, 0, ksize=3)

dy = cv2.Sobel(gray, cv2.CV\_64F, 0, 1, ksize=3)

magnitude = cv2.magnitude(dx, dy)

magnitude = np.uint8(np.clip(magnitude, 0, 255))

state.fixed\_canvas = cv2.cvtColor(magnitude, cv2.COLOR\_GRAY2BGR)

state.primitives = []

save\_state()

redraw\_canvas()

print("Применён фильтр Sobel")

def apply\_laplacian() -> None:

"""Применяет оператор Лапласа."""

gray = cv2.cvtColor(state.canvas, cv2.COLOR\_BGR2GRAY)

laplacian = cv2.Laplacian(gray, cv2.CV\_64F)

abs\_laplacian = np.uint8(np.absolute(laplacian))

state.fixed\_canvas = cv2.cvtColor(abs\_laplacian, cv2.COLOR\_GRAY2BGR)

state.primitives = []

save\_state()

redraw\_canvas()

print("Применён фильтр Laplacian")

def apply\_canny() -> None:

"""Применяет детектор границ Canny."""

def on\_ok():

try:

low = int(entry\_low.get())

high = int(entry\_high.get())

nonlocal thresholds

thresholds = (low, high)

dlg.destroy()

except ValueError:

messagebox.showwarning("Ошибка", "Введите целые числа")

thresholds = None

dlg = tk.Toplevel()

dlg.title("Пороги Canny")

dlg.geometry("300x150")

dlg.transient()

dlg.grab\_set()

tk.Label(dlg, text="Нижний порог:").pack(pady=5)

entry\_low = tk.Entry(dlg)

entry\_low.pack(pady=2)

entry\_low.insert(0, "50")

tk.Label(dlg, text="Верхний порог:").pack(pady=5)

entry\_high = tk.Entry(dlg)

entry\_high.pack(pady=2)

entry\_high.insert(0, "150")

tk.Button(dlg, text="Применить", command=on\_ok).pack(pady=10)

dlg.wait\_window()

if thresholds is None:

return

gray = cv2.cvtColor(state.canvas, cv2.COLOR\_BGR2GRAY)

edges = cv2.Canny(gray, thresholds[0], thresholds[1])

state.fixed\_canvas = cv2.cvtColor(edges, cv2.COLOR\_GRAY2BGR)

state.primitives = []

save\_state()

redraw\_canvas()

print(f"Применён Canny с порогами {thresholds[0]}, {thresholds[1]}")

def apply\_equalize\_hist() -> None:

"""Применяет эквализацию гистограммы."""

if len(state.canvas.shape) == 3:

lab = cv2.cvtColor(state.canvas, cv2.COLOR\_BGR2LAB)

l, a, b = cv2.split(lab)

l\_eq = cv2.equalizeHist(l)

lab\_eq = cv2.merge((l\_eq, a, b))

state.fixed\_canvas = cv2.cvtColor(lab\_eq, cv2.COLOR\_LAB2BGR)

else:

state.fixed\_canvas = cv2.equalizeHist(state.canvas)

state.fixed\_canvas = cv2.cvtColor(state.canvas, cv2.COLOR\_GRAY2BGR)

state.primitives = []

save\_state()

redraw\_canvas()

print("Применена эквализация гистограммы")

def load\_image() -> None:

"""Загружает изображение из файла."""

file\_path = filedialog.askopenfilename(

title="Выберите изображение",

filetypes=[("Изображения", "\*.jpg \*.jpeg \*.png \*.bmp \*.tiff")]

)

if not file\_path:

return

img = cv2.imread(file\_path)

if img is None:

messagebox.showerror("Ошибка", "Не удалось загрузить изображение")

return

dim = None

(h, w) = img.shape[:2]

if h > 768:

r = 768 / float(h)

dim = (int(w \* r), 768)

else:

r = 1024 / float(w)

dim = (1024, int(h \* r))

img = cv2.resize(img, dim, interpolation = cv2.INTER\_AREA)

state.fixed\_canvas = img.copy()

state.primitives.clear()

state.history.clear()

state.redo\_stack.clear()

save\_state()

redraw\_canvas()

print(f"Загружено изображение: {file\_path}")

def create\_ui() -> tk.Tk:

"""Создаёт главное окно управления."""

root = tk.Tk()

root.title("Панель управления редактором")

root.geometry("300x1000")

root.resizable(False, False)

style = ttk.Style()

style.theme\_use('clam')

tk.Label(root, text="Инструменты:", font=("Arial", 12, "bold")).pack(pady=(10,5))

tools = [

("Линия", "line"),

("Прямоугольник", "rectangle"),

("Круг", "circle"),

("Ластик", "eraser"),

]

for text, tool in tools:

btn = ttk.Button(root, text=text, command=lambda t=tool: set\_tool(t))

btn.pack(pady=2, fill='x', padx=20)

ttk.Separator(root, orient='horizontal').pack(fill='x', pady=10)

tk.Label(root, text="Цвет:").pack()

global color\_btn

hex\_color = '#%02x%02x%02x' % (state.color[2], state.color[1], state.color[0])

color\_btn = tk.Button(root, text="Выбрать цвет", bg=hex\_color, command=choose\_color)

color\_btn.pack(pady=5, fill='x', padx=40)

ttk.Separator(root, orient='horizontal').pack(fill='x', pady=10)

tk.Label(root, text="Толщина линии:").pack()

thickness\_scale = ttk.Scale(root, from\_=1, to=20, orient='horizontal', command=update\_thickness)

thickness\_scale.set(state.thickness)

thickness\_scale.pack(pady=5, fill='x', padx=20)

ttk.Separator(root, orient='horizontal').pack(fill='x', pady=10)

tk.Label(root, text="Фильтры:", font=("Arial", 12, "bold")).pack(pady=(10,5))

filters = [

("Фильтр резкости", apply\_filter2d),

("Blur", apply\_blur),

("Median Blur", apply\_median\_blur),

("Gaussian Blur", apply\_gaussian\_blur),

("Erode", apply\_erode),

("Dilate", apply\_dilate),

("Sobel", apply\_sobel),

("Laplacian", apply\_laplacian),

("Canny", apply\_canny),

("Equalize Histogram", apply\_equalize\_hist),

]

for text, func in filters:

btn = ttk.Button(root, text=text, command=func)

btn.pack(pady=2, fill='x', padx=20)

ttk.Separator(root, orient='horizontal').pack(fill='x', pady=10)

tk.Label(root, text="Действия:", font=("Arial", 12, "bold")).pack(pady=(10,5))

actions = [

("Загрузить изображение", load\_image),

("Очистить", clear\_canvas),

("Сохранить", save\_canvas),

("Отменить", undo\_action),

("Повторить", redo\_action)

]

for text, cmd in actions:

btn = ttk.Button(root, text=text, command=cmd)

btn.pack(pady=3, fill='x', padx=40)

ttk.Separator(root, orient='horizontal').pack(fill='x', pady=10)

return root

def main() -> None:

"""Главная функция."""

global state

state.canvas = create\_canvas(1024, 768)

state.background = state.canvas.copy()

save\_state()

cv2.namedWindow("Canvas", cv2.WINDOW\_AUTOSIZE)

cv2.setMouseCallback("Canvas", mouse\_callback)

cv2.imshow("Canvas", state.canvas)

root = create\_ui()

print("="\*50)

print("РЕДАКТОР ПРИМИТИВОВ ЗАПУЩЕН")

print("="\*50)

root.mainloop()

cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

main()