Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ВИКОРИСТАННЯ КЛІТИННИХ АВТОМАТІВ ДЛЯ ОБРОБКИ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ ГРАФІЧНИХ КАРТ

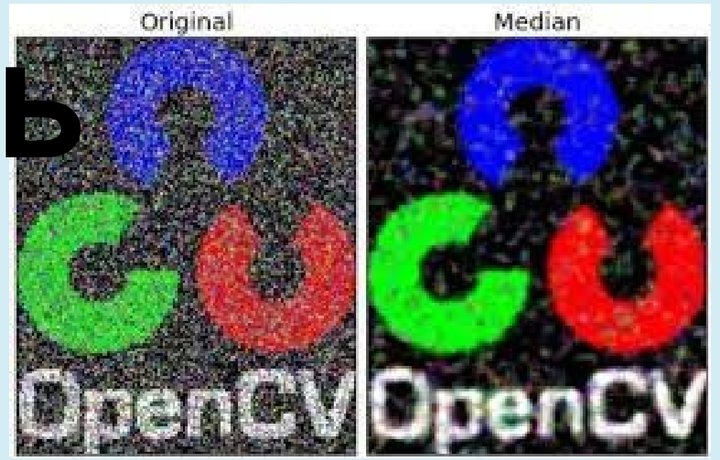
Мошковський Іван, ІПЗ-3
Науковий керівник - Жежерун Олександр Петрович
2025

Цілі дослідження

1. Дослідити модель клітинних автоматів, проаналізувати їх сумісність з графічними картами та дослідити принцип їх застосування в обробці зображень
2. Дослідити принципи роботи та будови графічних карт, визначити їх переваги у застосуванні з клітинними автоматами та як завдяки ним обробляти зображення
3. Розібратись у реалізації роботи з графічними картами та обробці зображень у мові програмування Python
4. Створити код, що виконуватиме розмиття зображення, використовуючи поєднання клітинних автоматів та графічних карт для максимальної ефективності

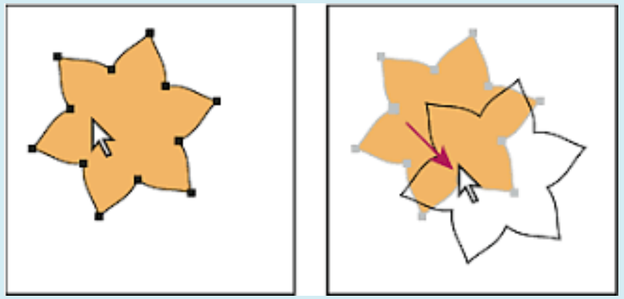
Обробка зображень

Обробка зображень —це розділ прикладної інформатики, що займається аналізом та перетворенням цифрових зображень з метою покращення їх якості, виявлення структур або підготовки до подальшої обробки. Основними задачами є фільтрація шуму, зміна яскравості й контрасту, сегментація зображень, виділення об'єктів та класифікація



Фільтрація зображення

Видалення шуму

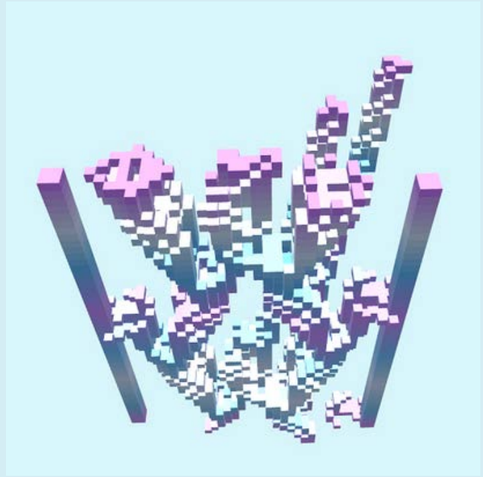


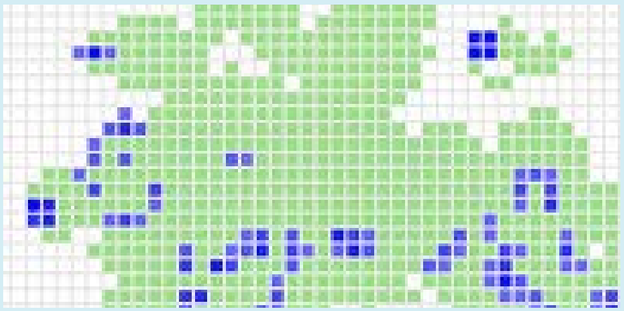
Виділення об’єкту

З/..

Клітинні автомати

Клітинні автомати (КА) — це дискретні моделі, що складається з клітинок, кожна з яких змінює свій стан паралельно на основі свого локального оточення за певними правилами





Приклади різніх інтерпретацій гри “Життя” Конвея - найвідомішого прикладу клітинних автоматів

Графічні карти

Графічні карти (або графічні процесори, GPU) — це спеціалізовані апаратні засоби, першоначально призначені для швидкого обчислення й відтворення зображень

Переваги використання клітинних автоматів з графічними картами в обробці зображень

- Висока паралельність
- Простота реалізації
- Природне масштабування
- Гнучкість моделей
- Висока швидкодія

Використання Python для обробки зображень з GPU та КА

```
import pyopencl as cl
```

Бібліотеки: Pillow, PyOpenCL, Tkinter(візуалізація), CuPy

```
import tkinter as tk
from tkinter import filedialog
```

```
from PIL import Image
```

7/ ...

Огляд застосунку Image blur

Структура коду

Застосунок складається з двох файлів:

image_blurrer.py

- Реалізація алгоритму розмиття
- Трансформація зображення у масив NumPy
- Реалізація через графічні карти завдяки OpenCL (підвантаження зображення, створення буферів, запуск ядра)

gui_app.py

- Створення користувацького інтерфейсу
- Збереження та виведення зображення

image_blurrer.py

```python
def _get_kernel_code(self):  1 usage
    return f"""
    __kernel void blur(__global uchar* input, __global uchar* output,
                       int width, int height, int channels) {{
        int x = get_global_id(0);
        int y = get_global_id(1);
        int k = {self.kernel_size};
        int half_k = k / 2;

        for (int c = 0; c < channels; ++c) {{
            int sum = 0;
            int count = 0;

            for (int dx = -half_k; dx <= half_k; ++dx) {{
                for (int dy = -half_k; dy <= half_k; ++dy) {{
                    int nx = x + dx;
                    int ny = y + dy;

                    if (nx >= 0 && ny >= 0 && nx < width && ny < height) {{
                        int index = (ny * width + nx) * channels + c;
                        sum += input[index];
                        count++;
                    }}
                }}
            }}

            int out_index = (y * width + x) * channels + c;
            output[out_index] = sum / count;
        }}
    }}
    """
```

```python
class ImageBlurrer:  3 usages
    def __init__(self, kernel_size=3):
        self.kernel_size = kernel_size
        self.ctx = cl.create_some_context()
        self.queue = cl.CommandQueue(self.ctx)

        self.program = cl.Program(self.ctx, self._get_kernel_code()).build()
```

```python
def blur_image(self, input_image_path, output_image_path):  2 usages
    # Завантаження та підготовка зображення
    image = Image.open(input_image_path).convert("RGB")
    img_array = np.array(image).astype(np.uint8)
    height, width, channels = img_array.shape

    # Буфери
    mf = cl.mem_flags
    input_buf = cl.Buffer(self.ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=img_array)
    output_buf = cl.Buffer(self.ctx, mf.WRITE_ONLY, img_array.nbytes)

    # Запуск ядра
    self.program.blur(
        self.queue,
        (width, height),
        None,
        input_buf,
        output_buf,
        np.int32(width),
        np.int32(height),
        np.int32(channels)
    )

    result = np.empty_like(img_array)
    cl.enqueue_copy(self.queue, result, output_buf)

    Image.fromarray(result).save(output_image_path)
```

10/...

gui_app.py

```python
class BlurApp:  1 usage
    def __init__(self, root):
        self.root = root
        self.root.title("GPU Blur with Cellular Automata")

        self.img_label = tk.Label(self.root, text="No image loaded")
        self.img_label.pack()

        self.btn_frame = tk.Frame(self.root)
        self.btn_frame.pack(pady=10)

        self.load_btn = tk.Button(self.btn_frame, text="Load Image", command=self.load_image)
        self.load_btn.grid(row=0, column=0, padx=5)

        self.blur_btn = tk.Button(self.btn_frame, text="Apply Blur", command=self.apply_blur, state=tk.DISABLED)
        self.blur_btn.grid(row=0, column=1, padx=5)

        self.image_path = None
```

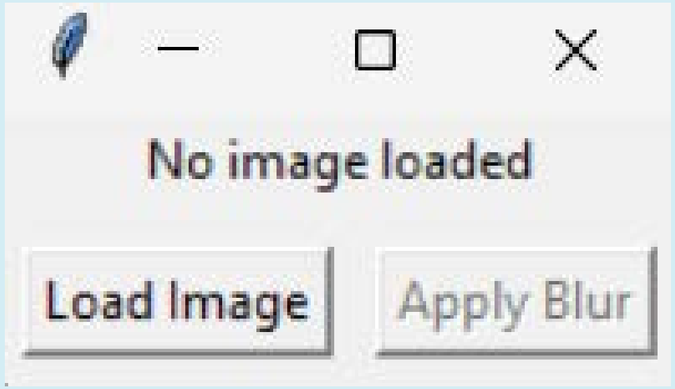
```python
def apply_blur(self):  1 usage
    output_path = "output.png"
    blurrer = ImageBlurrer(kernel_size=5)
    blurrer.blur_image(self.image_path, output_path)
    self.display_image(output_path)
```

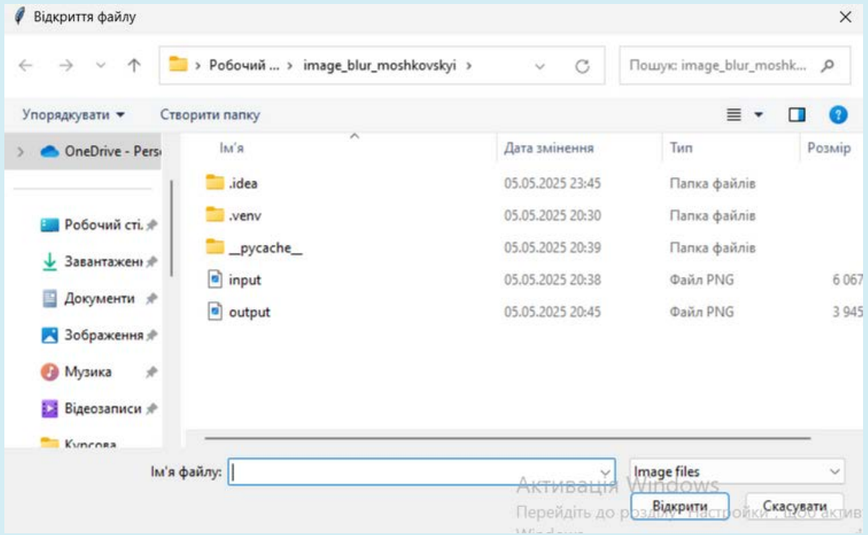
```python
def display_image(self, path):  2 usages
    img = Image.open(path).resize((400, 400))
    tk_img = ImageTk.PhotoImage(img)
    self.img_label.config(image=tk_img, text="")
    self.img_label.image = tk_img
```

```python
def load_image(self):  1 usage
    file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.png;*.jpg;*.jpeg")])
    if not file_path:
        return
    self.image_path = file_path
    self.display_image(file_path)
    self.blur_btn.config(state=tk.NORMAL)
```

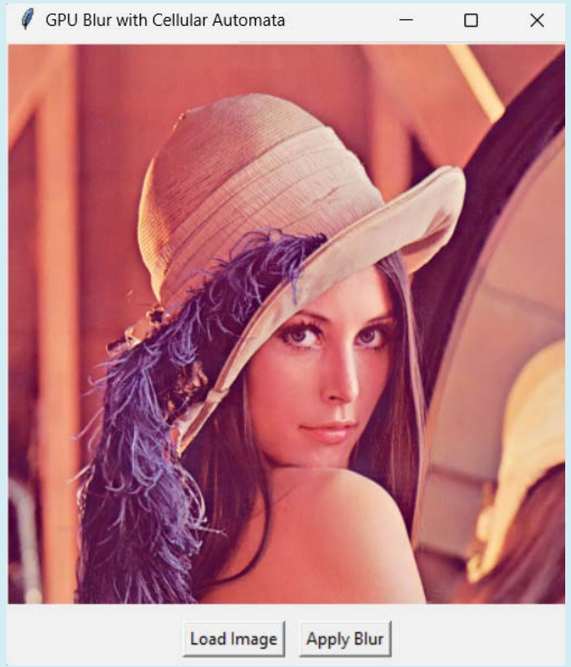
Робота застосунку

Після запуску користувач отримує вікно, де має завантажити зображення натиснувши на кнопку “Load image”





Обравши файл, буде виведене зображення, на якому після натискання на кнопку “Apply Blur” буде виконано розмиття, і результуюче зображення буде виведено на екран та збережене у папці проєкту



Результат