Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ВИКОРИСТАННЯ КЛІТИННИХ АВТОМАТІВ ДЛЯ ОБРОБКИ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ ГРАФІЧНИХ КАРТ

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення”

Керівник курсової роботи
к.т.н., доц.

_____
(прізвище та ініціали)

_____
(підпис)
“____” _____ 2025 р.

Виконав студент ІПЗ-3

Мошковський Іван Олегович
“____” _____ 2025 р.

Київ 2025

Календарний план-графік підготовки кваліфікаційної роботи на тему:

Тема: Використання клітинних автоматів для обробки зображень за допомогою графічних карт

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	07.11.2024	
2.	Огляд технічної літератури за темою роботи.	15.11.2024	
3.	Виконати аналіз сучасних методів обробки зображень	25.11.2024	
3.	Розробка алгоритму розмиття зображення	25.12.2024	
4.	Програмування розробленого алгоритму	15.01.2025	
5.	Застосування розробленого алгоритму до тестового зображення	15.02.2025	
6.	Виконання порівняльного аналізу результатів прогнозування, отриманих за допомогою розробленого алгоритму на основі нейромережі, та результатів, отриманих за допомогою регресійних моделей.	30.03.2025	
7.	Написання пояснювальної роботи.	20.04.2025	
8.	Створення слайдів для доповіді та написання доповіді.	22.04.2025	
8.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист магістерської роботи.	25.04.2025	
10.	Корегування роботи за результатами попереднього захисту.	1.05.2025	
11.	Остаточне оформлення пояснювальної роботи та слайдів.	3.05.2025	
12.	Захист курсової роботи (проекту)	12.05.2025	

Студент **Мошковський Іван Олегович**

Керівник **Жежерун Олександр Петрович**

“_____”_____

Зміст

Вступ

Зображення — це представлення візуальної інформації у вигляді двовимірної дискретної сітки пікселів, кожен з яких має певне значення, що відповідає кольору або яскравості у відповідній точці. **Обробка зображень** у свою чергу є сукупністю певних алгоритмів та методів, що застосовуються до зображення з метою його покращення, реконструкції, виявлення або виділення його частин, а також подальшого його аналізу. Основною задачею обробки, є перетворення зображення у більш зручну для сприйняття людиною форму або його налаштування під подальшу комп'ютерну обробку та аналіз.

У сучасному світі інформаційних технологій обробка зображень посідає важливе місце, охоплюючи такі сфери як: медицина, топографія та навігація, тренування штучного інтелекту, обробка космічних зображень із супутників, військова справа(особливо робота з дронами та навігаційними приладами) та багато інших.

Одним з провідних підходів для роботи в сфері обробки зображень є використання **клітинних автоматів**, які являють собою дискретну систему з решітки клітин, чим природньо підходять до роботи із зображеннями (адже всі пікселі у своїй сукупності є матрицею, по суті, аналогом клітинної решітки). Використання клітинних автоматів дозволяє виконувати обробку зображення через фільтрування шумів, виділення контурів, покращення його якості та має потенціал у сфері моделювання (анімації) різних ефектів та явищ, таких як: вогонь, дим, дифузія чи розмиття.

Однією з найсильніших переваг клітинних автоматів є їх потужна здатність до паралелізації процесів, що робить їх використання разом з **графічними картами** ще більш ефективними, адже графічні карти містять тисячі ядер, що можуть паралельно виконувати різноманітні обчислення та алгоритми.

Ця робота складається з трьох розділів:

Перший розділ присвячено вивченню принципу роботи та будови клітинних автоматів та аналізу того, як їх можна використати при роботі з обробкою зображень. Наведено приклади, що показують дію клітинних автоматів як у своєму загалі, так і конкретно при обробці цифрових зображень. Наданий короткий опис процесів, що відбуваються.

У другому розділі йдеться про графічні карти. Розглядається їх будова, та приділяється особлива увага аналізу їх можливостей у паралелізації та тому, які переваги надає їх використання при роботі з клітинними автоматами.

У третьому розділі буде описаний алгоритм на мові програмування Python, що завдяки користувацькому інтерфейсу дозволить завантажити зображення та виконати на ньому розмиття, отримавши результат у окремому файлі.

Постановка задачі

1) Вивчити та проаналізувати структуру та принципи роботи клітинних автоматів та графічних карт. Зробити стислий огляд їх використання
2) Визначити їх переваги та недоліки для роботи з зображеннями, та зрозуміти як найефективніше досягати того чи іншого результату, обробляючи зображення
3) Створити алгоритм, що розмиватиме зображення використавши клітинні автомати та технологію OpenCL

Розділ 1. Клітинні автомати

1.1 Поняття клітинних автоматів та принцип їх роботи

Клітинні автомати – це дискретні динамічни системи, поведінка яких повністю визначається в термінах локальних залежностей[1]. Вони визначаються набором клітинок, що утворюють сітку та мають визначені правила переходу із одного стану в інший, які є **локальними**, тобто залежать тільки від найближчого оточення клітинки(інакше кажучи сусідів), а не усієї системи. На ітерації кожна з клітинок, маючи обмежену кількість станів (наприклад, “живий/мертвий”), взаємодіє зі своїми сусідами за певними правилами, в результаті чого може або змінити свій стан або залишитись без змін. Таким чином після кожної ітерації створюється нове покоління клітинок, причому всі вони взаємодіють паралельно.

1.2 Гра “Життя”

Розглянемо як працюють клітинні автомати на прикладі найвідомішого з них – Гри “Життя”. Цей автомат винайшов англійський математик Джон Конвей 1970 року і по своїй суті він примітивно симулює життя. Клітина в цьому автоматі має два стани: вона може бути або живою або мертвою, а за сусідів беруться клітинки в квадраті 3x3 навколо конкретної клітини. Встановлені такі правила переходу:

1. Якщо жива клітинка має два або три сусіди – вона залишається живою
2. Якщо жива клітинка має менше двох сусідів – то вона помирає від “самотності”
3. Якщо в живої клітинки більше трьох сусідів – вона помирає від “перенаселення”
4. Якщо в мертвої клітинки рівно три живих сусідів – вона оживає

Гравець не бере безпосередньої участі у грі, а лише розставляє початкову конфігурацію із “живих” клітин, а ітерації з взаємодіями клітин відбуваються вже без його участі. В результаті маючи таку просту конфігурацію можна отримувати найрізноманітніші результати. Цю гру активно використовують в математиці для різних цілей:

- Ця гра є Тюрінг-повною, а отже в її межах можна реалізувати будь-який алгоритм або обчислення, що сильно допомагає, наприклад, вивчати які проблеми можна вирішити, а які – ні
- “Життя” – це детермінована динамічна система, тобто маючи фіксовані правила вона може створювати непердбачувані результати. Цим вона дозволяє вивчати хаотичні системи та емерджентну складність
- Вивчається, як всі можливі розташування клітин впливають на систему завдяки комбінаторним методам
- Можна займатися оптимізацією алгоритмів, через пошук оптимальних комбінацій і визначення того наскільки важко (обчислювально) знайти певну поведінку у грі

1.3 Класифікація клітинних автоматів

За різними ознаками: характером правил переходу, кількістю вимірів, складністю поведінки, типом сусідства; клітинні автомати класифікують на велику кількість видів. Завдяки цій кваліфікації легше зрозуміти можливості конкретного типу автомата, та підібрати його відповідно до поставленої задачі.

- **За характером правил переходу**
  1) **Детерміновані автомати**

     Для набору сусідів існує однозначний результат. Інакше кажучи, привила є фіксованими (як у грі “Життя)”.

2) **Стохастичні (випадкові) автомати**

   Результат переходу залежить не тільки від сусідів клітинки, а й від певної ймовірності (наприклад клітинка помирає з 55% ймовірністю). Це використовується в моделюванні росту тканин, шуму і т.д.

- **За кількістю вимірів**

1) **Одновимірні автомати**

   Поле представлене одним рядом клітинок. Використовується здебільшого для теоретичних досліджень. Прикладом є автомат Вольфрама.

2) **Двовимірні клітинні автомати**

   Двовимірне поле (у вигляді сітки). Цей вид є найпоширенішим й найефективнішим. Прикладом є гра “Життя”.

3) **Багатовимірні**

   Мають від трьох і більше вимірів. Такі автомати використовуються доволі рідко, адже є дуже складними для коректної реалізації, проте саме завдяки своїй складності їх використовують для комплексного та об’ємного моделювання (здебільшого у фізиці).

- **За складністю поведінки (класифікація Вольфрама)**

Дослідник Стівен Вольфрам створив класифікацію, що базується на поведінці клітинних автоматів при довготривалому запуску:

1) **Клас 1 –** Система сходиться до статичного стану, тобто всі клітинки стають однаковими і перестають змінювати свої стани.
2) **Клас 2 –** Формуються стабільні (незмінні) або періодичні (з певною кількісно-ітераційною періодичністю) структури.
3) **Клас 3 –** Поведінка непередбачувана (хаотична).
4) **Клас 4 –** Поведінка є складною, та важко передбачуваною.

Утворюються локальні структури, що взаємодіють між собою. Цей клас має найбільший потенціал, адже своєю поведінкою нагадує інтелектуальну систему.

- **За типом сусідства (для двовимірних систем)**
  1) **Сусідство фон Неймана**
     4 сусіди (згори, знизу, зліва та справа).
  2) **Сусідство Мура**
     8 сусідів (клітинки навколо). Саме такий тип використовується в грі “Життя”.
  3) **Розширене сусідство**
     Охоплюються клітинки віддалені від головної на кілька кроків за різними принципами.

Також існує два більш спеціалізованих типи клітинних автоматів – тоталістичні та зовнішні тоталістичні.

Тоталістичні клітинні автомати – це клітинні автомати, для яких новий стан визначається або сумою, або загальною кількістю станів її сусідів, при цьому сама клітина може бути включена у це значення. Основна ідея цього автомату в тому, що для нього не важливо які саме клітини були активні навколо певної, важливою є їхня кількість. До цього типу належить гра “Життя” Конвея.

Зовнішні тоталістичні клітинні автомати – це підвид тоталістичних клітинних автоматів, де новий стан клітини залежить від стану самої клітини та суми станів її сусідів окремо, тобто обробляються вони незалежно.

Ці види автоматів особливо зручні для моделювання біологічних процесів (росту, дифузії) і також використовуються для обробки зображень, де враховується кількість активних пікселів навколо.

1.4 Теоретичні основи використання клітинних автоматів в обробці зображень

Як вже було зазначено у вступній частині, клітинні автомати своєю будовою (сітка з клітинок) дуже схожі з зображенням (матриця з пікселів), але ще одним важливим пунктом в темі зручності використання КА при обробці зображень є те, що клітинні автомати оновлюють стан кожної клітинки на основі **локального оточення** (інкаше кажучи – сусідів). Це зручно, адже багато алгоритмів для обробки зображень (наприклад, фільтрація, контурне виділення) ґрунтуються на зміні числових параметрів пікселів, відштовхуючись від їхнього **локального контексту.**

На прикладі фільтрації математично зобразимо відповідність між зображеннями та клітинними автоматами:

Нехай P(x, y) – значення пікселя з координатами (x, y)

Тоді операцію фільтрації можна спрощено зобразити так:

$$P1(x, y) = f(S(x, y))$$

де S(x, y) – локальне оточення пікселя (сусіди), а f – метод (правило), що задає нове значення

Для клітинних автоматів вийде аналогічно:

$$C1(x, y) = F(N(x, y))$$

де N(x, y) – це множина сусідів клітинки (x, y), а F – правило переходу

З цього випливає, що правила клітинного автомату є ідентичними до правил фільтрації зображення

Визначимо переваги використання клітинних автоматів при обробці зображень:

1) **Паралелізація**: усі клітинки змінюють свій стан одночасно, що є дуже ефективним, плюс, використавши графічні карти, цей процес можна пришвидшити в кілька разів.

2) **Гнучкість та легка зміна правил**: клітинні автомати можна легко підлаштувати під будь-яку задачу змінивши правила переходу.
3) **Локальність**: робота виключно із сусідами значно спрощує створення потрібного алгоритму та покращує якість виконаних завдань.
4) **Стійкість до шуму**: більшість тоталістичних КА мають стабільну поведінку навіть за ненайкращих вхідних даних.

Тепер варто зрозуміти які саме напрямки застосування клітинних Автоматів є в галузі обробки зображень:

1) **Фільтрація шуму**: клітинні автомати можуть реалізовувати **медіанні фільтри**, тобто враховуючи локальну структуру(сусідів) КА підлаштовує клітинки до певного значення, тим самим прибираючи одиночні “викиди”.
2) **Контурування**: визначивши контур, як зміну кольорової інтенсивності між сусідами, в локальних правилах, можна автоматично виділяти контури всіх об’єктів.
3) **Сегментація**: завдяки КА, можна, задавши в правилах, наприклад, прийняття пікселя-сусіда, що по яскравості відрізняється від центрального не більше як на 10 одиниць, отримати виділений від фону об’єкт. Цей метод використовується в медицині (для виділення пухлин або органів на знімках МРТ), аналізі супутникових знімків (сегментація лісних ділянок, води, тощо) або розпізнавання будь-яких інших об’єктів.
4) **Морфологічні операції**: наприклад, ерозія (зменшення об’єктів шляхом “з’їдання” пікселів по краям фігур), дилатація (збільшення об’єктів, процес протилежний до ерозії), закриття (процес виконання дилатації, а потім ерозії, щоб заповнити малі отвори у великих структурах) і т.д.

1.5 Висновок

В цьому розділі було розглянуто клітинні автомати, принцип їх роботи, та проаналізовано, як можна їх використати для обробки зображень. Розглянувши далі графічні карти і поєднавши всі знання можна буде максимально ефективно виконувати обробку зображень завдяки різноманітним алгоритмам.

Розділ 2. Графічні карти

2.1 Графічні карти: визначення, структура та принцип роботи

За своїм визначенням **графічні карти** (GPU, з англ. Graphics Processing Unit) – це спеціалізовані електронні пристрої, створені для швидкого та ефективного виконання різноманітних графічних операцій. Однак стрімке зростання продуктивності графічного обладнання в поєднанні з нещодавніми покращеннями його програмованості зробили графічне обладнання переконливою платформою для виконання складних обчислювальних завдань у найрізноманітніших галузях застосування (наукових обчисленнях, машинному навчанні, обробці зображень)[2].

Основною перевагою GPU у порівнянні з 4-16 ядерним CPU (центральним процесором) є їхня масова **паралельна архітектура**. Графічні карти складаються з сотень або тисяч високоефективних обчислювальних елементів (ядер), що дозволяє одночасно обробляти великі об’єми даних, виконуючи однакову операцію над великою кількістю елементів (цей принцип має назву SIMD – single instruction, multiple data), наприклад роботу з пікселями зображення. Також CPU мають складну ієрархію пам’яті, пов’язану з багатопоточністю, розбиттям на блоки та сітки, що при грамотному програмуванні дозволяє ефективно розділити дані, мінімізувавши затримки та покращивши швидкодію програм та алгоритмів.

2.2 Загальне призначення графічних процесорів у науці

Як вже було зазначено, нині GPU використовується не лише для рендерингу графіки, але й для виконання різних за складністю обчислювальних задач загального призначення. Ця концепція отримала назву GPGPU (англ. General-Purpose computing on Graphics Processing Units). Це стало можливим завдяки технологіям **CUDA** (з англ. Compute Unified Device Architecture, представлена компанією Nvidia 15 лютого 2007 року) та OpenCL (з англ. Open Computing Language, винайдена компанією Apple Inc. І показана світові 16 червня 2008 року).

CUDA є мінімальним розширенням мов програмування C та C++, де програміст пише послідовну програму, яка викликає паралельні ядра, що можуть бути простими функціями або повноцінними програмами [3]. Запуск цієї програми на графічній карті дозволяє ефективно використовувати тисячі потоків для одночасного виконання.

Аналогічно і OpenCL. Цей фреймворк створений для написання програм пов'язаних з паралельним обчисленням. Він забезпечує паралельність на рівні інструкцій та на рівні даних і є реалізацією техніки GPGPU. Мета OpenCL полягає в тому, щоб доповнити OpenGL і OpenAL, які є відкритими галузевими стандартами для тривимірної комп'ютерної графіки і звуку, користуючись можливостями GPU.

Основними перевагами GPGPU є:

1) Висока пропускна здатність пам'яті (до кількох терабайт за секунду);

2) Масова паралельність (до близько 10 000 потоків одночасно);

3) Висока енергоефективність при великій продуктивності;

4) Можливість обробки великих масивів структурованих даних (матриці, зображення, 3-D сцени).

GPGPU використовується в різноманітних сферах: наукових симуляціях (моделювання клімату, астрономічних процесів, фізики частинок), машинному навчанні та нейронних мережах (тренування великих моделей типу GPT, ResNet, BETR), обробці зображень та відео (фільтрація, трансформація, розпізнавання об'єктів у відео в реальному часі), біоінформатиці (складання геномів, моделювання білкових структур), комп'ютерних іграх та віртуальній реальності (фізичні симуляції, розрахунок поведінки NPC, рендеринг сцен з фотореалізмом).

2.3 Графічні карти в обробці зображень

Обробка зображень є однією з найпоширеніших сфер використання графічних карт, адже паралельна природа графічного процесора ідеально узгоджується з її основною суттю – виконанням однакових операцій над великою кількістю пікселів.

Більшість алгоритмів обробки зображень можуть бути подані у вигляді **локальних операцій над пікселем,** що залежать від нього та його “сусідів” (наприклад згладжування, контурування). Тут дуже допомагає багатопоточність. На GPU можна реалізувати модель, де кожний піксель обробляється окремим потоком, масштабуючи її на десятки мільйонів точок.

Водночас, у фільтрах, що використовують **сусідні пікселі** (наприклад операція дилатації – “нарощення” області об’єктів), можна підвищити швидкодію шляхом розміщення даних в спільній пам’яті блоку потоків (shared memory). Завдяки цьому, кожен потік може звертатися до даних, зчитаних іншими потоками, що, в порівнянні зі зчитуванням данних з глобальної пам’яті, забезпечує пришвидшення виконання процесу більш як у 10 разів.

Для задач з інтерполяцією (обчислення проміжних значень між відомими точками), фільтрацією чи вибіркою фрагментів зображень можна скористатись **текстурною пам’яттю**, оптимізованою безпосередньо для читання даних зображення. Вона надає низку переваг: кешування доступів, апаратну підтримку інтерполяції (як лінійної так і бікубічної), можливість до повторення та відзеркалення текстур. Цей підхід активно використовується при роботі з масштабуванням зображень, зміні перспективи або обертанні, фільтрації зі зміщенням координат.

При оброці зображень на GPU використовуються **різні типи даних**:

1) uchar – для одноканальних зображень (градації сірого)

2) uchar4 – для кольорових зображень (RGBA)

3) float або float4 – для обробки HDR та математичних фільтрів із плаваючою точністю

Важливим є те, що графічні карти дозволяють працювати з векторними типами даних так само ефективно, як і зі скалярними. Завдяки цьому можна одночасно обробляти різні кольорові канали.

Для прикладу покроково розглянемо, як можна ефективно реалізувати **розмиття по Гаусу** (фільтр згладжування зображення, що використовує гаусівський розподіл для зважування пікселів). Для цього алгоритм фільтрації побудуємо за принципом **розгортки згортки (convolution)**:

1) Кожен піксель зчитує дані своїх сусідів у певному радіусі (наприклад, у квадраті 5x5 пікселів);

2) Далі отримані значення сумуються згідно з гаусівськими коефіцієнтами;

3) Результат записується у вихідне зображення.

Завдяки GPU цей алгоритм можна виконати у реальному часі для зображень 4K і більше без надмірних навантажень на систему.

2.4 GPU – реалізація клітинних автоматів для обробки зображень

Завдяки своїх будові у вигляді сітки з кітинок, клітинні автомати виступають **гнучкою моделлю локальної обробки пікселів** зображення, що легко поєднується з паралельною моделлю обчислення графічних карт.

Зображення по своїй суті можна розглядати як двовимірну сітку, де кожен піксель відповідає одній клітині автомату. GPU дозволяють працювати за принципом “один піксель = один потік”, повністю відповідаючи моделі клітинних автоматів. Таким чином можна оновлювати всі клітинки, паралельно використовуючи спільні правила переходу.

При реалізації клітинного автомату на графічній карті зазвичай використовується такий порядок дій:

1) Визначення координат потоку;

2) Зчитування станів сусідів (з глобальної пам’яті або пам’яті сусідів за принципом shared memory);

3) Обчислення нового стану за правилом переходу;

4) Запис результату, наприклад, у новий масив.

Однією з обов’язкових умов для правильного оновлення клітинних автоматів при роботі з графічними картами є повна ізоляція старих станів від нових. Щоб цього досягти використовують два масиви і після кожної ітерації масиви міняються місцями. Цей підхід називається **подвійною буферизацією** (double buffering).

Останнім, проте не менш важливим моментом є врахування граничних умов та сусідства клітин. Для клітинних автоматів існує дві типові топології – матрична сітка з прямокутним (сусідство фон Неймана) або квадратним сусідством (сусідсітво Мура). Важливо обережно поводитися з граничними клітинами, адже вихід за межі пам’яті може викликати помилки. Для цього використовують 3 основні підходи:

1) **Обмеження краю (clamping)**. Якщо координата сусіда виходить за межі зображення, вона автоматично стискається до найближчого граничного значення. Простіше кажучи, якщо звернення виходить за межу, береться крайній піксель. Наприклад:

x1 = min(max(x, 0), width - 1);

y1 = min(max(y, 0), height - 1);

Якщо, будучи на нульовому пікселі (0;0), звернутись до (-1;-1), замість неіснуючого пікселя ми отримаємо (0;0).

2) **Дзеркальне відображення (mirroring)**. Значення за межами зображення береться як дзеркальне відображення внутрішнього. Завдяки цьому обчислення по краях залишаються симетричними. Приклад:

x1 = abs(x);

if (x1 >= width) x1 = 2 * width – x1 - 1;

y1 = abs(y);

if (y1 >= height) y1 = 2 * height – y1 - 1;

Тут координата 1 відображатиметься як 0, координата width – як 1 і width + 1 – як width-2.

3) **Тороїдальна сітка (wrap-around)**. Сітка розглядається згорнутою в кільце і по горизонталі, і по вертикалі. При виході координати за межі, вона “обмотується” (wrap-around) навколо і переходить на протилежний бік. Розглянемо:

x1 = (x + width) % width;

y1 = (y + height) % height;

Звернувшись до пікселя (-1;0) ми отримаємо (width-1;0).

Звернувшись до (width;height) отрмаємо (0;0).

2.5 Висновок

Графічні карти – це потужні електронні пристрої, що підходять для обчислень різних видів. Завдяки своїй здатності до масової паралельної обробки, високій пропускній здатності та оптимізації для великих обсягів однотипних обчислень вони є дуже ефективними при обробці зображень, особливо у поєднанні з технологією клітинних автоматів, що своєю будовою ідеально поєднуються з їх роботою, забезпечуючи максимальну продуктивність та мінімально можливі витрати часу. Наступний розділ поєднає всі досліджені структури та процеси, використавши які буде створено застосунок для розмиття зображення.

Розділ 3. Розмиття зображення на графічній карті завдяки клітинним автоматам

Тепер, завдяки знанням, отриманим під час написання двох попередніх розділів, буде створено алгоритм на мові програмування Python з використанням OpenCL, що прийматиме на вхід зображення, виконуватиме на ньому розмиття та повертатиме результат користувачу.

Застосунок складається з двох клас-файлів:

1) **gui_app.py** – завдяки використанню бібліотеки Tkinter забезпечує простий GUI, що містить дві кнопки: “Load Image”(вибрати зображення з диску) та “Apply Blur” (запуск обробки через графічну карту). Після завантаження зображення воно відображається на екрані користувача, а після виконання обробки показується відредагована картинка.

2) **image_blurrer.py** – містить клас ImageBlurrer, що завантажує зображення через PIL.Image, завдяки методу _get_kernel_code(self) генерується та повертається текст ядра OpenCL, що виконуватиметься на GPU, і в ньому безпосередньо реалізований алгоритм розмиття за принципом клітинних автоматів (для вікна обраного розміру, наприклад 5x5,берчуи центральний піксель, обраховується його нове значення, як середнє значення всіх його сусідів). Метод blur_image бере вхідне зображення, конвертує його в RGB та перетворює його в масив NumPy для подальшої роботи.

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)

Далі завдяки наведеному вище фрагменті коду, визначаються прапорці для роботи з пам’яттю OpenCL та створюються буфери для вхідних та вихідних даних.

self.program.blur виконує ядро OpenCL, передаючи буфери на GPU для подальшої обробки зображення, після чого результуюче зображення у вигляді масиву записується у порожній масив, копіюється в пам’ять центрального процесору та перетворюється у .png формат.

В додатку А наведений повний код застосунку.

3.1 Висновок

Створений застосунок використовує різні бібліотеки мови Python завдяки яким, використовуючи клітинні автомати, виконує розмиття наданого користувачем зображення і робить все це безпосередньо на графічній карті завдяки технології OpenCL.

Висновок

В цій курсовій роботі було досліджено клітинні автомати, графічні карти, принципи їх роботи, будови та те як їх ефективно використовувати для обробки зображень.

Перший розділ був повністю присвячений клітинним автоматам. Було вивчено як вони працюють, де і як використовуються, та проаналізовано як ефективно їх застосувати обробляючи зображення.

У другому розділі йшлося про графічні карти, було вивчено як вони працюють, проаналізовано їх сильні сторони та особлива увага була присвячена аналізу їх роботи у тандемі з клітинними автоматами. Було доведено ефективність цього поєднання як у загалі так і безпосередньо у сфері обробки зображень.

У третьому розділі був наведений опис застосунку на Пайтоні, що використовуючи паралелізацію, властиву як клітинним автоматам так і графічним картам, завдяки графічному інтерфейсу, використанню алгоритму OpenCL для роботи з графічними картами та виконання алгоритму клітинних автоматів дозволяє користувачу завантажити власне зображення та виконати на ньому розмиття, отримавши результат.

Отримані дослідження дозволяють детально розібратися у клітинних автоматах і графіних картах. Незважаючи на те, що сферою, для якої проводився аналіз є обробка зображень, отриманий матеріал дозволить, заглибившись у тему працювати у найрізноманітніших сферах таких як: топографія (робота з картами, виченням та фіксуванням ландшафту), навчання штучного інтелекту (через аналіз зображень та роботу з ними), військова справа (робота дронів, та локаційних пристроїв для огляду місцевосці та наведення), графічний дизайн та анімація, комп’ютерні ігри, дослідження ефективності алгоритмів та обчислювальна математика (завдяки клітинним автоматам), фізика (виконання складних моделювань та аналіз різних процесів та явищ), біологія та медицина (робота з рентгенівськими

зображеннями, їх аналіз та різни види відображення). У наші часи досліджена тема активно використовується, а можливості клітинних автоматів досі вивчаються та допомагають і різних наукових дослідженнях.

Список використаних джерел

1. Машини клітинних автоматів, Т. Тофоллі, Н. Марголус (переклад з англійської) [Фізичний ресурс]
2. A Survey of General-Purpose Computation on Graphics Hardware, Computer Graphics Forum, 2007 Owens J. D., Houston M., Luebke D [Фізичний ресурс]
3. Scalable Parallel Programming with *CUDA*, ACM Queue, 2008 Nickolls J., Buck I., Garland M., Skadron K [Фізичний ресурс]
4. Python Documentation [Електронний ресурс] https://docs.python.org/uk/3/
5. Python OpenCL [Електроннийй ресурс] https://pypi.org/project/pyopencl/
6. Еволюція двовимірних клітинних автоматів [Електронний ресурс] https://science.lpnu.ua/uk/ujit/vsi-vypusky/vypusk-3-tom-1/evolyuciya-dvovymirnyh-klitynnyh-avtomativ-novi-formy-podannya
7. What is a GPU [Електронний ресурс] https://www.intel.com/content/www/us/en/products/docs/processors/what-is-a-gpu.html
8. A Systematic Literature Review on Graphics Processing Unit Accelerated Realm of High-Performance Computing [Електронний ресурс] https://www.researchgate.net/publication/379951830_A_Systematic_Literature_Review_on_Graphics_Processing_Unit_Accelerated_Realm_of_High-Performance_Computing
9. Методи обробки зображень [Електронний ресурс] https://web.posibnyky.vntu.edu.ua/fksa/2kvetnyj_komp'yuterne_modelyuvannya_system_procesiv/t2/2..htm
10. Image processing in Python [Електронний ресурс] https://www.geeksforgeeks.org/image-processing-in-python/

11. Image Processing with Python: Blurring Images [Електронний ресурс] https://datacarpentry.github.io/image-processing/06-blurring
12. Tkinter – Python interface to Tcl/Tk [Електронний ресурс] https://docs.python.org/uk/3.13/library/tkinter.html
13. Implementing an interface in Python [Електронний ресурс] https://realpython.com/python-interface/

Додаток А. Код застосунку image_blur_moshkovskyi

Код класу image_blurrer.py

```
import numpy as np
import pyopencl as cl
from PIL import Image

class ImageBlurrer:
    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()

    def _get_kernel_code(self):
        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;
    }}
}}
"""

    def blur_image(self, input_image_path, output_image_path):
        # 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)


if __name__ == "__main__":
    blurrer = ImageBlurrer(kernel_size=5)  # Можна вибрати 3, 5, 7 тощо
    blurrer.blur_image("input.png", "output.png")
    print("Розмиття завершено. Збережено у output.png")
```

Код класу gui_app.py

```
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
import os
from image_blurrer import ImageBlurrer

class BlurApp:
```

```
    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

    def load_image(self):
        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)

    def apply_blur(self):
```

```
        output_path = "output.png"
        blurrer = ImageBlurrer(kernel_size=5)
        blurrer.blur_image(self.image_path, output_path)
        self.display_image(output_path)

    def display_image(self, path):
        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

if __name__ == "__main__":
    root = tk.Tk()
    app = BlurApp(root)
    root.mainloop()
```