НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено:
В.о. завідувача кафедри
_____ **Микола ГРАЙВОРОНСЬКИЙ**
“___”_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною программою «Математичні методи моделювання, розпізнавання образів та безпеки даних»
спеціальності: 113 «Прикладна математика»

на тему: Відстеження періодичного руху об'єктів на статичному фоні на прикладі відеоігор.

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФІ-71
(шифр групи)

Павлюк Андрій Дмитрович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к. е. н. Ткач Володимир Миколайович _____ _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Консультант м. н. с. Кригін Валерій Михайлович _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Рецензент к. т. н., с. н. с. Кийко Володимир Михайлович _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім’я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 113 «Прикладна математика»
Освітньо-професійна програма «Математичні методи моделювання, розпізнавання образів та безпеки даних»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Микола ГРАЙВОРОНСЬКИЙ
(підпис)
«___»_____2021 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Павлюк Андрій Дмитрович
(прізвище, ім'я, по батькові)

1. Тема роботи: Відстеження періодичного руху об'єктів на статичному фоні на прикладі відеоігор,
керівник роботи: доцент, к. е. н. Ткач Володимир Миколайович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

консультант: м. н. с. Кригін Валерій Михайлович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « » _____2021 р. №

2. Термін подання здобувачем вищої освіти роботи 07 червня 2021 р.

3. Вихідні дані до роботи: існуючі алгоритми пошуку оптичного потоку, алгоритм імітації відпалу, методи пошуку максимального потоку на графі, відеозапис ігрового процесу зі статичним фоном, інтегроване середовище розробки для мови програмування Python, JIT компілятор Numba.

4. Зміст роботи: Огляд літератури та існуючих методів пошуку руху та оптичного потоку, виокремлення та вирівнювання об'єктів, що рухаються на відео, за допомогою задач max-flow та min-cut, реалізація алгоритму пошуку оптичного потоку на основі алгоритму імітації відпалу із розподілом Ґіббса.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація (12 слайдів)

6. Дата видачі завдання: 30.09.2020

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Отримання завдання	30.09.2020	
2	Огляд літератури та існуючих методів	01.10.2020 – 25.01.2021	
3	Пошук руху з використанням max-flow	27.01.2021 – 05.03.2021	
4	Огляд варіантів пошуку оптичного потоку	15.03.2021 – 16.04.2021	
5	Опис та реалізація алгоритму пошуку оптичного потоку на основі алгоритму імітації відпал з розподілом Ґіббса	16.04.2021 – 29.04.2021	
6	Написання статті до наукової конференції, її захист та публікація	29.04.2021 – 13.04.2021	
7	Оформлення результатів дипломної роботи	13.04.2021 – 01.06.2021	
8	Отримання допуску до захисту	03.06.2021	
9	Подання на рецензію до ММНЦ ІТіС НАН і МОН України	03.06.2021	
10	Захист дипломної роботи	14.06.2021	

Здобувач вищої освіти _____ Андрій ПАВЛЮК
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ Володимир ТКАЧ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Кваліфікаційна робота містить: 38 сторінок, 6 рисунків і 16 джерел.

Метою роботи є розробка методу вилучення зображень об’єктів, що рухаються на відео зі статичним фоном. Головною складовою цього методу є пошук оптичного потоку. В ході роботи проаналізовано існуючі методи пошуку оптичного потоку, реалізовано алгоритм знешумлення бінарного зображення руху з використанням алгоритму пошуку мінімального розрізу на графі, узагальнено до роботи з кольором та спрощено для конкретного випадку алгоритм співставляння шаблону (template matching), що базуєтсья на швидких перетвореннях Фур’є (FFT), і головне — реалізовано алгоритм пошуку оптичного потоку на основі алгоритму імітації відпалу (simulated annealing) з розподілом Ґіббса (семплювання за Ґіббсом).

Ключові слова: оптичний потік, максимальний потік, мінімальний розріз, імітація відпалу, семплювання за Ґіббсом, розпізнавання образів

ABSTRACT

This thesis contains: 38 pages, 6 figures and 16 references.

The aim of this work is to develop a method of motion tracking from video with static background on example of video games. The main component of this method is the optical flow searching. We've analyzed the existing optical flow methods, implemented the algorithm of noise reduction for binary motion images using the min-cut/max-flow algorithm, generalized the template matching algorithm for colored images based on Fast Fourier Transforms (FFT), and most importantly — implemented optical flow algorithm based on simulated annealing with Gibbs distribution (Gibbs sampling).

Keywords: optical flow, max-flow, min-cut, simulated annealing, Gibbs sampling, pattern recognition

ЗМІСТ

ВСТУП

Актуальність дослідження. Відеоігри та кіберспорт є одними з багатьох популярних у всьому світі явищ. Наразі існує велика кількість відеозаписів старих турнірів досить низької якості й інколи немає іншої можливості покращити їх, окрім як повністю перемалювати. У даній роботі представлено спосіб виокремлення зображень ігрових об’єктів для подальшої кластеризації і заміни. Отримані зображення називають спрайтами (від англ. sprite — фея, ельф) — двовимірні растрові зображення ігрових об’єктів, що використовуються в комп’ютерній графіці і зазвичай представляють собою послідовність зображень, що при періодичному відтворенні зображає об’єкт, що рухається. Після отримання таких спрайтів з відео їх можна використати для отримання фону відео більш ефективно, ніж, наприклад, медіанним фільтром. Самі спрайти можна замінити на інші або векторизувати як у статті Depixelizing Pixel Art [1]. Маючи чистий фон та спрайти, можна перемалювати сцени з відео для покращення його якості. Слід зазначити, що представлений алгоритм можна використовувати для довільних відеозаписів зі статичним фоном і об’єктами, що рухаються на ньому.

Метою дослідження є розробка алгоритму вилучення анімаційних зображень об’єктів з відео зі статичним фоном з метою подальшої кластеризації і заміни на більш якісні або просто інші. Для досягнення цієї мети необхідно реалізувати алгоритм пошуку оптичного потоку на основі алгоритму імітації відпалу з розподілом Ґіббса. Для цього необхідно вирішити такі завдання:

1) провести огляд літератури та існуючих методів пошуку руху на відео, пошуку оптичного потоку тощо;
2) отримати зображення об’єктів, що рухаються;
3) вирівняти зображення відносно одне одного;
4) описати алгоритм пошуку оптичного потоку на основі алгоритму імітації

відпалу з розподілом Ґіббса;

5) реалізувати алгоритм програмно;
6) провести експерименти, визначити штрафні функції та параметри, їх залежність від вхідних даних.

Об'єктами дослідження є алгоритм пошуку оптичного потоку, алгоритм імітації відпалу з розподілом Ґіббса, задачі пошуку максимального потоку і мінімального розрізу на графі, швидкі перетворення Фур'є.

Предметами дослідження є комп'ютерний зір, оптичний потік.

Практичне значення. Отриманий алгоритм пошуку оптичного потоку впорався із завданням краще, ніж відомий метод Лукаса-Канаде [2], в тому числі і за рахунок того, що він працює не з інтенсивністю пікселів, а з самим кольором. Також є можливість регулювати кількість ітерацій, радіус пошуку і параметри вагів штрафних функцій, що відповідають «важливості» різниці кольорів відповідних пікселів або різниці зсувів сусідніх пікселів. В подальшому можна переписати код програми на більш низькорівневу мову програмування (наприклад на C++) для значного покращення часу роботи програми.

Публікації. Роботу було представлено на XIX Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених в секції «Математичні методи комп'ютерного моделювання та кібернетичної безпеки» [3].

1 ОГЛЯД ЛІТЕРАТУРИ ТА ІСНУЮЧИХ МЕТОДІВ

Роботу з отримання спрайтів з відео слід розпочати з огляду методів відстеження об'єктів та пошуку оптичного потоку. З існуючих методів пошуку оптичного потоку можна виділити дві групи: дискретні та диференціальні. В дискретних методах простір квантується, кожному пікселю ставиться відповідна мітка таким чином, щоб відстань між сусідніми кадрами була мінімальною. Розв'язок таких задач шукається за допомогою лінійного програмування, алгоритмів пошуку максимального потоку і мінімального розрізу на графі. Диференціальні методи базуються на часткових похідних сигналу (наприклад інтенсивності пікселя) і на деяких припущеннях, зроблених задля обходу неоднозначності складених систем рівнянь. Найвідоміший з таких методів — метод Лукаса-Канаде.

1.1 Задачі лінійного програмування. Задачі max-flow та min-cut

Нехай $x \in \mathbb{R}^m$ — точка з m-вимірного простору дійсних чисел, а I — множина індексів $(1,2,\ldots,m)$. На деякі координати вектора x накладено обмеження: координати з індексами з множини $I^+ \subset I$ можуть приймати лише невід'ємні значення. На всі інші координати з індексами з множини $I^0 = I \setminus I^+$ обмежень не накладено. Позначимо множину $X = \{x \in \mathbb{R}^m : x_i \geq 0; i \in I^+\}$. Нехай $y \in \mathbb{R}^n$ — точка з n-вимірного простору дійсних чисел, а J — множина індексів $(1,2,\ldots,n)$. На деякі координати вектора y накладено обмеження: координати з індексами з множини $J^+ \subset I$ можуть приймати лише невід'ємні значення. На всі інші координати з індексами з множини $J^0 = J \setminus J^+$ обмежень не накладено. Позначимо множину $Y = \{y \in \mathbb{R}^n : y_j \geq 0; j \in J^+\}$. Нехай на множині $X \times Y$ визначено функцію двох змінних $f : X \times Y \to \mathbb{R}$. Таку функцію називають *білінійною* і

вона має вигляд

$$f(x,y) = \sum_{i \in I} a_i x_i - \sum_{i \in I} \sum_{j \in J} b_{ij} x_i y_j + \sum_{j \in J} c_j y_j. \tag{1.1}$$

Точку x' будемо називати допустимою відносно функції (1.1), якщо функція $f(x',y)$ однієї змінної y обмежена зверху на множині Y. Позначимо $\widetilde{X} \subset X$ множину всіх допустимих точок x. Тоді функція $\varphi : \widetilde{X} \to \mathbb{R}$ зі значеннями

$$\varphi(x) = \max_{y \in Y} f(x,y)$$

визначена на множині $\widetilde{X}$ [4].

Пряма задача лінійного програмування полягає у наступному: спершу треба перевірити, чи є множина $\widetilde{X}$ непорожньою, потім, якщо $\widetilde{X} \neq \emptyset$, то слід перевірити, чи є функція $\varphi : \widetilde{X} \to \mathbb{R}$ обмеженою знизу. Якщо результат цих перевірок позитивний, то необхідно знайти таку точку $x^* \in \widetilde{X}$, яка мінімізує φ. Тоді кажуть, що задача лінійного програмування має розв'язок.

Двоїста задача лінійного програмування визначається симетрично. Множина $\widetilde{Y}$ — множина таких точок $y' \in Y$, що функція однієї змінної $f(x,y')$ обмежена зверху на множині X, а для будь-якого $y \in \widetilde{Y}$ функція

$$\psi(y) = \min_{x \in X} f(x,y).$$

Так само, як і у випадку прямої задачі, спочатку перевіряється умова $\widetilde{Y} \neq \emptyset$. Далі, якщо функція ψ обмежена зверху на множині $\widetilde{Y}$, то шукається точка y^*, що максимізує ψ. Після знайдення такої точки кажуть, що двоїста задача лінійного програмування має розв'язок [4].

Теорема 1.1. ***Перша теорема двоїстості.*** *Якщо пряма задача лінійного*

програмування має розв'язок, до двоїста до неї задача теж має розв'язок. При цьому зберігається рівність

$$\min_{x\in\widetilde{X}} \varphi(x) = \max_{y\in\widetilde{Y}} \psi(y).$$

Пряма задача лінійного програмування полягає у розв'язанні наступної задачі на умовний екстремум: знайти

$$\min_{x\in X} \sum_{i\in I} a_i x_i$$

за умов

$$\sum_{i\in I} b_{ij} x_i \geq c_j, j \in J^+,$$
$$\sum_{i\in I} b_{ij} x_i = c_j, j \in J^0,$$
$$x_i \geq 0, i \in I^+.$$

Подібним чином можна представити і двоїсту задачу: необхідно знайти

$$\max_{y\in Y} \sum_{j\in J} c_j y_j$$

за умов

$$\sum_{j\in J} b_{ij} y_j \leq a_i, i \in I^+,$$
$$\sum_{j\in J} b_{ij} y_j = a_i, i \in I^0,$$
$$y_j \geq 0, j \in J^+.$$

Вище описані формули, що визначають пару взаємно двоїстих задач лінійного програмування і правила переходу від прямої задачі до двоїстої і навпаки. Перехід від прямої задачі до двоїстої означає перехід від мінімізації однієї функції багатьох змінних з обмеженнями, що накладено на ці змінні, до максимізації функції від інших змінних з обмеженнями, що накладено на ці змінні, причому кожному обмеженню прямої задачі відповідає змінна в двоїстій, а кожній змінній прямої задачі відповідає обмеження в двоїстій. Також, якщо обмеження в прямій задачі мають вигляд $\leq$, то в двоїстій вони будуть обов'язково мати вигляд $\geq$. Коефіцієнти a_i в функції, яку треба мінімізувати в прямій задачі, будуть в правій частині обмежень двоїстої задачі. Так само і з c_j для двоїстої. Відповідні коефіцієнти біля змінних прямої задачі в лівій частині рівнянь та нерівностей $b_{i,j}$ будут зберігатись біля змінних в лівій частині рівнянь та нерівностей двоїстої задачі. Наступна теорема наводить важливі властивості точок x^* та y^* (розв'язок прямої та двоїстої задач лінійного програмування) [4].

Теорема 1.2. ***Друга теорема двоїстості (про доповнюючу жорсткість)***. *Нехай* $x^* = (x_i^*, i \in I)$ *— розв'язок прямої, а* $y^* = (y_j^*, j \in J)$ *— розв'язок двоїстої задачі лінійного програмування. Тоді:*

якщо деяка координата x_i^* *в точці* x^* *не дорівнює нулю, то в точці* y^* *і-те обмеження двоїстої задачі виконується як рівність;*

якщо j-те обмеження прямої задачі в точці x^* *виконується як строга нерівність, то змінна* y_i^* *двоїстої задачі буде дорівнювати нулю.*

Фактично це означає, що для будь-якого $i \in I$ справедлива рівність

$$x_i^* \left(a_i - \sum_{j \in J} b_{ij} y_j^* \right) = 0,$$

а для будь-якого $j \in J$

$$y_j^* \left(c_j - \sum_{i \in I} b_{ij} x_i^* \right) = 0.$$

Задача пошуку максимального потоку має наступний вигляд: нехай T множина вершин, $s, e \in T$ — джерело та сток відповідно, $\tau \subset T^2$ — множина дуг, пропускна спроможність дуги $tt' \in \tau$ дорівнює $c_{tt'}$, а потік, що проходить через неї, дорівнює $f_{tt'}$. Також для вершини t визначимо множини $P_t = \{t' : t't \in \tau\}$ — множина вершин, з яких йде ребро у t та $N_t = \{t' : tt' \in \tau\}$ — множина вершин, у які йде ребро з t. Маємо задачу

$$\sum_{t' \in N_s} f_{st'} \to \max_{f:\tau \to \mathbb{R}}$$

$$\begin{cases} f_{tt'} \le c_{tt'}, \forall tt' \in \tau, \\ \sum\limits_{p \in P_t} f_{pt} - \sum\limits_{n \in N_t} f_{tn} = 0, \forall t \in T \setminus \{s,e\}, \\ f_{tt'} \ge 0, \forall tt' \in \tau. \end{cases}$$

Двоїста до неї задача — задача пошуку мінімального розрізу

$$\sum_{tt' \in \tau} \varphi_{tt'} c_{tt'} \to \min_{\varphi:\tau \to \mathbb{R}}$$

$$\begin{cases} \varphi_{tt'} \ge \theta_t - \theta_{t'}, \forall tt' \in \tau, \\ \theta_s = 1, \\ \theta_e = 0, \\ \varphi_{tt'} \ge 0, \forall tt' \in \tau. \end{cases}$$

Нехай $B = \{tt' \in \tau : f_{tt'} = c_{tt'}\}$ — множина всіх ребер, потік в яких дорівнює пропускній спроможності (максимально завантажені труби). Тоді

задачу лінійного програмування можна представити у вигляді

$$\sum_{tt' \in B} \varphi_{tt'} c_{tt'} \to \min_{\varphi: \tau \to \mathbb{R}}$$

$$\begin{cases} \varphi_{tt'} \geq \theta_t - \theta_{t'}, \forall tt' \in B, \\ \theta_{t'} - \theta_t \geq 0, \forall tt' \notin B, \\ \theta_s = 1, \\ \theta_e = 0, \\ \varphi_{tt'} \geq 0, \forall tt' \in \tau. \end{cases}$$

Теорема 1.3. ***Теорема Форда-Фалкерсона (max-flow min-cut theorem).*** *Величина максимального потоку в графі дорівнює пропускній спроможності його мінімального розрізу.*

В термінах лінійного програмування це означає, що задача про максимальний потік є двоїстою до задачі пошуку мінімального розрізу на графі [5]. Це дає змогу обирати ефективні методи вирішення задач пошуку максимального потоку для вирішення задач пошуку мінімального розрізу. Одним з таких методів є метод Бойкова-Колмогорова [6].

1.2 Метод Лукаса-Канаде

В основі поняття оптичного потоку лежить основне найголовніше припущення про те, що при руху об'єкту між кадрами інтенсивність його пікселів не змінюється. Позначимо інтенсивність пікселя з координатами (x,y) в момент часу t як $I(x,y,t)$. Тоді в наступний момент часу $t + \Delta t$ рівняння потоку буде мати вигляд

$$I(x,y,t) = I(x + \Delta x, y + \Delta y, t + \Delta t), \tag{1.2}$$

де Δx та Δy — зміщення конкретного пікселя з попередьного кадру на наступний, а Δt — зміна часу (наприклад, наступний кадр, або через кадр). Розклавши праву частину (1.2) в ряд Тейлора до першої похідної, маємо

$$I(x,y,t) = I(x,y,t) + \Delta x\frac{\partial I}{\partial x} + \Delta y\frac{\partial I}{\partial y} + \Delta t\frac{\partial I}{\partial t}. \tag{1.3}$$

Оскільки за невеликий проміжок часу піксель зміщується на невелику відстань, рівняння (1.3) можемо записати у вигляді

$$\Delta x\frac{\partial I}{\partial x} + \Delta y\frac{\partial I}{\partial y} + \Delta t\frac{\partial I}{\partial t} = 0. \tag{1.4}$$

Рівняння (1.4) називають рівнянням оптичного потоку. З кожним пікселем зображення пов’язаний вектор швидкості

$$\vec{u} = (u_x,u_y) = \left(\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}\right), \tag{1.5}$$

де u_x, u_y — координати вектора швидкості. Рівняння (1.5) представляє собою оптичний потік.

Розглядаючи задачу пошуку оптичного потоку, слід згадати відомий метод Лукаса-Канаде [7]. Рівняння (1.4) не може бути розв’язано однозначно, тому що містить дві невідомі. Алгоритм Лукаса-Канаде вирішує цю проблему, використовуючи ще одне припущення: в локальному околі пікселя p з координатами (x_p,y_p) значення оптичного потоку таке ж саме. Тоді отримаємо

систему рівнянь

$$\begin{cases} \nabla I(p_1) \cdot (u_x, u_y) = -I'_t(p_1) \\ \nabla I(p_2) \cdot (u_x, u_y) = -I'_t(p_2) \\ \dots \\ \nabla I(p_n) \cdot (u_x, u_y) = -I'_t(p_n) \end{cases} \tag{1.6}$$

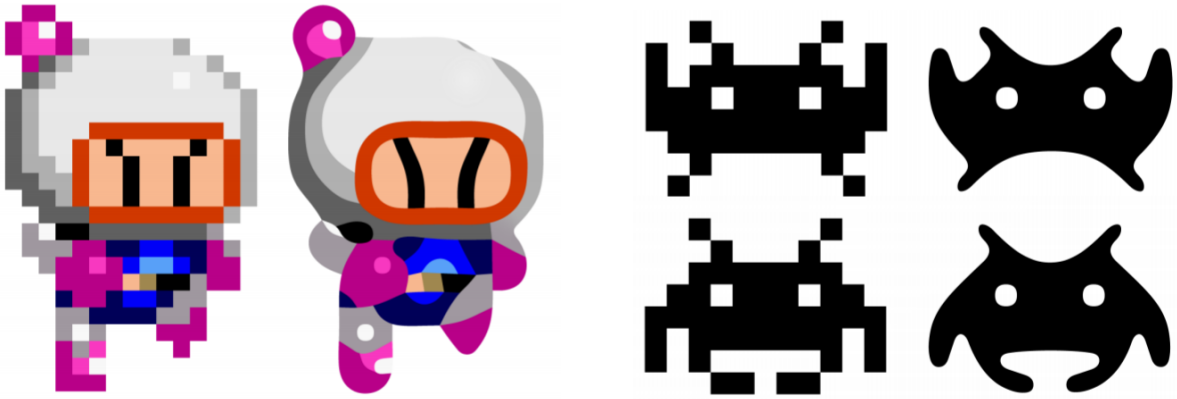
для всіх пікселів в околі, яку представимо у вигляді $A\vec{u} = \vec{b}$. Якщо до матриці A існує зворотня, то можемо знайти вектор $\vec{u} = A^{-1}\vec{b}$. Цю систему лінійних рівнянь зазвичай вирішують за допомогою методу найменших квадратів.

Метод Лукаса-Канаде має свої недоліки, адже він, як і саме поняття оптичного потоку, заснований на припущеннях, які виконуються не завжди. Перш за все — головне припущення про те, що значення пікселів об'єкту не змінюється. Ми розуміємо, що майже завжди це не так: на відео часто змінюється глобальне освітлення, вже не кажучи про зміни форми й кольору об'єктів і фону. При розкладанні в ряд Тейлора ми припустили, що нам вистачить тільки першої похідної, але в загальному випадку це не так, і таке обмеження не завжди дає бажану точність. Також припущення про локальний окіл не завжди буде працювати у випадку руху об'єктів, розмір яких більший за розмір локального околу і, наприклад, зафарбованих в один колір. Тоді всередині цього об'єкту рух може визначатися некоректно: він є, але його напрямок може відрізнятись від істиного напрямку руху об'єкту. Також проблеми виникають, коли ділянка зображення дає невироджену матрицю A, для якої немає зворотньої. В такому випадку ми не можемо визначити рух (він є, але невідомо, в якому напрямку). І головний на мій погляд недолік полягає у тому, що алгоритм спирається лише на інтенсивність пікселів, яка може відповідати декільком різним кольорам. В такому випадку алгоритм не використовує більш повну інформацію про зображення, дістати яку зовсім не важко. Узагальнення задачі до роботи з трьома каналами [8] замість одного призвело б з одного боку до падіння продуктивності, але з іншого — до

значного збільшення точності. Але, незважаючи на всі недоліки, даний метод на практиці дуже часто працює достатньо добре, навідь в умовах високої зашумленості.

1.3 Покращення якості

Після вилучення анімаційних зображень з ігрових об’єктів (спрайтів) можна перейти до заміни їх на інші, перемалювавши їх або ж покращивши їхню якість. Для останнього існує спосіб, представлений у статті Depixelizing pixel art [1]. На вхід було подано різні зображення персонажів старих ігор у низькій роздільній здатності. У статті представлено алгоритм векторизації цих зображень — процесу перетворення растрових зображень на векторні, що дозволяє наближати зображення без втрати якості або перемальовувати їх у довільній роздільній здатності. Приклад роботи алгоритму можемо побачити на рисунку 1.1.



(а) (б)

Рисунок 1.1 — Приклади роботи алгоритму з Depixeling pixel art [1]

1.4 Інші способи вилучення спрайтів із комп’ютерних ігор

Взагалі існують методи вилучення ігрових об’єктів шляхом безпосереднього проникнення в файли гри, а деякі компанії навіть схвально до цього ставляться, створюючи для цього відповідне програмне забезпечення, відкривачи цілі майданчики для користувацької творчості у вільному доступі і дозволячи використовувати користувацькі модифікації у власних іграх. Але по-перше, в загальному випадку це незаконно та (або) неможливо, а по-друге — це не є задачею розпізнавання, комп’ютерного зору або іншого підрозділу математики, який нас цікавив. Також є робота з отримання спрайтів із відеозапису на зеленому фоні [9]. Вона цікава з точки зору застосування — отриману анімацію змушують реалістично переміщуватись в просторі на іншому фоні за заданою траєкторією.

Висновки до розділу 1

Ми розглянули дискретні та диференціальні методи розв’язку задачі пошуку оптичного потоку, а також варіант покращення якості отриманих спрайтів шляхом векторизації. Ми навели означення прямої та двоїстої задачі лінійного програмування і показали, що задача пошуку максимального потоку на графі, що був побудований на основі бінарного зображення, зводиться до задачі пошуку мінімального розрізу на цьому графі і їх розв’язки співпадають (за теоремою Форда-Фалкерсона). Ми прийшли до висновку, що більшість методів можна легко узагальнити до роботи з кольором [8], а також спростити їх в окремих випадках. В наступному розділі мова йтиме про алгоритми на їх основі, детальний опис, спрощення і реалізацю.

2 ВИОКРЕМЛЕННЯ ТА ВИРІВНЮВАННЯ ОБ'ЄКТІВ, ЩО РУХАЮТЬСЯ

Роботу з відстеження руху було розбито на етапи: основний — реалізація алгоритму пошуку оптичного потоку на основі алгоритму імітації відпалу з розподілом Ґіббса та підготовчі — ті, які оброблять вхідні дані для основного.

Найперший етап — пошук руху *(motion detection)*. На вході — відео зі статичної камери і об'єктами, що рухаються, на ньому. На виході ми отримаємо зображення всіх об'єктів, що рухались на відео, значно меншого розміру, ніж саме відео. На цих зображеннях ділянки, що не рухались, було видалено задля відсутності впливу їх на подальшу роботу наступних алгоритмів.

Другий етап — вирівнювання отриманих на попередньому кроці зображень для подальшої роботи з ними. Як було сказано раніше, анімаційні зображення об'єктів являють собою періодичну послідовність кадрів. На попередньому кроці ми отримали уривки цих анімацій і нам потрібно вирівняти їх між собою (знайти оптимальне взаємне розташування).

На вході маємо множину зображень різних об'єктів, деякі з них зображають одні й ті самі об'єкти у різний момент часу. На виході для кожного з цих зображень отримуємо вирівняні відносно нього усі інші зображення або, для економії місця та часу, тільки координати необхідних зсувів.

2.1 Відстеження об'єктів, що рухаються

Поступово розбиваючи відео на послідовність кадрів, на кожному кроці, починаючи з другого, маємо два зображення розміру $h \times w$: $F \in C^{h \times w}$ і $S \in C^{h \times w}$, де C — множина кольорів у форматі *RGB*. Нехай D — бінарне

зображення розміру $h \times w$, таке, що $\forall p \in [1,h] \times [1,w]$:

$$D_p = \begin{cases} 1, & \|F_p - S_p\| > \alpha, \\ 0, & \|F_p - S_p\| \leqslant \alpha, \end{cases} \tag{2.1}$$

де α — деякий коефіцієнт, зазвичай у межах 5–30, що виникає внаслідок шуму на відео.

На отриманому бінарному зображенні можемо побачити деякі зашумлені області через те, що зазвичай рух об'єкту не означає зміну кольору кожного пікселя в ньому. Для отримання потрібних нам зображень виникає необхідність модифікувати D.

Задачу знешумлення бінарного зображення можна представити у вигляді задачі дискретної оптимізації

$$\sum_{p \in P} q_p(k_p) + \sum_{pp' \in N_p} g_{pp'}(k_p,k_{p'}) \to \min_{k:P \to K}, \tag{2.2}$$

де штрафні функції

$$q_p(k_p) = |D_p - k_p|\,, \tag{2.3}$$

$$g_{pp'}(k_p,k_{p'}) = \alpha \cdot |k_p - k_{p'}|\,, \tag{2.4}$$

а множина $K = \{0, 1\}$.

Відомо [6, 10, 11], що задача (2.2) зводиться до задачі пошуку мінімального розрізу на графі, а ця задача в свою чергу є двоїстою до задачі пошуку максимального потоку, що було показано в розділі 1. Це важливо в першу чергу тому, що ця задача має ефективний розв'язок.

Наведемо задачу знешумлення зображення (2.1) з використанням задачі пошуку мінімального розрізу на графі. На вхід подається бінарне зображення

D (2.1). Побудуємо граф, вузли $p_i \in P$ в якому будуть відповідати пікселям цього зображення, додамо ще дві вершини: джерело — мітка $k_1 \in K$, та сток — мітка $k_2 \in K$. Із джерела k_1 в усі вузли (пікселі зображення) виходять орієнтовані ребра. Для об'єкту p_i вага вхідного ребра — $q_{p_i}(k_2)$. Також з усіх вузлів p_i виходять ребра у сток, який ми визначили як k_2. Ваги таких ребер дорівнюють $q_{p_i}(k_1)$. Між сусідніми вершинами проведено по два орієнтованих ребра в обидві сторони. Ваги таких ребер, як видно із формули бінарного штрафу (2.3), дорівнюють α. Сусіди пікселя p з координатами (x,y) — пікселі з координатами $(x+1,y)$, $(x-1,y)$, $(x,y+1)$, $(x,y-1)$. Звичайно, для пікселів на границях сусідів менше. На рисунку 2.1 можемо побачити приклад графу у випадку бінарного зображення розміром 2×3 пікселі для задачі пошуку мінімального розрізу на графі.

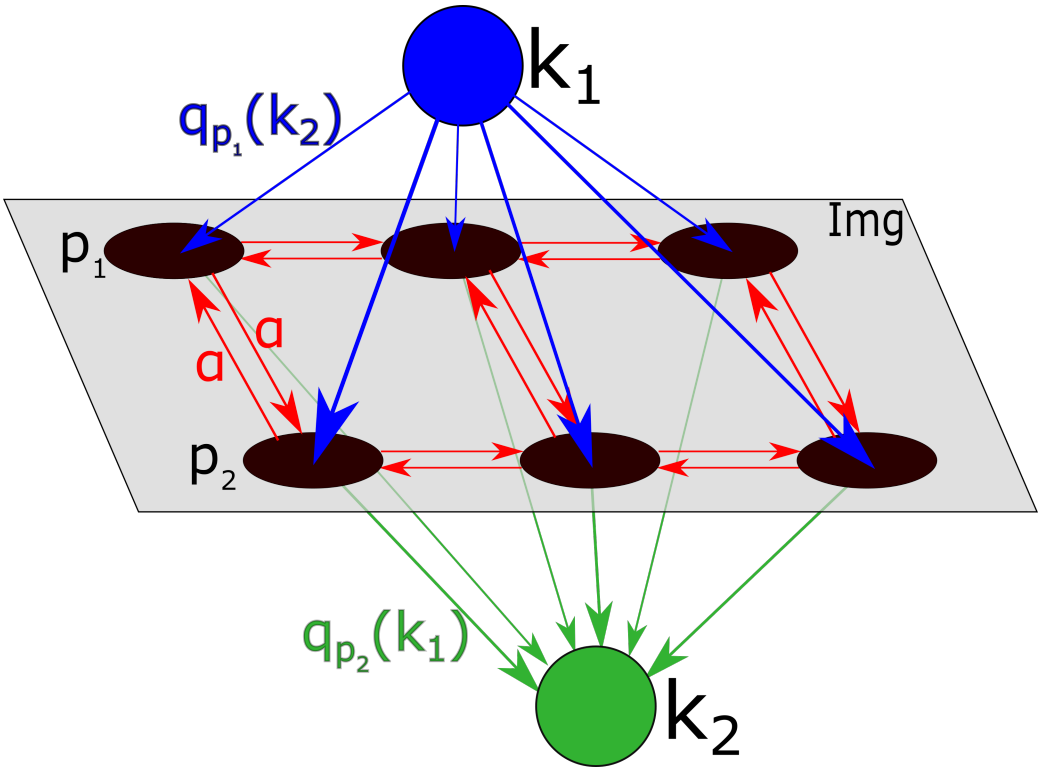


Рисунок 2.1 — Приклад графу для задачі пошуку мінімального розрізу на основі бінарного зображення

Задача пошуку мінімального розрізу на графі полягає у пошуку такого розрізу графу, що:

1) розріз ділить граф на два класи так, щоб сток і джерело були у різних класах;
2) розріз має проходити через ті ребра, для яких потік дорівнює пропускній спроможності (максимально навантажені труби);
3) у розрізі сума потоку з дуг, що направлені до стоку (сумарна кількість рідини, що витікає з розрізаних труб), має бути мінімальною.

Нагадаємо, що теорема Форда-Фалкерсона каже, що розріз, який відділяє джерело від стоку і мінімізує сумарний потік у дугах, що направлені з підграфу із джерелом у підграф із стоком, має вагу, рівну максимальному потоку, який можна пустити із джерела у сток через цей граф [5].

Якщо застосувати задачу мінімального розрізу на графі, побудованого на основі бінарного рображення D (2.1), то, розбивши граф на два класи, отримаємо бінарне зображення, пікселі першого класу у якому будуть одного кольору k_1, а пікселі іншого класу — іншого кольору k_2.

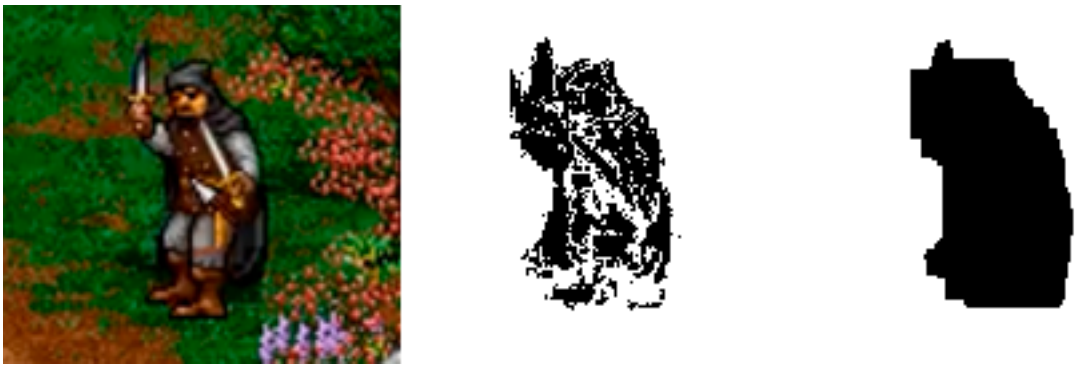


Рисунок 2.2 — Приклад роботи алгоритму пошуку максимального потоку. Оригінал відео: https://youtu.be/g42gSMad09Y

Оскільки ми з’ясували, що величина максимального потоку дорівнює потоку мінімального розрізу (за теоремою Форда-Фалкерсона), то було прийнято рішення використовувати алгоритм Бойкова-Колмогорова для пошуку максимального потоку на графі. Цей алгоритм має ефективний розв’язок і працює набагато швидше на відміну від аналогів — алгоритмів Форда-Фалкерсона, що використовує пошук у глибину, чи Едмондса-Карпа,

що використовує пошук у ширину. На рисунку 2.2 бачимо об'єкт, що рухається, початкове бінарне зображення D та розв'язок задачі видалення бінарного шуму.

2.2 Оптимальне взаємне розташування зображень

На попередньому кроці ми отримали деяку кількість різних зображень різних об'єктів. Для виконання наступних кроків необхідно визначити їх оптимальне взаємне розташування.

Нехай є два зображення $F = C^{h_1 \times w_1}$ і $S = C^{h_2 \times w_2}$. Треба знайти таку пару $(x^*,y^*) \in \mathbb{Z}^2$, що зміщення зображення F на x^* по горизонталі і на y^* по вертикалі відносно зображення S дасть вирівняне зображення F^* відносно S.

Нехай F' — зображення, що зміщено на таке (x',y'), що

$$(x',y') \in [-h_1, H] \times [-w_1, W],$$
$$H = h_1 + h_2 - 1,$$
$$W = w_1 + w_2 - 1.$$

Маємо задачу мінімізації

$$(x^*, y^*) = \underset{\substack{x' \in [-h_1, H] \\ y' \in [-w_1, W]}}{\operatorname{argmin}} \sum_{\substack{1 \leqslant i \leqslant h_2 \\ 1 \leqslant j \leqslant w_2}} \left(F'_{i+x',j+y'} - S_{i,j}\right)^2,$$

а, розкривши квадрат, отримуємо

$$(x^*, y^*) = \underset{\substack{x' \in [-h_1, H] \\ y' \in [-w_1, W]}}{\operatorname{argmin}} \sum_{\substack{1 \leqslant i \leqslant h_2 \\ 1 \leqslant j \leqslant w_2}} F'^2_{i+x',j+y'} + S^2_{i,j} - 2F'_{i+x',j+y'} \cdot S_{i,j},$$

де $F'_{i+x',j+y'} \cdot S_{i,j}$ — кореляція двох зображень. Спираючись на те, що за будь-якого зсуву F сума квадратів кольорів кожної з картинок залишається незмінною (вважаємо, що за межами S всі пікселі чорні), можемо прибрати суму квадратів кольорів окремих зображень і перейти до задачі максимізації кореляції двох зображень

$$(x^*, y^*) = \underset{\substack{x' \in [-h_1, H] \\ y' \in [-w_1, W]}}{\operatorname{argmax}} \sum_{\substack{1 \leqslant i \leqslant h_2 \\ 1 \leqslant j \leqslant w_2}} F'_{i+x',j+y'} \cdot S_{i,j}. \tag{2.5}$$

Задачу (2.5) можна оптимізувати, використовуючи швидкі перетворення Фур'є (FFT) [12, 13]. З теореми про згортку знаємо, що після перетворень Фур'є згортку двох зображень можна представити у вигляді простого поелементного перемноження

$$F \circledast G = \mathcal{F}^{-1}\left(\mathcal{F}(F \circledast G)\right) = \mathcal{F}^{-1}\left(\mathcal{F}(F) \cdot \mathcal{F}(G)\right), \tag{2.6}$$

де $\circledast$ — оператор згортки, $\mathcal{F}(x)$ та $\mathcal{F}^{-1}(x)$ — прямі та зворотні перетворення Фур'є. Прямі перетворення Фур'є визначаються формулою

$$F_k = \sum_{n=0}^{N-1} f_n \cdot e^{\frac{-2\pi i}{N} kn},$$

а зворотні — формулою

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k \cdot e^{\frac{2\pi i}{N} kn},$$

де N — кількість значень сигналу за період, f_n — значення сигналу в точках з номерами $n = 0,1,\ldots,N-1$ а F_k — k-й компонент перетворень Фур'є ($k =$

$0,1,\ldots,N-1$).

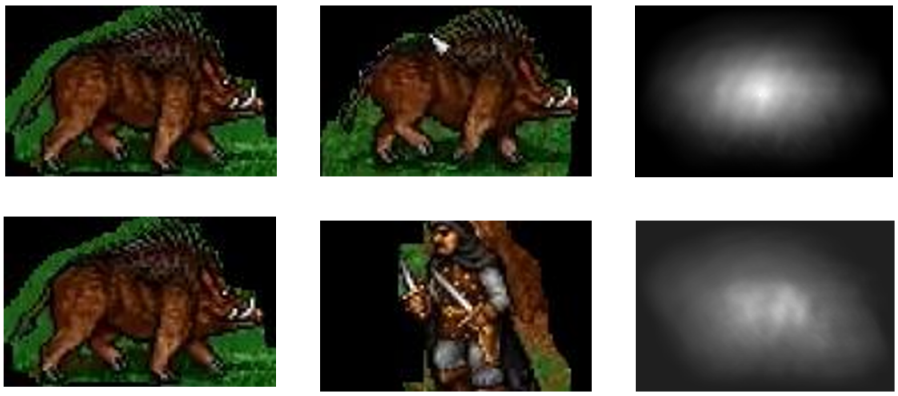


Рисунок 2.3 — Приклади мап згорток. Оригінал відео: https://youtu.be/g42gSMad09Y

На рисунку 2.3 бачимо зображення F, два можливих S та мапи згорток для них. На першій мапі видно максимум. Його координати і стануть зсувом (x^*,y^*) для F.

За великої кількості картинок у випадку пошуку взаємного розташування «всіх зі всіма» прямі перетворення Фур'є можна обчислювати один раз для кожного окремого зображення.

Висновки до розділу 2

В цьому розділі ми використали алгоритм пошуку максимального потоку для знешумлення бінарного зображення, що зображає рух (2.1). Цей алгоритм має ефективний розв'язок. Відомі алгоритми Форда-Фалкерсона або Едмондса-Карпа теж можна було використати, але вони працюють довго відносно того, який ми обрали, а саме — алгоритм Бойкова-Колмогорова [6]. На виході з програми ми отримали зображення ігрових об'єктів без значної частини фону, не потрібного для подальшої роботи.

Реалізовано алгоритим вирівнювання ігрових об'єктів на зображеннях, отриманих на попередньому кроці. За його основу було взято ідею алгоритму

співставляння шаблону (template matching), але із урахуванням не інтенсивності пікселя (grayscale), а інтенсивності трьох каналів кольорів, що дає більше інформації про зображення. Також, враховуючи особливості отриманих зображень (зображення схожого розміру, навколо об’єкта чорний фон, за межами зображень вважаємо, що теж чорний фон), ми значно спростили штрафну функцію: з квадрату різниці до простого перемноження. Це прискорило час роботи програми і компенсувало падіння продуктивності через роботу з кольором.

На основі отриманих результатів (вирівняні зображення різних об’єктів) перейдемо до опису задачі пошуку оптичного потоку.

3 ПОШУК ОПТИЧНОГО ПОТОКУ

На попередньому кроці для всіх зображень, отриманих з відео за допомогою алгоритму пошуку руху, ми отримали множини зображень, кожна з яких містить унікальне основне зображення та множину інших зображень такого ж розміру, але з вирівняними об'єктами на них відносно об'єкту на основному. Зрозуміло, що в таких множинах є зображення одного й того ж самого об'єкту і зображення інших об'єктів, які трапили туди випадково і не цікавлять нас. Для зображень одного об'єкту маємо знайти оптичний потік підоб'єктів (наприклад — руки або ноги солдата) для подальшого формування коректних анімаційних зображень.

Розглянутий у розділі 1 алгоритм Лукаса-Канаде не дає необхідного результату за рахунок своїх недоліків, хоча працює швидко. Розглянемо алгоритм імітації відпалу з розподілом Ґіббсу і на його основі побудуємо ітеративний алгоритм пошуку оптичного потоку зі зменшенням «температури системи» з часом.

3.1 Випадкове поле з розподілом Ґіббса

Нехай є випадкове поле на множині пікселів P зі структурою сусідства другого порядку $\tau \subset P \times P$, множиною міток K, унарною ваговою функцією $q : P \times K \to \mathbb{R}$, бінарною ваговою функцією $g : K^2 \to \mathbb{R}$ та ймовірністю

$$\mathbb{P}(k) \;=\; \frac{1}{Z(n)} \exp\left\{ \frac{\sum\limits_{p \in P} q(p,k_p) + \sum\limits_{pp' \in \tau} g(k_p,k_{p'})}{T(n)} \right\}, \tag{3.1}$$

де $Z(n)$ — нормуючий множник, а $T(n)$ — температура системи, що зменшується з часом n. Мітка k_p — така пара $(x,y) \in \mathbb{Z}^2$, яка відповідає

ймовірному двовимірному зсуву об'єкта (пікселя) p.

3.2 Структура сусідства

Структуру сусідства $\tau \in P^2$ було обрано наступним чином: для об'єкту p з координатами (x, y) є чотири сусіда з координатами $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$ та $(x,y - 1)$. Пікселі на краях зображень мають меншу кількість сусідів. Так, наприклад, піксель з координатами $(0,0)$ має лише двох сусідів: $(0,1)$ та $(1,0)$. Позначимо множину всіх сусідніх до p пікселів як $N_p \subset P$. За допомогою формули маргінального розподілу було показано, що для будь-якого фіксованого об'єкту p ймовірність мітки k_p в ньому за умови, що мітки в усіх інших об'єктах відомі, така сама, як і за умови, що відомі лише мітки всіх сусідів p

$$\mathbb{P}(k_p \mid k_{p'}, p' \in P \setminus \{p\}) = \mathbb{P}(k_p \mid k_{p'}, p' \in N_p). \tag{3.2}$$

Враховуючи чотири об'єкти замість тисяч або навіть мільйонів, величину (3.2) значно приємніше та швидше обчислювати.

3.3 Вибір штрафних функцій

З розподілу (3.1) можемо побачити в показнику експоненти суми двох штрафних функцій — унарну q та бінарну g. Очевидно, що збільшення (за модулем) штрафу зменшує ймовірність мітки в фіксованому об'єкті (самі штрафні функції завжди від'ємні або нульові). Унарна штрафна функція q має відповідати розбіжності (несхожості) кольорів зміщеного пікселя попереднього кадру і пікселя з тими ж координатами наступного кадру:

більша розбіжність — більший штраф. Було обрано штрафну функцію

$$q(p,k_p) = -\theta \cdot \|F(p + k_p) - S(p)\|,$$

де $F(p + k_p)$ — колір пікселя, зміщеного на мітку K_p відносно p на першому зображенні, а $S(p)$ — колір пікселя p на другому зображенні. Параметр θ — деякий ваговий коефіцієнт важливості для q — чим він більший, тим більше алгоритм буде враховувати саме цей штраф.

В якості бінарного штрафу було обрано узагальнення Поттс-моделі. В ній мітка-кандидат штрафується за кожного сусіда, в якого мітка відрізняється більше ніж на вибране значення параметру (тобто норма різниці міток k_p та $k_{p'}, p' \in N_p$ більша ніж параметр), і дає нуль за кожного сусіда, в якого така сама мітка. Функція g має вигляд

$$g(k_p,k_{p'}) = -\alpha \cdot \mathbb{1}\{\|k_p - k_{p'}\| > \beta\},$$

де α — коефіцієнт згладжування, а β — певний діапазон допустимого відхилення для міток у сусідніх об'єктах.

3.4 Побудова алгоритму пошуку оптичного потоку

Після того, як ми визначили структуру сусідства та штрафні функції для нашого випадкового поля, можемо перейти до опису алгоритму пошуку оптичного потоку. Він полягає у знаходженні таких координат зсуву для кожного пікселя зображення F, що якщо кожен піксель змістити по цим координатам, то маємо отримати зображення, максимально зхоже на зображення S (йдеться не тільки про візуальну схожість, а і схожість по нормі — сума квадратів різниці по кожному пікселю зміненого зображення F'

та зображення S має бути якомога меншою). Слід також зауважити, що задача алгоритму полягає не тільки у тому, щоб знайти найкращі координати зсуву кожного пікселя окремо, а ще й у тому, щоб знайти такі ділянки, множина пікселей у яких буде рухатись в одному напрямку. Для цього потрібно вибирати координати зсуву, орієнтуючись не тільки на бажаний колір, а й на координати зсуву сусідніх об'єктів. Цього можна досягнути з вибраними штрафними функціями шляхом регулювання їх вагових коефіцієнтів α та θ.

Для значного прискорення роботи програми було прийнято рішення обмежити множину міток K максимальною довжиною зсуву кожного пікселя величиною r

$$K = \{k : l_\infty(k) \leq r\}.$$

Тут K — множина таких $k_p = [x_p, y_p]$, що x_p — можливий зсув пікселя p по вісі x, а y_p — по вісі y, l_∞ — норма Чебишова.

Алгоритм ітеративний: на кожній ітерації він буде давати точніший і ймовірніший результат за рахунок випадковості. Координати зсуву для об'єкту будемо шукати шляхом випадкового вибору з дискретного розподілу, який сформуємо з ймовірностей всіх міток в цьому об'єкті. Якщо вибрати об'єкт p та зафіксувати всі $k_{p'} : p' \neq p$, то для кожної мітки $k_p \in K$ маємо ймовірність

$$\mathbb{P}(k_p \mid k_{p'}, p' \in N_p) = \frac{f(k_p)}{h(k_{p_i}; p_i \in N_p)},$$

де

$$f(k_p) = \mathbb{P}(k_p) = \exp\left\{\frac{q(p,k_p) + \sum\limits_{p' \in N_p} g(k_p, k_{p'})}{T(n)}\right\},$$

а

$$h(k_{p_i}; p_i \in N_p) = \mathbb{P}(k_{p_1}, \ldots, k_{p_{|N_p|}}) = \sum_{k \in K} \exp\left\{\frac{q(p,k) + \sum\limits_{p' \in N_p} g(k, k_{p'})}{T(n)}\right\}.$$

Слід зазначити, що для фіксованого p величина $h(k_{p_i}; p_i \in N_p)$ буде однаковою для всіх можливих зсувів k_p і не впливатиме на знаходження оцінки найбільшої правдоподібності, тому її можна прибрати як константу й обчислювати лише величину $f(k_p)$.

Алгоритм пошуку оптичного потоку наступний:

1) для кожного пікселя обираємо довільну (наприклад — нульову) мітку та визначаємо кількість ітерацій

$$\forall p \in P : k_p \leftarrow (0,0),$$
$$n \leftarrow 0,$$
$$n_{max} = 10;$$

2) для пікселя p знаходимо всі можливі ймовірності переходів k_p в радіусі r

$$\forall k_p \in K : \mathbb{P}(k_p) = \exp\left\{ \frac{q(p,k_p) + \sum\limits_{p' \in N_p} g(k_p,k_{p'})}{T(n)} \right\};$$

3) генеруємо нове k_p з розподілу, що було отримано на попередньому кроці;
4) кроки 2–3 виконуємо для всіх пікселів $p \in P$;
5) якщо $n < n_{max}$, то $n \leftarrow n+1$ та повертаємось до кроку 2. Інакше — кінець алгоритму.

Кількість ітерацій та параметри слід обирати в залежності від вхідного відео та розміру об'єктів на ньому. Порядок обходу пікселів може бути довільний. Початкову розмітку, що обирається на першому кроці, було обрано нульову — кожен піксель має мітку $(0,0)$, що означає, що він нікуди не рухається.

Температура системи $T(n)$ зменшувалась за законом

$$T(n) = 10 \cdot \lambda^n,$$

де λ — швидкість охолодження системи. Разом із падінням температури буде зростати ймовірність найбільш правдоподібної мітки і падати ймовірність всіх інших міток.

3.5 Програмна реалізація

Мовою програмування було обрано Python [14], а для зменшення часу роботи програми було додано JIT-компілятор Numba [15]. На вхід програма приймає два зображення, що йдуть одне за іншим у відео, і наступні параметри:

- кількість операцій n_{max};
- радіус *r*;
- параметр a для функції g ;
- параметри β та θ для функції q;
- параметр λ для обчислення поточної температури $T(n)$.

На виході програма дає мапу зсувів у вигляді масиву того ж розміру, що й зображення, але з координатами зсуву замість кольору, або ж у вигляді картинки у форматі *HSV*. Також на виході програма дає зображення, що було отримано в результаті зсуву всіх пікселів першого зображення за мапою зсувів.

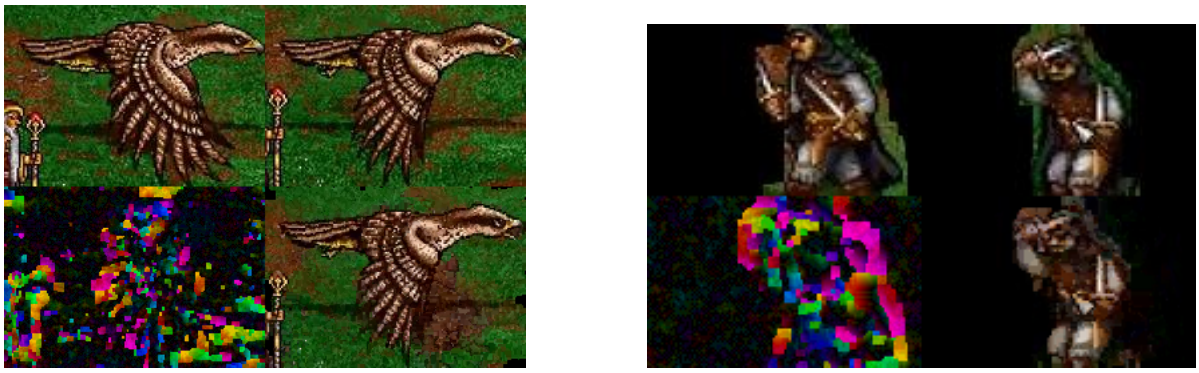
На рисунку 3.1 можемо побачити приклад роботи програми. Колір на мапі зсувів означає кут зсуву, а інтенсивність кольору — відстань зсуву. Можна побачити деякі однорідні ділянки на мапах зсувів — переміщення груп пікселів в одному напрямку.

На рисунку 3.2(а) можна побачити результат роботи програми, коли

(а) (б)

Рисунок 3.1 — Приклади оптичного потоку: (а) — ігровий об’єкт, (б) — фотографія

обрано занадто малий радіус r. Алгоритм не може знайти рух крил, які між кадрами здолали відстань більшу, ніж радіус пошуку. На рисунку 3.2(б) зображено результат роботи програми, якщо на вхід давати зображення без зайвого фону, який ми прибрали в розділі 2.



(а) (б)

Рисунок 3.2 — Приклади оптичного потоку: (а) — малий радіус r, (б) — зображення без зайвого фону

Висновки до розділу 3

Ми реалізували алгоритм пошуку оптичного потоку на основі семплеру Ґіббса, підібрали штрафні функції та їхні параметри. В ході тестування не

було помічено випадків, коли оптичний потік не було знайдено, але були помічені випадки некоректно знайденого оптичного потоку. Це усувалося за рахунок корегування параметрів штрафних функцій. Навіть для зовсім різних зображень алгоритм намагається знайти оптичний потік, але за рахунок того, що сусідні пікселі часто хочуть піти різними напрямками, алгоритм орієнтується лише на різницю в кольорі і, не дивлячись на все, попереднє зображення, отримане в результаті зсуву по мапі, часто схоже на наступне, хоча оптичний потік представляє собою шум без монотонних ділянок.

Недоліком реалізації можна вважати обрану мову програмування Python. Програма працює хоч і швидше за рахунок JIT-компілятора numba [15], але це все одно недостатньо швидко в порівнянні, наприклад, з мовою C++. В подальшому розвитку цієї роботи можна переписати код більш низькорівневою мовою, застосувати багатопоточність або CUDA [16].

ВИСНОВКИ

В ході роботи було розроблено алгоритм вилучення анімаційних зображень об’єктів з відео зі статичним фоном для подальшого покращення їхньої якості (наприклад, за допомогою Depixelizing pixel art [1]), стилізації або просто заміни на інші.

Проведено детальний огляд літератури та існуючих методів пошуку руху та оптичного потоку. Слід відмітити ефективний метод Лукаса-Канаде для пошуку оптичного потоку та різні методи пошуку руху на основі лінійного програмування.

Поставлено задачу знешумлення бінарного зображення руху та зведено її до пошуку мінімального розрізу на графі. Використовуючи теорему Форда-Фалкерсона про те, що величина максимального потоку дорівнює потоку мінімального розрізу, програмно реалізовано алгоритм знешумлення бінарного зображення з використанням алгоритму пошуку максимального потоку методом Бойкова-Колмогорова, що має ефективний розв’язок. Програмна реалізація саме цього алгоритму виявилась значно швидшою, ніж реалізація алгоритмів Форда-Фалкерсона та Едмондса-Карпа.

Узагальнено алгоритм співставляння шаблону (template matching) для трьох каналів кольору, спрощено його до конкретної задачі і також реалізовано програмно з використанням швидких перетворень Фур’є (FFT).

Головне досягнення цього дослідження — реалізація алгоритму пошуку оптичного потоку на основі алгоритму імітації відпалу з розподілом Ґіббса (семплювання Ґіббса). Ми детально описали цей алгоритм, підібрали найкращі штрафні функції для ефективної програмної реалізації, а також підібрали найкращі параметри для конкретної задачі.

В подальшому можна продовжити дослідження: розробити алгоритм вилучення фону з урахуванням отриманих рухомих об’єктів, який буде ефективнішим за медіанний фільтр, який припускає, що фон в кожній окремій

ділянці наявний більше половини від всього часу відео. Також можна об’єднати пошук руху і пошук оптичного потоку: для послідовних ділянок послідовних кадрів, на яких було знайдено рух, одразу застосовувати пошук оптичного потоку. І головне — знайти спосіб заміни отриманих зображень об’єктів на відео на інші зображення, стилізовані або якісно відмальовані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1 Kopf, Johannes. Depixelizing Pixel Art. [Текст] / Johannes Kopf, Dani Lischinski // *ACM Trans. Graph.* — 2011. — 07. — Vol. 30. — P. 99.

2 Barron, John. Performance Of Optical Flow Techniques. [Текст] / John Barron, David Fleet, S. Beauchemin // *International Journal of Computer Vision*. — 1994. — 02. — Vol. 12. — Pp. 43–77.

3 Павлюк, А. Д. Відстеження періодичного руху об’єктів зі статичної камери на прикладі відеоігор. [Текст] / А. Д. Павлюк, В. М. Кригін, Ткач В. М. // *XIX Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених*. — 2021. — Pp. 255–257.

4 Шлезингер, М. Десять лекций по статистическому и структурному распознаванию. [Текст] / М. Шлезингер, В. Главач. — Киев: Наукова думка, 2004.

5 Вступ до алгоритмів. [Текст] / Т. Кормен, Ч. Лейзерсон, Р. Рівест et al. — К.І.С., 2019. — Pp. 718–775.

6 Boykov, Yuri. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. [Текст] / Yuri Boykov, Vladimir Kolmogorov // *IEEE Trans. Pattern Anal. Mach. Intell.* — 2004. — . — Vol. 26, no. 9. — P. 1124–1137.

7 Fleet, David J. Optical Flow Estimation. [Текст] / David J. Fleet, Y. Weiss // Handbook of Mathematical Models in Computer Vision. — 2006.

8 Aires, Kelson. Optical flow using color information. [Текст] / Kelson Aires, Andre Santana, Adelardo Medeiros. — 2008. — 01. — Pp. 1607–1611.

9 Schödl, A. Controlled Animation of Video Sprites. [Текст] / A. Schödl, I. Essa // ACM/Eurographics Symposium on Computer Animation (SCA) / ACM SIGGRAPH. — San Antonio, TX, USA: ACM Press, 2002. — August.

10 Savchynskyy, Bogdan. Discrete Graphical Models — An Optimization Perspective. [Текст] / Bogdan Savchynskyy // *Foundations and Trends® in Computer Graphics and Vision*. — 2019. — Vol. 11, no. 3-4. — Pp. 160–429.

11 Maximum Flows by Incremental Breadth-First Search. [Текст] / Andrew Goldberg, Sagi Hed, Haim Kaplan et al. // 19th European Symposium on Algorithms (ESA 2011). — Springer Verlag, 2011. — January.

12 Real-valued fast Fourier transform algorithms. [Текст] / H. Sorensen, D. Jones, M. Heideman, C. Burrus // *IEEE Transactions on Acoustics, Speech, and Signal Processing*. — 1987. — Vol. 35, no. 6. — Pp. 849–863.

13 Winograd, Shmuel. On computing the Discrete Fourier Transform. [Текст] / Shmuel Winograd // *Proceedings of the National Academy of Sciences*. — 1976. — Vol. 73, no. 4. — Pp. 1005–1006.

14 Team, Python Development. Learning Python: Crash Course Tutorial. [Текст] / Python Development Team, G. Van Rossum. — Medina University Press International, 2020.

15 Lam, Siu Kwan. Numba: A LLVM-Based Python JIT Compiler. [Текст] / Siu Kwan Lam, Antoine Pitrou, Stanley Seibert // Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. — LLVM '15. — New York, NY, USA: Association for Computing Machinery, 2015. — 6 pp.

16 Cook, Shane. CUDA Programming: A Developer's Guide to Parallel Computing with GPUs. [Текст] / Shane Cook. — 1st edition. — San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012.