

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

«До захисту допущено»

В.о. завідувача кафедри

_____ Микола ГРАЙВОРОНСЬКИЙ

“ ____ ” 2020 р.

Дипломна робота
на здобуття ступеня бакалавра

зі спеціальності: 113 Прикладна математика

на тему: Задача стереозору в умовах повільного надходження даних

Виконав (-ла): студент (-ка) 4-го курсу, групи ФІ-61
(шифр групи)

Кириленко Іван Андрійович

(прізвище, ім'я, по батькові)

(підпис)

Керівник доцент кафедри ІБ, к.ф.-м.н. Орєхов О. А.

(посада, науковий ступінь, вчене звання, прізвище та ініціали) —

(підпис)

Рецензент

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) —

(підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Київ – 2020 року

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

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

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри

_____ Микола ГРАЙВОРОНСЬКИЙ
(підпис)

«____» 2020 р.

**ЗАВДАННЯ
на дипломну роботу студенту**

(прізвище, ім'я, по батькові)

1. Тема роботи Задача стереозору в умовах повільного надходження даних

_____ ,
керівник роботи доцент кафедри ІБ, к.ф.-м.н. Орехов О. А. _____ ,

_____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «____» 2020 р. №

2. Термін подання студентом роботи 10 червня 2020 р.

3. Вихідні дані до роботи _____

4. Зміст роботи _____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____

6. Дата видачі завдання _____

Календарний план

Студент

(підпис)

(ініціали, прізвище)

Керівник роботи

(підпис)

(ініціали, прізвище)

РЕФЕРАТ

Обсяг роботи 47 сторінки, 15 ілюстрацій, 4 додатки, 10 джерел.

Об'єкт дослідження — прихована модель Маркова.

Предмет дослідження — ефективна оцінка прихованих параметрів моделі Маркова.

У роботі розглянуто задачу бінокулярного стереозору для випадку повільного надходження даних. Розроблено алгоритми для ефективного розв'язання задачі стереозору для різних випадків повільного надходження даних. Розроблена програмна імплементація запропонованих алгоритмів на мові C++. Порівняно час роботи алгоритмів після надходження всіх даних для різних випадків та характерів надходження цих даних. Проведено аналіз отриманих результатів.

КЛЮЧОВІ СЛОВА: КОМП'ЮТЕРНИЙ ЗІР, БІНОКУЛЯРНИЙ СТЕРЕОЗІР, ОНЛАЙН АЛГОРИТМИ, РОЗПІЗНАВАННЯ ОБРАЗІВ.

ABSTRACT

This thesis contains 47 pages, 15 figures and 10 references. The object of this study is Markov's hidden model. The subject of this study is effective estimation of Markov's model hidden parameters.

This thesis examines stereovision problem for slowly incoming data. Effective algorithms for solving this problem are developed and implemented in C++. Run time of algorithms are compared for different data transmission cases. The analysis of acquired results is conducted.

COMPUTER VISION, BINOCULAR STEREOSCOPE, ONLINE ALGORITHMS, PATTERN RECOGNITION.

ЗМІСТ

Вступ	7
1 Теоретичні відомості	9
1.1 Знаходження глибини методом бінокулярного паралаксу	9
1.2 Оффлайн алгоритм пошуку скалярного поля зсувів	12
Висновки до першого розділу	16
2 Online алгоритм пошуку скалярного поля зсувів	17
2.1 Постановка задачі	17
2.2 Поступове упорядковане надходження даних	18
2.3 Неупорядковане надходження даних при одному відомому зображені	21
Висновки до другого розділу	28
3 Практичні результати	29
3.1 Умови тестування	29
3.2 Поступове упорядковане надходження даних	30
3.3 Неупорядковане надходження даних при одному відомому зображені	31
Висновки до третього розділу	32
Висновки	33
Перелік посилань	34
Додаток А	36
Додаток Б	39
Додаток В	43
Додаток Г	46

ВСТУП

Задача бінокулярного стереозору — одна з актуальних проблем у сфері комп’ютерного зору. Це метод оцінки відстані від камери до об’єктів шляхом порівняння пари їх знімків, зроблених під різними кутами.

Від доповненої реальності[1] до автономної навігації[2] — задачі стереозору мають багато застосувань в сучасному світі. Будучи менш точними, ніж лазерне сканування, методи стерео зору часто використовуються для отримання топографічних мап через те, що дозволяють охопити одним знімком великий регіон земної поверхні.

В цій роботі розглядаються алгоритми пошуку одновимірної карти зсувів, що на вхід приймають одновимірне ректифіковане зображення. Такі алгоритми дають непогане уявлення про глибину сцени (рис. 11), будучи при цьому відносно швидкими.

Актуальність роботи. Offline алгоритми розв’язання задачі стереозору можуть почати свою роботу тільки після надходження всіх даних. Проте, з розвитком мережевих технологій все частіше трапляється, що необхідна інформація завантажується безпосередньо з інтернету по мірі необхідності, а не зберігається локально. Також можлива ситуація, коли дані для обрахунку відправляються до обчислювального центру. Сучасні дата-центри мають дуже швидкі та надійні канали доступу до мережі, цього не можна сказати про їх клієнтів. Тож до часу роботи самого алгоритму буде додаватися ще й час надходження всіх даних. Цю проблему можна вирішити, використовуючи алгоритми, які можуть починати обрахунки вже після надходження першої частини даних, зменшуючи тим самим сумарний час обробки даних.

Об’єкт дослідження — прихована модель Маркова.

Предмет дослідження — ефективна оцінка прихованих параметрів моделі Маркова.

Мета дослідження. Розробка online алгоритму для задачі стереозору.

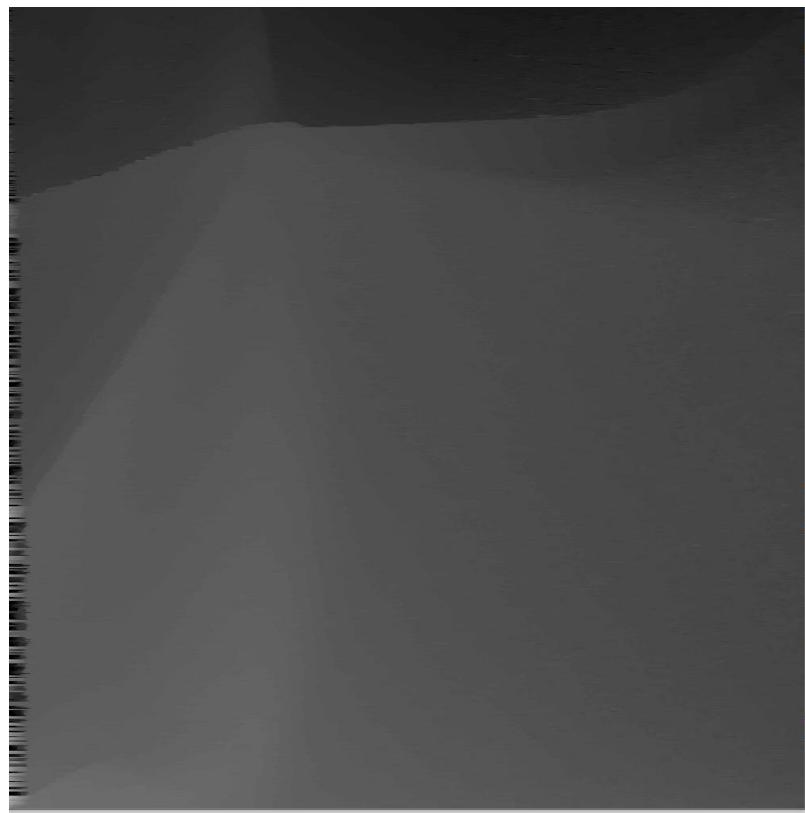


Рисунок 1 — Результат роботи одновимірного алгоритму

Практичне значення результатів. Розроблений алгоритм можна використовувати для ефективного вирішення задачі стереозору в умовах повільного надходження даних.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Перший розділ присвячено опису бінокулярного стереозору, постановці та розв'язку задачі стереозору для випадку повних даних.

1.1 Знаходження глибини методом бінокулярного паралаксу

Розглянемо принцип роботи бінокулярного стереозору.

1.1.1 Будова камери

Сучасні цифрові фото та відео камери мають куди більш складну будову, ніж їх попередники. Для подальших викладок нам буде достатньо розглянути роботу найпростішої можливої камери — камери-обскури (від лат. *camera obscura* — «темна кімната») (рис. 1.1). Це просто темне приміщення з одним малим отвором (його називають *точковою діафрагмою*), через який на протилежну стіну проектується перевернуте зображення предметів ззовні. Для нас протилежна стіна — матриця цифрової камери.

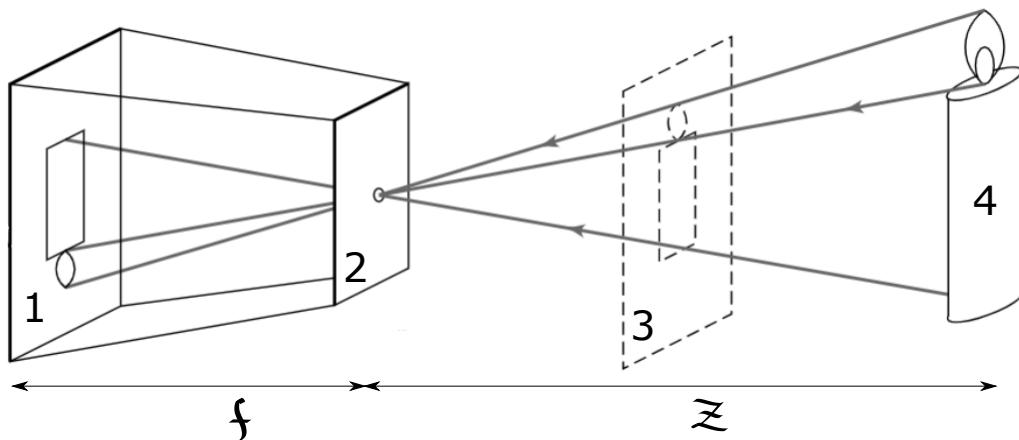


Рисунок 1.1 — Будова камери-обскури.

1 — результатуюче зображення, 2 — отвір, 3 — віртуальне зображення, 4 — об'єкт

1.1.2 Пошук відстані методом бінокулярного паралаксу

В силу оптичних особливостей результуюче зображення в камері-обскурі буде перевернутим. Тому замість нього будемо розглядати віртуальне зображення, що знаходиться на відстані f з права від діафрагми, але вже не перевернуте (рис. 1.1). Це ніяк не вплине на результат, але спростить подальші викладки. Покажемо тепер, як за допомогою двох таких камер можна знайти абсолютну відстань до об'єкта. Камери розташовані паралельно одна одній. Тоді маємо таку конфігурацію (рис. 1.2). Тут L та R — діафрагми лівої та правої камери, O_L, O_R — центри віртуальних зображень, f — фокусна від-

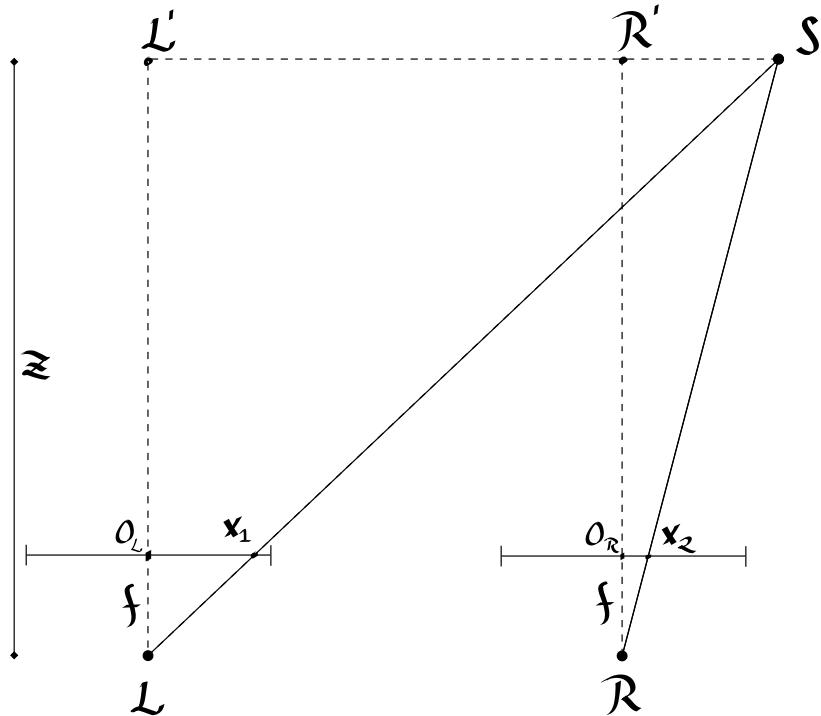


Рисунок 1.2 — Геометрія розташування камер.

стань, z — відстань до об'єкта, Δ — відстань між камерами, x_1 та x_2 — координати проекцій об'єкта на лівому та правому зображеннях, S — об'єкт. Тоді

з подібності трикутників LSL', Lx_1O_L та RSR', Rx_2O_R маємо

$$\begin{cases} \frac{f}{x_1} = \frac{z}{\Delta+x}, \\ \frac{f}{x_2} = \frac{z}{x}. \end{cases}$$

Виразивши Δ з першого рівняння та x з другого, отримаємо

$$\begin{cases} \Delta = \frac{z \cdot x_1}{f} - x, \\ x = \frac{z \cdot x_2}{f}. \end{cases}$$

Підставимо x в перше рівняння і отримаємо вираз для z

$$z = \frac{\Delta \cdot f}{(x_1 - x_2)}.$$

Оскільки Δ і f не залежать від розташування об'єкта, то відстань до нього буде залежати тільки від різниці $x_1 - x_2$, тобто тільки від зсуву проекцій об'єкту між двома знімками. Тож для знаходження глибини сцени нам необхідно знайти зсув для кожного пікселя між зображеннями.

1.2 Оффлайн алгоритм пошуку скалярного поля зсувів

1.2.1 Постановка задачі

Нехай n — довжина зображення, $I = \{1, 2, \dots, n\}$ — множина координат пікселів, $D = \{0, \dots, D_{max}\}$ — множина зсувів, де D_{max} — значення максимального зсуву, що обирається емпіричним шляхом. Ліве і праве зображення задамо як функції

$$\mathcal{L} : I \rightarrow \mathcal{C},$$

$$\mathcal{R} : I \rightarrow \mathcal{C}.$$

Тобто, $\mathcal{L}(i)$ — інтенсивність i -го пікселя на лівому зображені, а $\mathcal{R}(i)$ — інтенсивність i -го пікселя на правому зображені, а \mathcal{C} — множина кольорів зображення.

Для вирішення задачі кожному пікселю лівого зображення потрібно знайти відповідний йому піксель на правому зображені (1.3). Введемо сюр'єктивне відображення $d : I \rightarrow D$, де $d(i)$ — такий зсув $d_i \in D$ для пікселя з номером i лівого зображення, що піксель правого зображення з номером $i - d_i$ відповідає йому. Проте, не будь-яка пара пікселів може знаходитися у

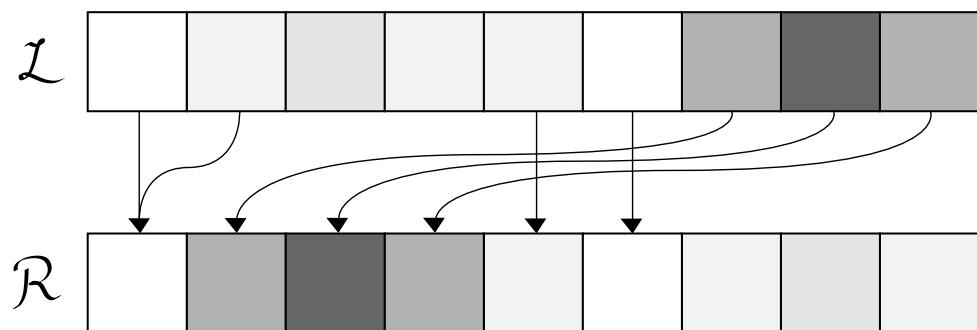


Рисунок 1.3 — Пошук відповідних пікселів

відповідності — пікселю з номером i на лівому зображені можуть відповідати

пікселі правого зображення з номером j , для яких $i \geq j$. Переконатися в цьому можна, тримаючи перед собою олівець, та по черзі закриваючи то праве, то ліве око. Для правого ока олівець буде знаходитися лівіше ніж для лівого

Треба знайти таку послідовність $\bar{d} \in D^n$, яка мінімізує штрафну функцію

$$\mathcal{W}(\bar{d}) = \sum_{i=1}^n h(i, d_i) + \sum_{i=1}^{n-1} g(d_i, d_{i+1}), \quad (1.1)$$

де $h(i, d_i)$ відповідає за схожість кольору пікселів, а $g(d_i, d_{i+1})$ за гладкість поля зсувів.

1.2.2 Представлення задачі як пошук найкоротшого шляху у графі

Зведемо задачу до пошуку послідовності \bar{d} , що мінімізує штрафну функцію (1.1), як пошук найкоротшого шляху через орієнтований зважений граф $G = \langle \mathcal{V}, \mathcal{E} \rangle$ (рис. 1.4), [3]. Його множина вершин

$$\mathcal{V} = \{\sigma(i, d) \mid i \in I, d \in D\} \cup \{S, E\}.$$

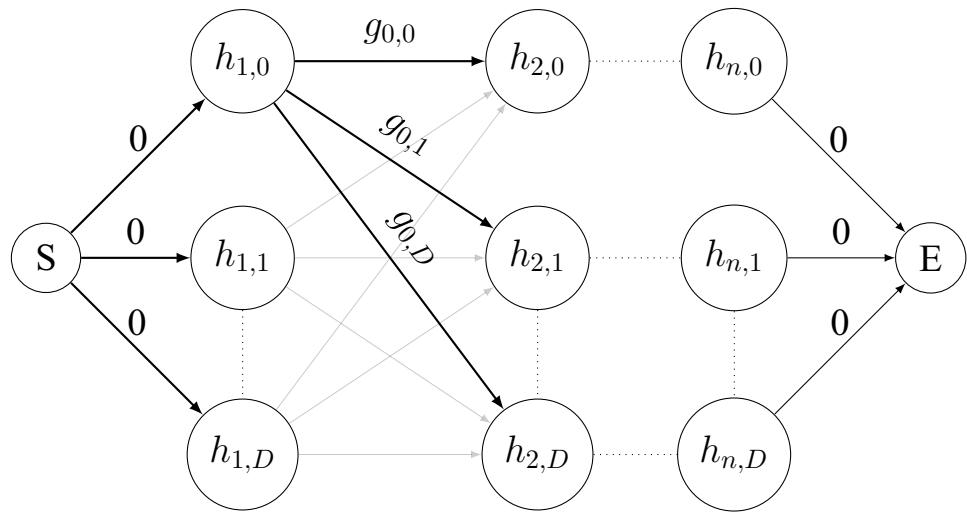
Введемо функцію ваг вершин

$$v : \mathcal{V} \setminus \{S, E\} \rightarrow \mathbb{R},$$

$$v(\sigma(i, d)) = h(i, d), \quad \forall i \in I, d \in D.$$

Множина його ребер

$$\mathcal{E} = \bigcup_{d \in D} \{ \langle S, \sigma(1, d) \rangle \} \cup \bigcup_{d \in D} \{ \langle \sigma(n, d), E \rangle \} \cup \bigcup_{\substack{i=1..n-1 \\ d \in D \\ d' \in D}} \{ \langle \sigma(i, d), \sigma(i+1, d') \rangle \}.$$

Рисунок 1.4 – Граф G

Введемо функцію ваг ребер

$$w : \mathcal{E} \rightarrow \mathbb{R},$$

де $w(v_1, v_2)$ – вага ребра $\langle v_1, v_2 \rangle$, $\langle v_1, v_2 \rangle \in \mathcal{E}$,

$$w(\sigma(i, d), \sigma(i + 1, d')) = g(d, d'), \quad \forall d, d' \in D, i \in I,$$

$$w(S, \sigma(1, d)) = 0, \quad \forall d \in D,$$

$$w(\sigma(n, d), E) = 0, \quad \forall d \in D.$$

Послідовність \bar{d} – послідовність вершин, через які проходить найкоротший шлях з S в E , мінімізує штрафну функцію $\mathcal{W}(\bar{d})$ (1.1).

1.2.3 Пошук найкоротшого шляху у графі

Позначимо довжину найкоротшого шляху з вершини S в вершину $\sigma(i, d)$ як $f_i(d)$. Тоді

$$\begin{aligned} f_1(d) &= h(1, d), \forall d \in D \\ f_2(d) &= \min_{d' \in D} \left(f_1(d') + g(d', d) \right) + h(2, d), \forall d \in D \\ &\vdots \\ f_i(d) &= \min_{d' \in D} \left(f_{i-1}(d') + g(d', d) \right) + h(i, d), \forall d \in D. \end{aligned}$$

Тоді елементи послідовності \bar{d} знаходимо за формулами (якщо кращих елементів декілька, вибираємо будь-який)

$$\begin{aligned} d_n &\in \arg \min_{d' \in D} (f_n(d')), \\ d_i &\in \arg \min_{d' \in D} (f_i(d') + g(d', d_{i+1})), \quad i = \overline{n-1, 1}. \end{aligned}$$

Висновки до первого розділу

У першому розділі було розглянуто основні теоретичні відомості необхідні для подальшого розуміння та постановки одновимірної задачі стереозору. Був описаний принцип роботи бінокулярного стереоэффекту та його математичне представлення для машинного зору. Також була розглянута постановка одновимірної задачі стереозору для випадку повних даних та зведення цієї задачі до задачі пошуку найкоротшого шляху на орієнтованому зваженому графі спеціального вигляду. Для задачі пошуку найкоротшого шляху був наведений алгоритм її вирішення, що використовує динамічне програмування.

Постановка одновимірної задачі стереозору для випадку повних даних та її зведення до задачі пошуку найкоротшого шляху на графі, що наведена в цьому розділі, буде узагальнена на випадки повільного надходження даних у наступному розділі.

2 ONLINE АЛГОРИТМ ПОШУКУ СКАЛЯРНОГО ПОЛЯ ЗСУВІВ

Другий розділ присвячено постановці задачі стереозору для випадку по-вільного надходження даних. Пропонуються алгоритми розв'язку задачі стереозору, які для такого випадку є більш ефективними, ніж алгоритми наведені в розділі 1.

Метод, описаний в підрозділі 1.2, потребує наявності всіх даних до початку обчислень. В ситуації, коли дані надходять повільно, цей метод не є ефективним, адже доводиться чекати надходження всіх даних і тільки потім починати обчислення, бо ми не можемо розрахувати ваги деяких вершин. Замість цього можна проводити деякі розрахунки з частиною даних, що вже надійшла, цим самим зменшивши час роботи алгоритму після отримання всіх даних.

2.1 Постановка задачі

Як і в частині 1.2, n — довжина зображення, $I = \{1, 2, \dots, n\}$ — множина координат пікселів, $D = \{0, \dots, D_{max}\}$ — множина зсувів, D_{max} — значення максимального зсуву. Проте ліве і праве зображення вже задаємо як функції

$$\mathcal{L} : I \rightarrow \mathcal{C} \cup \{\varepsilon\},$$

$$\mathcal{R} : I \rightarrow \mathcal{C} \cup \{\varepsilon\}.$$

Якщо i -й піксель лівого зображення вже надійшов, то $\mathcal{L}(i)$ — інтенсивність i -го пікселя на лівому зображені. Якщо ще не надійшов, то $\mathcal{L}(i) = \varepsilon$. Так само, як і у 1.2, нам потрібно знайти таку послідовність $\bar{d} \in D^n$, яка мінімізує штрафну функцію (1.1).

Цю задачу ми теж представимо як пошук найкоротшого шляху на графі

$G = \langle \mathcal{V}, \mathcal{E} \rangle$. Його множина вершин, множина ребер та ваги ребер такі ж, як і в підрозділі 1.2.2. Але ваги вершин

$$v(\sigma(i, d)) = \begin{cases} h(i, d), & \text{якщо } \mathcal{L}(i) \neq \varepsilon, \\ \infty, & \text{якщо } \mathcal{L}(i) = \varepsilon. \end{cases}$$

Будемо називати вершину $\sigma(i, d)$ **відкритою**, якщо $\mathcal{L}(i) \neq \varepsilon$, інакше назовемо її **закритою**.

Отже, поки нам не надійшли всі дані, ми не можемо знайти таку послідовність $\bar{d} \in D^n$, що мінімізує штрафну функцію

$$\omega(\bar{d}) = \sum_{i=1}^n h(i, d_i) + \sum_{i=1}^{n-1} g(d_i, d_{i+1}).$$

Але, щоб зменшити загальний час роботи алгоритму, ми можемо проводити деякі розрахунки з тими даними, що вже надійшли.

2.2 Поступове упорядковане надходження даних

Нехай на початку нам не відомі жодні пікселі обох зображень (рис. 2.1), та існує стохастичний автомат, що на кожному кроці $t \in I$ генерує пари \langle

$$\mathcal{L} \quad \boxed{\text{X} \mid \text{X} \mid \text{X}}$$

$$\mathcal{R} \quad \boxed{\text{X} \mid \text{X} \mid \text{X}}$$

Рисунок 2.1 – Немає даних

c_L, c_R — кольори наступних пікселів лівого та правого зображень ($c_L, c_R \in \mathcal{C}$). Автомат завершує свою роботу після генерації n пар кольорів. Тоді на

кожному кроці t нам буде відкриватися стовпчик вершин $h(t, d), \forall d \in D, t \in I$ (рис. 2.4).

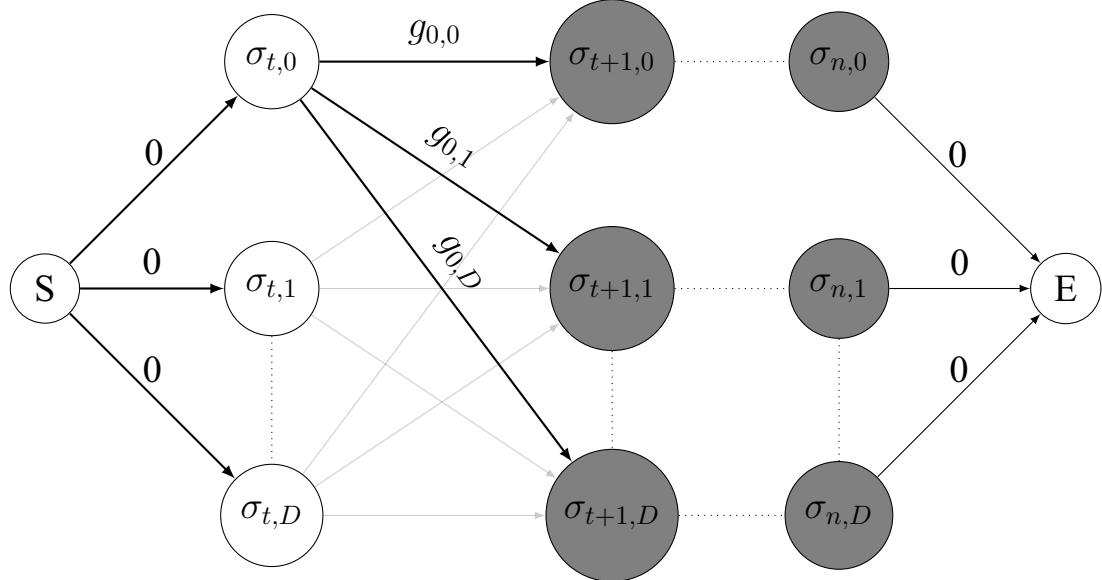


Рисунок 2.2 – Граф G

Перед тим, як навести сам алгоритм, неформально опишемо ідею його роботи. Припустимо, нам відомо, що найкоротший шлях з S в E проходить через вершину $\sigma(t + 1, d), d \in D$. Тоді, якщо всі вершини $\sigma(t, d), \forall d \in D$ відкриті, ми можемо точно сказати, через яку з них буде проходити найкоротший шлях. Таку вершину назовемо *оптимальною попередньою* для вершини $\sigma(+1t, d)$. На жаль, ми не знаємо, через яку вершину $\sigma(t + 1, d), d \in D$ проходить найкоротший шлях, тож нам необхідно для кожної такої вершини знайти її оптимальну попередню вершину. Зате після цього нам вже не потрібні вершини $\sigma(t, d), \forall d \in D$, і ми можемо їх відкинути разом з їх ребрами, замінивши на D_{max} нових ребер. Цим самим ми зменшимо загальну кількість ребер на D_{max}^2 .

Нехай $G^0 = \langle \mathcal{V}^0, \mathcal{E}^0 \rangle = \langle \mathcal{V}, \mathcal{E} \rangle$. Нехай також $T = \{1, \dots, n - 1\}$ – множина ітерацій, $s : T \times D \rightarrow R$, $s^0(d) = 0, \forall d \in D$. Для відновлення послідовності \bar{d} введемо функцію *оптимальних вершин* $\hat{\mathcal{P}} : I \times D \rightarrow D$.

$$p_{1,d} = 0, \forall d \in D.$$

Тільки-но автомат на кроці $t \in T$ генерує нову пару $\langle cL, cR \rangle$, шукаємо новий граф G^t :

1) $\forall d \in D :$

$$s^t(d) = \min_{d' \in D} (s^{(t-1)}(d') + h(t, d') + g(d', d)).$$

$$p_{t+1,d} \in \arg \min_{d' \in D} (s^{(t-1)}(d') + h(t, d') + g(d', d)),$$

(якщо кращих елементів декілька, оберемо будь-який з них).

2) $\mathcal{V}^t = \mathcal{V}^{t-1} \setminus \{\sigma(t, d) \mid d \in D\}$.

3) Позначимо множину $\bigcup_{d \in D} \langle S, \sigma(t, d) \rangle$ як \mathcal{A} ,

множину $\bigcup_{\substack{d \in D \\ d' \in D}} \langle \sigma(t, d), \sigma(t+1, d') \rangle$ як \mathcal{B} ,

а множину $\bigcup_{d \in D} \langle S, \sigma(t+1, d) \rangle$ як \mathcal{C} . Тоді

$$\mathcal{E}^t = (\mathcal{E}^{t-1} \setminus (\mathcal{A} \cup \mathcal{B})) \cup \mathcal{C}.$$

Вага ребра $\langle S, \sigma(t+1, d) \rangle = s^t(d)$

4) $G^t = \langle \mathcal{V}^t, \mathcal{E}^t \rangle$

Коли ж на кроці n автомат згенерує останні пікселі зображення, нам залишиться тільки знайти

$$d_n \in \arg \min_{d' \in D} \{s^{(n-D_{max}-1)}(d') + h(n, d')\},$$

та відновити послідовність \bar{d} через матрицю $\hat{\mathcal{P}}$

$$d_i = p_{i+1, d_{i+1}}, \quad i = \overline{n-1, 1}.$$

Кожен новий граф G^t матиме на $(D_{max})^2$ менше ребер, ніж граф G^{t-1} .

Таким чином, на момент приходу останніх пікселів, нам треба буде опрацюва-

ти лише D_{max} ребер. Використання ж offline алгоритму потребує опрацювання $(n - 1) \cdot D_{max}^2$ ребер після надходження всіх даних.

2.3 Неупорядковане надходження даних при одному відомому зображені

Нехай на початку нам відомі всі пікселі лише одного з зображень. Без втрати загальності припустимо, що нам відомо ліве зображення (рис. 2.3).

\mathcal{L}	✓	✓	✓	✓	✓	✓	✓	✓	✓
\mathcal{R}	✗	✗	✗	✗	✗	✗	✗	✗	✗

Рисунок 2.3 – Немає даних

Введемо множину ітерацій $T = \{1, \dots, n\}$, та стохастичний автомат, що на кожному кроці $t \in T$ генерує пари $\langle i, c \rangle$, де c – інтенсивність пікселя з координатою i для невідомого зображення ($i \in I, c \in \mathbb{R}$). Автомат ніколи не генерує одне й те саме i більше одного разу, та завершує свою роботу після генерації останнього пікселя. Тоді на кожному кроці t нам буде відкриватися стовпчик вершин $h(i, d), \forall d \in D$ (рис. 2.4).

Перед тим, як навести сам алгоритм, неформально опишемо ідею його роботи. Припустимо, нам відомо, що найкоротший шлях з S в E проходить через вершини $\sigma(i + 1, d')$ та $\sigma(i - 1, d'')$, $d', d'' \in D, i \in I$. Коли вершини $\sigma(i, d), \forall d \in D$ стають відкритими, ми можемо точно сказати, через яку з них буде він буде проходити. Проте, d' та d'' нам не відомі, тому нам необхідно для кожної пари вершини $\sigma(i - 1, d')$ та $\sigma(i + 1, d'')$ знайти для них *оптимальну проміжну вершину*. Після цього вершини $\sigma(i, d), \forall d \in D$ нам вже не потрібні, і ми можемо їх відкинути разом з їх ребрами, замінивши на D_{max}^2 нових ребер,

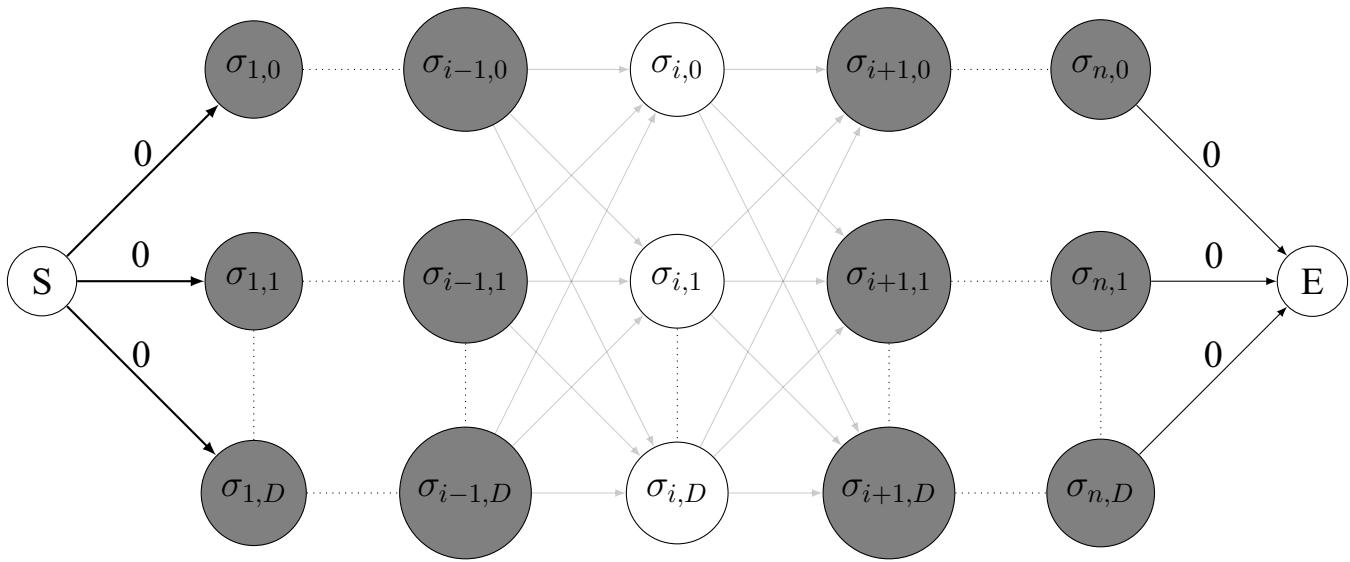


Рисунок 2.4 – Граф G (відкриті вершини зафарбовані білим).

зменшивши загальну кількість ребер теж на D_{max}^2 .

Праве зображення задамо так, щоб знати які пікселі правого зображення відкриті на кроці t .

$$\mathcal{R} : I \times T \cup \{0\} \rightarrow \mathcal{C} \cup \{\varepsilon\},$$

$$\mathcal{R}^0(i) = \varepsilon, \quad \forall i \in I.$$

$\mathcal{R}^t(i)$ – Колір i -го пікселя правого зображення на кроці t . Ліве зображення задамо як і раніше

$$\mathcal{L} : I \rightarrow \mathcal{C} \cup \{\varepsilon\}.$$

Для відстежування порядку надходження пікселів правого зображення введемо функцію

$$o : T \rightarrow I,$$

де o^t – номер пікселя, що надійшов на кроці t . Також введемо функції *сусідніх оптимальних вершин*

$$r : I \rightarrow I \cup \{0\},$$

$$l : I \rightarrow I \cup \{0\},$$

де $r(i)$ — номер найближчого правого, а $l(i)$ — номер найближчого лівого закритого стовпчика вершин на момент надходження пікселя i . Якщо на момент надходження пікселя i , всі пікселі правіше від нього відкриті, то $r(i) = 0$. Якщо всі лівіше від нього відкриті, то $l(i) = 0$. Для знаходження сусідніх стовпчиків закритих вершин у графі G^t вводимо

$$j : I \times T \rightarrow I,$$

$$j^t_i = \sum_{k=1}^i \mathbb{1}(\mathcal{R}^t(k) = \varepsilon),$$

Введемо граф G^t як $\langle \mathcal{V}^t, \mathcal{E}^t \rangle$, $G^0 = \langle \mathcal{V}^0, \mathcal{E}^0 \rangle$. Множина вершин \mathcal{V}^0 та множина ребер \mathcal{E}^0 співпадають з множинами \mathcal{V} та \mathcal{E} з підрозділу 1.2.2. Введемо функцію ваг ребер та ваг вершин

$$w : T \times \mathcal{E} \rightarrow \mathbb{R},$$

$$v : T \times \mathcal{V} \rightarrow \mathbb{R},$$

де $w^t(\sigma_1, \sigma_2)$ — вага ребра $\langle \sigma_1, \sigma_2 \rangle$, а $v^t(\sigma_1)$ — вага вершини σ_1 у графі G^t ($\sigma_1 \in \mathcal{V}^t, \langle \sigma_1, \sigma_2 \rangle \in \mathcal{E}^t$).

Для відновлення послідовності \bar{d} введемо функцію оптимальних проміжних вершин

$$\hat{\mathcal{P}} : I \times D^2 \rightarrow D.$$

На кожній ітерації t будемо замінювати граф G^{t-1} на новий граф G^t , що матиме менше ребер та вершин.

Тільки-но автомат на кроці $t \in \{1, \dots, n-1\}$ генерує нову пару $*i, c>*$, будуємо новий граф G^t

- 1) Запам'ятовуємо номер пікселя, що надійшов на кроці t та знаходимо стовпчик вершин у графі G^t , що відповідає пікселю i

$$o(t) = i$$

$$\begin{aligned} j^t_i &= \sum_{k=1}^i \mathbf{1}(\mathcal{R}^t(k) = \varepsilon), \\ \hat{n}^t &= \sum_{k=1}^n \mathbf{1}(\mathcal{R}^t(k) = \varepsilon). \end{aligned}$$

- 2) Відкидаємо останній стовпчик вершин у графі G^t

$$\mathcal{V}^t = \mathcal{V}^{t-1} \setminus \{\sigma(\hat{n}^{t-1}, d') \mid \forall d \in D\}.$$

- 3) Переписуємо ваги вершин

$$v^t(\sigma(k, d)) = v^{t-1}(\sigma(k, d)), \quad 1 \leq k < j^t_i, \quad \forall d \in D,$$

$$v^t(\sigma(k, d)) = v^{t-1}(\sigma(k+1, d)), \quad j^t_i < k \leq \hat{n}^{t-1} - 1, \quad \forall d \in D.$$

- 4) Позначимо множину $\bigcup_{d', d'' \in D} \{<\sigma(\hat{n}^{t-1} - 1, d'), \sigma(\hat{n}^{t-1}, d'')>\}$ як \mathcal{A} , множину $\bigcup_{d' \in D} \{<\sigma(\hat{n}^{t-1}, d'), e>\}$ як \mathcal{B} , а множину $\bigcup_{d' \in D} \{<\sigma(\hat{n}^{t-1} - 1, d'), e>\}$ як \mathcal{C} . Тоді

$$\mathcal{E}^t = \left(\mathcal{E}^{t-1} \setminus (\mathcal{A} \cup \mathcal{B}) \right) \cup \mathcal{C}.$$

5) Якщо $j^t_i = 1$, залишаємо без змін вагу ребер

$$w^t(\sigma(k, d), \sigma(k + 1, d')) = w^{t-1}(\sigma(k, d), \sigma(k + 1, d')), \quad 2 \leq k < \hat{n}^{t-1},$$

$$w^t(\sigma(\hat{n}^t, d), e) = w^{t-1}(\sigma(\hat{n}^t, d), e).$$

Запам'ятуємо сусідні до i закриті пікселі

$$r^i = \min\{k \in I \mid k > i, \mathcal{R}^t = \varepsilon\},$$

$$l^i = 0.$$

Обчислюємо вагу нових ребер, та знаходимо оптимальні вершини

$$w^t(s, \sigma(1, d)) = \min_{\hat{d} \in D} (w^{t-1}(s, \sigma(1, \hat{d})) + v^{t-1}(\sigma(1, \hat{d})) + w^{t-1}(\sigma(1, \hat{d}), \sigma(2, d))),$$

$$p^i_{d', 0} \in \arg \min_{\hat{d} \in D} (w^{t-1}(s, \sigma(1, \hat{d})) + v^{t-1}(\sigma(1, \hat{d})) + w^{t-1}(\sigma(1, \hat{d}), \sigma(2, d))).$$

6) Якщо $j^t_i = \hat{n}^t$, залишаємо без змін вагу ребер

$$w^t(\sigma(k, d), \sigma(k + 1, d')) = w^{t-1}(\sigma(k, d), \sigma(k + 1, d')), \quad 1 \leq k < \hat{n}^t,$$

$$w^t(s, \sigma(1, d)) = w^{t-1}(s, \sigma(1, d)).$$

Запам'ятуємо сусідні до i закриті пікселі

$$r^i = 0,$$

$$l^i = \max\{k \in I \mid k < i, \mathcal{R}^t = \varepsilon\}.$$

Обчислюємо вагу нових ребер, та знаходимо оптимальні вершини

$$\begin{aligned}
w^t(\sigma(\hat{n}^t, d), e) &= \\
&= \min_{\hat{d} \in D} (w^{t-1}(\sigma(\hat{n}^{t-1} - 1, d), \sigma(\hat{n}^{t-1}, \hat{d})) + v^{t-1}(\sigma(\hat{n}^{t-1}, \hat{d})) + \\
&\quad + w^{t-1}(\sigma(\hat{n}^{t-1}, \hat{d}), e)), \\
p^i_{d', 0} &\in \\
&\in \arg \min_{\hat{d} \in D} (w^{t-1}(\sigma(\hat{n}^{t-1} - 1, d), \sigma(\hat{n}^{t-1}, \hat{d})) + v^{t-1}(\sigma(\hat{n}^{t-1}, \hat{d})) + \\
&\quad + w^{t-1}(\sigma(\hat{n}^{t-1}, \hat{d}), e)).
\end{aligned}$$

7) Якщо $1 < j^t_i < \hat{n}^t$,

$$\begin{aligned}
w^t(\sigma(k, d), \sigma(k + 1, d')) &= w^{t-1}(\sigma(k, d), \sigma(k + 1, d')), \quad 1 \leq k < j^t_i - 1, \\
w^t(\sigma(k, d), \sigma(k + 1, d')) &= w^{t-1}(\sigma(k, d), \sigma(k + 1, d')), \quad j^t_i \leq k < \hat{n}^t, \\
w^t(s, \sigma(1, d)) &= w^{t-1}(s, \sigma(1, d)). \\
w^t(\sigma(\hat{n}^t, d), e) &= w^{t-1}(\sigma(\hat{n}^t, d), e).
\end{aligned}$$

Запам'ятовуємо сусідні до i закриті пікселі

$$r^i = \min\{k \in I \mid k > i, \mathcal{R}^t = \varepsilon\},$$

$$l^i = \max\{k \in I \mid k < i, \mathcal{R}^t = \varepsilon\}.$$

Обчислюємо вагу нових ребер

$$\begin{aligned}
 w^t(\sigma(j_i^t - 1, d), \sigma(j_i^t, d')) = \min_{\hat{d} \in D} \{ & \\
 w^{t-1}(\sigma(j_i^t - 1, d), \sigma(j_i^t, d')) + & \\
 + v^{t-1}(\sigma(j_i^t, \hat{d})) + & \\
 + w^{t-1}(\sigma(j_i^t, d), \sigma(j_i^t + 1, d')) \},
 \end{aligned}$$

та знаходимо оптимальні вершини

$$\begin{aligned}
 p_{d',0}^i \in \arg \min_{\hat{d} \in D} \{ & \\
 w^{t-1}(\sigma(j_i^t - 1, d), \sigma(j_i^t, d')) + & \\
 + v^{t-1}(\sigma(j_i^t, \hat{d})) + & \\
 + w^{t-1}(\sigma(j_i^t, d), \sigma(j_i^t + 1, d')) \}.
 \end{aligned}$$

Коли на кроці n автомат згенерує останню пару $< i, c >$, тоді $o(n) = i$, і нам залишається лише відновити послідовність \bar{d} через матрицю $\hat{\mathcal{P}}$. Нехай $d_0 = 0$, тоді для $(n - 1) \geq k \geq 1$

$$\begin{aligned}
 d_{o(n)} \in \arg \min_{d' \in D} \{ & s^{(n-D_{max}-1)}(d') + h(n, d') \}, \\
 d_{o(k)} = p^{o(k)}_{d_{l^{o(k)}}, d_{r^{o(k)}}}
 \end{aligned}$$

Кожен новий граф G^t матиме на $(D_{max})^2$ менше ребер, ніж граф G^{t-1} . Таким чином, на момент приходу останніх пікселів, нам треба буде опрацювати лише $2 \cdot D_{max}$ ребер. Використання ж offline алгоритму потребує опрацювання $(n - 1) \cdot (D_{max})^2$ ребер після надходження всіх даних.

Висновки до другого розділу

У другому розділі було запропоноване узагальнення постановки задачі з другого розділу на випадки повільного надходження даних. Також було розглянуто зведення одновимірної задачі стереозору для випадку повільного надходження даних, до задачі пошуку найкоротшого шляху на орієнтованому зваженому графі спеціального вигляду.

Були запропоновані спеціальні алгоритми вирішення задачі пошуку найкоротшого шляху графі, що для повільного надходження даних є більш ефективними, ніж алгоритм наведений у першому розділі.

3 ПРАКТИЧНІ РЕЗУЛЬТАТИ

У третьому розділі показані практичні результати роботи алгоритмів, запропонованих в розділі 2. Порівнюється час роботи запропонованих алгоритмів з часом роботи алгоритмів, наведених в розділі 1.

3.1 Умови тестування

Всі алгоритми були реалізовані на мові **C++**, з використанням бібліотеки **opencv** [4], для зручної роботи з зображеннями.

Алгоритми тестувалися на зображеннях з **Middlebury Stereo Datasets** [5],[6–10] з $D_{max} = 100$. В якості штрафів були вибрані

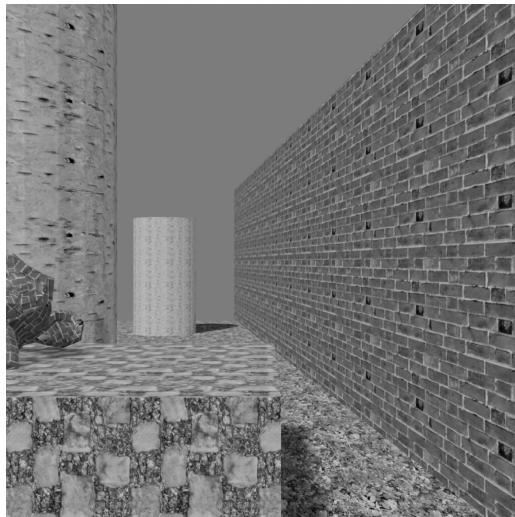
$$h(i,d) = |\mathcal{L}(i) - \mathcal{R}(i - d)|, \quad (3.1)$$

$$g(d, d') = \alpha|d - d'|, \quad (3.2)$$

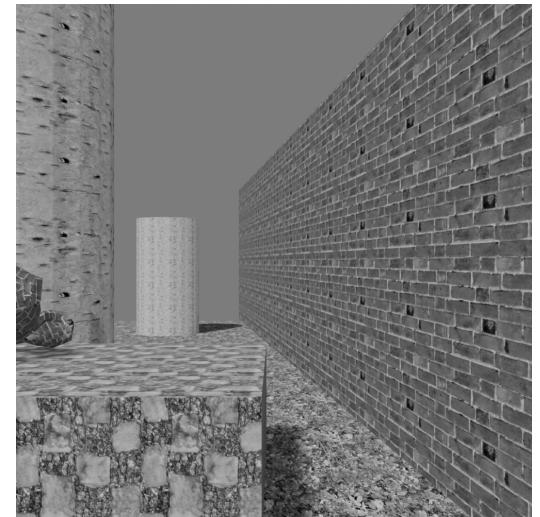
де α – коефіцієнт згладжування, що підбирається експериментально.

3.2 Поступове упорядковане надходження даних

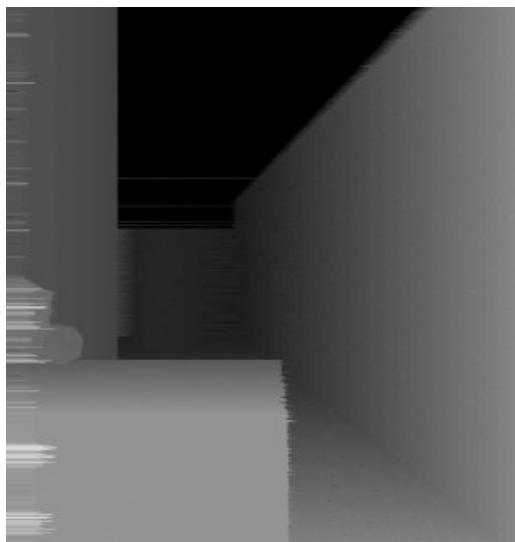
На кожній ітерації $t \in \{1, \dots, 1000\}$ відкриваються пікселі лівого та правого зображення з координатою t . Зображення 1000×1000 .



(а) Ліве зображення



(б) Праве зображення

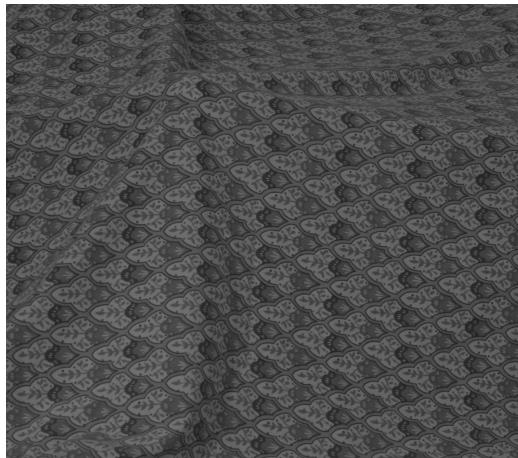


(в) Результат

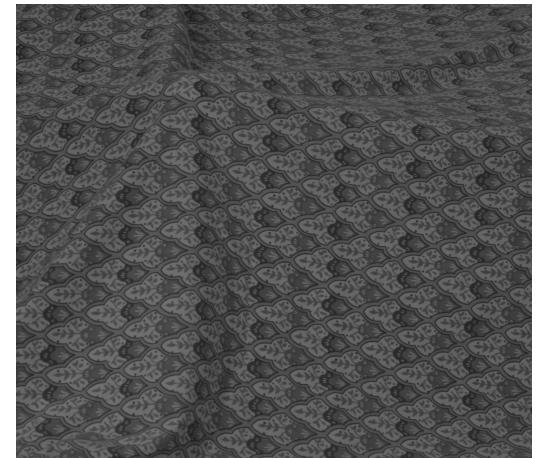
Час роботи offline алгоритму після надходження всіх даних $\mathcal{T}_{offline} = 5.6$ секунд і при відсутній затримці, і при затримці в 10^{-5} секунди. Час роботи online алгоритму після надходження всіх даних $\mathcal{T}_{online} = 6.9$ секунд при відсутній затримці, і $\mathcal{T}_{online} = 0.04$ при затримці в 10^{-5} секунди.

3.3 Неупорядковане надходження даних при одному відомому зображені

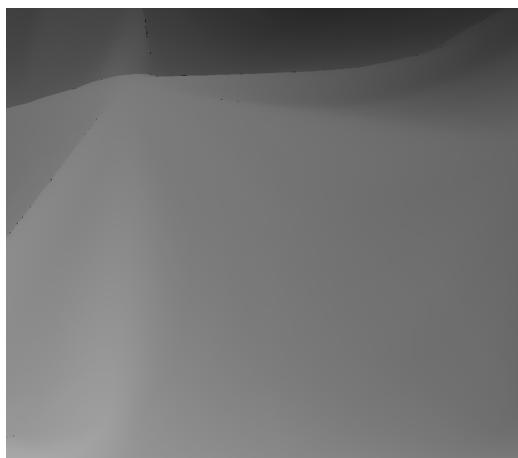
На кожній ітерації $t \in \{1, \dots, 1000\}$ відкривається випадковий піксель правого зображення, $latency$ — затримка між ітераціями. Зображення 1000×1000 .



(а) Ліве зображення



(б) Праве зображення



(в) Результат

Час роботи offline алгоритму після надходження всіх даних $\mathcal{T}_{offline} = 5.6$ секунд і при відсутній затримці, і при затримці в 10^{-5} секунди. Час роботи online алгоритму після надходження всіх даних $\mathcal{T}_{online} = 7.3$ секунд при відсутній затримці, і $\mathcal{T}_{online} = 0.06$ при затримці в 10^{-5} секунди.

Висновки до третього розділу

У третьому розділі було продемонстровано результат роботи запропонованих алгоритмів, та порівняно час їх роботи з offline алгоритмом, введеним у розділі 1.2. Була показана їх перевага для випадків повільного надходження даних.

ВИСНОВКИ

В даній роботі було розглянуто одновимірну задачу стереозору для випадку повільного надходження даних, її зведення до задачі пошуку найкоротшого шляху на орієнтованому зваженому графі спеціального вигляду та offline алгоритм розв'язання цієї задачі. Для різних випадків повільного надходження даних було розроблено окремі online алгоритми, які для цих випадків, є ефективнішими за offline алгоритм.

Було створено програмна реалізація цих алгоритмів, та проведено порівняння швидкості їх роботи для випадку повних даних, та для випадку повільного надходження даних. Практичні результати підтвердили теоретичні здогадки: для випадку повних даних offline алгоритм є більш ефективним, а для випадків повільного надходження даних більш ефективними є запропоновані online алгоритми.

ПЕРЕЛІК ПОСИЛАНЬ

- 1 Real Time Tracking using Stereo Vision for Augmented Reality / Francesco Cosco, Francesco Caruso, Fabio Bruno, Maurizio Muzzupappa. — 2008. — 01. — Pp. 613–620.
- 2 Outdoor Mapping and Navigation Using Stereo Vision / Kurt Konolige, Motilal Agrawal, Robert Bolles et al. — Vol. 39. — 2006. — 01. — Pp. 179–190.
- 3 Schlesinger, Michail. Ten Lectures on Statistical and Structural Pattern Recognition / Michail Schlesinger, Vaclav Hlavac. — 2002. — 01. — Vol. 24.
- 4 Bradski, G. The OpenCV Library / G. Bradski // *Dr. Dobb's Journal of Software Tools*. — 2000.
- 5 Middlebury Stereo Datasets. <http://vision.middlebury.edu/stereo/data/>.
- 6 Scharstein, Daniel. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms / Daniel Scharstein, Richard Szeliski // *International Journal of Computer Vision*. — 2002. — 04. — Vol. 47. — Pp. 7–42.
- 7 Scharstein, Daniel. High-accuracy stereo depth maps using structured light / Daniel Scharstein, R. Szeliski // *Comput. Vision Pattern Recognit.* — 2003. — 07. — Vol. 1. — Pp. I–195.
- 8 Scharstein, Daniel. Learning Conditional Random Fields for Stereo / Daniel Scharstein, Chris Pal // *CVPR*. — 2007. — 06.
- 9 Hirschmüller, Heiko. Evaluation of Cost Functions for Stereo Matching / Heiko Hirschmüller, Daniel Scharstein // *IEEE Conference on Computer Vision and Pattern Recognition, CVPR'07*. — 2007. — 06.

- 10 High-Resolution Stereo Datasets with Subpixel-Accurate Ground Truth / Daniel Scharstein, Heiko Hirschmüller, York Kitajima et al. — 2014. — 09. — Vol. 8753. — Pp. 31–42.

ДОДАТОК А

Код програми.

```

1  #include "opencv2/opencv.hpp"
2  #include "functions.h"
3  #include <stdint.h>
4
5  // #include <cassert>
6  // #define assertm(exp, msg) assert(((void)msg, exp))
7
8  using namespace cv;
9
10 float Scale = 1; // Windows settings
11 int WindowStartX = 1500, WindowStartY = 100, WindowMargin = 10;
12
13 float alpha = 1;
14
15 int main(int argc, char** argv)
16 {
17     //Tests
18     int a[3][3] = { {0, 1, 2},
19                     {1, 0, 1},
20                     {2, 1, 0} };
21
22     Mat testBP = Mat(3, 3, CV_32S);
23     initBinaryPenalty(testBP, alpha);
24     showInt(testBP, 3, 3);
25
26     for (int i = 0; i < 3; i++)
27     {
28         for (int j = 0; j < 3; j++)
29         {
30             int w = testBP.at<int>(i, j);
31             //std::cout << w << " " << a[i][j] << " " << i << j << std::endl;
32             assert( w == a[i][j] );
33         }
34     }
35
36     Mat imL = imread("../imgs/3d/view2.png", IMREAD_GRAYSCALE);
37     Mat imR = imread("../imgs/3d/view1.png", IMREAD_GRAYSCALE);
38     //Mat disp = imread("../imgs/sawtooth/disp2.pgm", IMREAD_ANYDEPTH);
39
40     // Getting image parameters
41     int width = imL.cols, height = imL.rows;
42     //double MAX_DISP;
43     //minMaxLoc(disp, nullptr, &MAX_DISP);
44     // MAX_DISP = static_cast<int>(MAX_DISP/8); // TO DO round
45     int MAX_DISP = 50;
46
47

```

```

48
49     //std::cout << result.at<uchar>(0,2) << std::endl;
50     std::cout << "Max disparity: " << MAX_DISP << std::endl;
51     std::cout << "Width: " << width << " Height: " << height << std::endl;
52
53     //Initialising "g" - binary penalty
54     Mat g = Mat(MAX_DISP+1, MAX_DISP+1, CV_32S); // MAX_DISP, MAX_DISP -1, ... , 1 and 0;
55
56     std::clock_t start, end;
57     start = clock();
58
59     initBinaryPenalty(g, alpha);
60     //showInt(g, MAX_DISP + 1, MAX_DISP + 1);
61
62
63     Mat result = Mat(1, height, CV_32S);
64     for (int row = 0; row < height; row++)
65     {
66         progress(row, height);
67
68         //Initialising "H" - unary penalty
69         Mat H = Mat(width, MAX_DISP + 1, CV_32F);
70         initUnaryPenalty(H, row, imL, imR);
71         //showFl(H, 20, MAX_DISP + 1);
72
73         Mat Fi = Mat(width, MAX_DISP + 1, CV_32F);
74         initFi(Fi, row, H, g);
75         //showFl(Fi, 20, MAX_DISP + 1);
76
77         Mat prevInd = Mat(width, MAX_DISP + 1, CV_32S);
78         initPrevInd(prevInd, Fi, g, row);
79         //showInt(prevInd, width, MAX_DISP + 1);
80
81
82         //di
83         Mat D = Mat(1, width, CV_32S);
84
85         D.at<int>(0, width - 1) = argmin(row, width - 1, Fi, g);
86
87         for (int i = width - 2; i >= 0; i--)
88         {
89             D.at<int>(0, i) = prevInd.at<int>( i + 1, D.at<int>(0, i + 1) );
90         }
91         //showInt(D, 1, width);
92
93         for (int i = 0; i < width; i++)
94         {
95             imR.at<uchar>(row, i) = static_cast<uchar>(D.at<int>(0, i) * 255 / 50);
96         }
97     }
98     end = clock();

```

```
99     double time_taken = double(end - start) / double(CLOCKS_PER_SEC);
100    std::cout << "Time taken by program is : " << std::fixed
101        << time_taken << std::setprecision(5);
102    std::cout << " sec " << std::endl;
103
104    std::cout << "Max disparity: " << MAX_DISP << std::endl;
105    std::cout << "Width: " << width << " Height: " << height << std::endl;
106
107    namedWindow("Left", WINDOW_FREERATIO);
108    namedWindow("Right", WINDOW_FREERATIO);
109
110    imshow("Left", imL);
111    imshow("Right", imR);
112
113    //resizeWindow("Left", width * Scale, height * Scale);
114    //resizeWindow("Right", width * Scale, height * Scale);
115
116    moveWindow("Left", WindowStartX, WindowStartY);
117    moveWindow("Right", WindowStartX , WindowStartY);
118
119    waitKey();
120 }
```

ДОДАТОК Б

```

1  #include "functions.h"
2  #include <iostream>
3
4  using namespace std;
5
6  /////////////// Unary penalty ///////////////
7  void initUnaryPenalty(Mat &H, int row, Mat& imL, Mat& imR)
8  {
9      int maxdi = H.cols, width = H.rows;
10     //std::cout << "cols / rows :" << width << " " << maxdi << std::endl;
11     for (int i = 0; i < width; i++)
12     {
13         for (int di = 0; di < maxdi; di++)
14         {
15             if (i >= di)
16                 H.at<float>(i, di) =
17                     abs (static_cast<float>(imL.at<uchar>(row, i)) - static_cast<float>(imR.at<uchar>(row, i -
18             else
19                 H.at<float>(i, di) = std::numeric_limits<float>::infinity();
20         }
21     }
22     //std::cout << ">Unary penalty matrix for row " << row << " initialised!" << std::endl;
23 }
24
25 void showFl(Mat& M, int size1, int size2)
26 {
27     std::cout << std::endl;
28     for (int i = 0; i < size1; i++)
29     {
30         for (int j = 0; j < size2; j++) // TODO Possible mistakes here
31         {
32             std::cout << M.at<float>(i, j) << " ";
33         }
34         std::cout << std::endl;
35     }
36     std::cout << std::endl;
37 }
38
39
40
41 /////////////// Binary penalty ///////////////
42
43 void initBinaryPenalty(Mat &g, float alpha)
44 {
45     int maxdi = g.cols;
46     for (int di = 0; di < maxdi; di++)
47     {
48         for (int dj = 0; dj < maxdi; dj++)
49         {

```

```

50             g.at<int>(di, dj) = alpha*abs(di - dj);
51         }
52     }
53     std::cout << "Binary penalty matrix initialised!" << std::endl;
54 }
55
56 void showInt(Mat& M, int size1, int size2)
57 {
58     std::cout << std::endl;
59     for (int i = 0; i < size1; i++)
60     {
61         for (int j = 0; j < size2; j++) // TODO Possible mistakes here
62         {
63             std::cout << M.at<int>(i, j) << " ";
64         }
65         std::cout << std::endl;
66     }
67     std::cout << std::endl;
68 }
69
70
71
72 ////////////////////////////////////////////////////////////////// Cumulative penalty //////////////////////////////////////////////////////////////////
73 void initFi(Mat& Fi, int row, Mat& H, Mat& g)
74 {
75     int width = Fi.rows, maxd = Fi.cols;
76     for (int i = 0; i < width; i++)
77     {
78         for (int d = 0; d < maxd; d++)
79         {
80             Fi.at<float>(i, d) = f(row, i, d, Fi, H, g);
81         }
82     }
83 }
84
85 float f(int row, int i, int d, Mat &Fi, Mat &H, Mat &g)
86 {
87     int maxd = g.cols;
88     if (i == 0) return H.at<float>(i, d);
89
90     else
91     {
92         float minf = std::numeric_limits<float>::infinity();
93         for (int dj = 0; dj < maxd; dj++)
94         {
95             float temp = Fi.at<float>(i - 1, dj) + g.at<int>(d, dj);
96             if (temp < minf) minf = temp;
97         }
98         return minf + H.at<float>(i, d);
99     }
100 }
```

```

101
102
103
104 /////////// argmin_d( f_i(d) ) ///////////
105 int argmin(int row, int i, Mat &Fi, Mat &g)
106 {
107     int maxd = g.cols;
108
109     float min = std::numeric_limits<float>::infinity();
110     int mind;
111     for (int d = 0; d < maxd; d++)
112     {
113         float temp = Fi.at<float>(i, d);
114         if (temp < min)
115         {
116             min = temp;
117             mind = d;
118         }
119     }
120     return mind;
121 }
122
123
124
125 /////////// Previous index ///////////
126 void initPrevInd(Mat &prevInd, Mat &Fi, Mat &g, int row)
127 {
128     int width = prevInd.rows, maxd = prevInd.cols;
129
130     for (int d = 0; d < maxd; d++)
131     {
132         prevInd.at<int>(0, d) = 0;
133     }
134
135     for (int i = 1; i < width; i++)
136     {
137         for (int d = 0; d < maxd; d++)
138         {
139             // argmin_dj ( f_i-1(dj) + g(d, dj) )
140             float minf = std::numeric_limits<float>::infinity();
141             int mind = 0;
142             for (int dj = 0; dj < maxd; dj++)
143             {
144                 float temp = Fi.at<float>(i - 1, dj) + static_cast<float>(g.at<int>(dj, d));
145                 if (temp < minf)
146                 {
147                     minf = temp;
148                     mind = dj;
149                 }
150             }
151             prevInd.at<int>(i, d) = mind;

```

```
152         }
153     }
154 }
155
156
157 /////////////////// Progress bar ///////////////////
158 void progress(int p, int max)
159 {
160     system("cls");
161
162     if (p == max - 1)
163     {
164         std::cout << "Done" << endl;
165     }
166     else
167     {
168         int barWidth = 100;
169         float progress = p * 100 / max;
170
171         std::cout << "[";
172         //int pos = barWidth * progress;
173         for (int i = 0; i < barWidth; i++)
174         {
175             if (i < (int)progress) std::cout << "*";
176             else if (i == (int)progress) std::cout << "|";
177             else std::cout << " ";
178         }
179         std::cout << "] - " << p << " / " << max << endl;
180         std::cout << std::flush;
181     }
182 }
183
184 }
```

ДОДАТОК В

Код програми.

```

1  #include <iostream>
2  #include <opencv2/opencv.hpp>
3  #include <limits>
4  #include "functions.h"
5
6  int WindowStartX = 500, WindowStartY = 100, WindowMargin = 10, Scale = 1;
7
8  int main()
9  {
10         Mat imL = imread("../imgs/cloth1/view1.png", IMREAD_GRAYSCALE);
11         Mat imR = imread("../imgs/cloth1/view2.png", IMREAD_GRAYSCALE);
12         int width = imL.cols, height = imL.rows;
13         int MAX_DISP = 100; //< 255
14         Mat depth_map = Mat(height, width, CV_8U);
15
16
17         //For one row
18         for (int row = 0; row < height; row++)
19         {
20             Mat L = getRow(row, imL);
21             Mat R = getRow(row, imR);
22             Mat P = initPrevMatrix(MAX_DISP + 1, width);
23             Mat W = initWeightsMatrix(MAX_DISP + 1);
24             Mat W2 = initWeightsMatrix(MAX_DISP + 1);
25             Mat G = initBinaryPenalty(MAX_DISP, 10);
26             Mat H = Mat(MAX_DISP + 1, width, CV_32S);
27
28             //---Computing PrevMatrix---
29             for (int t = 1; t < width; t++)
30             {
31                 for (int d = 0; d < MAX_DISP + 1; d++)
32                 {
33                     int temp_sum;
34                     int Wmin = std::numeric_limits<int>::max();
35                     for (int b = 0; b < MAX_DISP + 1; b++)
36                     {
37                         //
38                         if (b > t)
39                             H.at<int>(b, t) = std::numeric_limits<int>::infinity();
40                         else
41                             H.at<int>(b, t) = abs(L.at<uchar>(0, t) - R.at<uchar>(0, t
42
43                         //
44                         if (H.at<int>(b, t) == std::numeric_limits<int>::infinity())
45                         {
46                             temp_sum = std::numeric_limits<int>::infinity();
47                             //std::cout << "H backfired" << std::endl;

```

```

48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
}
/*
else if (W.at<int>(b, 0) == std::numeric_limits<int>::infinity())
{
    temp_sum = std::numeric_limits<int>::infinity();
    //std::cout << "W backfired" << std::endl;
}

else if (G.at<int>(d, b) == std::numeric_limits<int>::infinity())
{
    temp_sum = std::numeric_limits<int>::infinity();
    std::cout << "G backfired" << std::endl;
}
*/
else
{
    temp_sum = W.at<int>(b, 0) + H.at<int>(b, t) + G.at<int>(d,
    //if (temp_sum < Wmin)
    {
        Wmin = temp_sum;
        W2.at<int>(d, 0) = Wmin;
        P.at<uchar>(d, t) = b;
    }
}
W = W2;
//std::cout << t << " / " << d << " / " << static_cast<int>(P.at<uchar>(d, t));
//std::cout << " / " << W2.at<int>(d, 0) << std::endl;
}

}

//--Restore depth_string--
Mat depth_string = Mat(1, width, CV_8U);
//Finding d_n
int temp_sum, d, n = width - 1;
int Wmin = std::numeric_limits<int>::max();
for (int b = 0; b < MAX_DISP + 1; b++)
{
    temp_sum = W.at<int>(b, 0) + h(n, b, L, R);
    //std::cout << d << " / " << b << " / " << temp_sum << std::endl;
    if (temp_sum < Wmin)
    {
        Wmin = temp_sum;
        d = b;
    }
}
//Computing depth_string
depth_string.at<uchar>(0, n) = d;
for (int i = n - 1; i > 0; i--)
{

```

```
99         int k = static_cast<int>(depth_string.at<uchar>(0, i + 1));
100        depth_string.at<uchar>(0, i) = P.at<uchar>(k, i + 1);
101        //std::cout << i << " / " << static_cast<int>(depth_string.at<uchar>(0, i)) << std::endl;
102    }
103
104
105    //Writing data to depth_map
106    for (int i = 0; i < width; i++)
107    {
108        depth_map.at<uchar>(row, i) = depth_string.at<uchar>(0, i);
109    }
110    std::cout << "[ " << row << " / " << height << " ] \r";
111 }
112
113
114
115
116
117
118    namedWindow("Window1", WINDOW_FREERATIO);
119    imshow("Window1", depth_map);
120    //resizeWindow("Window1", width * Scale, height * Scale);
121    //moveWindow("Window1", WindowStartX, WindowStartY);
122    waitKey();
123 }
```

ДОДАТОК Г

Код програми.

```

1  #include "functions.h"
2  #include <iostream>
3
4
5  int h(int i, int d, Mat L, Mat R)
6  {
7      if (d > i)
8          return abs(L.at<uchar>(0, i) - R.at<uchar>(0, 0));
9      else
10         return abs(L.at<uchar>(0, i) - R.at<uchar>(0, i - d));
11 }
12
13
14 int g(int d, int b, int alpha)
15 {
16     return alpha * abs(d - b);
17 }
18
19 Mat initBinaryPenalty(int MAX_DISP, float alpha)
20 {
21     Mat g = Mat(MAX_DISP + 1, MAX_DISP + 1, CV_32S);
22     int maxdi = g.cols;
23     for (int di = 0; di < maxdi; di++)
24     {
25         for (int dj = 0; dj < maxdi; dj++)
26         {
27             g.at<int>(di, dj) = alpha * abs(di - dj);
28         }
29     }
30
31     return g;
32 }
33
34
35 Mat initPrevMatrix(int height, int width)
36 {
37     Mat P = Mat(height, width, CV_8U);
38     for (int i = 0; i < height; i++)
39     {
40         for (int j = 0; j < width; j++)
41         {
42             P.at<uchar>(i, j) = 0;
43         }
44         //P.at<uchar>(i, 0) = 0;
45     }
46     return P;
47 }
```

```
48
49
50 Mat initWeightsMatrix(int height)
51 {
52     Mat W = Mat(height, 1, CV_32S);
53     for (int d = 0; d < height; d++)
54     {
55         W.at<int>(d, 0) = 0;
56     }
57     return W;
58 }
59
60
61 Mat getRow(int row, Mat im)
62 {
63     Mat R = Mat(1, im.cols, CV_8U);
64     for (int i = 0; i < im.cols; i++)
65     {
66         R.at<uchar>(0, i) = im.at<uchar>(row, i);
67     }
68     return R;
69 }
```