НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

«До захисту допущено»
Завідувач кафедри

_____
(підпис)
М. В. Грайворонський
(ініціали, прізвище)
“____” _____
2017 р.

Дипломна робота

освітньо-кваліфікаційного рівня “магістр”

за спеціальністю 8.04030101 «Прикладна математика»
на тему «Тема»
Виконав студент 6 курсу групи group
Name

Керівник Rank, Name _____
(підпис)

Рецензент Rank, Name _____
(підпис)

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

Київ — 2017 року

ЗМІСТ

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 до автономної навігації — 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) ?(while not being computationaly complex)?.

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 онлайн алгоритму для задачі стереозору.

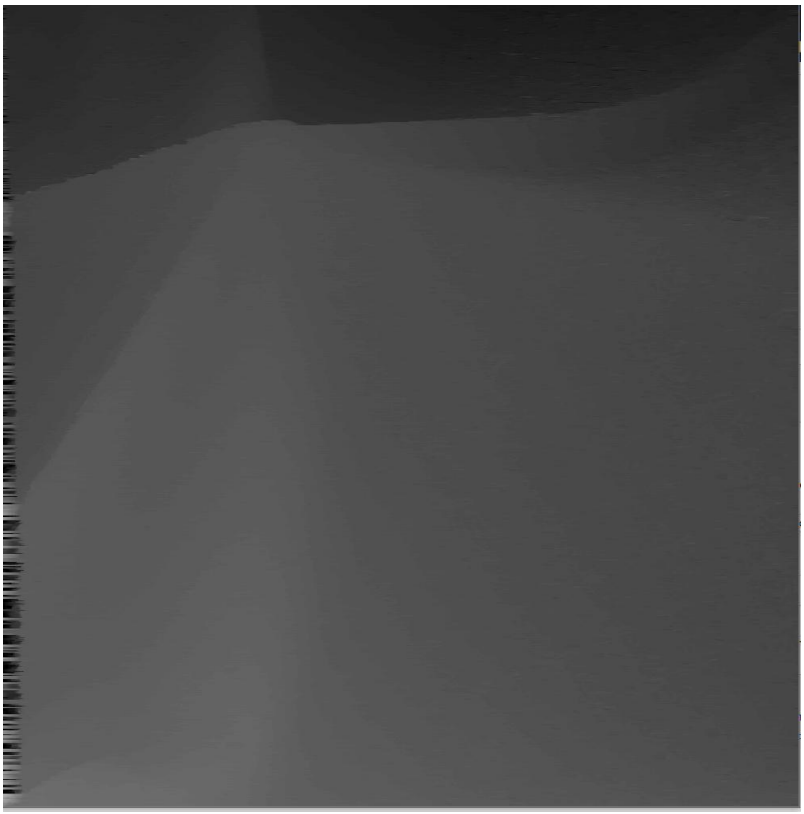

Рисунок 1 — Результат роботи одновимірного алгоритму

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

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Знаходження глибини методом бінокулярного паралаксу

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 obscūra — «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.

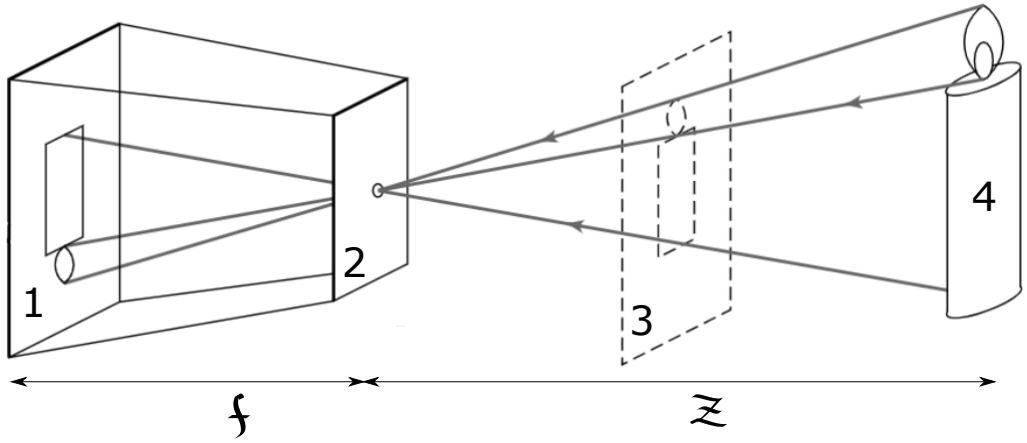

Рисунок 1.1 — Будова камери-обскури.
1 — результуюче зображення, 2 — отвір, 3 — віртуальне зображення, 4 — об’єкт

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 зобра-

ження, 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 з 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 відстань,

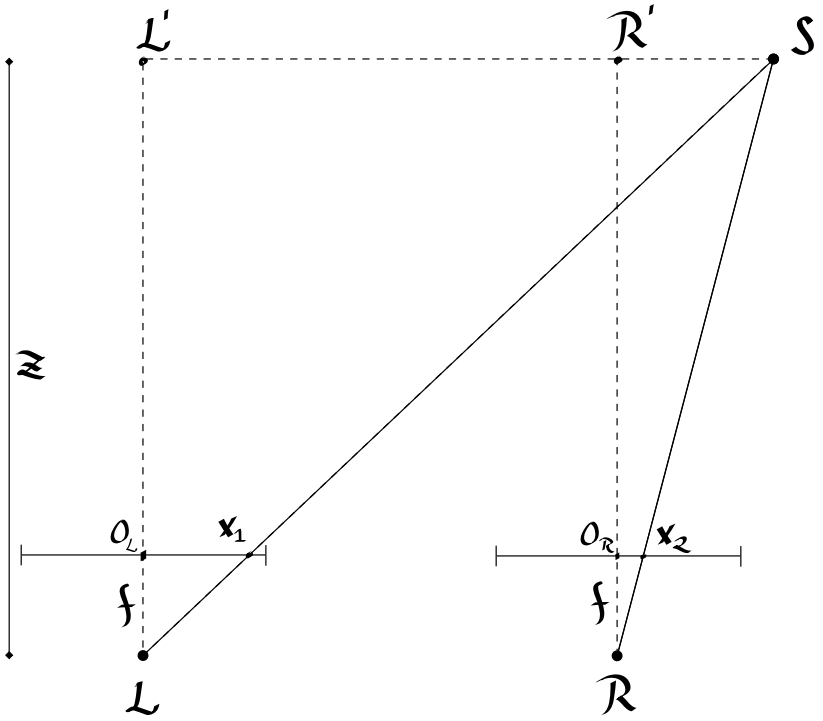

Рисунок 1.2 — Геометрія розташування камер.

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

$$\mathcal{L} : I \to \mathcal{C},$$
$$\mathcal{R} : I \to \mathcal{C}.$$

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 зображенні (1.3). Введемо сюр'єктивне відображення $d : I \to D$, 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

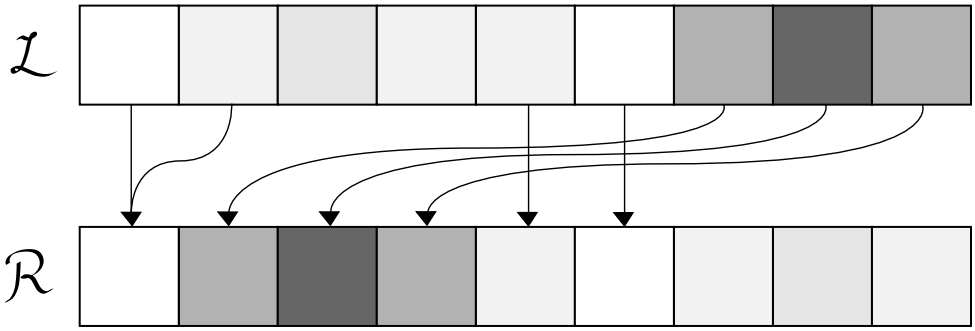

Рисунок 1.3 — Пошук відповідних пікселів

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 $\overline{d} \in D^n$, яка мінімізує штрафну функцію

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

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 пошуку послідовності $\overline{d}$, що мінімізує штрафну функцію (1.1), як пошук найкоротшого шляху через орієнтований зважений граф $G =< \mathcal{V}, \mathcal{E} >$ (рис. 1.4). Його множина вершин

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

Введемо функцію ваг вершин

$$v : \mathcal{V} \setminus \{S, E\} \to \mathbb{R},$$
$$v(\sigma(i,d)) = h(i,d),\ \forall i \in I,\, d \in D.$$

Множина його ребер

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

Введемо функцію ваг ребер

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

де $w(v_1, v_2)$ — вага ребра $< v_1, v_2 >,\ < v_1, v_2 > \in \mathcal{E}$,

$$w(\sigma(i,d), \sigma(i+1, d')) = g(d, d'),\ \forall d, d' \in D, i \in I,$$
$$w(S, \sigma(1,d)) = 0, \forall d \in D,$$
$$w(\sigma(n,d), E) = 0, \forall d \in D.$$

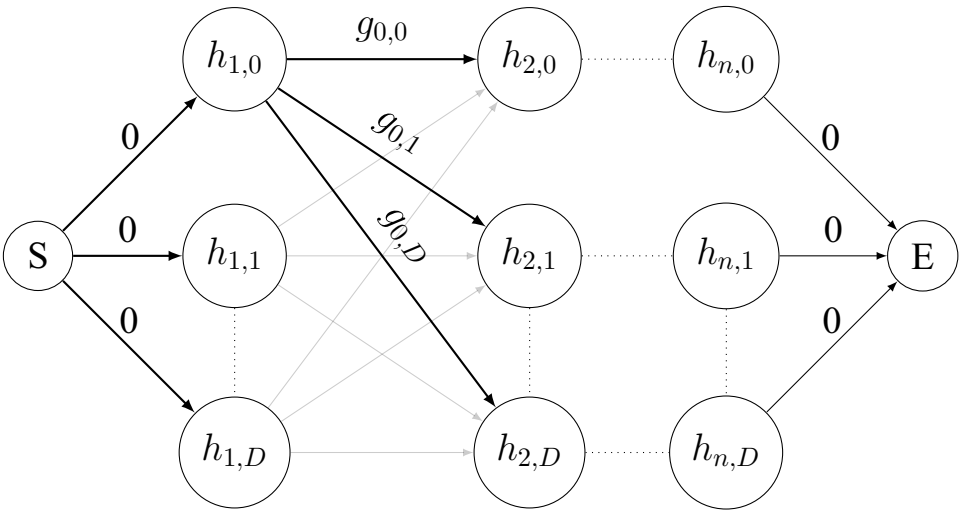

Рисунок 1.4 — Граф G

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

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

Тоді елементи послідовності $\overline{d}$ знаходимо за формулами

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

1.3 Онлайн алгоритм пошуку скалярного поля зсувів

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.

1.3.1 Постановка задачі

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

$$\mathcal{L} : I \to \mathcal{C} \cup \{\varepsilon\},$$
$$\mathcal{R} : I \to \mathcal{C} \cup \{\varepsilon\}.$$

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 $\overline{d} \in D^n$, яка мінімізує штрафну функцію (1.1).

Цю задачу ми теж представимо як пошук найкоротшого шляху на графі $G =< \mathcal{V}, \mathcal{E} >$. 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 назвемо її **закритою**.

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

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

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

1.3.2 Поступове упорядковане надходження даних

Нехай на початку нам не відомі жодні пікселі обох зображень (рис. 1.5), та існує деякий стохастичний автомат, що на кожному кроці $t \in I$ генерує

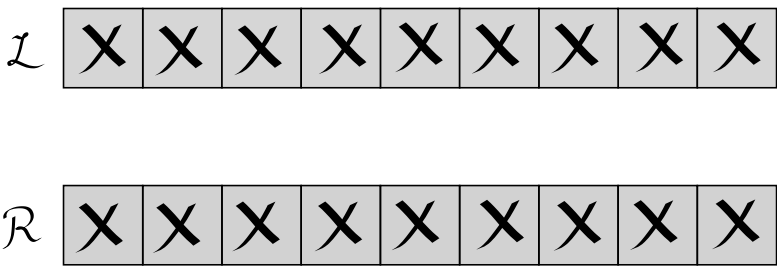

Рисунок 1.5 — Немає даних

пари $< c_L,\ c_R >$ — кольори наступних пікселів лівого та правого зображень ($c_L, c_R \in \mathcal{C}$). Тоді на кожному кроці t нам буде відкриватися стовпчик вершин $h(t,d), \forall d \in D, t \in I$ (рис. 1.8).

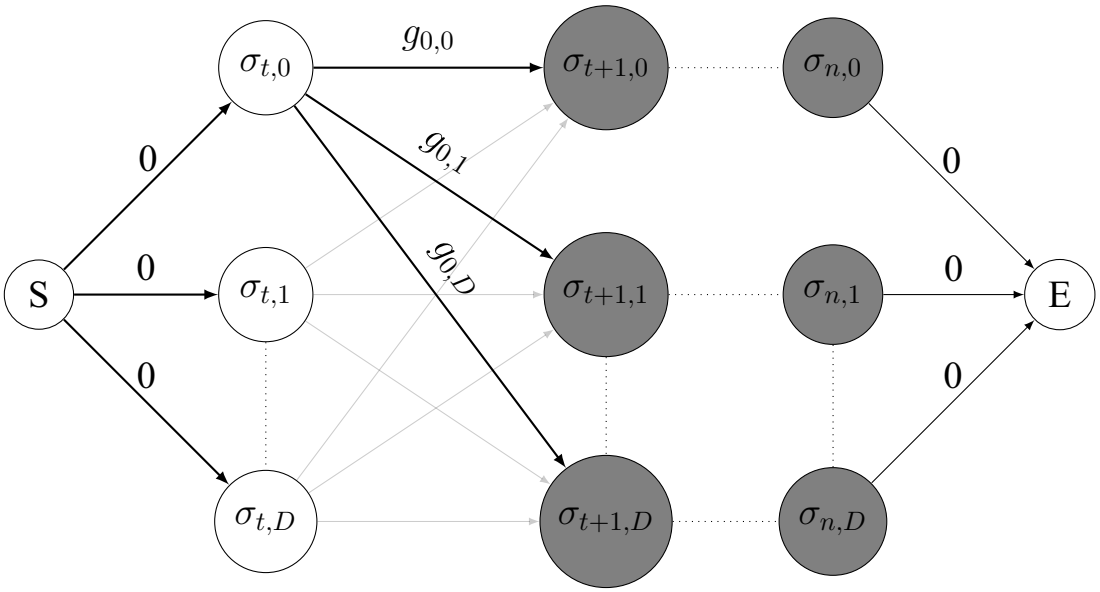

Рисунок 1.6 — Граф G

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 назвемо *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 $G^0 =< \mathcal{V}^0, \mathcal{E}^0 >=< \mathcal{V}, \mathcal{E} >$. Нехай також $T = \{1, ..., n-1\}$ — множина ітерацій, $s : T \times D \to R$, $s^0(d) = 0, \ \forall d \in D$. Для відновлення послідовності $\overline{d}$ введемо матрицю *попередніх оптимальних вершин* $\hat{\mathcal{P}}$, розмірності $n \times (D_{max}+1)$.

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

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

1) $\forall d \in D$:

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

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

   (якщо мінімізуючих 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 $\bigcup\limits_{d \in D} < S, \sigma(t, d) >$ як $\mathcal{A}$,

   множину $\bigcup\limits_{\substack{d \in D \\ d' \in D}} < \sigma(t, d), \sigma(t+1, d') >$ як $\mathcal{B}$,

   а множину $\bigcup\limits_{d \in D} < S, \sigma(t+1, d) >$ як $\mathcal{C}$. Тоді

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

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

4) $G^t =< \mathcal{V}^t, \mathcal{E}^t >$

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

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

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

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

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 опрацювати лише D_{max} ребер. Використання ж offline алгоритму потребує опрацювання $(n-1) \cdot {D_{max}}^2$ ребер після надходження всіх даних. А якщо б ми спочатку

чекали приходу всіх даних, а тільки потім починали обчислення, нам треба було опрацювати $(n-1)(D_{max})^2$ ребер.

1.3.3 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. 1.7). Введемо деякий стохастичний автомат, що на кожному кроці $t \in I$ генерує

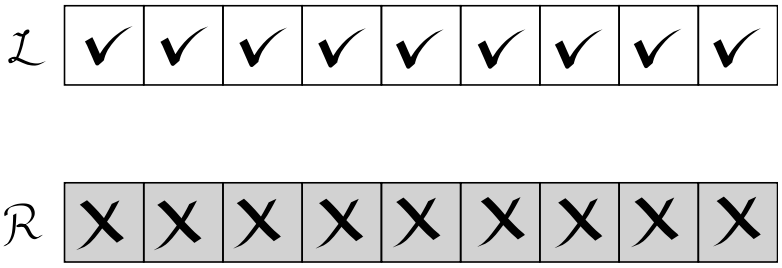

Рисунок 1.7 — Немає даних

пари $< i,\ c >$, де c — інтенсивність пікселя з координатою i для невідомого зображення ($i \in I, c \in \mathbb{R}$). Тоді на кожному кроці t нам буде відкриватися стовпчик вершин $h(i,d), \forall d \in D$ (рис. 1.8).

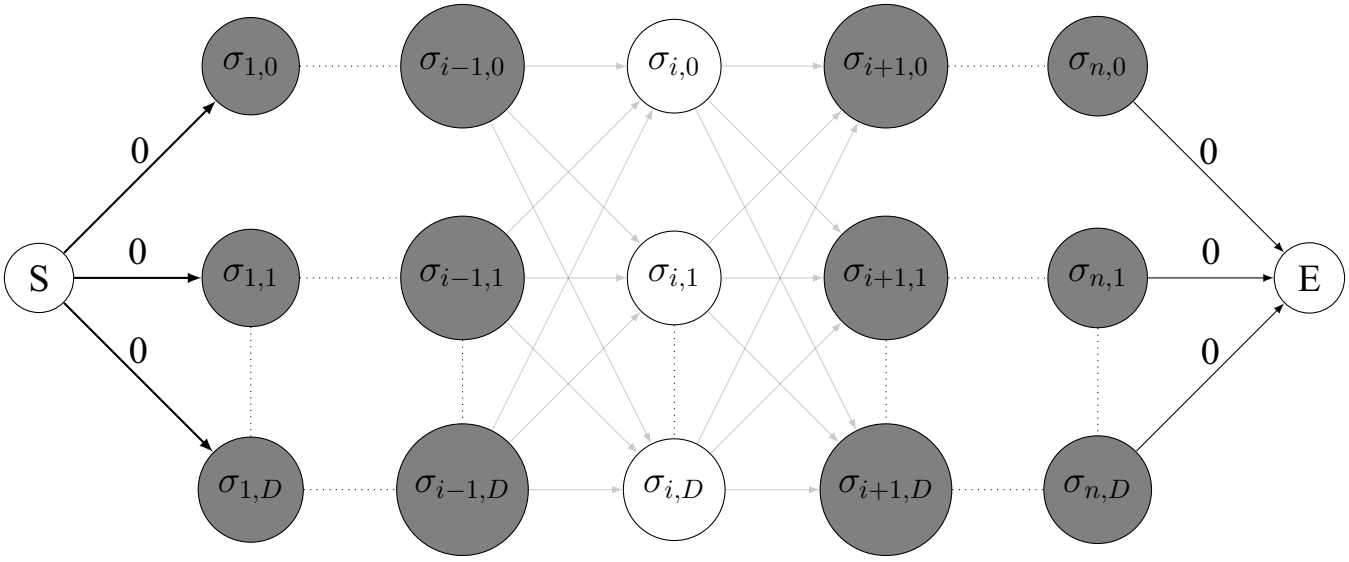

Рисунок 1.8 — Граф G (відкриті вершини зафарбовані білим).

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 їх оптимальну середню 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, зменшивши загальну кількість ребер теж на ${D_{max}}^2$.

Введемо $G^0 =< \mathcal{V}^0, \mathcal{E}^0 >=< \mathcal{V}, \mathcal{E} >= G$, $T = \{1, ..., n-1\}$ — множина ітерацій. На кожній ітерації t будемо замінювати граф G^{t-1} на новий граф G^t, що матиме менше ребер та вершин. Введемо функцію

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

де $w^t(v_1, v_2)$ — вага ребра $< v_1, v_2 >$ у графі G^t ($< v_1, v_2 > \in E^t$). Для знаходження сусідніх стовпчиків закритих вершин у графі G^t вводимо

$$j = i - \sum_{k=1}^{i} \mathbb{1}(\mathcal{L}(k) \neq \varepsilon)$$

Для відновлення послідовності $\overline{d}$ введемо матрицю *оптимальних вершин* $\hat{\mathcal{P}}$, розмірності $I \times (D_{max}+1) \times (D_{max}+1)$,

$${p^i}_{d',d''} = 0,\ \forall d', d'' \in D, \forall i \in I.$$

Як тільки автомат на кроці $t \in T$ генерує нову пару $< i,\ c >$, шукаємо новий граф G^t

1) $j = i - \sum\limits_{k=1}^{i} \mathbb{1}(\mathcal{L}(k) \neq \varepsilon)$,
$\hat{n} = n - \sum\limits_{k=1}^{n} \mathbb{1}(\mathcal{L}(k) \neq \varepsilon)$.

2) Якщо $j = 0$,

$$w^t(s, \sigma(j+1, d)) = \min_{\hat{d} \in D} \left(w^{t-1}(s, \sigma(i,\hat{d})) + h(i, \hat{d}) + w^{t-1}(\sigma(i, \hat{d}), \sigma(j+1,d))\right), \tag{1.2}$$

$$p^i{}_{d',d''} = \arg\min_{\hat{d} \in D} \left(w^{t-1}(s, \sigma(i,\hat{d})) + h(i, \hat{d}) + w^{t-1}(\sigma(i, \hat{d}), \sigma(j+1,d))\right). \tag{1.3}$$

3) Якщо $j = \hat{n}$,

$$w^t(\sigma(j, d), e) = \min_{\hat{d} \in D} \left(w^{t-1}(\sigma(j, d), \sigma(i,\hat{d})) + h(i, \hat{d}) + w^{t-1}(\sigma(i, \hat{d}), e)\right), \tag{1.4}$$

$$p^i{}_{d',d''} = \arg\min_{\hat{d} \in D} \left(w^{t-1}(\sigma(j, d), \sigma(i,\hat{d})) + h(i, \hat{d}) + w^{t-1}(\sigma(i, \hat{d}), e)\right); \tag{1.5}$$

4) Якщо $0 < j < \hat{n}$,

$$w^t(\sigma(j, d'), \sigma(j+1, d'')) = \tag{1.6}$$

$$\min_{\hat{d} \in D} \left(w^{t-1}(\sigma(j, d'), \sigma(i,\hat{d})) + h(i, \hat{d}) + w^{t-1}(\sigma(i, \hat{d}), \sigma(j+1, d'')\right); \tag{1.7}$$

$$p^i{}_{d',d''} = \arg\min_{\hat{d} \in D} \left(w^{t-1}(\sigma(j, d'), \sigma(i,\hat{d})) + h(i, \hat{d}) + w^{t-1}(\sigma(i, \hat{d}), \sigma(j+1, d'')\right); \tag{1.8}$$

5) $\mathcal{V}^t = \mathcal{V}^{t-1} \setminus \{\sigma(i,d) \mid \forall d \in D\}$;

6) Позначимо множину $\bigcup\limits_{d\in D} < S, \sigma(i,d) >$ як $\mathcal{A}(i)$,
множину $\bigcup\limits_{\substack{d\in D \\ d'\in D}} < \sigma(i,d), \sigma(i+1,d') >$ як $\mathcal{B}(i)$,
а множину $\bigcup\limits_{d\in D} < S, \sigma(t+1,d) >$ як $\mathcal{C}$. Тоді

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

Вага ребра $< S, \sigma(t+1,d) >= s^t(d)$;

7) $G^t =< \mathcal{V}^t, \mathcal{E}^t >$.

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

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

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

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

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 опрацювати лише D_{max} ребер. А якщо б ми спочатку чекали приходу всіх даних, а тільки потім починали обчислення, нам треба було опрацювати $(n-1)(D_{max})^2$ ребер.

1.4 Стереозрение при медленно поступающих данных

Описать что такое медленно приходящие данные...

1.4.1 Данные приходят по порядку

Пусть сначала все пиксели нам неизвестны (рис. 1.9), поэтому мы не можем вычислить вес ни одной из вершин графа G (**??**).

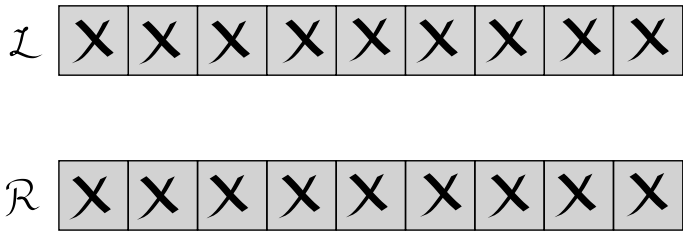

Рисунок 1.9 — Нет данных

Если мы не можем вычислить вес вершины — будем называть эту вершину "*закрытой*" и обозначать ее черным кружочком. В противном случае назовем вершину "*открытой*" и обозначим белым кружком. Для простоты рисунков положим $D_{max} = 2$ и не будем обозначать ребер. Тогда, в ситуации когда все данные нам неизвестны, граф G будет иметь следующий вид (рис. 1.10).

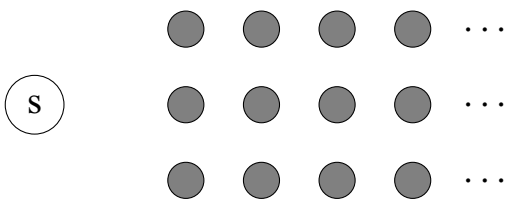

Рисунок 1.10 — граф G в отсутствии данных

Пусть теперь нам по порядку начинают приходить пары из пикселей правого и левого изображений с одинаковыми координатами. Когда нам известен только один первый пиксель каждого изображения (рис.1.11) то мы мо-

жем вычислить вес только одной вершины, и граф G будет иметь вид как на рис.1.12.

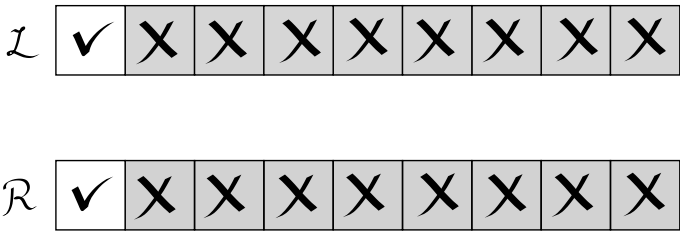

Рисунок 1.11 — Известны только первые пиксели

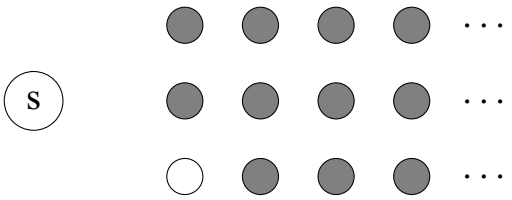

Рисунок 1.12 — Граф G когда известны только первые пиксели

Когда же нам будут известны уже первые два пикселя обоих изображений, мы сможем найти веса уже трёх вершин, и граф G будет выглядеть как на рис.1.13.

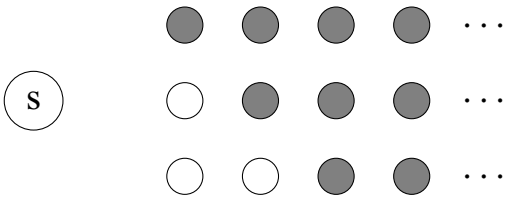

Рисунок 1.13 — Відомі перші два пікселі зображень

По мере прихода следующих пикселей в графе G будут открываться вершины на диагонали над уже открытыми вершинами. А как только нам станут известны $D_{max} + 1$ первых пикселей обоих изображений — то все вершины в первом столбике станут открытыми (рис. 1.14), и мы можем приступать к предвычислениям

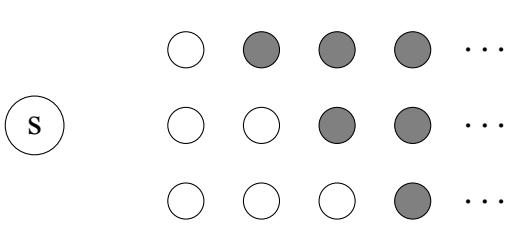

Рисунок 1.14 — Известны первые $D_{max} + 1$ пікселей

Предположим кратчайший путь проходит через какую-то вершину $\sigma(2, d)$ во втором столбике. Тогда точно можно утверждать, что он проходит через вершину $\sigma(1, k)$ первого столбика, где

$$k = \arg\min_{l=0}^{D_{max}} \big(h(1, l) + g(l, d)\big).$$

Тогда все остальные возможные пути из S в $\sigma(2, d)$ нам не нужны, и мы их отбрасываем, а вместо них вводим ребро $< S, \sigma(2, d) >$, с весом

$$g^{(1)}{}_d = \min_{l=0}^{D_{max}} \big(h(2, l) + g(l, d)\big),\ \forall d \in D.$$

Таким образом мы уменьшили количество рёбер в графе G на D_{max}.

Но мы не знаем через какую именно вершину второго столбика проходит кратчайший путь, поэтому вынуждены провести эту операцию для всех вершин $\sigma(2, d)\ \forall d \in D$.В результате такой оптимизации мы сократим количество рёбер на $(D_{max})^2$ (рис. 1.15 $\rightarrow$ рис. 1.16).

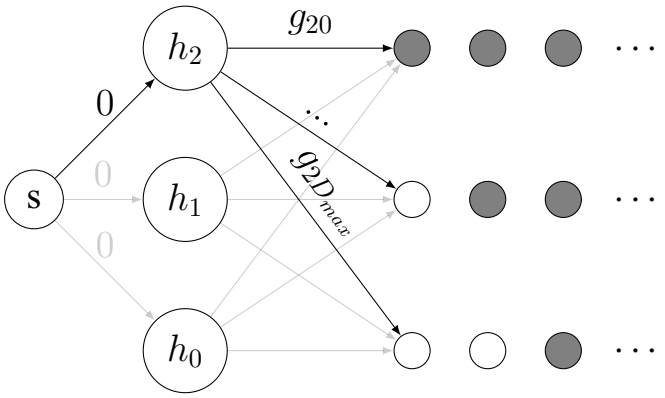

Рисунок 1.15 — граф G до оптимизации

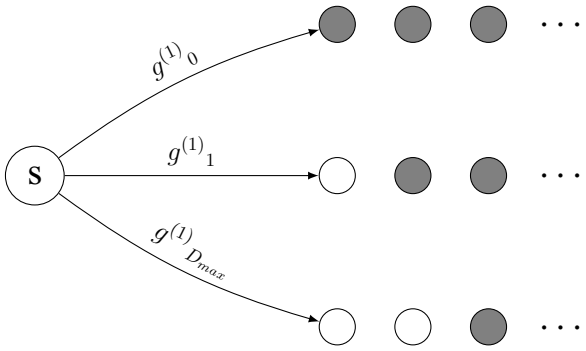

Рисунок 1.16 — граф G после оптимизации

По пришествию первых $D_{max} + 2$ стереопар пикселей проводим аналогичные действия, но теперь уже учитываем предыдущую оптимизацию. Для каждой вершины $\sigma(3, d)$ $d \in D$ находим вершину $\sigma(2, k)$, где

$$k = \arg\min_{l \in D} \left(g_l^{(1)} + h(2, l) + g(l, d) \right).$$

Отбрасываем все пути из S в $\sigma(3, d),\ d \in D$, и вводим рёбра $< S, \sigma(3, d) > d \in D$ с весами

$$g^{(2)}{}_d = \min_{l \in D} \left(g_l^{(1)} + h(2, l) + g(l, d) \right),\ \forall d \in D.$$

Этим самым снова уменьшив общее количество рёбер графа G на $(D_{max})^2$.

Таким образом, при как только становятся известны первые $D_{max} + m$

стереопар пикселей $(m \geqslant 2)$, отбрасываем все пути из S в $\sigma(m+1,d)\ d \in D$, заменяя их рёбрами $< S, \sigma(m+1,d) > \ \forall d \in D$, с весами

$$g^{(m)}{}_d = \min_{l \in D} \left(g_l^{(m-1)} + h(m,l) + g(l,d) \right),\ \forall d \in D, \tag{1.9}$$

уменьшая количество рёбер графа G ещё на $(D_{max})^2$.

Для восстановления конечной оптимальной последовательности $\overline{d}$, нам для каждой вершины $\sigma(i,d),\ \forall\ i = \overline{2,n},\ d \in D$ нужно запоминать предыдущую ей "*оптимальную*" вершину $\sigma(i-1,k)$, где

$$k = \arg\min_{l \in D} \left({g^{(i-1)}}_l + h(i-1,l) + g(l,d) \right).$$

Для этого введем матрицу предыдущих оптимальных вершин $\hat{\mathcal{P}}$ размера $n \times D_{max} + 1$, где $\sigma(i-1, p_{ij})$ — предыдущая $\sigma(i,j)$ оптимальная вершина, $\forall i \in I, j \in D$. Элементы этой матрицы находятся следующим образом:

- $p_{1j} = 0,\ \forall j \in D$ поскольку в вершины первого столбца можно попасть только из начальной вершины S, и предыдущая оптимальная вершина для них всегда одна. Конечно, можно просто исключить эти элементы из матрицы, но их наличие приводит к более красивому и простому определению остальных элементов матрицы $\hat{\mathcal{P}}$.
- $p_{ij} = \arg\min_{l \in D} \left({g^{(i-1)}}_l + h(i-1,l) + g(l,j) \right)$.

Элементы p_{ij} нужно вычислять при приходе $D_{max} + i$ пары пикселей, так как при приходе следующей пары, рёбра $g_l^{(i-1)}$ используются для расчёта рёбер $g_l^{(i)},\ \forall l \in D$ согласно формуле 1.9 и отбрасываются.

После нахождения всех элементов $\hat{\mathcal{P}}$, можем восстановить все элементы последовательности $\overline{d}$ —

$$\begin{aligned} d_n &= \arg\min_{l \in D} \left({g^{(n-1)}}_l + h(n,l) \right), \\ d_{n-1} &= p_{n,d_n}, \\ &\dots \\ d_i &= p_{i,d_{n-1}}. \end{aligned}$$

1.4.2 Одно из изображений известно

2 ПРАКТИЧНІ РЕЗУЛЬТАТИ

3 FINAL REMARKS

ВИСНОВКИ

В результаті виконання роботи вдалося.

REFERENCES

ПЕРЕЛІК ПОСИЛАНЬ