Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №8
з дисципліни
«Технології Computer Vision»
на тему
«Дослідження технологій тривимірної реконструкції об’єктів за цифровими зображеннями»

Виконала:
Студентка групи ІМ-21
Кривохата Марія Юріївна
Номер у списку групи: 12

Перевірив: Баран Д. Р.

Київ 2024

Мета: дослідити методологію і технології реконструкції 3D просторових об’єктів за їх 2D зображеннями методами багатовидової (стерео / сигнатурна) обробки.

Завдання

Завдання І рівня складності – максимально 8 балів.

Організувати та реалізувати роботу стереопари та отримати цифрове статичне зображення самостійно обраного об’єкту із двох каналів з різними значеннями кутового ракурсу. Або обрати із відкритих джерел результати роботи стереопари. Здійснити 3D реконструкцію обраного об’єкту та дослідити якість результату від параметрів стереопари: база, ракурс на об’єкт (за умов наявності стереопари, або відомих параметрів, що супроводжують відкриті джерела даних від стереопари).

Синтезована математична модель

Диспаритет визначається як різниця у положеннях певної точки на лівому та правому зображеннях. Він обчислюється за допомогою алгоритму Semi-Global Block Matching (SGBM).

Математична модель SGBM включає: $\mathrm{Cost}(x, y, d) = ||I_L(x, y) - I_R(x-d, y)||$, де:

- $I_L(x, y)$ та $I_R(x, y)$ - інтенсивності пікселів у лівому і правому зображеннях відповідно;
- d - тестований диспаритет (зміщення).

Алгоритм SGBM знаходить найкраще значення d, яке мінімізує вартість.

Глибина (Z) відновлюється із диспаритету (d) за формулою: Z = (f*B) / d, де:

- f - фокусна відстань камери.
- B - базова лінія (відстань між камерами).
- d - диспаритет.

Стереопара проектується у тривимірний простір через **матрицю переобчислення Q**:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & -1 & 0 & c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & 1/B & 0 \end{bmatrix}$$

Ця матриця перетворює 2D координати (з урахуванням диспаритету) у 3D простір.

Функція **cv2.reprojectImageTo3D()** використовує диспаритетну карту і матрицю Q для побудови 3D-точок у світовій системі координат:

$X = ((u - c_x) * Z) / f$

$Y = ((u – c_y) * Z) / f$

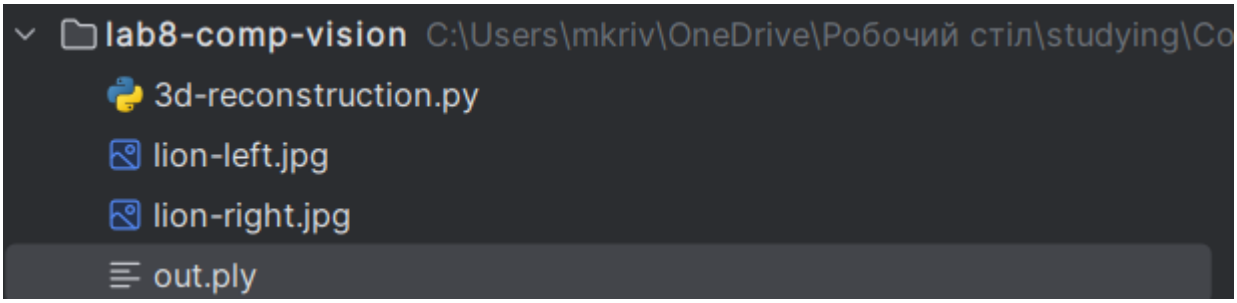
$Z = (f * B) / d$

де u,v — координати пікселя на зображенні.

Результат архітектурного проектування

Опис структури проекту

В проекті міститься програма, що вирішує завдання – 3d-reconstruction.py, ліва частина стерео зображення зі статуєю лева – lion-left.jpg, права частина стерео зображення – lion-right.jpg. Результат роботи програми при кожному запуску записується до out.ply.



Результат роботи програми та дослідження впливу параметрів на відображення

В ролі початкового стерео зображення (лівої та правої частини) було використано такі зображення:

Для того, щоб дослідити вплив різних параметрів на ефективність формування вихідного зображення, спочатку необхідно визначити за що відповідає кожен із параметрів, що використовується в програмі:

- $window\_size$ – Розмір вікна для порівняння блоків між двома зображеннями;

- min_disp – Мінімальне значення диспаритету;

- num_disp – Кількість можливих значень диспаритету;

- $blockSize$ – Розмір блоків для порівняння;

- $P1$ – Параметр для контролю згладжування диспаритету (менші значення);

- $P2$ – Параметр для контролю згладжування диспаритету (більші значення);

- $disp12MaxDiff$ – Максимальна різниця між лівим і правим диспаритетом для визнання пари валідною;

- $uniquenessRatio$ – Відсоток унікальності для фільтрації поганих матчів;

- $speckleWindowSize$ – Розмір вікна для видалення малих часток шуму;

- $speckleRange$ – Максимальна різниця диспаритету в межах вікна для видалення часток шуму.

- f – Фокусна відстань камери;

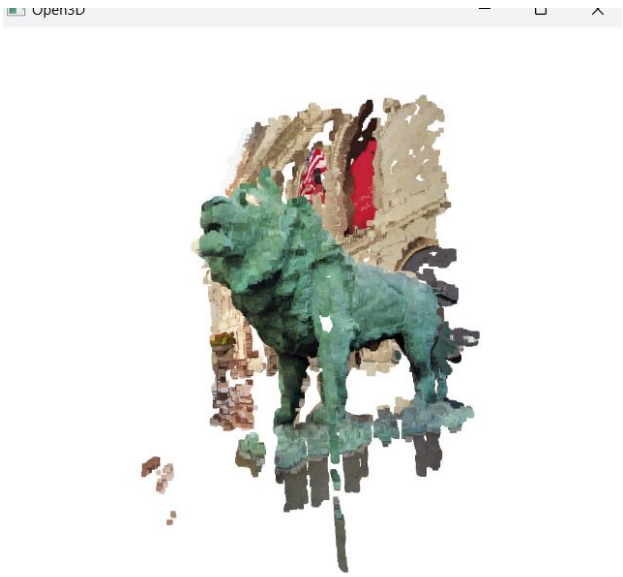
- Q – Матриця проекції для перетворення 2D диспаритету в 3D точки.

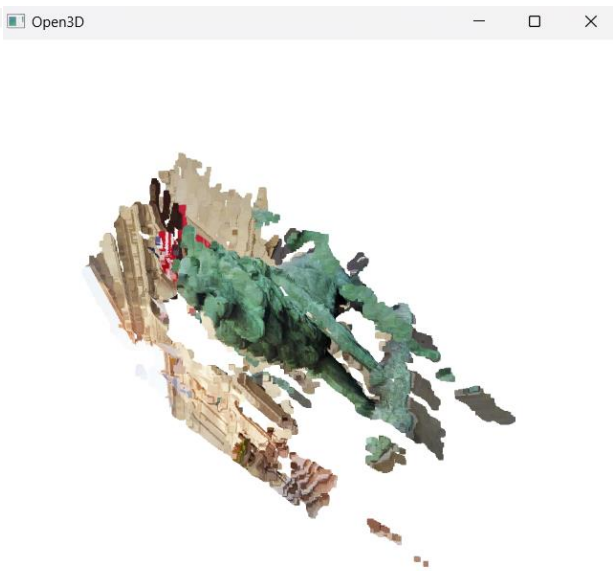
Тепер спробуємо запустити програму декілька разів і, експериментуючи зі значеннями різних параметрів, знайти найвигіднішу комбінацію та зрозуміти, який параметр як повпливає на результат. Для початку візьмемо набір параметрів, що пропонувався у прикладі для лабораторної роботи. Для зображення у прикладі ці параметри підійшли досить добре, тож перевіримо їх в іншій задачі.

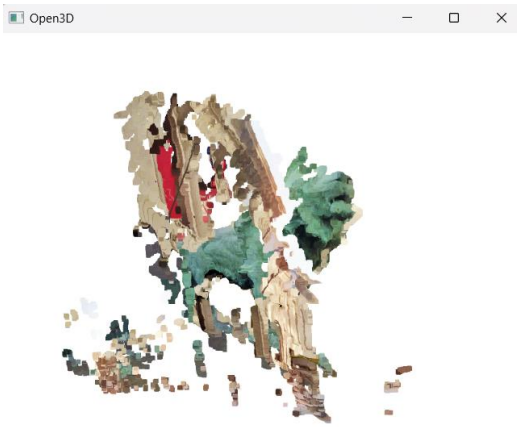
1) Отже, такі налаштування були взяті як стартова точка:

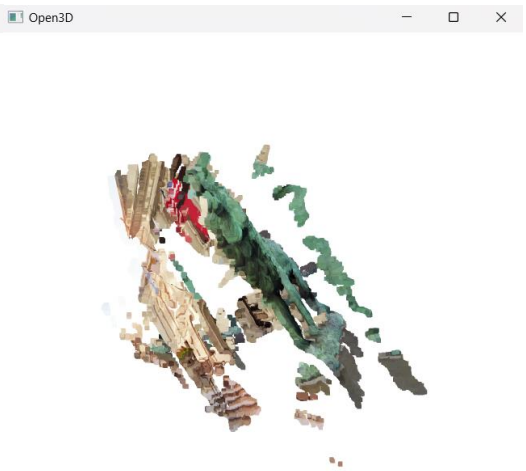
```
window_size = 3
min_disp = 16
num_disp = 112-min_disp
stereo = cv2.StereoSGBM_create(minDisparity = min_disp,
    numDisparities = num_disp,
    blockSize = 16,
    P1 = 8*3*window_size**2,
    P2 = 32*3*window_size**2,
    disp12MaxDiff = 1,
    uniquenessRatio = 10,
    speckleWindowSize = 100,
    speckleRange = 32
)
```

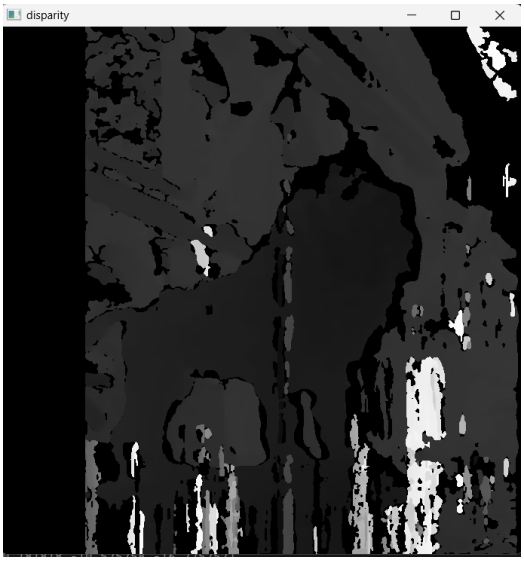
Із результатів бачимо, що з одного з ракурсів лев дійсно виглядає впізнавано, але частина постаменту стерта і позаду лева наявна занадто велика частина фону. До того ж посередині зображення є стовпчик із точок із некоректною глибиною.



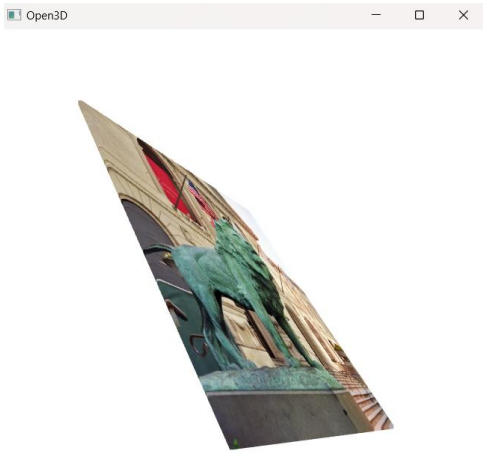




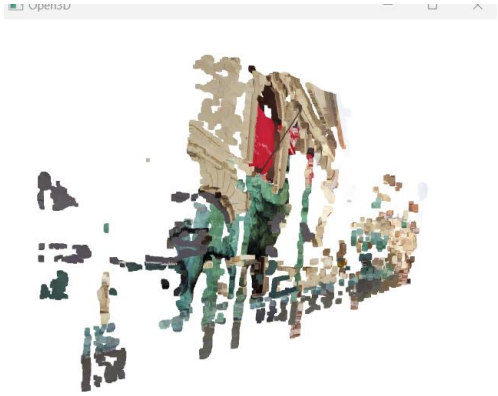


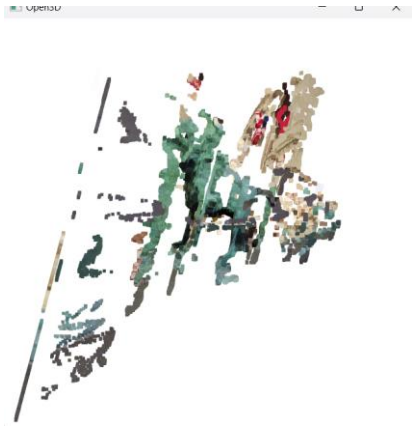

disparity

2) Спробуємо поекспериментувати із параметром window_size. Змінимо його із 3 до 20, щоб точно зрозуміти вплив. Бачимо, що при window_size=20 відображається повністю пласке зображення.



Якщо ж ми встановимо window_size=10, то бачимо, що результуюче зображення не пласке, але воно має занадто сильну деталізацію фону, через що результат стає збіркою невпізнаваного шуму:



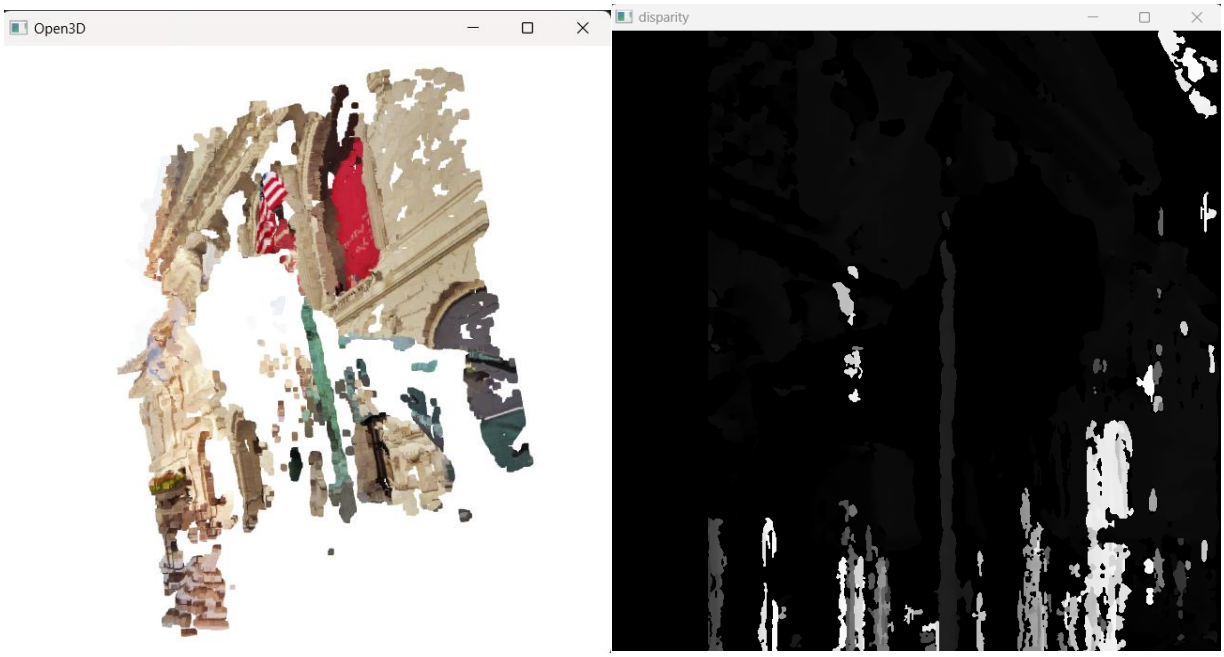


Отже, можна зробити висновок, що в даному випадку параметр window_size в першу чергу повпливав на фон. Відбулося це тому що цей параметр має безпосередній вплив на значення P1 та P2. Його зміна із дефолтного значення не принесла покращень, тож залишаємо його без змін: window_size = 3.

3) Тепер спробуємо поекспериментувати із параметром min_disp, Змінимо його із 16 на 32.

```
window_size = 3
min_disp = 32
num_disp = 112-min_disp
stereo = cv2.StereoSGBM_create(minDisparity = min_disp,
    numDisparities = num_disp,
    blockSize = 16,
    P1 = 8*3*window_size**2,
    P2 = 32*3*window_size**2,
    disp12MaxDiff = 1,
    uniquenessRatio = 10,
    speckleWindowSize = 100,
    speckleRange = 32
)
```

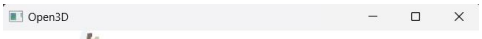
Бачимо, що за такого значення лев повністю зникає, що зовсім не є метою програми.



Зменшення min_disp до 8 також не принесло значних покращень. На зображеннях нижче можна помітити, що координати частини постаменту були розпізнані не зовсім коректно, через що точки постаменту були занадто сильно розтягнутими вглибину.

```
window_size = 3
min_disp = 8
num_disp = 112-min_disp
```

```
stereo = cv2.StereoSGBM_create(minDisparity = min_disp,
    numDisparities = num_disp,
    blockSize = 16,
    P1 = 8*3*window_size**2,
    P2 = 32*3*window_size**2,
    disp12MaxDiff = 1,
    uniquenessRatio = 10,
    speckleWindowSize = 100,
    speckleRange = 32
)
```

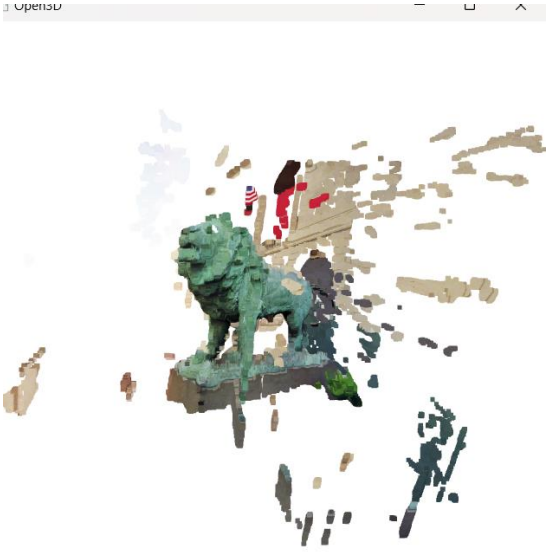



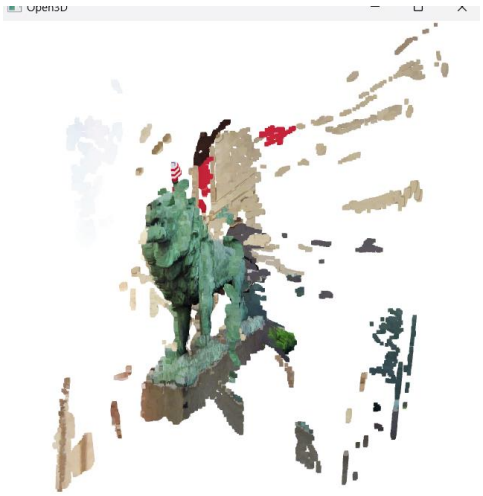
Отже, цей параметр мав вплив на розподіл глибин точок.

4) Спробуємо змінити num_disp: num_disp= 32 – min_disp = 32 – 16 = 16, при цьому залишивши всі інші параметри на тому самому рівні

```
window_size = 3
min_disp = 16
num_disp = 32 - min_disp
stereo = cv2.StereoSGBM_create(minDisparity=min_disp,
                               numDisparities=num_disp,
                               blockSize=16,
                               P1=8 * 3 * window_size ** 2,
                               P2=32 * 3 * window_size ** 2,
                               disp12MaxDiff=1,
                               uniquenessRatio=10,
                               speckleWindowSize=100,
                               speckleRange=32
                               )
```

Таке значення наразі виявилось найбільш оптимальним. Можемо переконатись в цьому, подивившись на постамент, на якому розташований лев – він став цілісним і перестав розділятись на велику кількість розірваних деталей. При цьому зник шум із заднього фону, що дає розгледіти статую краще. Глибина самої статуї також стала передаватись коректніше.





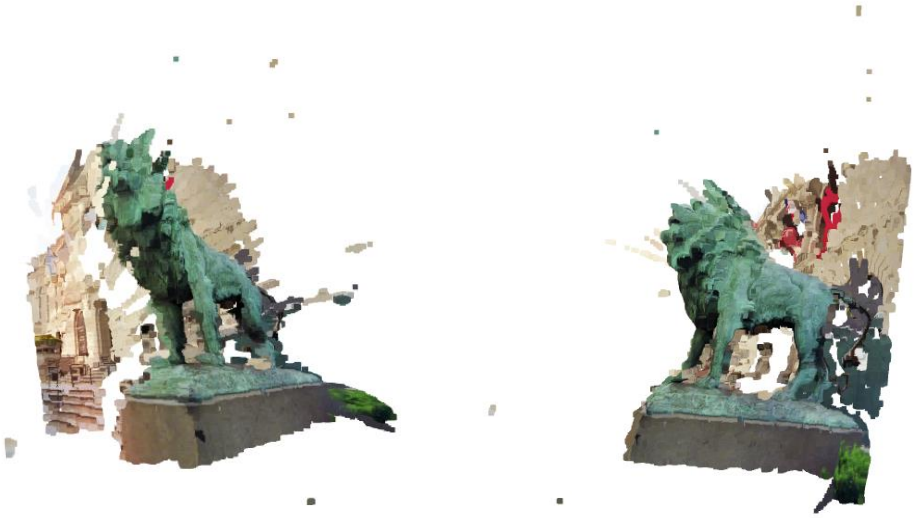
5) Після знаходження відносно дієвої комбінації параметрів, спробуємо внести зміни у blockSize, disp12MaxDiff, uniquenessRatio, speckleRange та фокусну відстань. Після цього знову підлаштуємо window_size, min_disp та num_disp. В ході декількох експериментів було знайдено таку **остаточну працюючу комбінацію**:

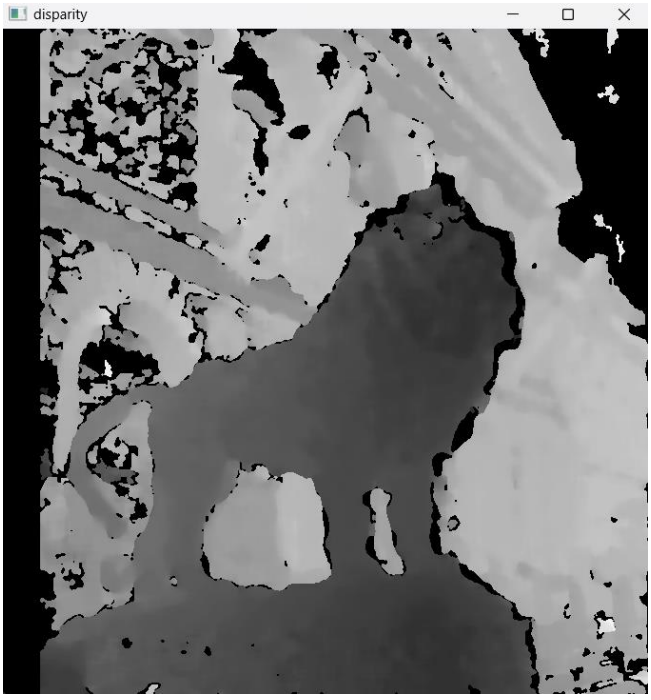
```
window_size = 1
min_disp = 20
num_disp = 41 - min_disp
stereo = cv2.StereoSGBM_create(minDisparity=min_disp,
                               numDisparities=num_disp,
                               blockSize=14,
                               P1=8 * 3 * window_size ** 2,
                               P2=32 * 3 * window_size ** 2,
                               disp12MaxDiff=120,
                               uniquenessRatio=10,
                               speckleWindowSize=100,
                               speckleRange=100
                               )

print('computing disparity...')
disp = stereo.compute(imgL, imgR).astype(np.float32) / 16.0

print('generating 3d point cloud...', )
h, w = imgL.shape[:2]
f = 1 * w  # guess for focal length
```

Як бачимо із зображень нижче, тепер частину фону видно також, але це не є критичним, адже тепер він цілісний і виглядає чистішим за всі попередні спроби. Також можна помітити, що зник стовпчик із точок із некоректним розміщенням, який до цього сильно псував загальне відображення. Постамент, на якому стоїть лев, також є цілісним і об’ємним. Загалом 3D зображення стало впізнаваним з усіх боків.





Програмний код

```
from __future__ import print_function
import numpy as np
import cv2
import open3d as o3d

ply_header = '''ply
format ascii 1.0
element vertex %(vert_num)d
property float x
property float y
property float z
property uchar red
```

```
property uchar green
property uchar blue
end_header
'''


def write_ply(fn, verts, colors):
    verts = verts.reshape(-1, 3)
    colors = colors.reshape(-1, 3)
    verts = np.hstack([verts, colors])
    with open(fn, 'wb') as f:
        f.write((ply_header % dict(vert_num=len(verts))).encode('utf-8'))
        np.savetxt(f, verts, fmt='%f %f %f %d %d %d ')


if __name__ == '__main__':
    print('loading images...')
    imgL = cv2.pyrDown(cv2.imread('lion-left.jpg'))  # downscale images for
faster processing
    imgR = cv2.pyrDown(cv2.imread('lion-right.jpg'))

    window_size = 1
    min_disp = 20
    num_disp = 41 - min_disp
    stereo = cv2.StereoSGBM_create(minDisparity=min_disp,
                                   numDisparities=num_disp,
                                   blockSize=14,
                                   P1=8 * 3 * window_size ** 2,
                                   P2=32 * 3 * window_size ** 2,
                                   disp12MaxDiff=120,
                                   uniquenessRatio=10,
                                   speckleWindowSize=100,
                                   speckleRange=100
                                   )

    print('computing disparity...')
    disp = stereo.compute(imgL, imgR).astype(np.float32) / 16.0

    print('generating 3d point cloud...', )
    h, w = imgL.shape[:2]
    f = 1 * w  # guess for focal length
    Q = np.float32([[1, 0, 0, -0.5 * w],
                    [0, -1, 0, 0.5 * h],  # turn points 180 deg around x-axis,
                    [0, 0, 0, -f],  # so that y-axis looks up
                    [0, 0, 1, 0]])
    points = cv2.reprojectImageTo3D(disp, Q)
    colors = cv2.cvtColor(imgL, cv2.COLOR_BGR2RGB)
    mask = disp > disp.min()
    out_points = points[mask]
    out_colors = colors[mask]
    out_fn = 'out.ply'
    write_ply('out.ply', out_points, out_colors)
    print('%s saved' % 'out.ply')

    cv2.imshow('left', imgL)
    cv2.imshow('disparity', (disp - min_disp) / num_disp)
    cv2.imshow('disparity', (disp - min_disp) / num_disp)

    print("Load a ply point cloud, print it, and render it")
```

```
    pcd = o3d.io.read_point_cloud("out.ply")
    print(pcd)
    print(np.asarray(pcd.points))
    o3d.visualization.draw_geometries([pcd], width=650, height=650, left=20,
top=20)

    cv2.waitKey()
    cv2.destroyAllWindows()
```

Висновки: виконавши лабораторну роботу №8, я отримала навички із перетворення 2D зображень стереопари у 3D. В рамках лабораторної роботи було розроблено скрипт, що ефективно реконструює 3D зображення статуї лева. Для досягнення ефективного результату також було проведено міні-дослідження щодо різних параметрів стерео перетворення та їхнього впливу. Методом експериментів було визначено найефективнішу комбінацію цих параметрів.