Politechnika Śląska

8 stycznia 2025,
Gliwice

Laboratorium ZAOWR

Rok akademicki	Termin	Rodzaj studiów	Kierunek	Prowadzący	Grupa
2024/2025	Czwartek, 10:00-13:15	Stacjonarne	Informatyka	Dr inż. Marcin Paszkuta Mgr inż. Mateusz Płonka	IGT

Ćwiczenie nr 4

Temat: Mapy głębi i chmury punktów

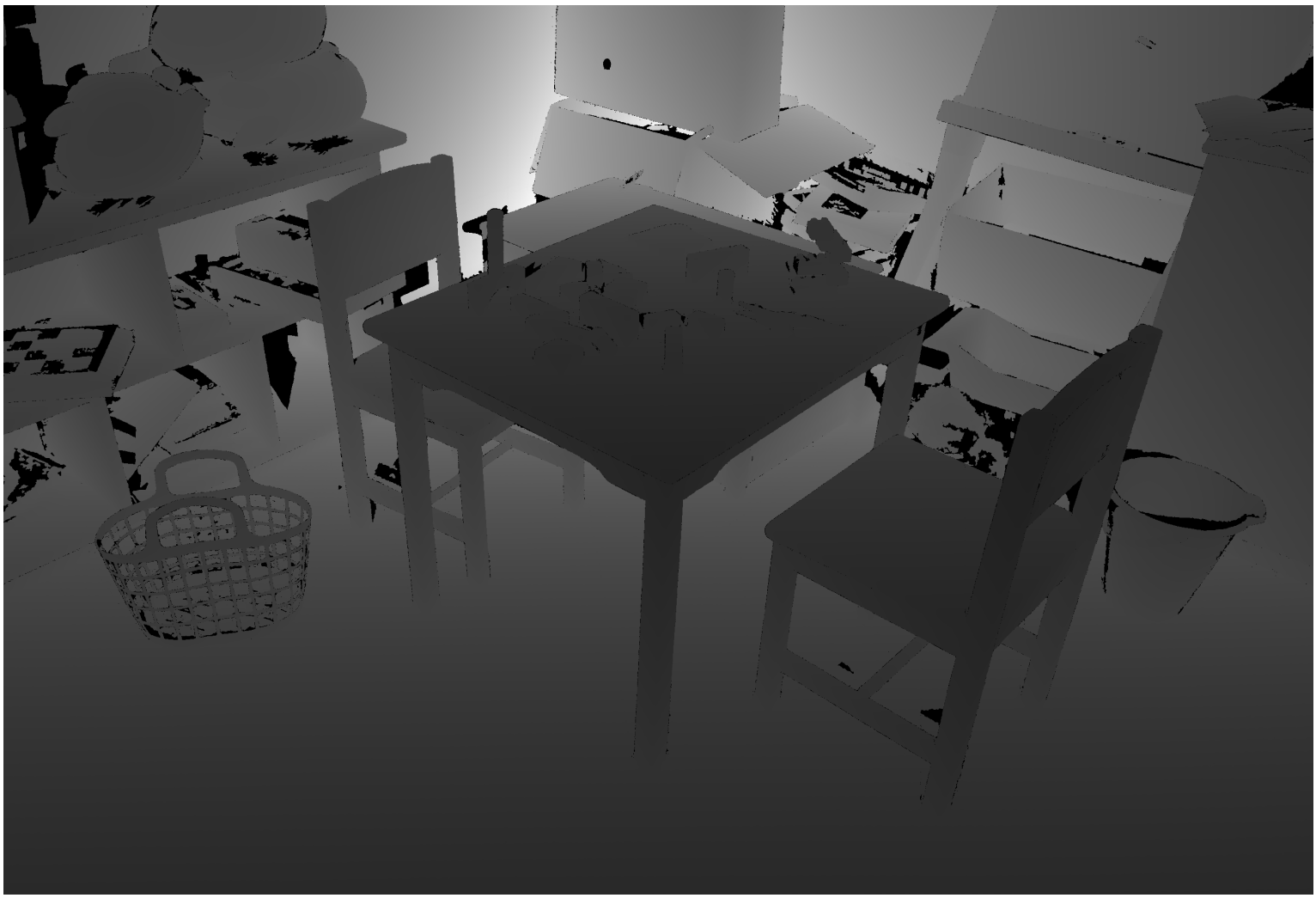
Informatyka, IGT

Maksymilian Kisiel

Spis treści

1 Zadanie 1 - Odczyt i konwersja mapy rozbieżności na mapę głębi

Aby wykonać to zadanie, należało odczytać referencyjną mapę rozbieżności i przekonwertować ją na mapę głębi. Stworzona mapa została znormalizowana do 8-bitowej skali szarości oraz zapisana za pomocą bezstratnego formatu **PNG** (Rys. 1). Poniżej znajduje się fragment kodu odpowiednich metod (Kod 1, pełny kod programu i funkcji wykonujących poszczególne zadania został dołączony do sprawozdania, a także jest dostępny w serwisie GitHub).



Rys. 1: Mapa głębi po konwersji

```python
def load_pfm_file(filePath: str = None) -> tuple[np.ndarray, float]:
    with open(filePath, "rb") as f:
        # Read the header
        header = f.readline().decode().rstrip()
        if header not in ["PF", "Pf"]:
            raise ValueError(Fore.RED + "\nNot a PFM file.\n")
        # Determine color (PF for RGB, Pf for grayscale)
        color = header == "PF"
        # Read width and height
        dims = f.readline().decode().strip()
        width, height = map(int, dims.split())
        # Read scale (endian and range)
        scale = float(f.readline().decode().strip())
        endian = "<" if scale < 0 else ">"
        scale = abs(scale)
        # Read the data
        data = np.fromfile(f, endian + "f")
        data = np.reshape(data, (height, width, 3) if color else (height, width))
        data = np.flipud(data)  # Flip vertically due to PFM format
        data = np.array(data)
        data[np.isinf(data)] = 0
        return data, scale

def disparity_to_depth_map(
        disparityMap: np.ndarray,
        baseline: float,
        focalLength: float,
        aspect: float = 1000.0
) -> np.ndarray:
    disparityMap = np.float32(disparityMap)
    # Adjust disparity with offset
    validDisparity = disparityMap > 0  # Only consider valid disparity values
    adjustedDisparity = disparityMap
    depthMap = np.zeros_like(disparityMap)
    depthMap[validDisparity] = baseline * focalLength /
    adjustedDisparity[validDisparity]
    depthMap = depthMap / aspect # to meters
    return depthMap

def depth_map_normalize(
        depthMap: np.ndarray,
        normalizeDepthMapRange: str = "8-bit"
) -> np.ndarray:
    depthMapNormalized = None
    if normalizeDepthMapRange == "8-bit":
        # Normalize depth map to 8-bit grayscale (0-255)
        depthMapNormalized = cv.normalize(depthMap, None, 0, 255, cv.NORM_MINMAX)
        depthMapNormalized = depthMapNormalized.astype(np.uint8)
    return depthMapNormalized

def display_img_plt(
        img: np.ndarray,
        pltLabel: str = 'Map',
        show: bool = False,
        save: bool = False,
        savePath: str = None,
        cmap: str = 'gray'
) -> None:
    plt.figure(figsize=(20, 10))
    plt.imshow(img, cmap=cmap)
    plt.title(pltLabel)
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    if save:
        plt.savefig(savePath, format='png', bbox_inches='tight')
        print(Fore.GREEN + f"\nPlot successfully saved at {savePath}")
        savePathCV2 = savePath.replace(".png", "_2.png")
        cv2.imwrite(savePathCV2, img)
        print(Fore.GREEN + f"\nOpenCV RAW image successfully saved at {savePathCV2}")
    if show:
        plt.show()
    plt.close()
```

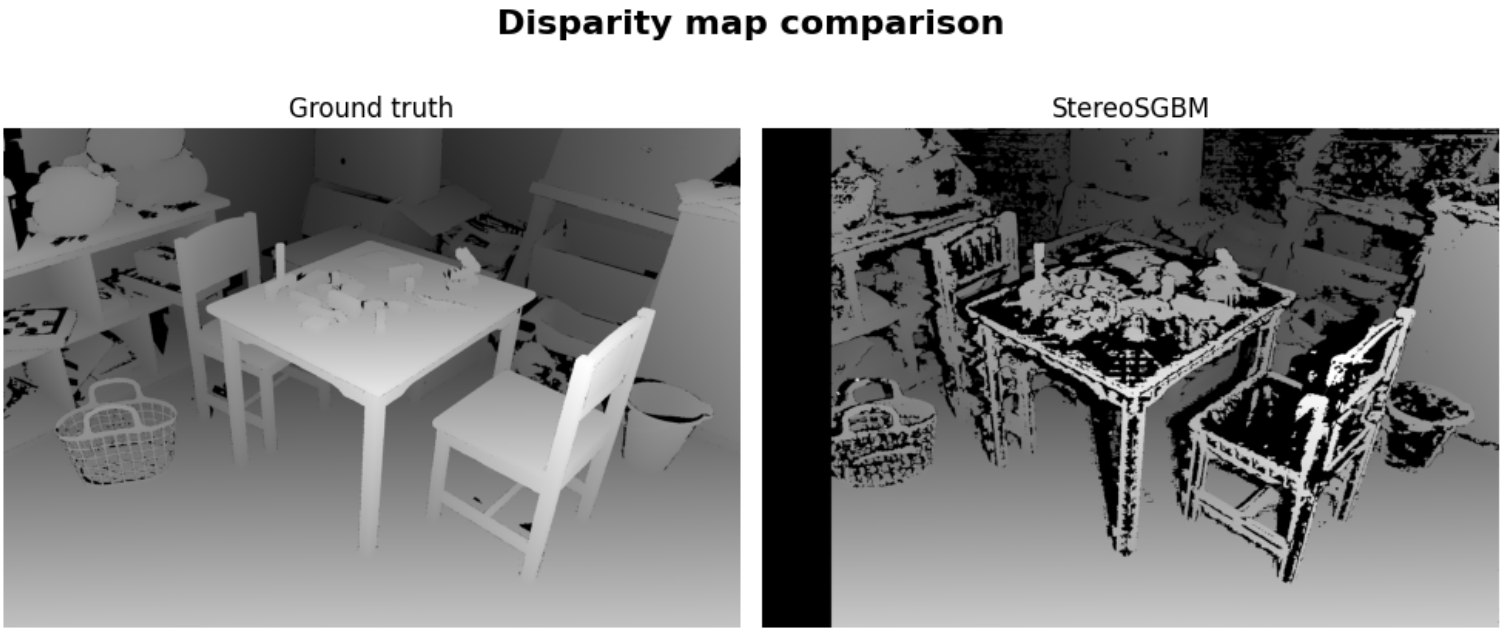
Kod 1: Fragment kodu funkcji odczytu, konwersji, normalizacji i zapisu

2 Zadanie 2 - Wyznaczanie mapy dysparycji i jej konwersja za pomocą danych kalibracyjnych

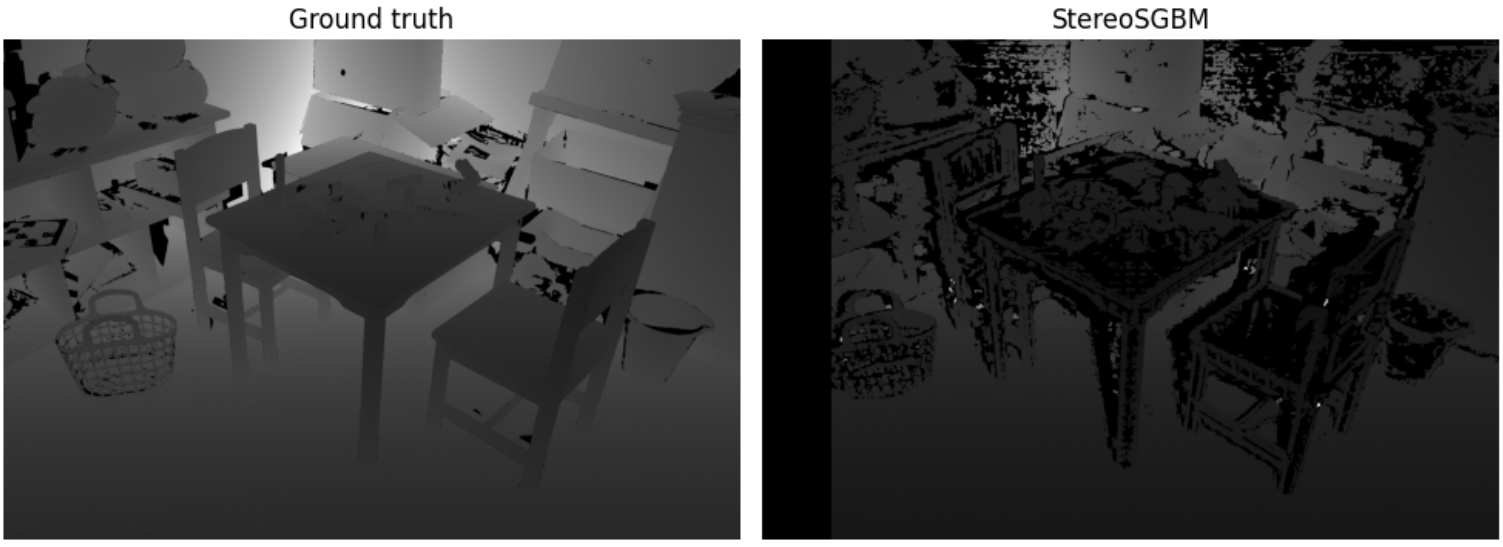
W tym zadaniu wykorzystałem kod do wyznaczania mapy dysparycji, który został przygotowany na potrzeby wcześniejszego laboratorium (funkcja *calculate_disparity_map()*). Wybrany przeze mnie algorytm to **StereoSGBM**. Poniżej znajduje się fragment kodu (Kod 2) wywołujący dane funkcje (przedstawione już wcześniej - Kod 1). Wyniki działania programu widoczne są na kolejnej stronie (Rys. 2 i 3).

```
def load_depth_map_calibration(calibFile: str) -> DepthCalibrationParams:
    # Parse the calibration data
    calibrationData = {}
    for line in lines:
        key, value = line.strip().split("=")
        key = key.strip()
        value = value.strip()
        if key.startswith("cam"):  # cam0 or cam1
            rows = value.strip("[]").split(";")
            calibrationData[key] = [[float(x) for x in row.split()] for row in rows]
        elif key in {"doffs", "baseline", "dyavg", "dymax", "vmin", "vmax"}:
            calibrationData[key] = float(value)  # Float values
        elif key in {"width", "height", "ndisp", "isint"}:
            calibrationData[key] = int(value)  # Integer values
        else:
            calibrationData[key] = value  # Fallback for unexpected keys
    if "cam0" in calibrationData:
        cam0FirstRow = calibrationData["cam0"][0]
        focalLength = cam0FirstRow[0]  # Extract the first element of the first row
        calibrationData["focalLength"] = focalLength
    return calibrationData  # Type-hinted as DepthCalibrationParams

################################################################################

calibrationParams = zw.load_depth_map_calibration(calibFile=calibrationFile)

disparityMapSGBM = zw.calculate_disparity_map(
    leftImagePath=img_left_path,
    rightImagePath=img_right_path,
    blockSize=9,
    numDisparities=256,
    minDisparity=0,
    disparityCalculationMethod="sgbm",
    normalizeDisparityMap=False,
)

disparityMapSGBM = disparityMapSGBM.astype(np.float32) / 16.0

depthMapSGBM = zw.disparity_to_depth_map(
    disparityMap=disparityMapSGBM,
    baseline=calibrationParams["baseline"],
    focalLength=calibrationParams["focalLength"],
    aspect=1000.0
)
depthMapSGBM_8bit = zw.depth_map_normalize(
    depthMap=depthMapSGBM,
    normalizeDepthMapRange="8-bit"
)
zw.display_img_plt(
    img=disparityMapSGBM,
    pltLabel="Disparity map",
    save=True,
    savePath=ex_2_disparityMapPath
)
zw.display_img_plt(
    img=depthMapSGBM_8bit,
    pltLabel="Depth map",
    save=True,
    savePath=ex_2_depthMapPath,
)
```

Kod 2: Fragment kodu odpowiedzialny za wywołanie odpowiednich funkcji



Rys. 2: Mapa dysparycji



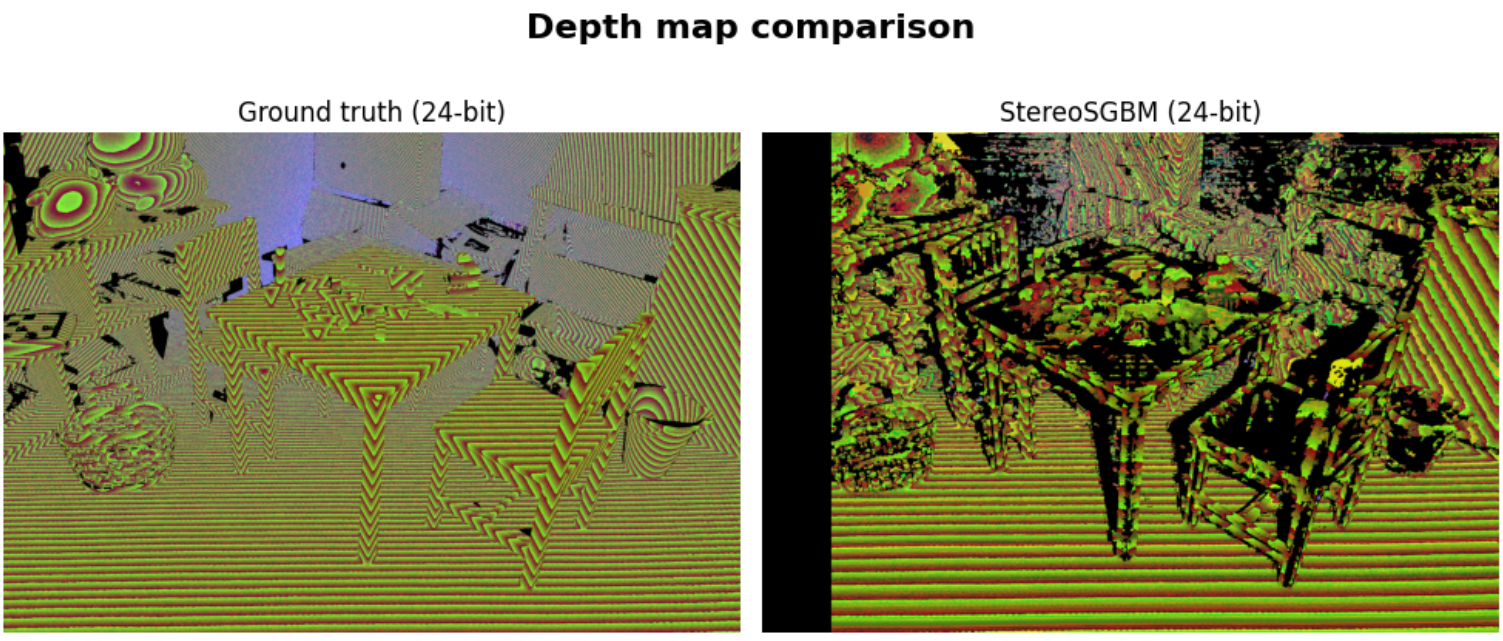
Rys. 3: Mapa głębi

3 Zadanie 3 - Zapis mapy głębi w formacie 24-bitowym

W tym zadaniu wykorzystano przygotowaną wcześniej funkcję do normalizacji i rozszerzono jej funkcjonalność do normalizacji 24-bitowej (Kod 3). Do zapisu wykorzystano wspomnianą wcześniej funkcję *display_img_plt()*.

```
def depth_map_normalize(
        depthMap: np.ndarray,
        normalizeDepthMapRange: str = "8-bit"
) -> np.ndarray:
    depthMapNormalized = None
    if normalizeDepthMapRange == "8-bit":
        # Normalize depth map to 8-bit grayscale (0-255)
        depthMapNormalized = cv.normalize(depthMap, None, 0, 255, cv.NORM_MINMAX)
        depthMapNormalized = depthMapNormalized.astype(np.uint8)
    elif normalizeDepthMapRange == "24-bit":
        # Normalize depth map to 24-bit color (0-16777215)
        depthMapNormalized = (cv.normalize(depthMap, None, 0, (256 ** 3 - 1),
    cv.NORM_MINMAX)).astype(np.uint32)
        # Extract RGB channels
        R = (depthMapNormalized & 0xFF).astype(np.uint8)
        G = ((depthMapNormalized >> 8) & 0xFF).astype(np.uint8)
        B = ((depthMapNormalized >> 16) & 0xFF).astype(np.uint8)
        # Combine channels into an RGB image
        depthMapNormalized = np.stack((R, G, B), axis=-1)
    return depthMapNormalized
################################################################################
depthMap_24bit = zw.depth_map_normalize(
    depthMap=depthMap, # Ground truth
    normalizeDepthMapRange="24-bit"
)
depthMapSGBM_24bit = zw.depth_map_normalize(
    depthMap=depthMapSGBM, # SGBM
    normalizeDepthMapRange="24-bit"
)
```

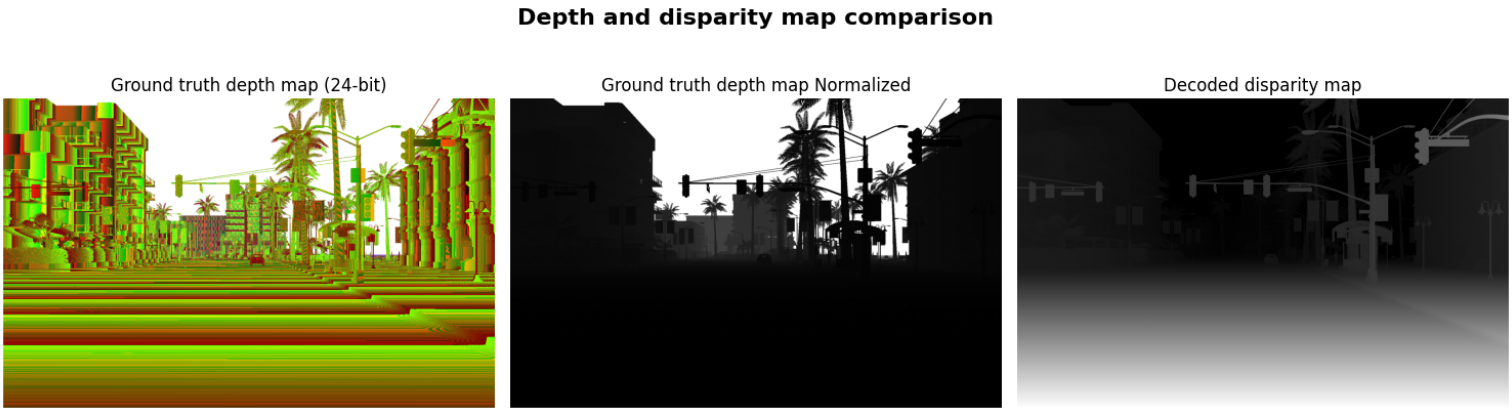
Kod 3: Fragment kodu odpowiedzialny za normalizację (po modyfikacji)



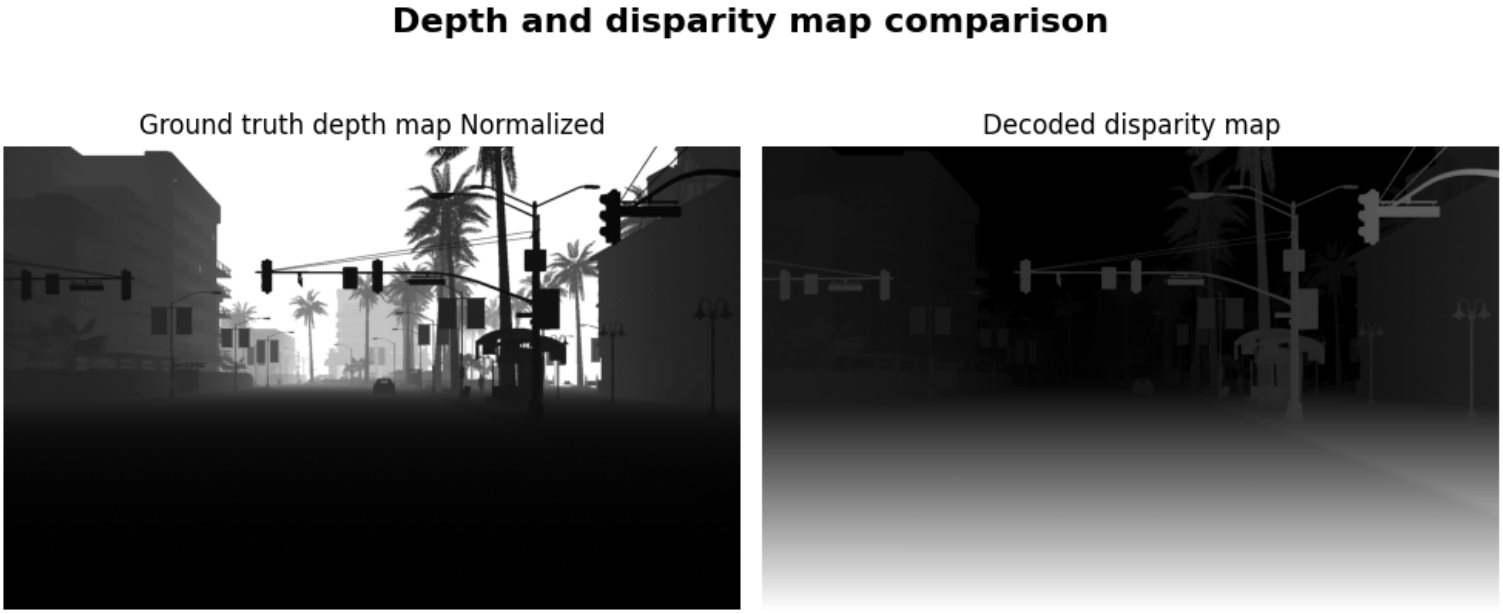
Rys. 4: Porównanie map głębi z kodowaniem 24-bitowym

4 Zadanie 4 - Wyznaczanie mapy dysparycji na podstawie mapy głębi

W tym zadaniu zaimplementowałem funkcje, które dekodują referencyjną mapę głębi, konwertują ją na mapę dysparycji, którą następnie normalizują do 8-bitowej szarości. Kod tych funkcji znajduje się poniżej (Kod 4), wraz z rysunkami przedstawiającymi otrzymane wyniki (Rys. 5 i 6).



Rys. 5: Przedstawienie map głębi i dysparycji wraz z referencyjną mapą głębi



Rys. 6: Przedstawienie map głębi i dysparycji

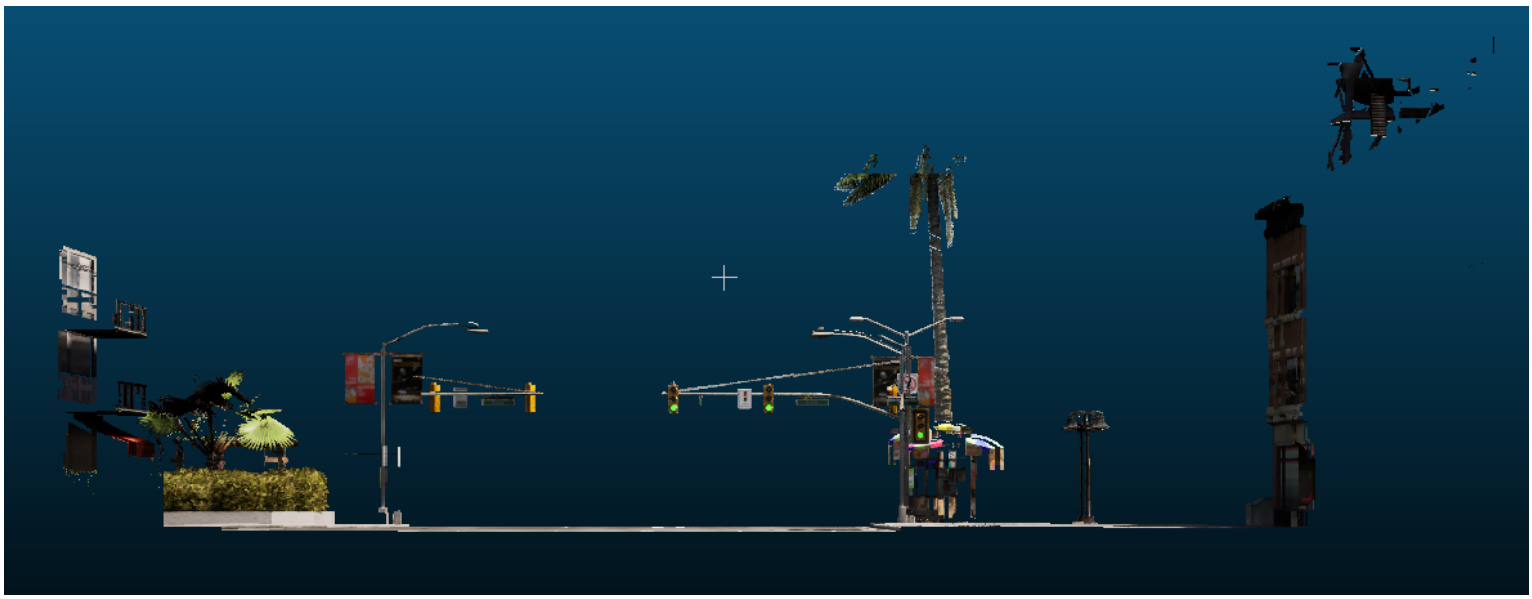
```python
def decode_depth_map(
        depthMap: np.ndarray,
        maxDepth: float = 1000.0,
        decodeDepthMapRange: str = "24-bit"
) -> np.ndarray:
    depthMapDecoded = None
    if decodeDepthMapRange == "8-bit":
        # Decode 8-bit depth map
        depthMap = depthMap[:, :, 0]
        depthMap = depthMap[:, :, ::-1]
        # Decode 8-bit depth map
        depthMapDecoded = (depthMap / 255) * maxDepth
    elif decodeDepthMapRange == "24-bit":
        # Decode 24-bit depth map
        depthMap = depthMap[:, :, :3]
        depthMap = depthMap[:, :, ::-1]
        # Decode 24-bit depth map
        R = (depthMap[:, :, 0]).astype(np.uint32)
        G = (depthMap[:, :, 1]).astype(np.uint32) * 256
        B = (depthMap[:, :, 2]).astype(np.uint32) * 256 ** 2
        depthMapDecoded = ((R + G + B) / (2 ** 24 - 1)) * maxDepth
    return depthMapDecoded

def disparity_map_normalize(
        disparityMap: np.ndarray,
        normalizeDisparityMapRange: str = "8-bit"
) -> np.ndarray:
    disparityMapNormalized = None
    if normalizeDisparityMapRange == "8-bit":
        # Normalize disparity map to 8-bit grayscale (0-255)
        disparityMapNormalized = cv.normalize(disparityMap, None, 0, 255,
    cv.NORM_MINMAX)
        disparityMapNormalized = disparityMapNormalized.astype(np.uint8)
    return disparityMapNormalized

def depth_to_disparity_map(
        depthMap: np.ndarray,
        baseline: float,
        focalLength: float,
        minDepth: float = 0.001,
        normalizeDisparityMapRange: str = "8-bit"
) -> np.ndarray:
    # depthMap = np.nan_to_num(depthMap)
    depthMap = np.maximum(depthMap, minDepth)
    disparityMap = (baseline * focalLength) / depthMap
    disparityMapNormalized = disparity_map_normalize(disparityMap,
    normalizeDisparityMapRange)
    return disparityMapNormalized

################################################################################

hFOV = 60
baseline = 0.1 # meters
maxDepth = 1000.0 # meters
depthMap_uint24 = cv2.imread(deptMapRef_24bit, cv2.IMREAD_UNCHANGED)
focalLength = (depthMap_uint24.shape[0] / 2) / np.tan(np.radians(hFOV) / 2)
# print(f"{type(focalLength) = }") # <class 'numpy.float64'>

depthMap_ex4 = zw.decode_depth_map(
    depthMap=depthMap_uint24,
    maxDepth=maxDepth,
    decodeDepthMapRange="24-bit",
)

disparityMap_ex4 = zw.depth_to_disparity_map(
    depthMap=depthMap_ex4,
    baseline=baseline,
    focalLength=focalLength,
    minDepth=0.0,
)
```

Kod 4: Fragment kodu odpowiedzialny za dekodowanie, konwersję i normalizację

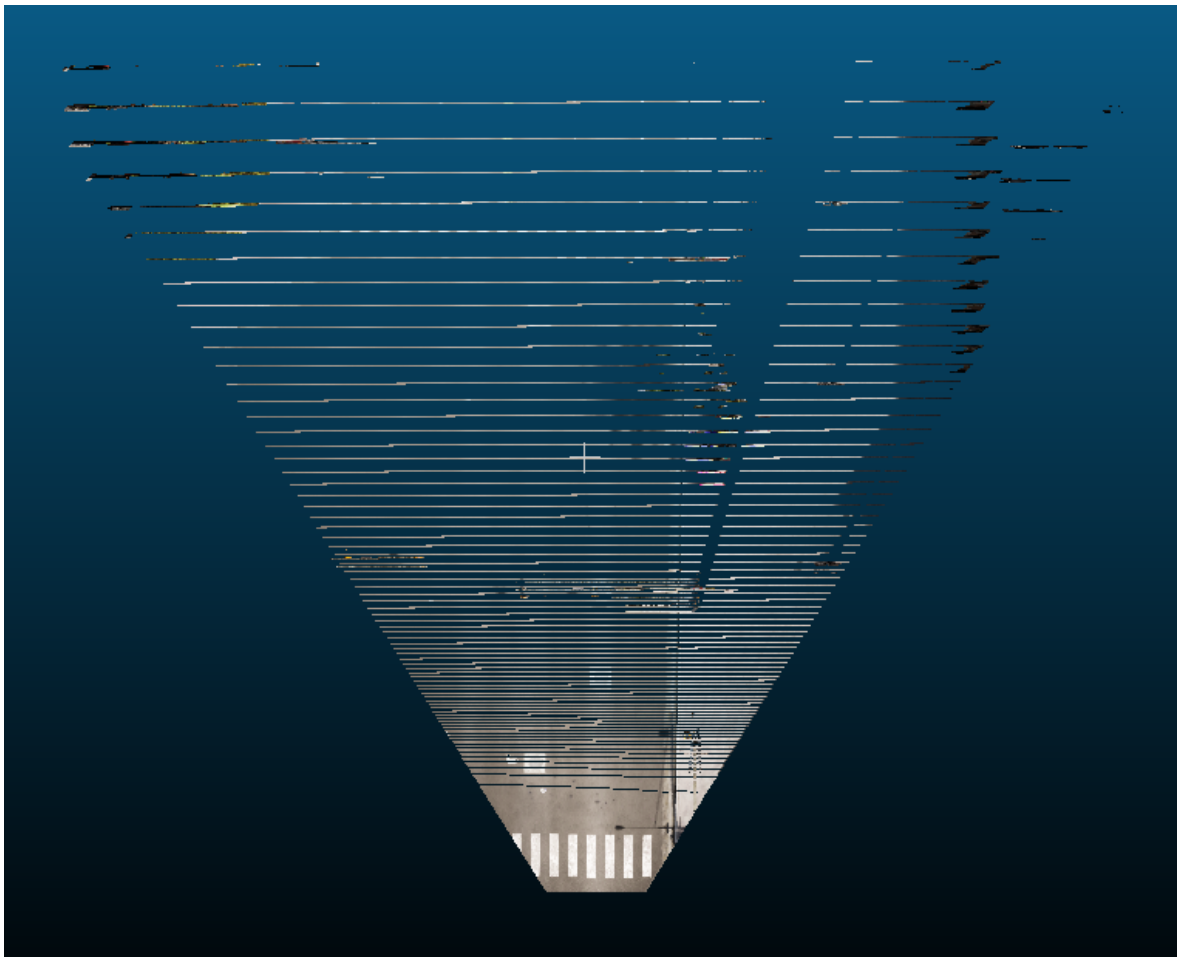
5 Zadanie 5 - Konwersja mapy głębi na kolorową chmurę punktów

W tym zadaniu wykorzystano referencyjną mapę głębi oraz wyznaczoną na jej podstawie mapę dysparycji. Dzięki mapom wyznaczono punkty reprojekcji oraz maskę, która pozwala na ograniczanie głębokości, do której chcemy wyznaczyć chmurę punktów (< 50 m). Wyznaczona chmura zapisywana jest w formacie **PLY**. Po jej wygenerowaniu użyto programu **CloudCompare**, aby wizualizować wyniki i wykonać zrzuty ekranu po ustawieniu odpowiedniego widoku.

Poniżej znajduje się fragment funkcji wyznaczający punkty i zapisujący je w formacie **PLY** (Kod 5), a także otrzymane wyniki (*z1c* - *z3c*, Rys. 7, 8 i 9).



Rys. 7: Widok chmury punktów **na wprost, z1c**



Rys. 8: Widok chmury punktów **z góry, z2c**

Rys. 9: Widok chmury punktów **z dowolnej perspektywy, z3c**

```
def write_ply_file(
        fileName: str,
        verts: np.ndarray,
        colors: np.ndarray
) -> None:

    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
    '''

    verts = verts.reshape(-1, 3)
    colors = colors.reshape(-1, 3)
    verts = np.hstack([verts, colors])
    with open(fileName, '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 ')

################################################################################

img = cv2.imread(imgPath, cv2.IMREAD_COLOR)
disparityMap_ex5 = cv2.imread(ex_4_disparityMapPath, cv2.IMREAD_GRAYSCALE)
depthMap_ex5 = cv2.imread(ex_4_depthMapPath, cv2.IMREAD_GRAYSCALE)

h, w = depthMap_ex5.shape[:2]
if img.shape[:2] != (h, w):
    img = cv2.resize(img, (w, h), interpolation=cv2.INTER_AREA)

if disparityMap_ex5.shape[:2] != (h, w):
    disparityMap_ex5 = cv2.resize(
        disparityMap_ex5, (w, h), interpolation=cv2.INTER_AREA
    )

f = 0.8 * w  # 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(disparityMap_ex5, Q)
colors = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
mask = depthMap_ex5 < 50

outPoints = points[mask]
outColors = colors[mask]

zw.write_ply_file(
    fileName=plyPath,
    verts=outPoints,
    colors=outColors,
)
```

Kod 5: Fragment kodu odpowiedzialny za wyznaczanie chmury punktów i jej zapis do pliku **PLY**

6 Kod

Przygotowana na potrzeby tego laboratorium paczka, została zaktualizowana, aby wykonywać wszystkie wymagane zadania z tego laboratorium:

- wczytywanie danych kalibracyjnych z pliku,
- wczytywanie referencyjnej mapy rozbieżności,
- wyznaczanie mapy głębi na podstawie mapy dysparycji,
- wyznaczanie mapy dysparycji na podstawie mapy głębi,
- normalizacja mapy głębi,
- normalizacja mapy dysparycji,
- zapis otrzymanych map,
- dekodowanie 24-bitowej mapy głębi i wyznaczanie mapy dysparycji na jej podstawie,
- tworzenie kolorowej chmury punktów za pomocą map,
- zapis chmury punktów w formacie PLY.

Kod paczki jest dostępny w serwisie GitHub, jednak kopia kodu źródłowego została załączona do sprawozdania w celu łatwiejszej weryfikacji wyników.