Coursework 3 - Report

Iva Jorgusheska, UID: 11114620

April 7, 2025

1 Part 1: Stereo Imagery

1.1 Focal Length Calculation

To determine the focal length of the cameras in millimetres, I begin with the focal length in pixels, which is obtained from the camera intrinsic matrix. From the data provided, the focal length for both camera 0 and camera 1 is:

$$f_{\text{pixels}} = 5806.559 \text{ pixels}$$

We are also given the following specifications for the camera sensor:

- Physical sensor width: 22.2 mm
- Original image width (horizontal resolution): 3088 pixels

Using these, I first compute the pixel size in millimetres:

$$\text{Pixel size} = \frac{\text{Sensor width (mm)}}{\text{Image width (pixels)}} = \frac{22.2}{3088} \approx 0.007188 \,\text{mm/pixel}$$

I now convert the focal length from pixels to millimetres by multiplying it with the pixel size:

$$f_{\text{mm}} = f_{\text{pixels}} \times \text{Pixel size} = 5806.559 \times 0.007188 \approx 41.74 \,\text{mm}$$

Final Result: The focal length the cameras were set to is approximately **41.74 mm**.

1.1.1 Camera Resolution Clarification

The original images were 2960×2016, and then they have been rescaled from the original resolution. The correct resolution to use for focal length conversion is the native sensor resolution, which is:

`Canon EOS 450D in 6 MP mode:` 3088×2056 `pixels`

This resolution aligns with the physical sensor dimensions of 22.2×14.8 mm and is used for accurate camera calibration computations.

1.2 Disparity Map

To compute the disparity map, I used the provided `disparity.py` script as a foundation, modifying the `getDisparityMap()` function to test both greyscale and edge-detected versions of the input stereo images. The function returns a floating point image where each pixel's value corresponds to the computed disparity, indicating the horizontal shift needed to match pixels between the left and right views.

Two critical parameters influence the quality of the disparity map:

- **numDisparities:** This parameter must be divisible by 16 and determines the maximum disparity range considered when searching for matching pixels. A higher value allows the detection of closer objects more accurately but increases computational cost and may introduce more noise. According to my experiments, the minimum `numDisparities` value required to obtain a usable result was 64. Lower values resulted in incomplete reconstructions, especially on the left side of the image, where matching pixels require larger shifts. This made it difficult for the algorithm to establish correspondences for foreground objects.

- **blockSize:** This odd-valued parameter defines the size of the window used for matching blocks between the two images. A smaller block size captures more fine detail but can result in noisy or mismatched disparities. Larger values produce smoother maps but may blur fine edges. In practice, I found that a block size of 5 provided the best balance between detail and smoothness.

To facilitate experimentation, I implemented real-time parameter tuning using trackbarsfor both `numDisparities` and `blockSize`, which allowed immediate visual feedback for different parameter configurations. Also I have a button for whether to use "edge image".

I was able to obtain good results where we get clear outline of the umbrellas with multiple parameter configurations.

Using a greyscale version of the input images with `numDisparities` = 64 and `blockSize` = 5 yielded combination clearly outlined the umbrellas, but with lot of noise in the background. Increasing the block size beyond this made the outlines excessively thick and less precise. In contrast, using edge-detected images instead of greyscale resulted in worse performance: not all umbrella edges were retained in the disparity map, and increasing the block size further thickened the outlines without improving structure.

The superior performance with greyscale input can be attributed to the richer intensity variation across the scene, which helps the block-matching algorithm better discriminate between regions. Edge-detected images, while highlighting prominent contours, discard much of the contextual information necessary for robust correspondence, leading to degraded disparity estimation.

Below I will show the results for both using edge-detected and the greyscale image, with varying the parameters on the trackbars.

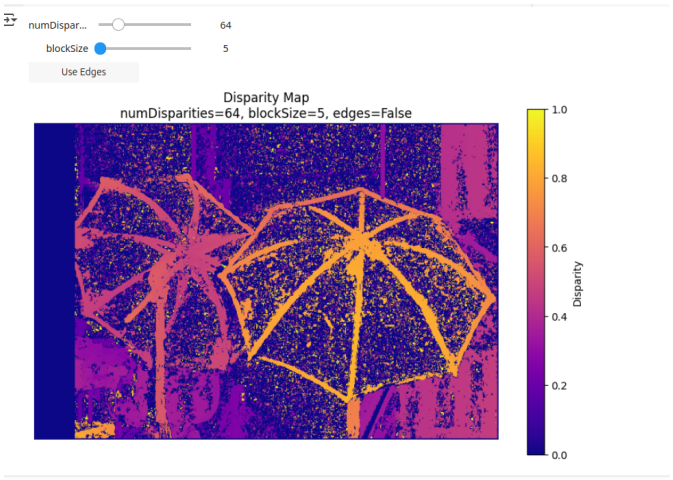

Figure 1: Normalised disparity map for the umbrella stereo images using greyscale input with `numDisparities` = 64 and `blockSize` = 5.

Overall, the disparity map using greyscale images yielded clearer object structures with acceptable noise levels, confirming it as the preferred input type for this task.

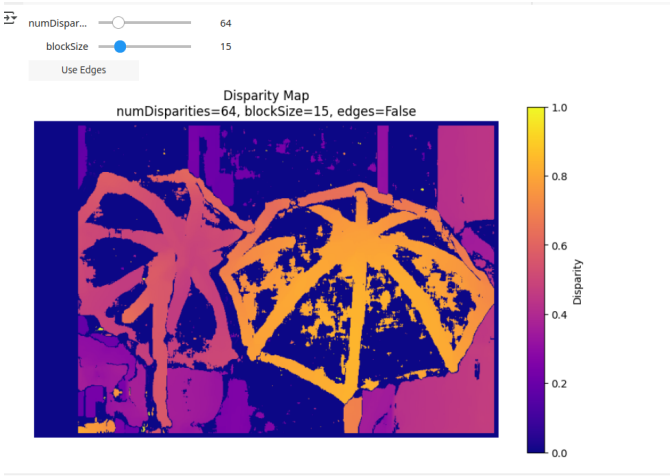

Figure 2: Normalised disparity map for the umbrella stereo images using greyscale input with `numDisparities` = 64 and `blockSize` = 15.

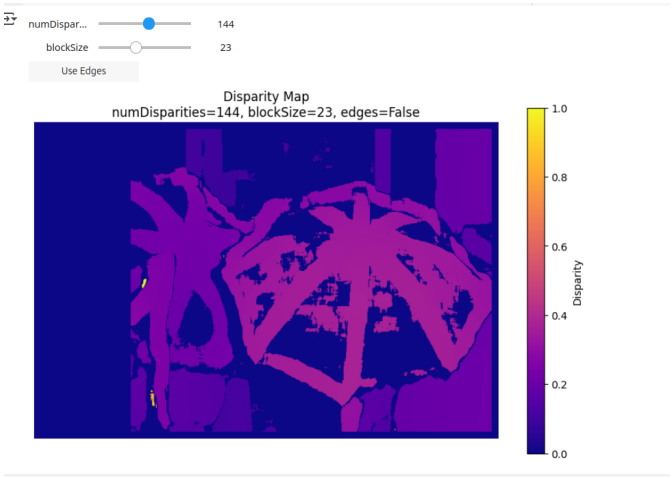

Figure 3: Normalised disparity map for the umbrella stereo images using greyscale input with `numDisparities` = 144 and `blockSize` = 23.

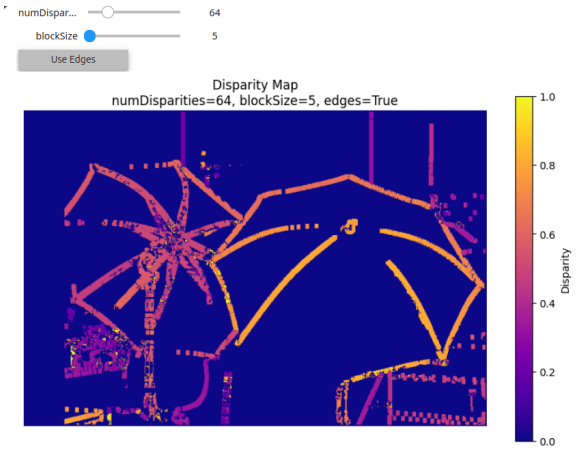

Figure 4: Normalised disparity map for the umbrella stereo images using EDGE-DETECTED input with `numDisparities` = 64 and `blockSize` = 5.

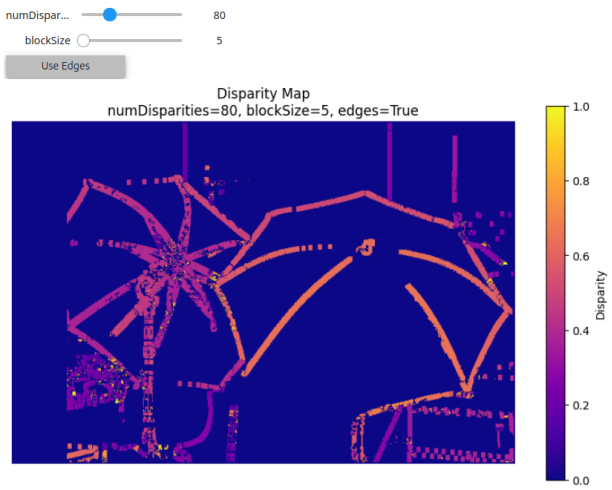

Figure 5: Normalised disparity map for the umbrella stereo images using EDGE-DETECTED input with `numDisparities` = 80 and `blockSize` = 5.

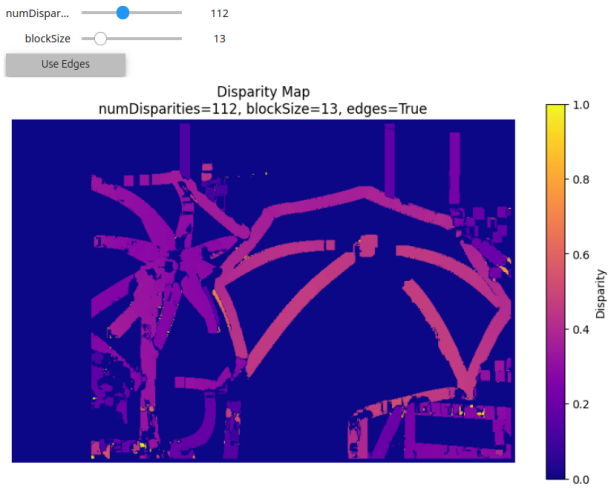

Figure 6: Normalised disparity map for the umbrella stereo images using EDGE-DETECTED input with `numDisparities` = 112 and `blockSize` = 13.

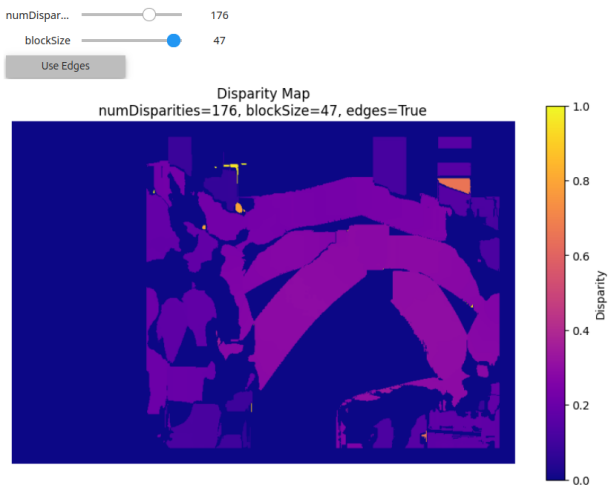

Figure 7: Normalised disparity map for the umbrella stereo images using EDGE-DETECTED input with `numDisparities` = 174 and `blockSize` = 35.

1.3 Views of the Scene

To reconstruct the 3D structure of the scene from the stereo image pair, I first computed a disparity map and then used triangulation to recover the real-world coordinates of each pixel. The depth into the scene (Z) is calculated using the formula:

$$Z = \frac{\text{baseline} \times f}{d + d_{\text{offset}}}$$

where **baseline** is the distance between the stereo cameras in mm, **f** is the focal length in pixels, **d** is the disparity at a given pixel, and $d_{\textbf{offset}}$ is used to avoid division by zero or very small values.

Implementation Details:

- I used the grayscale versions of `umbrellaL.png` and `umbrellaR.png`, resized to `740 × 505` pixels for faster processing.
- All intrinsic camera parameters (focal length f, principal points c_x, c_y, and d_{offset}) were scaled appropriately using the resize ratio.
- A point cloud was generated by looping through each valid pixel in the disparity map and computing its (X, Y, Z) coordinates using the pinhole camera model.

I filtered out disparity values that were too small or too large (in our experiments, below 20 and above 60), which corresponded to unreliable depth estimates due to occlusions or noise.

Visualization:

- I plotted the resulting (X, Y, Z) coordinates using `matplotlib`'s 3D scatter plot with color-mapping by depth (Z).
- Additionally, I generated 2D projections of the point cloud:
  - **Top-down view** of the scene using (X, Z).
  - **Side view** of the scene using (Y, Z).
- The `ax.view_init` function was used to set an appropriate viewpoint for the 3D plot (elevation = 20, azimuth = -70).

Result and Observations: The 3D plot reveals the structure of the umbrellas and surrounding objects in depth. The top-down and side views give a clear impression of the spatial layout of the scene. Points with invalid or extreme disparity values were excluded to improve the quality of the reconstruction. The resulting point cloud is sparse but accurately captures the geometry of prominent features.

The side view (Y-Z) of the reconstructed scene appears slightly noisier and less informative for one of the umbrellas than the top-down or 3D views. This is because disparity primarily varies along the horizontal axis, so vertical depth resolution is inherently limited, and small pixel inaccuracies are amplified in the Y-axis due to scaling by depth. Additionally, the image content lacks strong vertical structure, leading to overlapping points and poor spatial separation in the side projection.

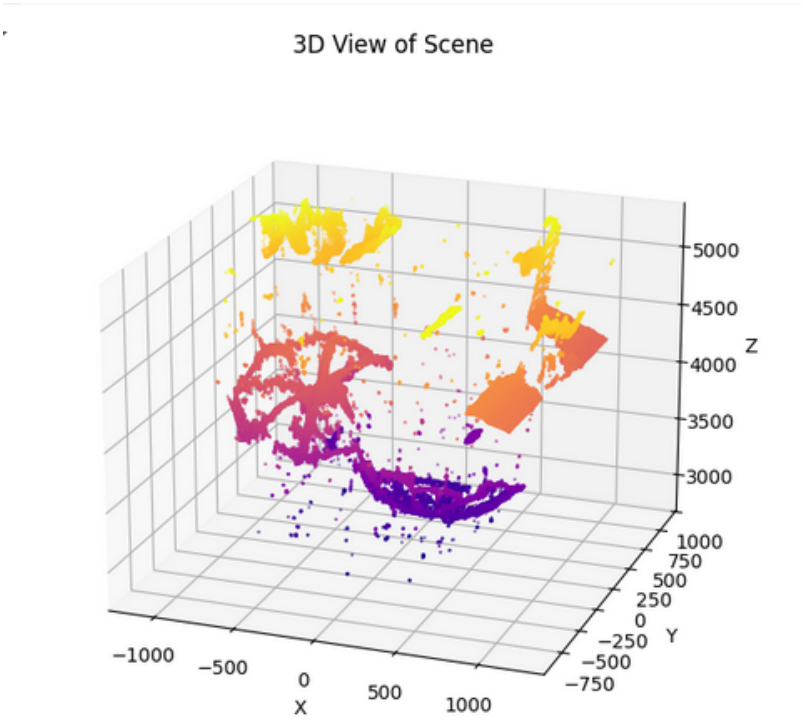

Figure 8: 3D view of reconstructed scene.

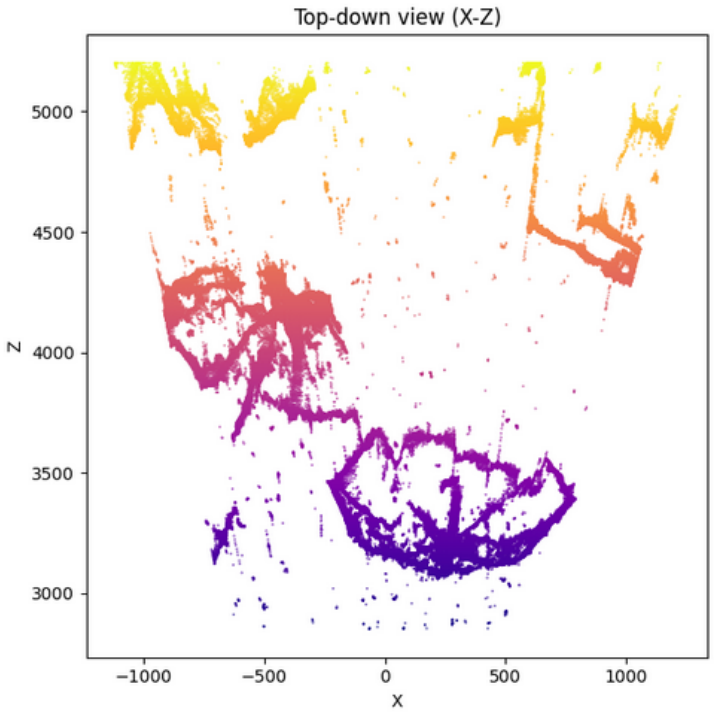

Figure 9: Top-down view (X-Z).

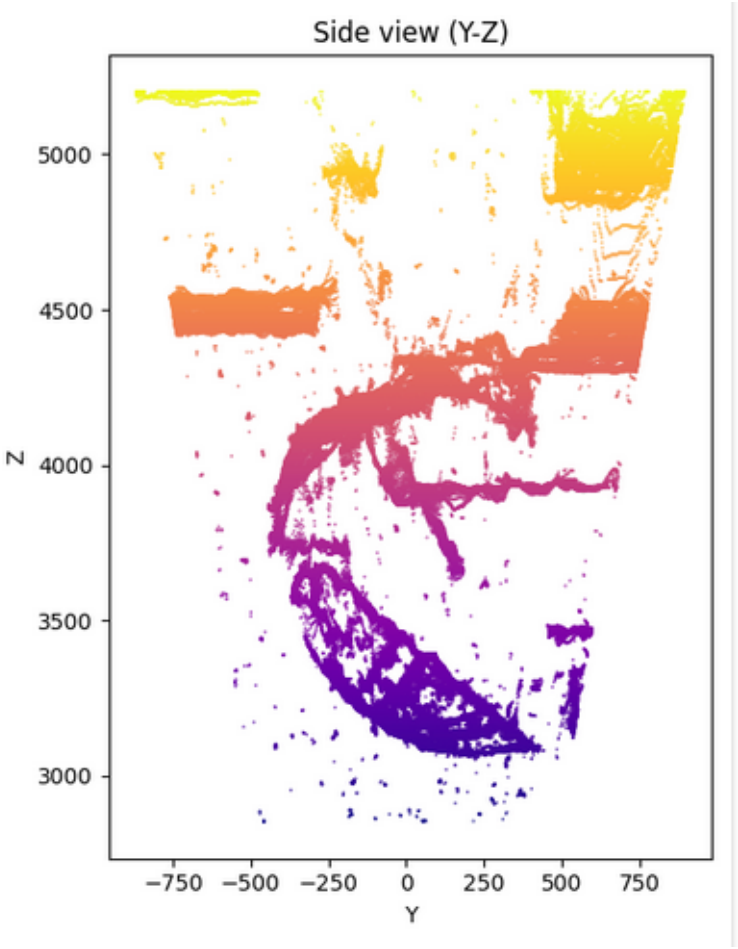

Figure 10: Side view (Y-Z).

2 Part 2: Selective Focus

In this task, we replicate the selective focus effect found in portrait mode on smartphones, where the foreground subject remains sharp and the background is heavily blurred. This is achieved by estimating relative depth from a stereo image pair and using it to selectively blend a blurred or greyscale background with the original color image.

2.1 Depth Estimation and Foreground Segmentation

The disparity map was computed using OpenCV's `StereoBM` algorithm on grayscale versions of the input stereo pair. A set of trackbars was implemented to allow real-time adjustment of key parameters: `numDisparities`, `blockSize`, the depth offset constant k, and the foreground segmentation `threshold`.

Each of these parameters plays an important role in refining the final result:

`numDisparities`: Controls the maximum disparity considered. This parameter was the most difficult to tune, as a value too high leads to excessive blurring and loss of detail, while a value too low results in a noisy and unreliable depth map.

`blockSize`: Determines the local window size used for matching. Increasing the block size leads to a smoother and more coherent disparity map, which in turn produces a cleaner and more natural-looking blur.

k: Used in the depth calculation formula depth $= \frac{1}{\text{disparity}+k}$. Decreasing k can help suppress noisy, low-disparity regions that would otherwise remain unblurred, enhancing the separation between foreground and background.

`threshold`: Applied on the depth map to segment the foreground subject. Proper tuning of this threshold is essential for accurate subject isolation.

The resulting disparity map was normalized and smoothed using a Gaussian filter to reduce artifacts. The computed depth map was then thresholded and further refined with morphological operations and blurring to create a clean binary mask of the foreground subject.

2.2 Blending and Output

Depending on the selected mode, the code can produce an image where the background is either heavily blurred or converted to greyscale. A pixel-wise mask was used to blend the original color image with the processed background, ensuring the foreground subject remained untouched. The final result is a visually appealing image with a sharply focused subject and a perceptually realistic defocused background.

2.3 Challenges

One of the main difficulties in this task was accurately separating the foreground subject from a visually complex background. The woman in the foreground is posing in front of a poster or picture of another girl with similar hair color and style, which makes it visually ambiguous—almost as if the hair in the background belongs to the foreground subject. This visual continuity significantly complicates depth-based segmentation.

Additionally, the metal chair with vertical bars poses a unique challenge. The narrow gaps between the bars often result in small background regions that are hard to capture accurately in the depth map. These areas are prone to misclassification and require fine-tuning of parameters to ensure clean segmentation and consistent background blurring.

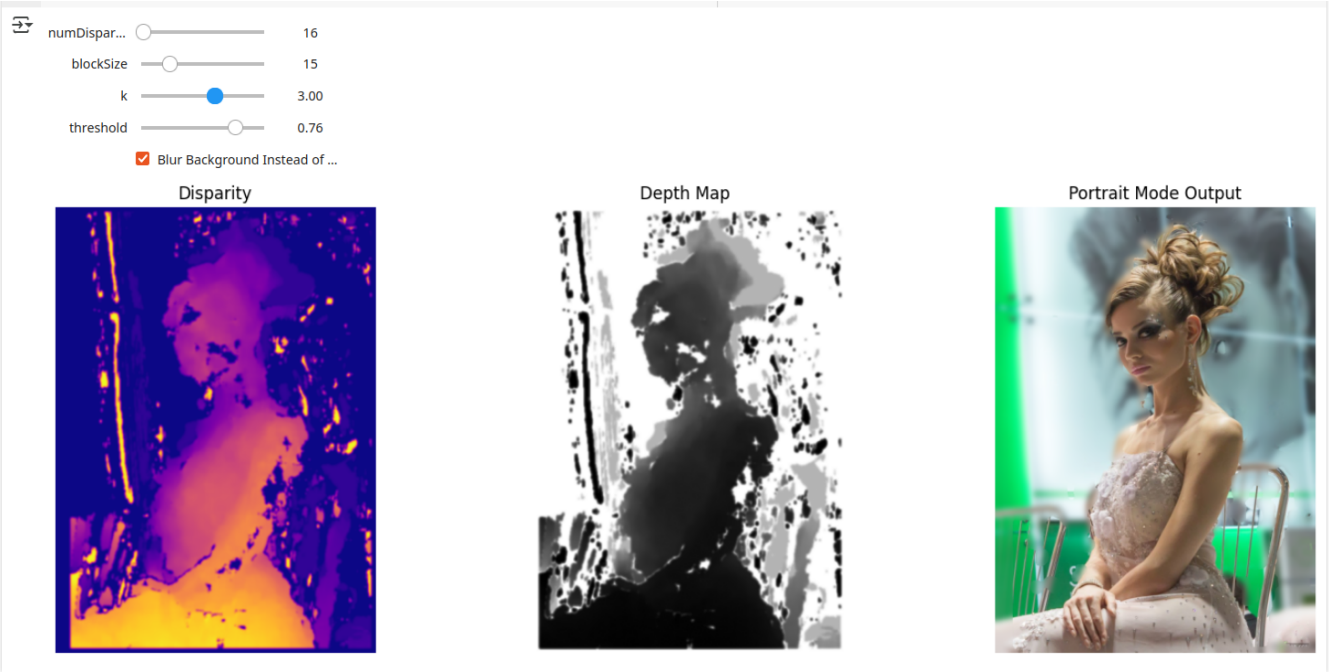

Figure 11: One of the best result obtained with hyperparameter tuning.

We can see that with these hyperparameter values we get the whole girl clearly as foreground, and only some space between the bars of the chair is mistakenly detected as foreground. The rest is blurred. Below are couple of pictures showing the results with different parameter's values with setting their values by sliding the trackbars.

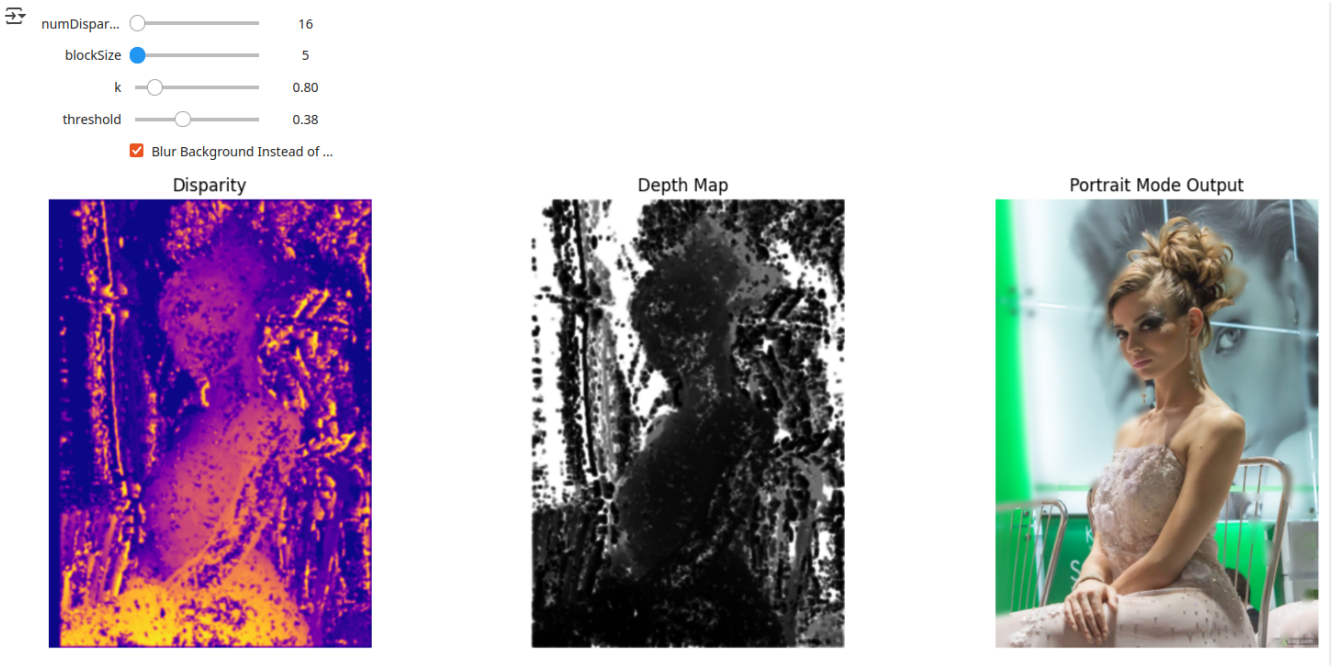

Figure 12: Selective focus results.

With can observe that with small values for both k, and blockSize the depth map we get is pretty noisy. That is why in this result above some background has the same depth as the foreground and theerefore is not blurred.

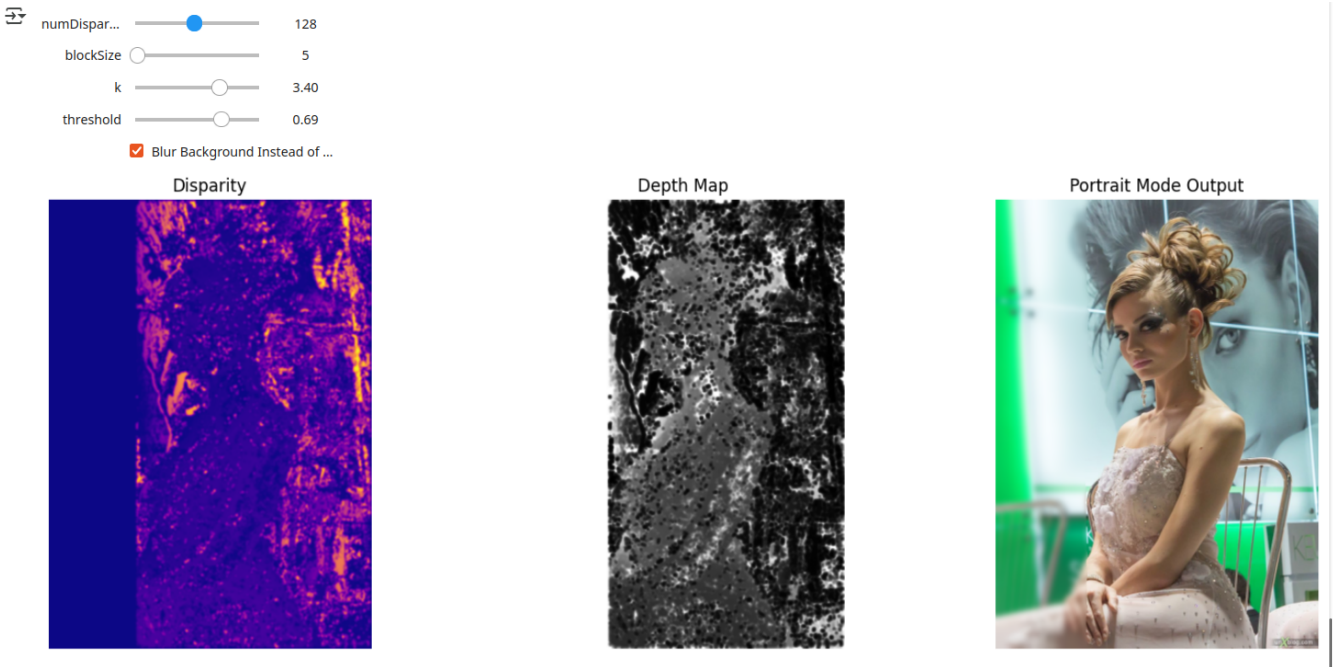

Figure 13: Selective focus results.

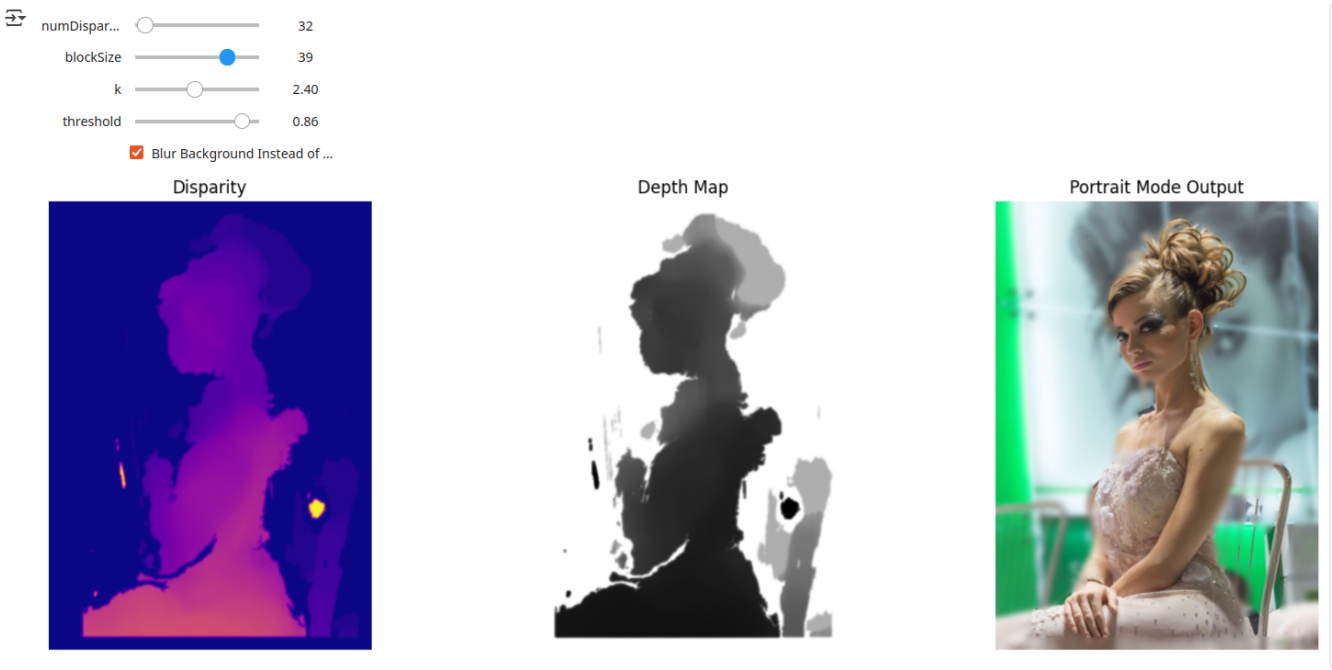

Figure 14: Selective focus results.

With bigger block size, we get smooth depth map, but we lose some very small details. However, this is also a pretty solid result.

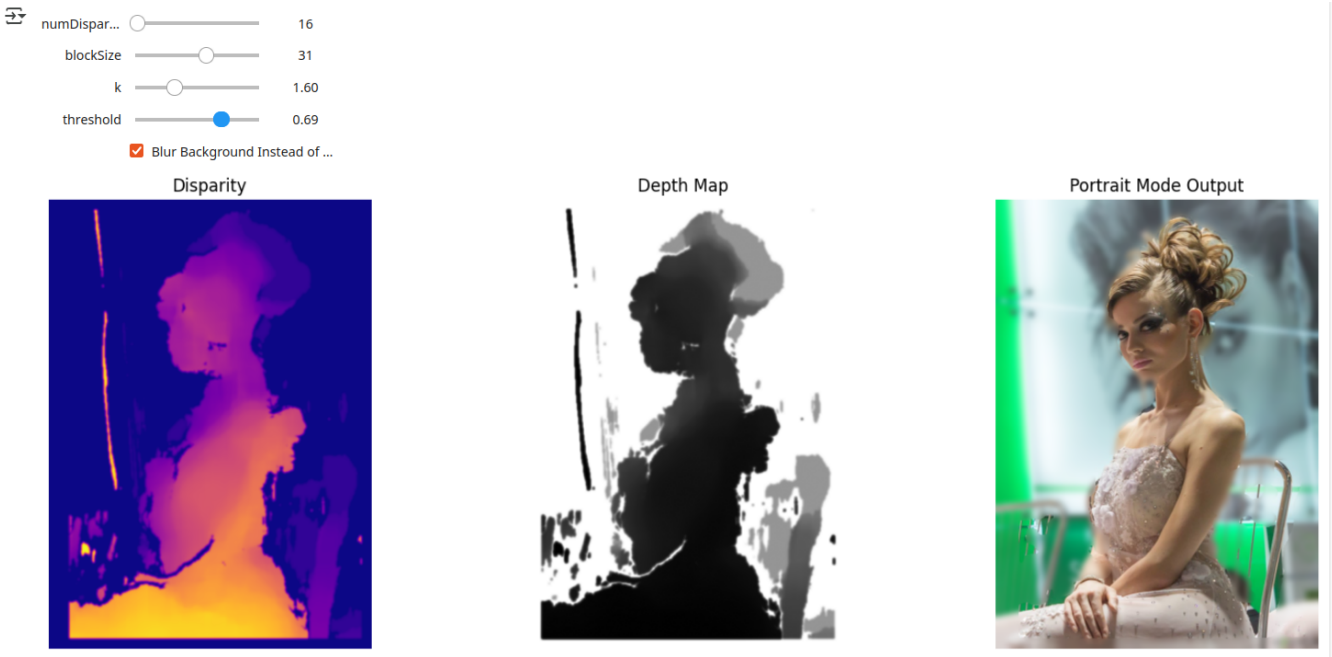

Figure 15: Selective focus results.

A Source Code

```
############################1.2#######################################

import cv2
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, IntSlider, ToggleButton
```

```
from IPython.display import display

# Load the stereo images in grayscale
imgL = cv2.imread('umbrellaL.png', cv2.IMREAD_GRAYSCALE)
imgR = cv2.imread('umbrellaR.png', cv2.IMREAD_GRAYSCALE)



def getDisparityMap(imL, imR, numDisparities, blockSize):
    stereo = cv2.StereoBM_create(numDisparities=numDisparities, blockSize=blockSize)
    disparity = stereo.compute(imL, imR)
    disparity = disparity - disparity.min() + 1  # avoid zero depth
    disparity = disparity.astype(np.float32) / 16.0
    return disparity

def display_disparity(numDisparities=64, blockSize=15, use_edges=False):
    left = imgL.copy()
    right = imgR.copy()

    if use_edges:
        left = cv2.Canny(left, 50, 150)
        right = cv2.Canny(right, 50, 150)

    if blockSize % 2 == 0:
        blockSize += 1
    if blockSize < 5:
        blockSize = 5
    if numDisparities < 16:
        numDisparities = 16
    if numDisparities % 16 != 0:
        numDisparities = (numDisparities // 16) * 16

    disp = getDisparityMap(left, right, numDisparities, blockSize)
    disp_norm = np.interp(disp, (disp.min(), disp.max()), (0.0, 1.0))

    plt.figure(figsize=(10, 6))
    plt.imshow(disp_norm, cmap='plasma')
    plt.title(f'Disparity Map\nnumDisparities={numDisparities}, blockSize={blockSize}, edges={
        use_edges}')
    plt.colorbar(label='Disparity')
    plt.axis('off')
    plt.show()

# Interactive sliders
interact(
    display_disparity,
    numDisparities=IntSlider(min=16, max=256, step=16, value=64),
    blockSize=IntSlider(min=5, max=51, step=2, value=15),
    use_edges=ToggleButton(value=False, description='Use Edges')
);



##############################1.3#################################33

# Imports
import cv2
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Load grayscale stereo images
imgL = cv2.imread('/content/umbrellaL.png', cv2.IMREAD_GRAYSCALE)
imgR = cv2.imread('/content/umbrellaR.png', cv2.IMREAD_GRAYSCALE)

# Resize to 740 x 505
imgL = cv2.resize(imgL, (740, 505))
imgR = cv2.resize(imgR, (740, 505))

# Camera parameters (rescaled)
scale = 740 / 2960  # = 0.25

f = 5806.559 * scale
cx0 = 1429.219 * scale
```

```
cx1 = 1543.51 * scale
cy = 993.403 * scale
doffs = 114.291 * scale
baseline = 174.019  # mm




def getDisparityMap(imL, imR, numDisparities, blockSize):
    stereo = cv2.StereoBM_create(numDisparities=numDisparities, blockSize=blockSize)
    disparity = stereo.compute(imL, imR)
    disparity = disparity - disparity.min() + 1  # avoid zero depth
    disparity = disparity.astype(np.float32) / 16.0
    return disparity

disparity = getDisparityMap(imgL, imgR, numDisparities=16*4, blockSize=13)

# Calculate 3D coordinates
def getPointCloud(disparity, f, cx, cy, baseline, doffs):
    h, w = disparity.shape
    X, Y, Z = [], [], []

    disparity_min_threshold = 20
    disparity_max_threshold = 60

    for y in range(h):
        for x in range(w):
            d = disparity[y, x]
            if d <= 0: continue
            if d < disparity_min_threshold:
                continue
            if d > disparity_max_threshold:
                continue
            z = (baseline * f) / (d + doffs)
            x_world = (x - cx) * z / f - baseline/2
            y_world = (y - cy) * z / f
            X.append(x_world)
            Y.append(y_world)
            Z.append(z)

    #return np.array(X), np.array(Y), np.array(Z)
    return X, Y, Z

X, Y, Z = getPointCloud(disparity, f, cx0, cy, baseline, doffs)

# 3D scatter plot
fig = plt.figure(figsize=(18, 6))
ax = fig.add_subplot(131, projection='3d')
ax.scatter(X, Y, Z, c=Z, cmap='plasma', s=0.5)
ax.set_title('3D View of Scene')
ax.set_xlabel('X'), ax.set_ylabel('Y'), ax.set_zlabel('Z')
ax.view_init(elev=20, azim=-70)

# Top-down view (X-Z)
ax2 = fig.add_subplot(132)
ax2.scatter(X, Z, s=0.1, c=Z, cmap='plasma')
ax2.set_title('Top-down view (X-Z)')
ax2.set_xlabel('X'), ax2.set_ylabel('Z')
ax2.set_aspect('equal')

# Side view (Y-Z)
ax3 = fig.add_subplot(133)
ax3.scatter(Y, Z, s=0.1, c=Z, cmap='plasma')
ax3.set_title('Side view (Y-Z)')
ax3.set_xlabel('Y'), ax3.set_ylabel('Z')
ax3.set_aspect('equal')

plt.tight_layout()
plt.show()

##################2#################################


import cv2
```

```python
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, IntSlider, FloatSlider, Checkbox

# Load stereo images
imgL_color = cv2.imread('girlL.png')
imgR_color = cv2.imread('girlR.png')

imgL = cv2.cvtColor(imgL_color, cv2.COLOR_BGR2GRAY)
imgR = cv2.cvtColor(imgR_color, cv2.COLOR_BGR2GRAY)

# Resize if needed to match dimensions
imgL = cv2.resize(imgL, (imgR.shape[1], imgR.shape[0]))
imgL_color = cv2.resize(imgL_color, (imgR.shape[1], imgR.shape[0]))




def getDisparityMap(imL, imR, numDisparities, blockSize):
    stereo = cv2.StereoBM_create(numDisparities=numDisparities, blockSize=blockSize)
    disparity = stereo.compute(imL, imR)
    disparity = disparity - disparity.min() + 1  # avoid zero depth
    disparity = disparity.astype(np.float32) / 16.0
    return disparity

def apply_portrait_mode(numDisparities=128, blockSize=21, k=1.0, threshold=0.18, blur_bg=False):
    disparity = getDisparityMap(imgL, imgR, numDisparities, blockSize)

    disparity = cv2.GaussianBlur(disparity, (7, 7), 0)

    depth = 1.0 / (disparity + k)
    depth = cv2.normalize(depth, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    # Threshold depth map to get foreground mask
    _, mask = cv2.threshold(depth, int(threshold * 255), 255, cv2.THRESH_BINARY)

    mask = cv2.GaussianBlur(mask, (7, 7), 0.5)

    # Morphological operations for better mask
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (12, 12))
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    mask = cv2.GaussianBlur(mask, (9, 9), 0)
    mask_3ch = cv2.merge([mask]*3)

    # Prepare background
    if blur_bg:
        background = cv2.GaussianBlur(imgL_color, (25, 25), 0)
    else:
        background_gray = cv2.cvtColor(imgL_color, cv2.COLOR_BGR2GRAY)
        background = cv2.merge([background_gray]*3)


    # mask_f = mask_3ch.astype(np.float32) / 255.0
    # output = (imgL_color * mask_f + background * (1 - mask_f)).astype(np.uint8)


    output = background.copy()
    for i in range(background.shape[0]):
        for j in range(background.shape[1]):
            if mask[i] [j] == 0:
                output[i][j] = imgL_color[i][j]



    # Display
    plt.figure(figsize=(15, 5))
    plt.subplot(1, 3, 1)
    plt.title("Disparity")
    plt.imshow(disparity, cmap='plasma')
    plt.axis('off')

    plt.subplot(1, 3, 2)
    plt.title("Depth␣Map")
```

```
    plt.imshow(depth, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.title("Portrait Mode Output")
    plt.imshow(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.tight_layout()
    plt.show()

# Interact
interact(
    apply_portrait_mode,
    numDisparities=IntSlider(min=16, max=256, step=16, value=128),
    blockSize=IntSlider(min=5, max=51, step=2, value=21),
    k=FloatSlider(min=0.1, max=5.0, step=0.1, value=1.0),
    threshold=FloatSlider(min=0.01, max=1, step=0.01, value=0.18),
    blur_bg=Checkbox(value=False, description='Blur Background Instead of Greyscale'),
);
```