Anna Dollbo 1843672
Flavius Marc 0862037

Report Assignment 2: Voxel-based 3D reconstruction

1. Calibration

1.1. Intrinsic parameters

To be able to find the intrinsic parameters using the code from Assignment 1, frames from the four videos were extracted. A png image was extracted every two seconds from the video with the help of FFmpeg. These images were thereafter manually screened, and images with reflections were removed to speed up the calibration process. The code developed for Assignment 1 was then modified in order to loop over images in each camera directory, and individual thresholds for the iterative calibration were set for each camera based on a trial run. The intrinsic camera parameters for each camera were written to individual xml-files.

1.2. Extrinsic parameters & Background Image

The extrinsic parameters were obtained by calling the `Camera::detExtrinsics()` method and manually selecting the corners, starting from the same upper left corner in each camera view. Once the program has these points it calls the openCV function `solvePnP()` to obtain the rotation and translation matrix. After this step both the intrinsic parameters (the camera matrix and distortion coefficients) and the extrinsic parameters were written to a config.xml file for each camera.

The background image used in the program for the background subtraction was created by taking 10 frames from each background.avi video with ffmpeg, then averaging all 10 images into one by using the function `addWeighted()`.

2. Background subtraction

We used BackgroundSubtractorMOG2, for this part. First, we “train” the foregroundMask on the background video of each camera, and then we apply the Subtractor and the foregroundMask to the main video. The learning stage is made once for each camera in Camera.cpp, `initialize()` function. The learning rate is set to -1, since it gives the best visual results. Next, the foregroundMask is applied for each frame in the processForeground function from Scene3DRenderer.cpp, where the learning rate is 0.

3. Volume Reconstruction

The voxel model was created with the help of the supplied framework in directory src. Additionally, post-processing has been made for a more accurate reconstruction. With the use of a morphological operation (opening) outliers noise was removed.

Additional Post-Processing Improvement:
We detected an unnecessary hole in the foregroundMask for the camera 3 and also some clutter in the region between the chair and the legs (see figure 1):

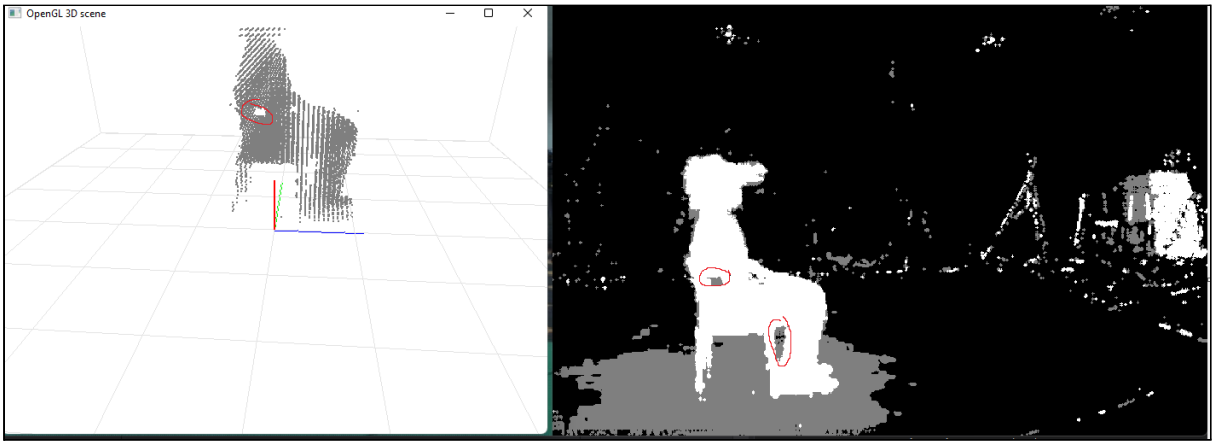

Figure 1. Before post-processing

To fix the issues, we used findContours and drawContours after some preprocessing has been made, which includes bitwise_not operations and thresholding. First time, we detected the largest blob by calculating the maximum area, and then subsequently, we eliminated from the contours vector the contours corresponding to the largest areas at the same level, until we found the regions of interest. The code for this improvement can be found in the processForeground function.

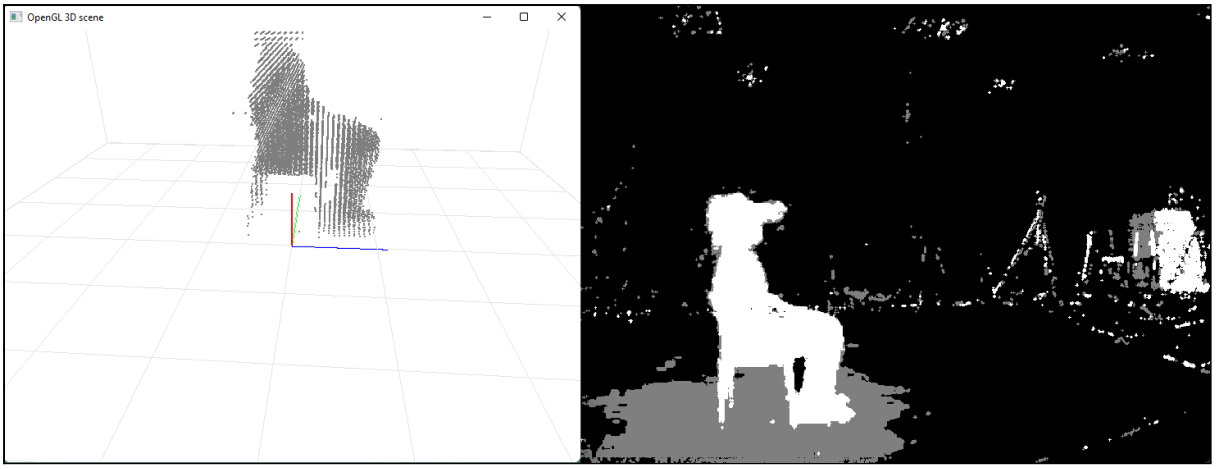

Figure 2. After post-processing

4. Choice Tasks

4.1. Voxel model construction optimisation

As a first step, the voxel space was trimmed by changing the values of `m_height` variable and manually changing the size of the camera field of view plane `m_plane_size`. This led to a reduction of the voxel space from 1048576 voxels to 821516, a decrease of 21.7%.

Secondly, we manage to get a good execution time for the processForeground function by calculating the Subtractor at the initialization of the camera (see figure 3):

Currently, the colorisation of the voxel model slows down the program. See next section for suggestions on how to make this part more time efficient.

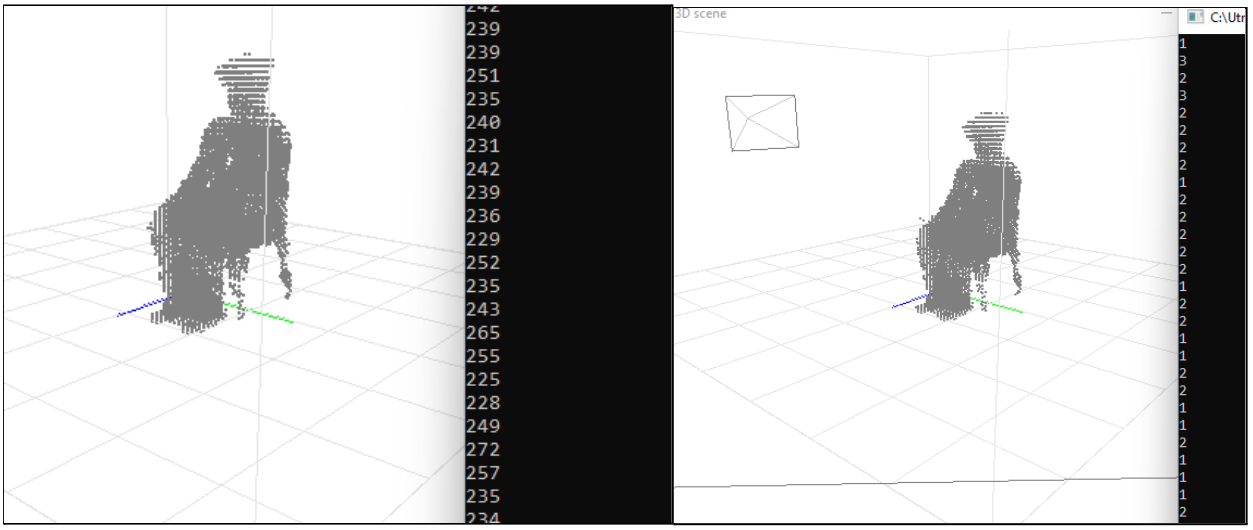

Figure 3. Right: 240 ms in average for execution (release mode). Left: 2 ms in average for execution (release mode)

Finally, we also wanted to implement the optimization where a XOR is used to detect "binary movement" between frames. Even though we did not manage to finish it, there is commented code in the `processForeground()` function, which may be used for increasing efficiency.

4.2. Colourisation of voxel model

To colour the voxels, a colour reference image was extracted from the video.avi file for each camera using ffmpeg. Each file was then read as a matrix and all matrices stored in a vector. During the initialisation of the voxels, the colour of each pixel was extracted from these images and stored in the `voxel->color` member of the voxel data type. During projection of the voxel volume, voxel colour was read for each voxel in the `Glut::drawVoxels()` method. On account of time pressure, the voxel colour was only extracted using the colour reference image of one camera. This led to an imbalance in the rgb values, which resulted in the final voxel model having different colours, and not really resembling the original image. For further development, an average of all rgb values for camera for each voxel would be implemented, as well as a reasoning about depth, where a projection of perhaps `cv::circle()` to the 3D scene only for the outmost pixels would be tested, which could deal with occlusions. Furthermore, the initalisation of the voxel colours is currently done each time the program runs. For a faster implementation, these values would be read to a separate file, so that the colour calculation would not be as costly. Additionally, pixels that are close to each other tend to have similar rgb-values, which means that each pixel might not have to be processed, and a heuristic could be used to assign the same values to *n* voxels in the same neighbourhood.

5. Results

Due to a well done background subtraction, the voxel model is adequate. All four chair legs are visible in the voxel model, with some being more apparent than others. The gap between the legs of the figure is clearly visible, as are the gaps between the arms and legs. The head with pointy ears and a long snout is also clearly visible and modelled correctly. We believe that the quality of the background subtraction made this voxel model successful. The colourisation of the voxel model as

well as the XOR operation between frames could be optimised further as discussed previously, which would make the voxel model more visually appealing as well as more efficient. But in general, we are content with the results.

Link to a demonstration of the voxel model: https://www.youtube.com/watch?v=L0rbiZ9zoMk