Computer Vision 2, Assignment 3

Davide Belli
11887532
davide.belli@student.uva.nl

Gabriele Cesa
11887524
gabriele.cesa@student.uva.nl

Lukas Jelínek
11896493
lukas.jelinek1@gmail.com

Introduction

In this assignment we are going to work at 3D Mesh Reconstruction and Texturing from 2D images and depth maps. For this purpose, we are going to use C++ libraries provided in PointCloudLibrary (PCL). The first step towards 3D reconstruction is to generate a 3D Point cloud of the scene from depth images, as we have seen in the previous assignments. Once the point cloud is generated we can choose a surface reconstruction algorithm to *interpolate* points by generating a 3D surface connecting them. Examples of such algorithms are Poisson Reconstruction and Marching Cube. Some post-processing techniques are then required, including smoothing surfaces and closing holes in volumes by connecting surfaces (watertighting). Once the 3D Mesh is generated, we will proceed with Texturing. This final step aims to map color information from 2D images to RGB polygons or points in the 3D reconstruction.

1 Depth-based and Texture-based 3D Reconstruction Comparison

1.1 Advantages and disadvantages of Depth-based and Texture-based approaches

In this course we have experimented with different approaches to solve 3D reconstruction of scenes from either depth maps or texture.

Iterative Cloud Point algorithm was introduced in the first assignment to iteratively generate 3D point clouds of an object from a set of depth maps generated from different perspectives. This algorithm is an example of depth-based approaches to 3D reconstruction, in contrast to texture-based ones. The main advantage of Depth-based reconstruction is that it is generally easier to work with depth information in comparison with 2D texture. This is because depth maps intrinsically describe the three dimensions of space. In addition, depth representation of scenes is invariant with respect to aspects such as lighting conditions, object textures and body reflectance, which, on the other way, negatively affect texture representation. Moreover, nowadays hardware like Microsoft Kinect makes it possible to generate depth maps in a reliable way, with accuracy levels up to millimeter scale. Looking at the disadvantages of ICP, we notice that it is easy for it to get stuck into local optima, especially with bad point cloud initializations. Although many solutions have been proposed to reduce this issue, steps in the algorithm such as selecting, matching and rejecting points should always be tuned with some grid search to find the best combination for a particular domain. In addition, because of the iterative nature of ICP algorithm which only matches 2 point clouds at a time, it often happens that reconstruction errors accumulate over a series of point cloud merges. Misalignment of few millimeters or degrees that are barely noticeable when merging a pair of cloud points would then build up on top of each other and produce very imprecise final reconstructions.

Looking now at Texture-based approaches to 3D reconstruction, we take as example ***Structure from Motion*** which we implemented in our second assignment. Since depth information is often not accessible, SfM provides a good alternative to Depth-based methods. 2D images are much less expensive, both because of hardware costs and for the availability of a large number of datasets online (e.g. Google Streetview). On the downside, Texture-based methods rely on the quality of images and strongly depend on the scenes conditions. Lighting conditions, texture appearances and self-similarities in scenes largely affect performance of Structure from Motion. Indeed, the results from SfM are based on the accuracy of feature descriptors and feature matches, which once again have known pitfalls and fault cases. A way to partially solve some of these issues is to include filtering of outliers matches using algorithms such as RANSAC.

1.2 Combining Depth-based and Texture-based approaches to improve 3D reconstruction

One possible way of combining these two approaches is to extract features from a point cloud and use structure from motion and epipolar geometry to combine multiple point clouds. The main disadvantage of this solution is that point cloud has only spacial information. This can be a crucial flaw if we have two objects only distinguishable based on texture or color. It could be resolved by enhancing point cloud with color information.

Another possibility is to use colored images only for structure from motion reconstruction. Afterwards, we could use all RANSAC verified key-points matches between two images to further optimize cloud alignment using ICP. This would avoid problems with pure ICP initialization. Potentially, it could reduce the noise in the reconstruction.

2 3D Meshing and Watertighting

In this project we experimented with two mesh generation methods: Poisson Surface Reconstruction and Marching Cube.

The first problem one has to deal with when working with points cloud data is to remove the background. To retrieve only the points belonging to the person we filtered out in each frame the points with a Z-value (the depth) greater than 1.2 meters (we found this value running and visualizing some experiments, trying to filter out the points outside the body).

Once isolated the points in each frame, we can transform them according to the known camera pose and, finally, merge all points clouds.

(a) Front view

(b) Back view

(c) Top view

Figure 1: Merged point cloud recolored using images RGB information

In Fig. 1, we show the merged point cloud obtained at this point. Since the point cloud itself is colorless, we remap RGB information from images to the reconstructed cloud to improve its visualization.

We take every depth image pixel's 2d coordinates and transform it to the coordinate system of rgb image. Than we take closest pixel to this location as our color information for depth cloud particle.

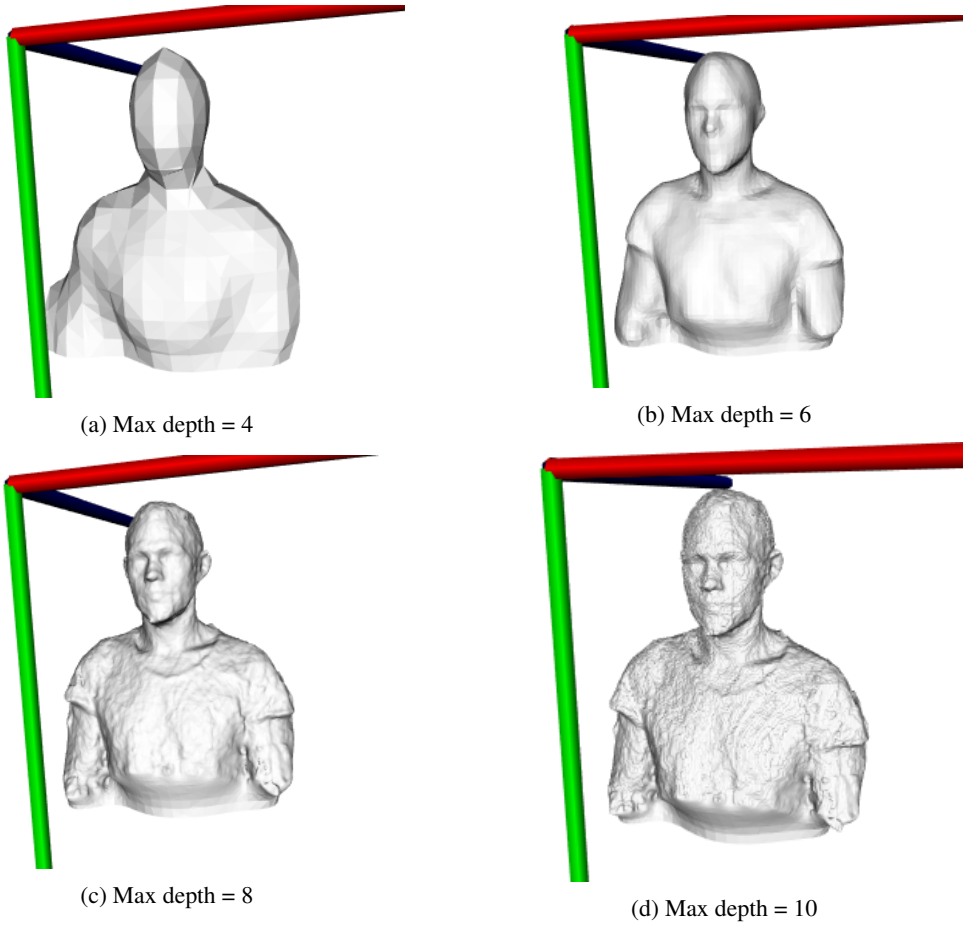
Although we can already clearly perceive the 3D reconstruction of the original scene, this representation does not actually generate any surface (and, consequently, volume). This will be done in the Mesh generation and Texturing tasks. From the plots it is also evident that some regions of the image are not visible from any point of view (notice in particular the lower part of the back). Those gaps in the point cloud may generate holes in the reconstructed meshes. We will see in the next sections how this problem can be solved.

After the cloud point is created, mesh generation algorithms require the normal vectors of the points to approximate the surface they belong to. In order to estimate the normals we used the *IntegralImageNormalEstimation* method provided by *Point Cloud Library*.

Finally, we can build a mesh from the point cloud using one of the two aforementioned methods.

2.1 Poisson Surface Reconstruction

This algorithm approaches the problem trying to solve (approximately) a *Poisson Equation* [1] which aims to match the gradient of the indicator function of the surface to the normals of the points cloud.



(a) Max depth = 4

(b) Max depth = 6

(c) Max depth = 8

(d) Max depth = 10

Figure 2: Poisson Surface reconstruction at different resolution changing the values of the octree maximum depth

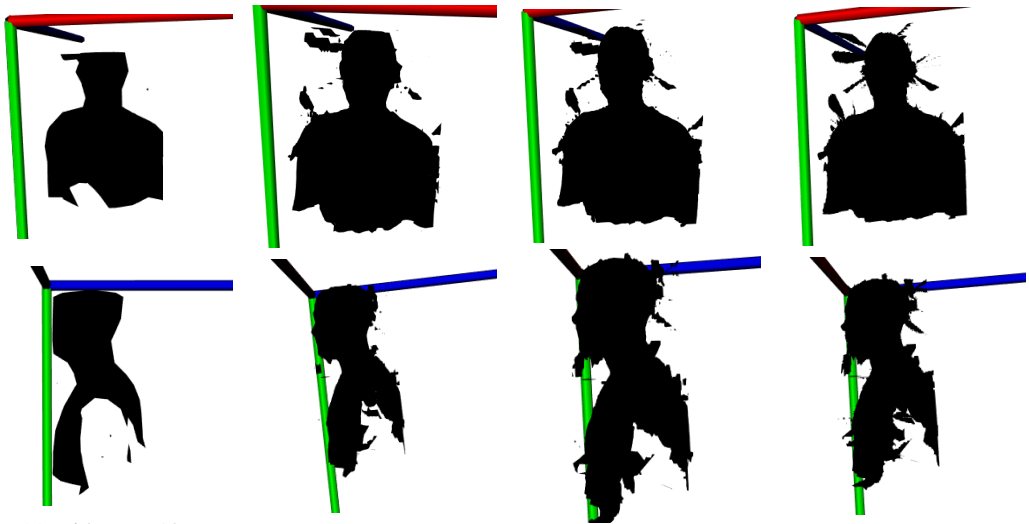
Once we implemented the Poisson Surface Reconstruction method, we noticed how the main parameter to be tuned to improve the results was the *maximum depth* allowed in the octree. This parameter, used in the Poisson Reconstruction to build the indicator function for the surface, is directly related to the maximum resolution in the output. Lower values produce a smoother surface but result in losing many details, while higher values result in a fine-graned reconstruction which is going to be more sensitive to noise in the Point Cloud representation.

In Fig. 2 we show how the depth parameter affect the final result. As explained, lower values for the maximum depth result in the loss of smaller details such as nose, eyes, mouth, t-shirt depth. On the other way, we notice that with the higher value reported, the surface becomes too noisy due to imprecision in the Point Cloud reconstruction. Since we want a good trade-off between smoothness of surfaces and detail representation, we pick the maximum value for depth equal to *8* and fix this parameter for the next experiments with Poisson Reconstruction.

We also experimented with other parameters for Poisson Reconstruction such as *scaling* and the *number of samples per node*. The first one regulates the ratio between the diameter of the cube used for reconstruction and the diameter of the samples' bounding cube. The latter defines the minimum number of sample points that should fall within an octree node, and should be used to regulate reconstructions in relation to the noise included in the samples. In both experiments we didn't find any relevant improvement by changing the parameters from their default values.

2.2 Marching Cube

This algorithm splits the space in a 3-dimensional grid made of small cubes. Given a isosurface as a scalar field (associating to each point in the space a scalar) built from the point cloud, it considers the 8 vertices of each cube and checks which of them are inside or outside of the isosurface based on an iso level (which is used as a threshold on the value in the scalar field to classify the point). Based on the configuration of points and its iso-status, the algorithm chooses a correct precomputed cube representing the local surface. The algorithm works with 256 possible cubes. Finally, all cubes are merged and surfaces are linearly interpolated. One limitation of this algorithm is the introduction of limited number of mesh types. Additionally, the linear interpolation technique smooths sharp features on the surface. This method also suffers from surface triangulation ambiguity which can yield to creation of holes in the mesh. [2].



(a) grid_res = 10 (b) grid_res = 40 (c) grid_res = 60 (d) grid_res = 100

Figure 3: Marching Cube surface reconstruction using different values for Grid resolution

In our implementation, we noticed how the default configuration of parameters doesn't yield good results. In particular, we found that the most relevant parameter to be tuned was the *Grid Resolution*. This parameter defines in how many cubes should the space be divided when running the algorithm. In particular, this number represents the amount of ticks on each space axis. While increasing the number of cubes enables a more detailed reconstruction, Marching Cube becomes much more sensitive to noise. The computational weight also increases significantly with the Grid Resolution value (with the default resolution value, the RAM fills before finishing the execution). Similarly to Poisson Reconstruction behavior, low values for the resolution generate very bad and approximate results, where only the general shape of the surface is preserved.

In Fig. 3, we show how changing the Grid Resolution affects the final mesh reconstruction by plotting frontal and side views for each value. We notice that with the lowest resolution there is little noise, but major parts of the object are missing from the mesh (back of the subject and top part of the head). While increasing the resolution, we see that noise in the reconstruction significantly increases but Marching Cube is able to preserve most of the object surfaces. We choose Grid Resolution = 60 as a good trade-off between the two corner cases and keep this configuration in further experiments.

We also tried to tune other parameters, for example the *ISO level* and *PercentageExtendGrid*, which respectively regulate the threshold on the scalar field that classifies points as inside or outside, and a value for enlargement of the resulting mesh. However, no significant improvements were found with those experiments.

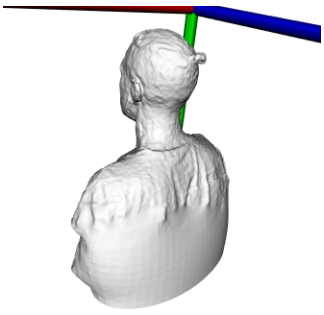
By comparing the outputs obtained from Marching Cube and Poisson Reconstruction, it is evident that the latter outperforms the first one. Indeed, most of the reports in literature agree on this results and we expected this to happen as well considering the reconstruction ambiguity intrinsic in the Marching Cube algorithm.

2.3 Filtering

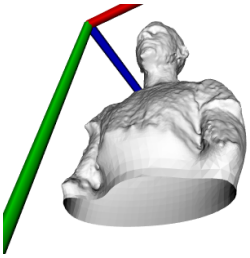
In order to improve the results produced by these two methods, we tried to fill the holes left in the meshes.

Since Poisson Reconstruction does some watertighting by itself, some holes in the Point Cloud are already filled by surface interpolation between existing points. We can see this by looking at the lower part of the back in the model of the person in Fig. 4a. However, biggest gaps as the one in the bottom of the model are not solved by Poisson Reconstruction. For example, in Fig.4b there are no points below the body since that part is not visible by any view; as a result, the mesh generated usually contains a large hole there.

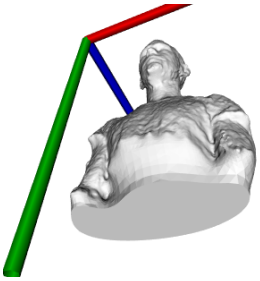
For this purpose, we employed *vtkFillHolesFilter* from *VTK* with very good results for the mesh built by Poisson Surface Reconstruction. Again, we tried to modify the parameter configuration for this class. For example, *HoleSize* parameter specifies the size of the hole to be filled. Any value greater or equal to 1 is enough to make the filling work, and no visual differences are found by using higher values.



(a) Poisson Reconstruction is able to reconstruct surfaces in the lower part of the back by interpolation between available points.



(b) Hole in the bottom part of the object after Poisson Reconstruction.



(c) The hole is filled using *vtkFillHolesFilter* from *VTK*

Figure 4

In Fig. 4c we show the effect of this Hole Filling procedure on the surface reconstructed with Poisson. A smooth flat surface is used to connect all the borders of the mesh terminating in the gap at the bottom of the figure.

We also tried to fill holes in the output of Marching Cube. However, we saw that this algorithm was not able to work with the generated mesh, probably because of large irregularities in the surfaces.

2.4 Smoothing

In order to remove high-frequency noise in the reconstructed mesh, we tried to use the *MeshSmoothingWindowedSincVTK* provided by *VTK* through *PCL*. More precisely, the algorithm corrects mesh's points' coordinates applying a windowed kernel over neighboring points. Nevertheless, we haven't been able to observe significant visual improvements.

3 Coloring 3d Model

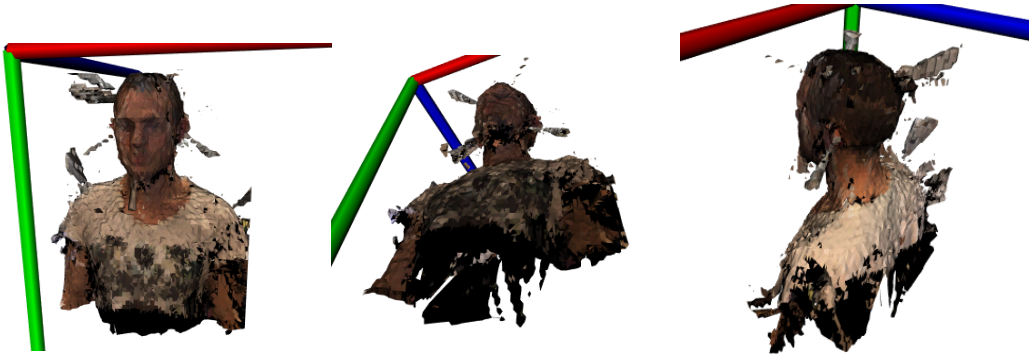
The model coloring is done by extracting all vertices from a previously constructed mesh. This creates a point cloud which is transformed into camera coordinate system for each camera position. We need to distinguish if a vertex is visible or occluded on current camera. This is done by ray-tracing in the octree data-structure which is created from vertex point cloud in camera frame. The octree has a resolution of 0.01. If a vertex is not occluded we project it into camera image plane. The size of the image plane is based on the RGB image provided together with depth information. In our dataset, the depth map has a smaller resolution than the RGB image. We have tried to resolve this issue in two ways. First, we calculated the ratio difference between two resolutions and multiplied the focal length with this constant. This offered a good reconstruction, but the reconstructed color included some noise. We improved on top of this experiment by downsampling the RGB image to the resolution of the depth image. This effectively reduced the noise in the image. We have also averaged colors from multiple views to get better approximation of real color information with limited highlights.

The color reconstruction on the resulting mesh from Poisson Reconstruction still contains some black holes 5. These holes appear because these points were not projected to any camera image or were occluded by other blocks in the octree. We could resolve it by using Kd-tree and fill black vertices of mesh with the closes color. Alternatively, we could try to fill the color of black vertices by averaging among neighboring polygons.

(a) Front view (b) Back view (c) Top view

Figure 5: Mesh created with Poisson reconstruction and recolored using RGB images

Similarly, we can recolor the final reconstruction from Marching Cube. Obviously, we expect it to be much less accurate than the one from Poisson Reconstruction, given the worse results obtained in the mesh generation. In Fig. 6 we show the same recoloring method described above applied to Marching Cube surface output.



(a) Front view

(b) Back view

(c) Top view

Figure 6: Mesh created with Marching Cube reconstruction and recolored using RGB images

Although the final result is worse than the one obtained with Poisson Reconstruction, the recoloring algorithm seems to be enough stable and resistant to the noise in the mesh, generating a meaningful recoloring of the surfaces.

Conclusion

Working at this assignment we had the chance to study and experience with fundamental algorithms for 3D Mesh reconstruction. In addition, we took some time to understand how and why different parameter configurations yield different final results. During the implementation, we had to analyze and solve intrinsic issues of the 3D Reconstruction task, such as thinking about how to match 2D RGB pixels to their 3D reconstructed voxel, or how to implement watertighting after the initial reconstruction. We then described advantages and disadvantages of texture-based and depth-based reconstruction methods and compared pros and cons of Marching Cube and Poisson Reconstruction. In conclusion, we employed some well-known algorithms and introduced some improvements described in literature to successfully generate 3D reconstruction of objects working with real-world depth and image datasets.

References

[1] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[2] S. Roy and P. Augustine. Comparative study of marching cubes algorithms for the conversion of 2d image to 3d. *International Journal of Computational Intelligence Research*, 13(3):327–337, 2017.