3D Mesh Generation and Texturing
Computer Vision 2

Nichita Diaconu (11737980), Radu Sibechi(11808527),
Ruben-Laurentiu Grab(11609923)

May 6, 2019

1 Introduction

In this assignment we use our previous knowledge and implement a 3D meshing algorithm with texturing and coloring of the mesh. As a result, the 3 assignments took us through most of the challenges that are involved in constructing 3D objects from images.

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

One advantage of the Structure from Motion (SFM) algorithm is the fact it does not require depth information, but it only requires RGB images, which are more prevalent. To accurately obtain structure from images with SFM, the algorithm requires many images with slightly variating viewpoints. Iterative Closest Points (ICP) requires depth images, which are not as commonly available as RGB images. One advantage of ICP is that it results in a higher quality 3D model when compared to SFM.

These methods can be combined to create a textured 3D model. The depth-based method can be used in order to generate the 3D mesh, while the texture-based method maps the color onto the mesh.

3 3D Meshing and Watertighting

Given the registered point clouds, we are generating a 3D mesh by merging them, using the algorithm below:

Algorithm 1 Merging

```
procedure MergingPointclouds(3DFrames)
    model_point_cloud ← emptyPointCloud()
    for frame in 3DFrames do
        depth_image ← frame.depth_image
        focal_length ← frame.focal_length
        camera_pose ← frame.camera_pose
        point_cloud ← depthToPointCloud(depth_image, focal_length)
        point_cloud_with_normals ← computeNormals(point_cloud)
        point_cloud_with_normals ← transformPointCloud(point_cloud_with_normals, camera_pose)
        model_point_cloud ← concatPointClouds(model_point_cloud, point_cloud_with_normals)
    end for
    return model_point_cloud                    ▷ Contains all points XYZ and normals
end procedure
```

In order to get the final 3D mesh, the *model_ point_ cloud* needs to be passed to a mesh generation method such as (**Poisson Surface Reconstruction** or **March-**

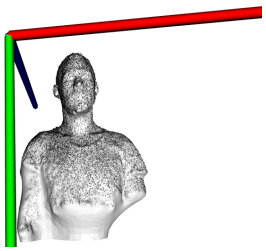
ing cube). In order to speed up the computation, we have made use of the KDTree. We will analyze how well these methods perform and the effect of various hyperparameters on the final output.

3.1 Poisson Surface Reconstruction

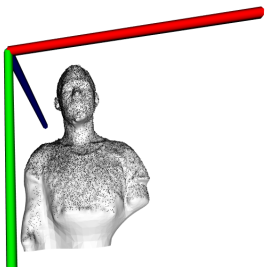
When reconstructing using the Poisson method, we have taken the following hyperparameters into account: depth and samples per node.

3.1.1 Samples per node

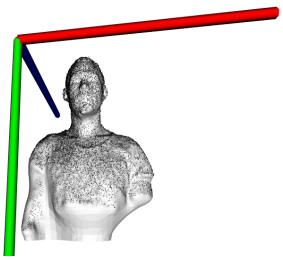
This parameter specifies the minimum number of sample points that should fall within an octree node as the octree construction is adapted to sampling density. For noise-free samples, small values in the range [1.0 - 5.0] can be used. For more noisy samples, larger values in the range [15.0 - 20.0] may be needed to provide a smoother, noise-reduced, reconstruction



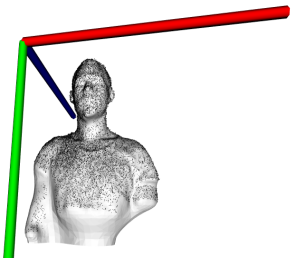
(a) SPN 1



(b) SPN 5



(a) SPN 10

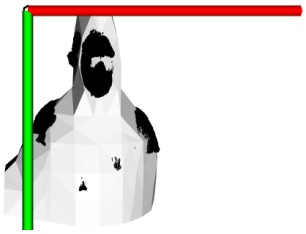


(b) SPN 20

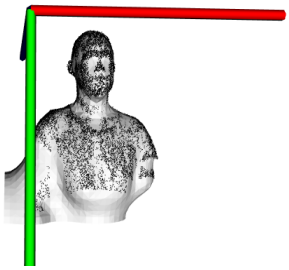
3.1.2 Depth

This parameter defines the level of detail of the resulting mesh. This integer is the maximum depth of the tree that will be used for surface reconstruction. Running at depth d corresponds to solving on a voxel grid whose resolution is no larger than $2^d x 2^d x 2^d$. Note that since the reconstructor adapts the octree to the sampling density, the specified reconstruction depth is only an upper bound.

(a) Depth 1

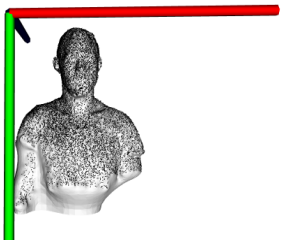


(b) Depth 3

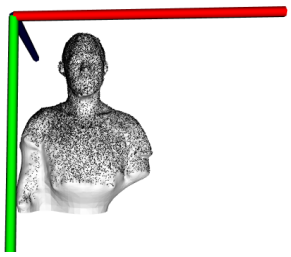


(a) Depth 5

(b) Depth 7



(a) Depth 8



(b) Depth* 10

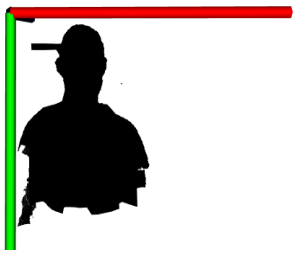
As we increase the depth, the results becomes smoother. A low depth results in very few polygons being used for the reconstruction. A larger search depth results in a better representation. However, this comes at a computational cost. We have found that there is minimal difference between a depth of 8 and a depth of 10. As the computational cost increases by a large margin between 8 and 10, we have decided to select 8 as our depth parameter.

3.2 Marching Cubes

When reconstructing using the Marching Cubes method, we have taken the grid size parameter into account

3.2.1 Grid resolution

The grid resolution determines the amount of cubes that are used in order to create the mesh.



(a) Grid resolution 10

(b) Grid resolution 50

(a) Grid resolution 100

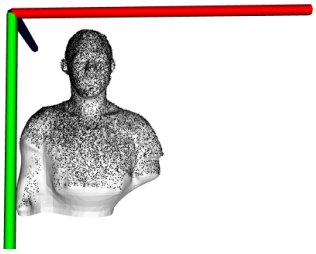
(b) Grid resolution 150

3.3 Discussion

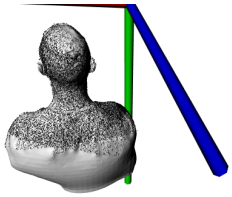
We can observe that the textures meshes obtained with Poisson method achieves better looking result compared to the Marching Cubes method. The Poisson method has no outliers and less irregularities. Moreover, the "Poisson" method shows no artifacts.

3.4 Final model

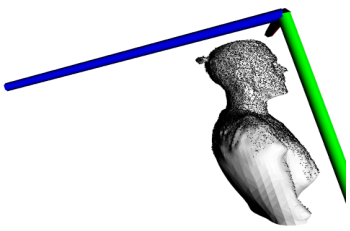
In the end we have chosen Poisson with a depth of 8 and Samples per node of 10. Below you can see more pictures of our model:



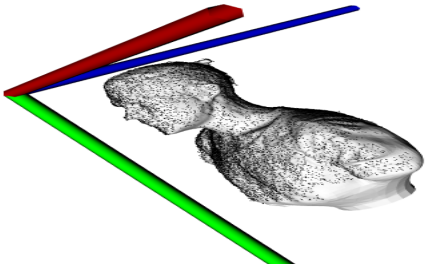
(a) Front



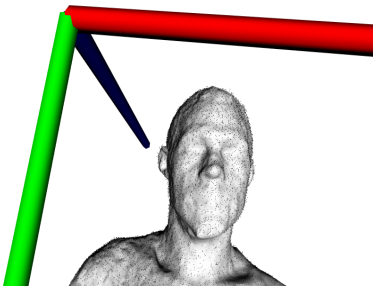
(b) Back



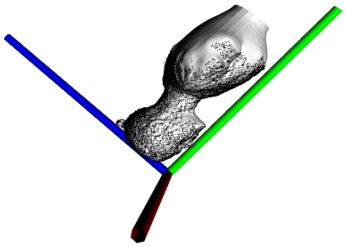
(a) Left



(b) Right



(a) Front face



(b) Side back

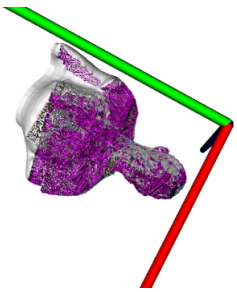
4 Coloring 3D Model

Given a constructed 3D model we tried to add texture and color to it. Our approach is following the algorithm:

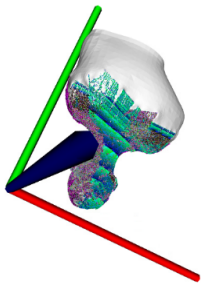
Algorithm 2 Texturing

```
1:  procedure TEXTURE(mesh, 3DFrames)
2:      polygons ← mesh.polygons
3:      point_cloud ← mesh.point_cloud
4:      for frame in 3DFrames do
5:          depth_image ← frame.depth_image
6:          focal_length ← frame.focal_length
7:          camera_pose ← frame.camera_pose
8:          transformed_point_cloud ← transformPointCloud(point_cloud, camera_pose.inverse())
9:          for polygon in polygons do
10:             if polygon visible to this camera then
11:                 uv_coordinates ← getUVCoordinates(polygon, transformed_point_cloud)
12:                 assign uv_coordinates of this camera to the polygon
13:             end if
14:         end for
15:     end for
16: end procedure
```

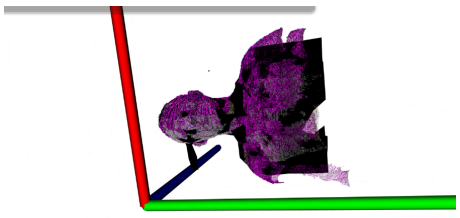
Our results depend on the method used for meshing.



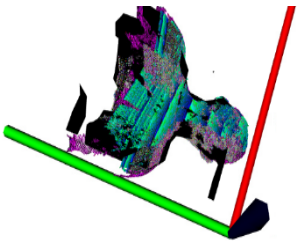
(a) Poisson followed by Coloring. View 1



(b) Poisson followed by Coloring. View 2



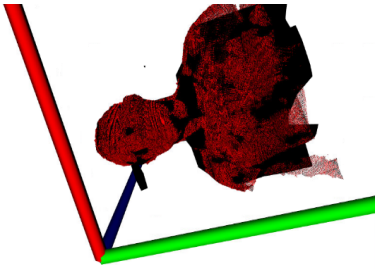
(a) Marching Cubes; Coloring. View 1



(b) Marching Cubes; Coloring. View 2

Unfortunately, our results with coloring did not match the performance of part 3. We tried to color in different ways, namely by first constructing the whole mash

and then coloring, or coloring each point cloud, while adding it then to the mash. Nonetheless, these did not achieve better results and we also get a segmentation fault error in the latter case for marching cubes. We also tried using only one color to get an intuition of what it going wrong:



(a) Marching Cubes; Red coloring

5 Conclusion

Working on this task to create 3D mashes, we experimented with different algorithims, each of them with pros and cons. We saw first reconstruct the 3D object from the depth and the focal length and a mesh generation method. Then we try to color the mash. We saw first hand the effect the hyperparamters have in the resulting mesh. Besides that, as stated before, we had to find a balance between the performance of the algorithm and and it's speed. We are surprised that the coloured mesh did not arise to our expectations. Although we tried different configurations in order to get the same proficiency as at the uncolored mesh, we were unsuccessful. Despite this, we still believe that there is an error using in plain sight that prevented us from getting the expected results. As a future work, we would like to find a solution to this issue.

6 Self-contribution

We all feel that the work has been shared equally among ourselves.