Point Cloud Library - Toyota Code Sprint
Final Report

Alexandru E. Ichim

April 1, 2012

1 Work Done

This section will present the work I have done in addition to the results presented in the midterm report.

1.1 Updated *pcl::surface* Architecture

We have updated the *pcl::surface* architecture again (since the TOCS midterm report) for more increased flexibility. The new structure is presented below.

CloudSurfaceProcessing (new class) - base class for algorithms that take a point cloud as an input and produce a new output cloud that has been modified towards a better surface representation. These types of algorithms include surface smoothing, hole filling, cloud upsampling etc. Currently, the classes inheriting from this are:

- MovingLeastSquares
- BilateralUpsampling

MeshConstruction - reconstruction algorithms that always preserve the original input point cloud data and simply construct the mesh on top (i.e. vertex connectivity). They input a point cloud and produce a PolygonMesh that has the same vertices as the input. The current classes in PCL that inherit from this are:

- ConcaveHull
- ConvexHull
- OrganizedFastMesh
- GreedyProjectionTriangulation

SurfaceReconstruction - reconstruction methods that generate a new surface or create new vertices in locations different than the input point cloud. The input is a point cloud, and the output a PolygonMesh with a different underlying vertex set. Classes in PCL that accord to this are:

- GridProjection
- MarchingCubes
- SurfelSmoothing

MeshProcessing - methods that modify an already existent mesh structure and output a new mesh. They take in a PolygonMesh and produce a new PolygonMesh with possibly different vertices and different connectivity. Classes currently inheriting from this are:

- EarClipping
- MeshSmoothingLaplacianVTK
- MeshSmoothingWindowedSincVTK
- MeshSubdivisionVTK

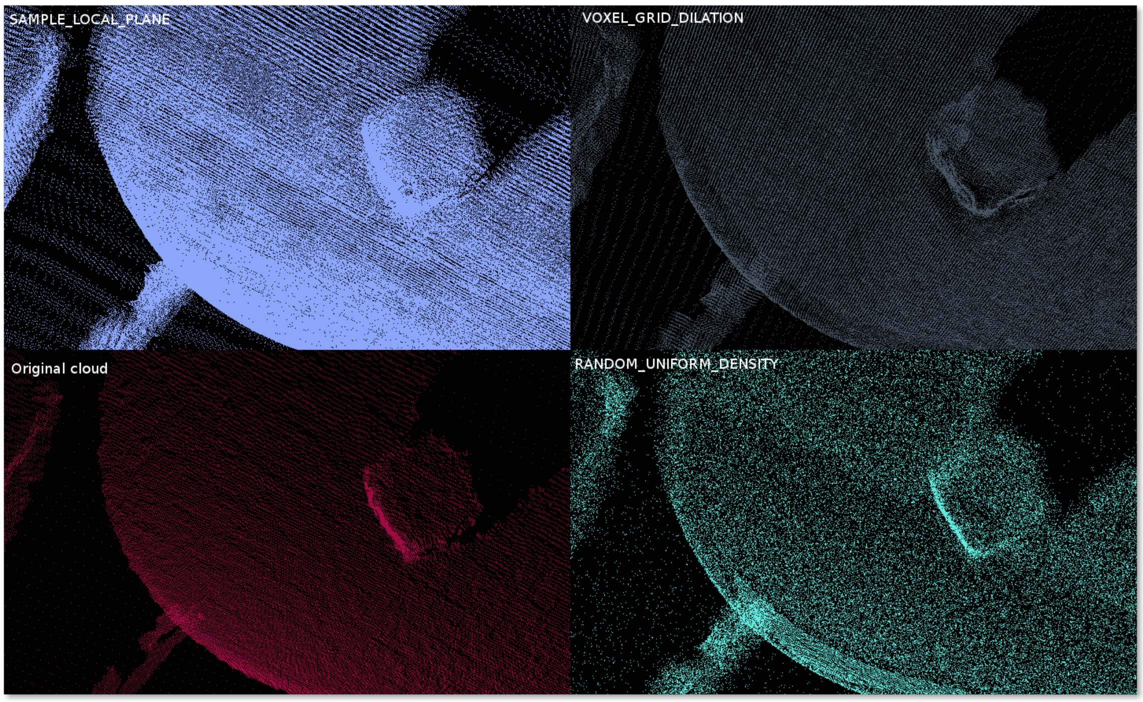


Figure 1: Examples of all the upsampling methods present in the Moving Least Squares implementation in PCL.

1.2 Moving Least Squares - Upsampling Methods

1.2.1 Introduction

In the Toyota Code Sprint midterm report, we presented the smoothing effects of the Moving Least Squares algorithm we currently had in PCL, as proposed in the original paper by Alexa et al [1], and extended in Rusu and colleagues in [10]. A thorough set of results can be found in the afore mentioned document.

During this development period, we enhanced the approach to also handle hole filling. By definition, the algorithm fits a polynomial of a certain degree (usually 2^{nd} or 3^{rd} degree) to the point set, so the possibility of sampling these local polynomials comes naturally. We propose three methods of doing so, as they will be explained in the next subsections. Figure 1 shows an overview of the effects of each method.

1.2.2 NONE

No additional points are created in this case. This is just the Moving Least Squares point cloud smoothing, the output will contain the same number of points as the input, possibly moved to better approximate the underlying smooth surface.

1.2.3 SAMPLE_LOCAL_PLANE

For each point, its local plane is sampled by creating points inside a circle with fixed radius and fixed step size. Then, using the polynomial that was fitted to the input cloud, compute the normal at the sample position and add the displacement along the normal. To reject noisy points, we increased the threshold for the number of points we need in order to estimate the local polynomial fit. This guarantees that points with a weak neighborhood (i.e., noise) do not appear in the output.

Figure 2 shows the reconstruction of coke bottles on a table. Please notice the correction for the quantization effects. The table surface is now planar and the objects look gripable. Figure 3 is a similar scenario, now using tupperware. The contour of the tupperware is much smoother and contains a lot

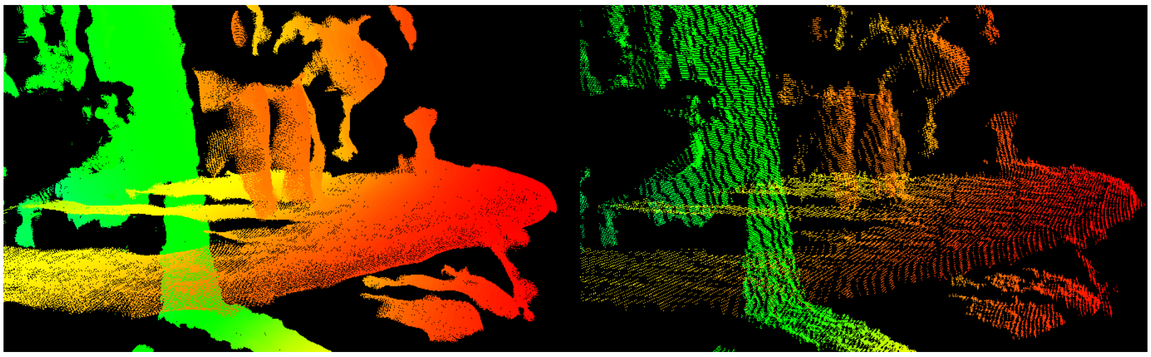


Figure 2: Moving Least Squares with SAMPLE_LOCAL_PLANE upsampling on a point cloud representing a table with bottles on top. On the right, the raw input cloud, and on the left, the processed version.

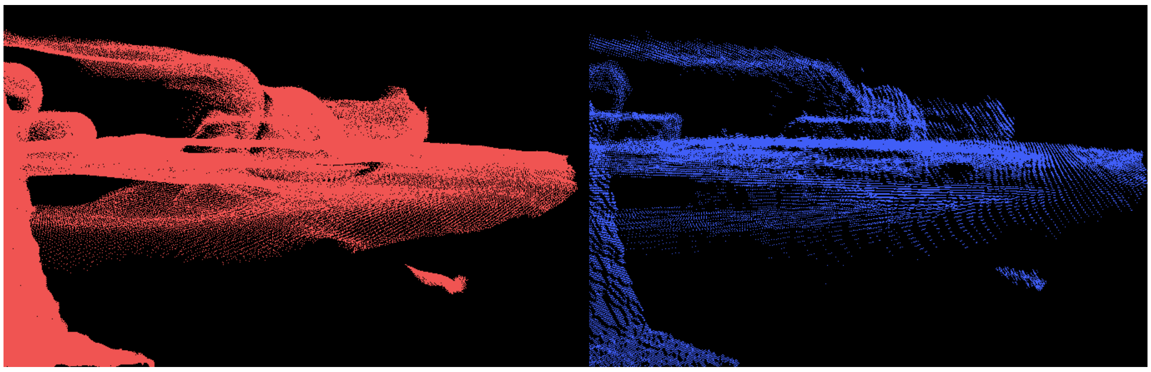


Figure 3: Moving Least Squares with SAMPLE_LOCAL_PLANE upsampling on a point cloud representing a table with tupperware on top. On the right, the raw input cloud, and on the left, the processed version.

more information. Lastly, Figure 4 shows how well the door handle is reconstructed. We conclude that visually, the results are very good.

An immediate problem is that this method adds the same amount of new samples to all points, not taking into account the local point density (i.e., points closer to the sensor will have much denser neighborhoods in the input cloud, and this will be increased even more in the output; the opposite applies for points further away from the sensor). An improvement we can make on this approach is to filter it with a voxel grid in order to have a uniform point density.

1.2.4 UNIFORM_DENSITY

This method takes as parameters a desired point density within a fixed-radius neighborhood. For each point, based on the density of its vicinity, add more points on the local plane using a random number generator with uniform distribution until the specified density is reached. We then apply the same procedure as for SAMPLE_LOCAL_PLANE to project the new samples to the MLS surface.

The results are satisfying. As compared to the previous method, we do not need to apply the expensive voxel grid filter anymore in order to get an uniformly sampled point cloud. An issue might be the fact that, because we generate the points using a random number generator, the output point cloud looks a bit messy (as compared to SAMPLE_LOCAL_PLANE, where the points are generated on a grid determined by the step size), but the surface is still well preserved. We need to note that the time performance is a slightly poorer due to the random number generator. Figure 5 shows an example on a point cloud of some curtains.

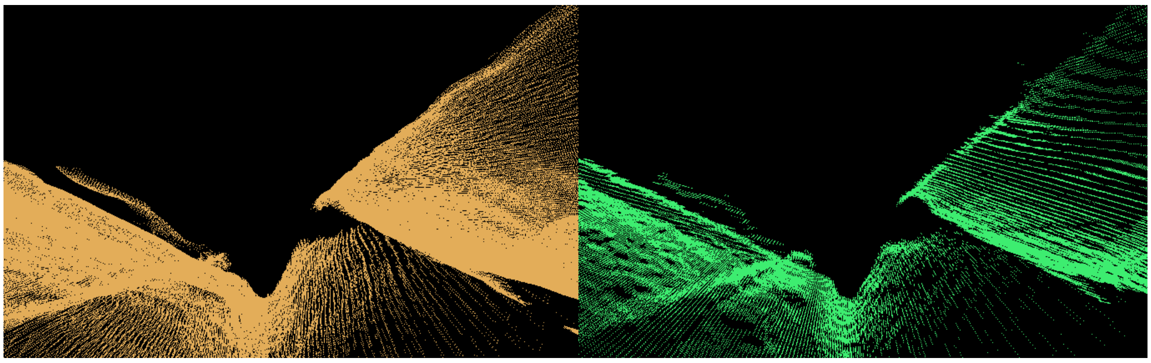


Figure 4: Moving Least Squares with SAMPLE_LOCAL_PLANE upsampling on a point cloud representing a close-up of a door handle. On the right, the raw input cloud, and on the left, the processed version.

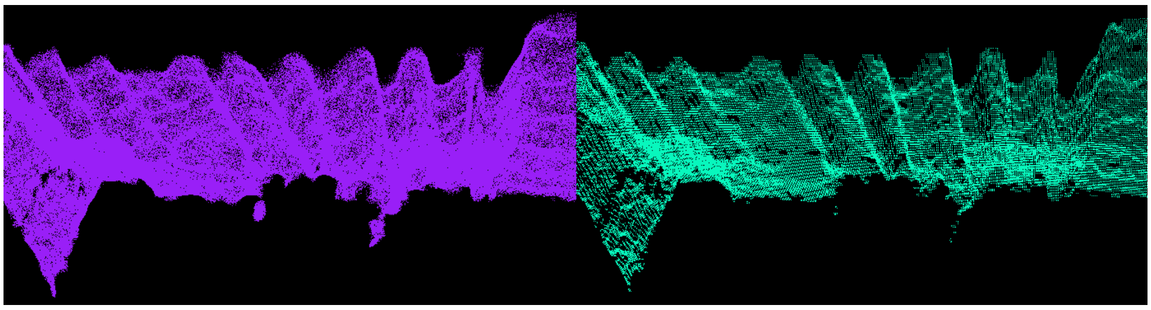


Figure 5: Moving Least Squares with UNIFORM_DENSITY upsampling on a point cloud representing curtains. On the right, the raw input cloud, and on the left, the processed version.

1.2.5 VOXEL_GRID_DILATION

The input cloud is inserted into a voxel grid which will be dilated a user-set number of times. The resulting points will be projected to the MLS surface of the closest point in the input cloud. The output is a point cloud with constant point density, and holes of various sizes are uniformly filled, based on the size of the voxel grid and the number of dilation iterations we apply. Figure 6 shows an example.

1.2.6 A Fourth Method

At the suggestions of one of the authors of paper [10], we tried the following idea, but did not succeed to make it feasibly memory efficient to handle large point clouds. Because of this reason, we did not spend a lot of time on it and did not include it in the repository.

The idea behind it is to take each point pair within a fixed radius neighborhood and to uniformly sample the line connecting these two points. Ideally, this would fill up any small holes inside the cloud. The downside is that it also creates a lot of additional points in already dense areas. A solution would have been to discretize the locations of the sampled points, but the previous methods gave us fairly good results, so we abandoned this direction.

1.2.7 Quantitative Analysis of the results

The project requirements generally stated that the results should be inspected visually, and the quality of the algorithms will mostly be assessed this way.

Table 1 shows the timing results of the MLS upsampling methods we implemented, all ran on a Microsoft Kinect scan (about 300.000 points). We tweaked the parameters such that the running time would be around 35 seconds, so that to evaluate the algorithms by the number of points they produce in the same time interval.

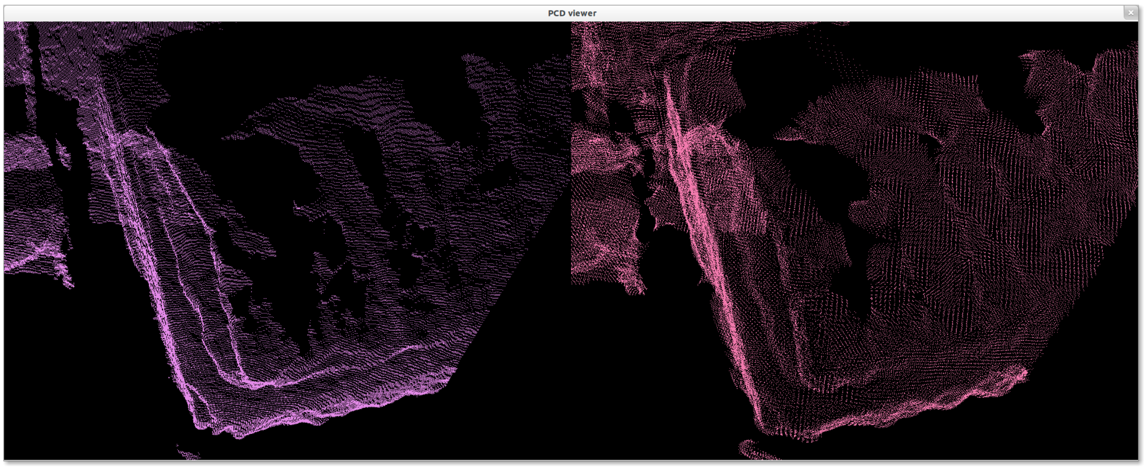


Figure 6: Moving Least Squares with VOXEL_GRID_DILATION upsampling on a point cloud. On the right, the raw input cloud, and on the left, the processed version.

Upsampling method	Time [sec]	Resulting number of points
NONE	35	256.408
SAMPLE_LOCAL_PLANE	36	2.051.264
RANDOM_UNIFORM_DENSITY	36	740.510
VOXEL_GRID_DILATION	38	1.225.989

Table 1: Performance results of the various upsampling methods for MovingLeastSquares, ran on a 300.000 points Kinect cloud.

We devised a simple method for interpreting how well our upsampled clouds represent the scanned surface. In order to do so, we scanned a wall at a distance where the noise is large (about 3m) and tried to fit a plane in each of the resulting upsampled clouds. In order to make the experiment more realistic, we took the picture of the wall at an angle, such that the quantization effects would increase along the wall. The numeric results are presented in Table 2.

Unfortunately these numerical values do not represent the actual quality of the fit, because of the varying point density across the cloud in the different upsampling methods (i.e., the parts of the wall closer to the sensor had a larger density and precision in the original cloud, and as points get farther from the sensor, the sparsity and noise increase; however, in VOXEL_GRID_DILATION and RANDOM_UNIFORM_DENSITY, the density is constant across the cloud, meaning that the noisier part of the wall has the same amount of points as the more precise part).

As such, in order to analyze the quality of the fit, we did a visual analysis of the inliers/outliers ratio, as shown in Figure 7. The conclusion is that our upsampling methods do improve the plane fitting, and that VOXEL_GRID_DILATION performs best.

1.3 Bilateral Filtering

The *BilateralUpsampling* implementation we did in PCL during this code sprint was inspired by [8]. The approach is to use the information in the RGB image (uniformly colored regions and sharp gradients) in order to enhance the depth image, in a joint bilateral filtering, based on the following formula:

$$\tilde{S}_p = \frac{1}{k_p} \sum_{q_d \in \Omega} S_{q_d} f(||p_d - q_d|| g(||\tilde{I}_p - \tilde{I}_q||)$$

where S is the depth image and I the RGB image.

The Microsoft Kinect sensor is usually used in its native mode: 640x480 RGB image, 640x480 depth image at a rate of 30 Hz. There is, however, another mode, which is useful for applying the bilateral filtering algorithm: 1280x1024 RGB image, 640x480 depth image at 15 Hz. The quality of the color

Upsampling method	Number of points	Percentage of inliers
original	275.140	81.3
NONE	275.140	81.1
SAMPLE_LOCAL_PLANE	2.201.120	81.2
RANDOM_UNIFORM_DENSITY	732.186	73
VOXEL_GRID_DILATION	1.050.394	73

Table 2: Results of plane fitting a scan of a wall, after being upsampled with MLS.

image is much higher. We implemented a new feature in the *OpenNIGrabber* (the PCL interface for the OpenNI drivers) that publishes 1280x1024 point clouds at 15 Hz, with the inherent characteristics of having each second row and column of points with nan depth values, and the last 64 rows are all nans for the depth.

We then apply bilateral filtering, in order to fill those empty rows and columns. The results we are expecting are point clouds of double the resolution in both directions (i.e., 4 times more points), smoother surfaces and filling for small holes (based on the number of iterations of the algorithm).

Figures 8, 9, and 10 show examples we ran. Please note that the algorithm did not improve the points corresponding to the glasses in Figure 8, as there was almost no depth information recorded by the sensor for the algorithm to start from. However, in the case of the bottle, we did not get a lot of points because the color difference between the label and cap on the bottle and the rest of the bottle is very large (bilateral filtering considers that only points with small color differences should belong to the same depth). Figure 10 shows the smoothing effect of the algorithm. The top image was filtered with a small σ_{color}, and the bottom image used a large value for this parameter, obtaining a much smoother wall surface.

1.4 Poisson Surface Reconstruction

The method proposed by Kazhdan in [7] poses surface reconstruction as a spatial Poisson problem, that solves for all the points at once, and allows for a hierarchy of locally supported basis functions, that reduces to a well conditioned sparse linear system. This approach is ubiquitous in the computer graphics community, being considered one of the most reliable solutions for the surface reconstruction problem. As the implementation of this method is very involved, we decided to port the code written by the author. We struggled with a few issues in adapting the code, but in the end we got it fully working within the *pcl::surface* module. We have created an interface for the algorithm that allows the user to set all the parameters of the initial cloud, and the running performance is similar to the individual application offered by the author on his website.

Unfortunately, this method cannot use raw Kinect scans to produce mesh representations, because it is targeted at producing watertight meshes. We need registered clouds for this, and Figure 11 shows such an example.

2 Future Work

There are a lot of things to be explored in the field of surface reconstruction, as the *pcl::surface* module did not receive too much attention before this code sprint. I am currently working in the Computer Graphics and Geometry Laboratory at EPFL under the supervision of Prof. Dr. Mark Pauly, one of the authors of PointShop3D [9]. This application is a collection of interactive surface editing tools that might contain algorithms of interest for the PCL community, and is definitely worth having a more thorough look into.

During the code sprint, we analyzed the surface reconstruction methods adopted by MeshLab [4], and concluded that they could not be applied to individual scans (as was the case with the Poisson surface reconstruction), but gave good results on registered clouds. As such, we are suggesting to implement algorithms such as Ball Pivoting [2], a flavor of Alpha Shapes [6] or [5], Radial Basis Functions surface reconstruction [3] etc.

References

[1] Marc Alexa, Johannes Behr, Daniel Cohen-or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9:3–15, 2003.

[2] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Claudio Silva, Gabriel Taubin, and Senior Member. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5:349–359, 1999.

[3] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA, 2001. ACM.

[4] Paolo Cignoni, Massimiliano Corsini, and Guido Ranzuglia. Meshlab: an open-source 3d mesh processing system. *ERCIM News*, (73):45–46, April 2008.

[5] Herbert Edelsbrunner. Weighted alpha shapes. Technical report, Champaign, IL, USA, 1992.

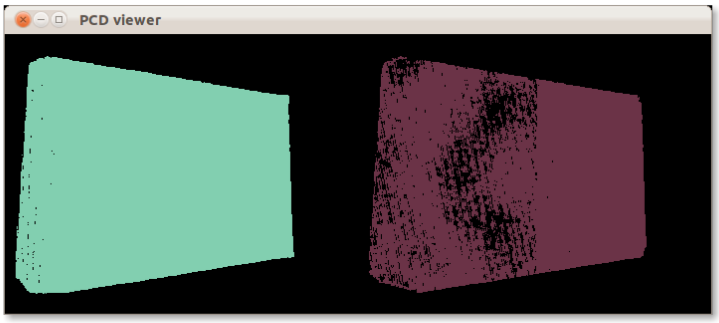
[6] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, January 1994.

[7] Michael Kazhdan, Matthew Bolitho, and Hugues 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.

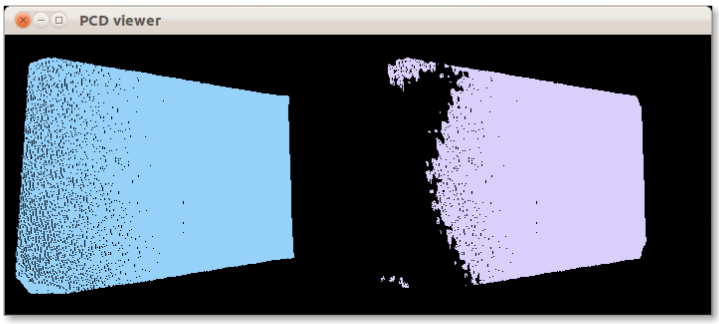
[8] Johannes Kopf, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):to appear, 2007.

[9] Oliver Knoll Tim Weyrich Richard Keiser Markus Gross Mark Pauly, Matthias Zwicker.

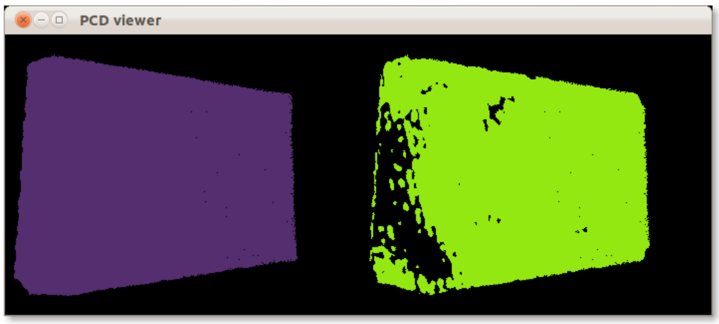
[10] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3d point cloud based object maps for household environments. *Robot. Auton. Syst.*, 56(11):927–941, November 2008.



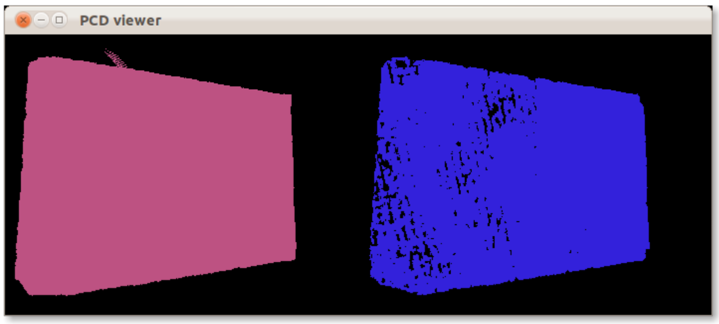
(a) Original



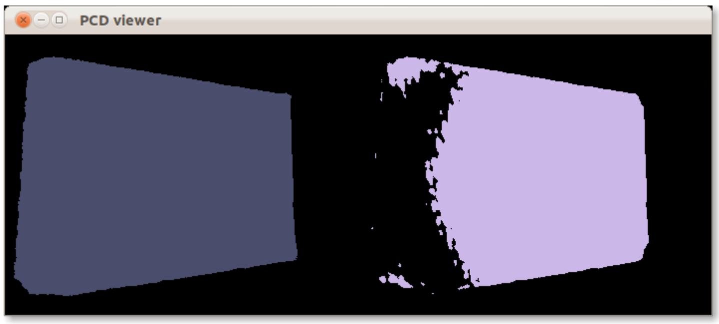
(b) NONE



(c) RANDOM_UNIFORM_DENSITY

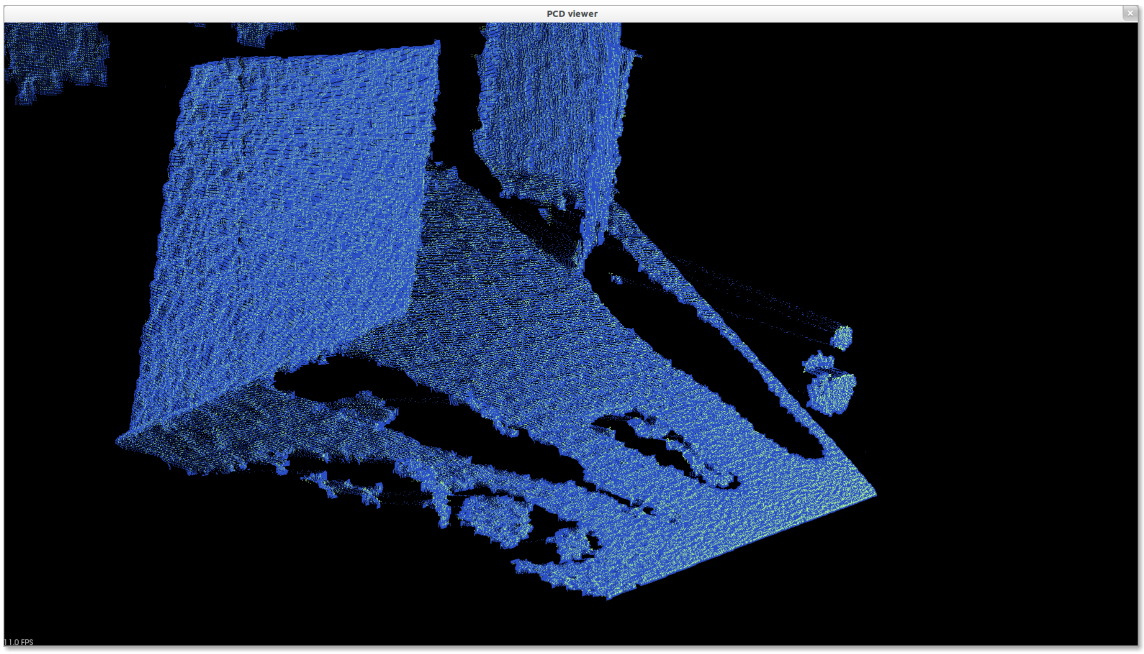


(d) VOXEL_GRID_DILATION

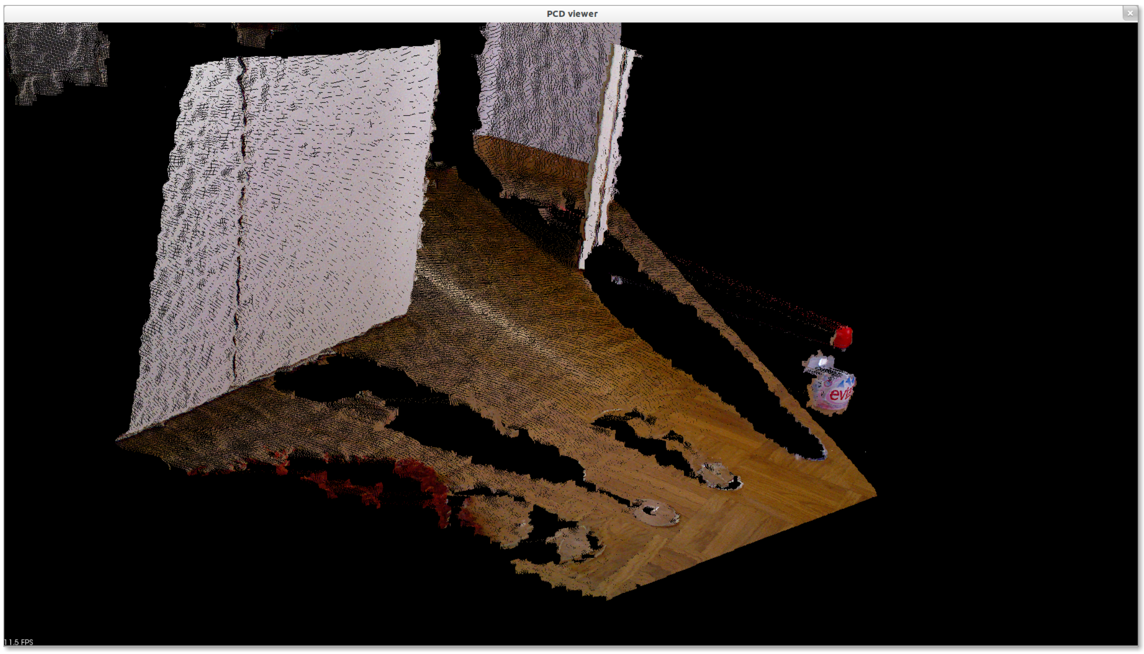


(e) SAMPLE_LOCAL_PLANE

Figure 7: Plane fitting on a wall after being smoothed and upsampled using the methods present in the Moving Least Squares implementation. On the left, there is the smoothed and upsampled cloud for each case and on the right, the plane inliers of that cloud.

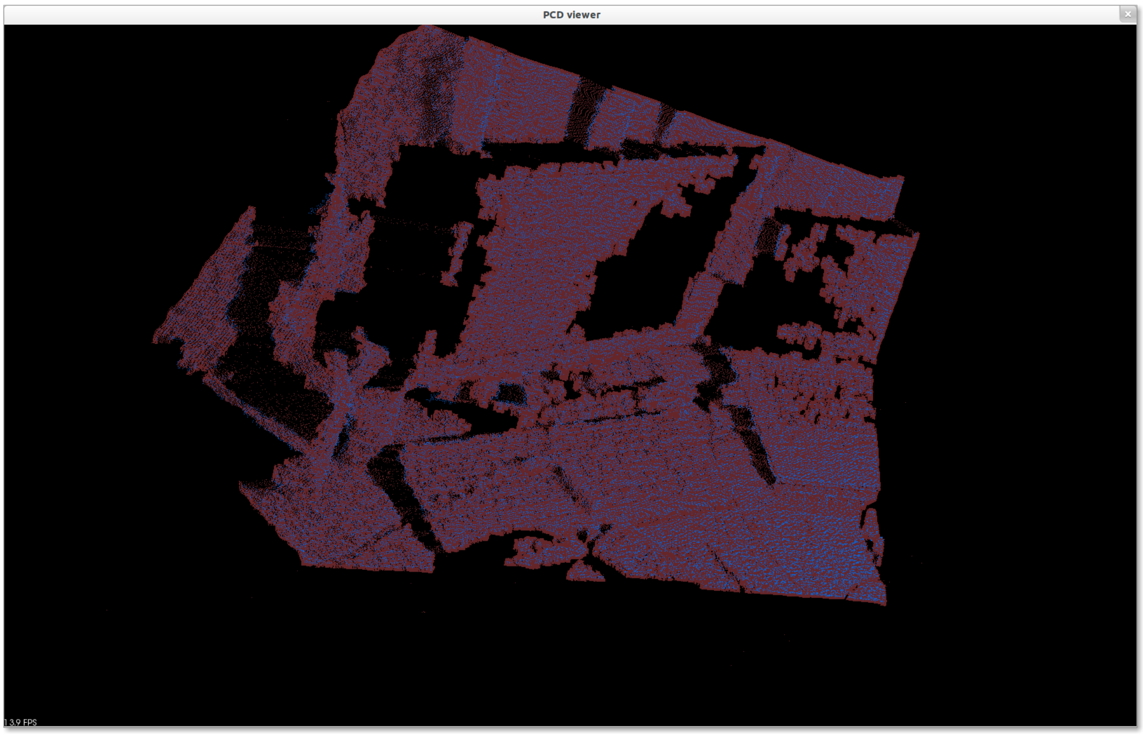


(a)

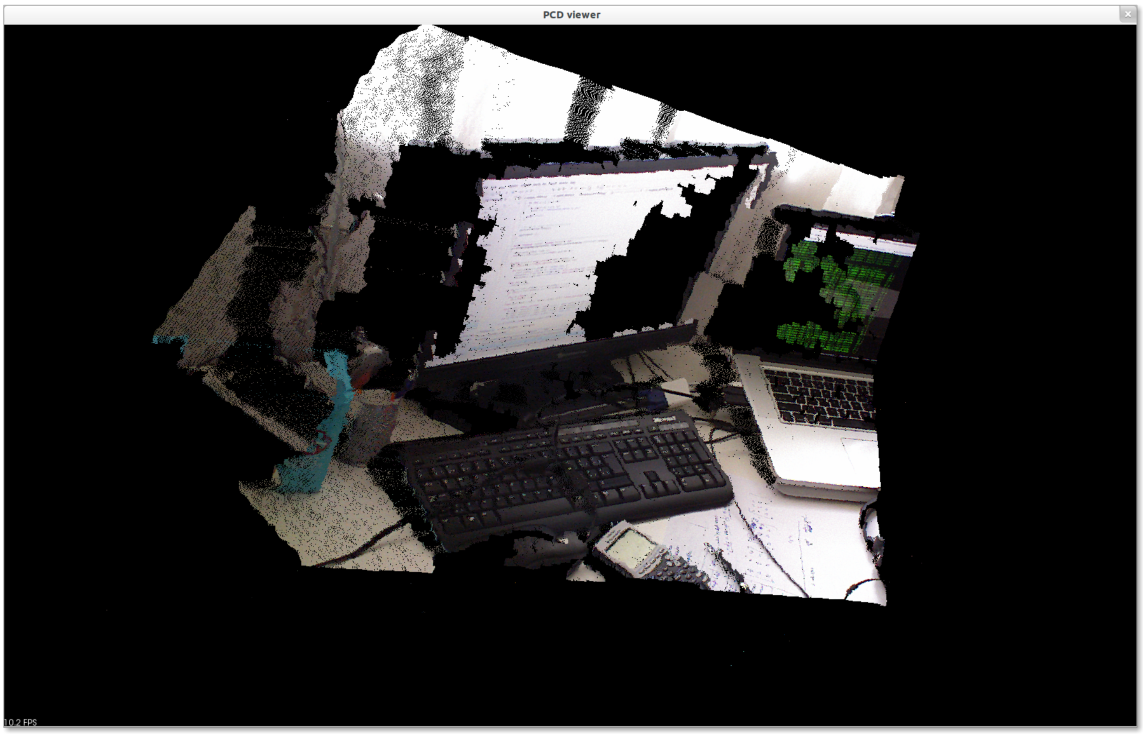


(b)

Figure 8: Bilateral Filtering results on a scan of glasses and a bottle.

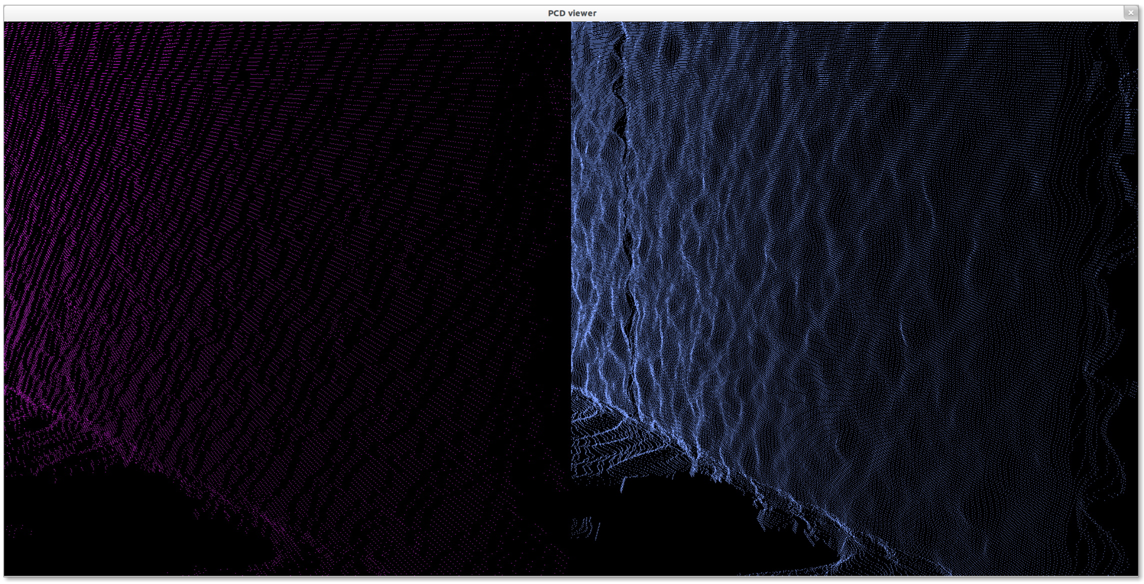


(a)

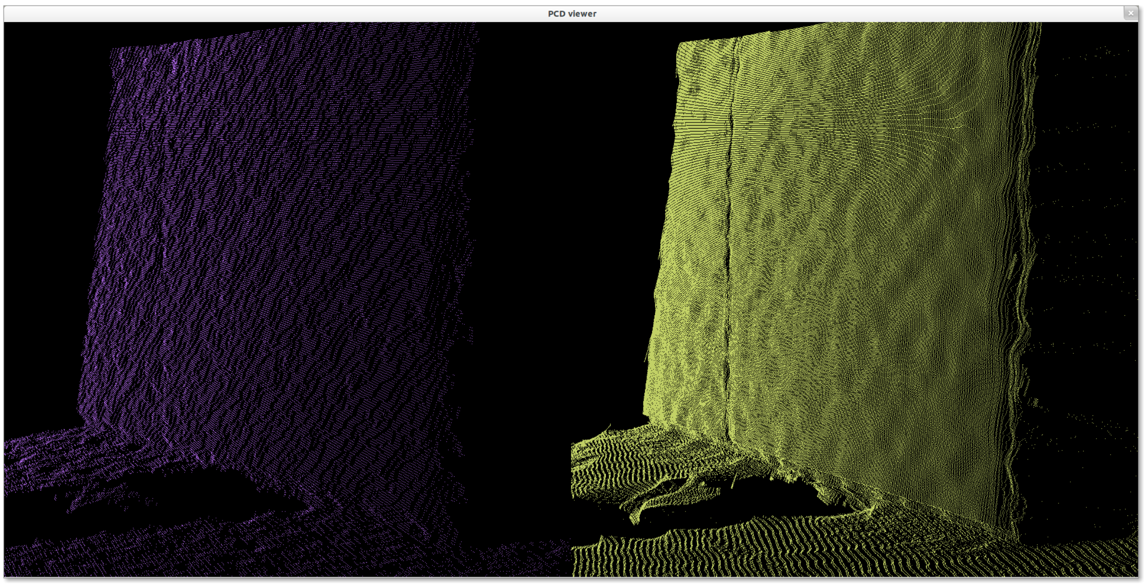


(b)

Figure 9: Bilateral Filtering results on a scan of two computer screens.



(a)



(b)

Figure 10: Bilateral Filtering results on a scan of wall, with a low value for the σ_{color} parameter, and a large value, respectively.

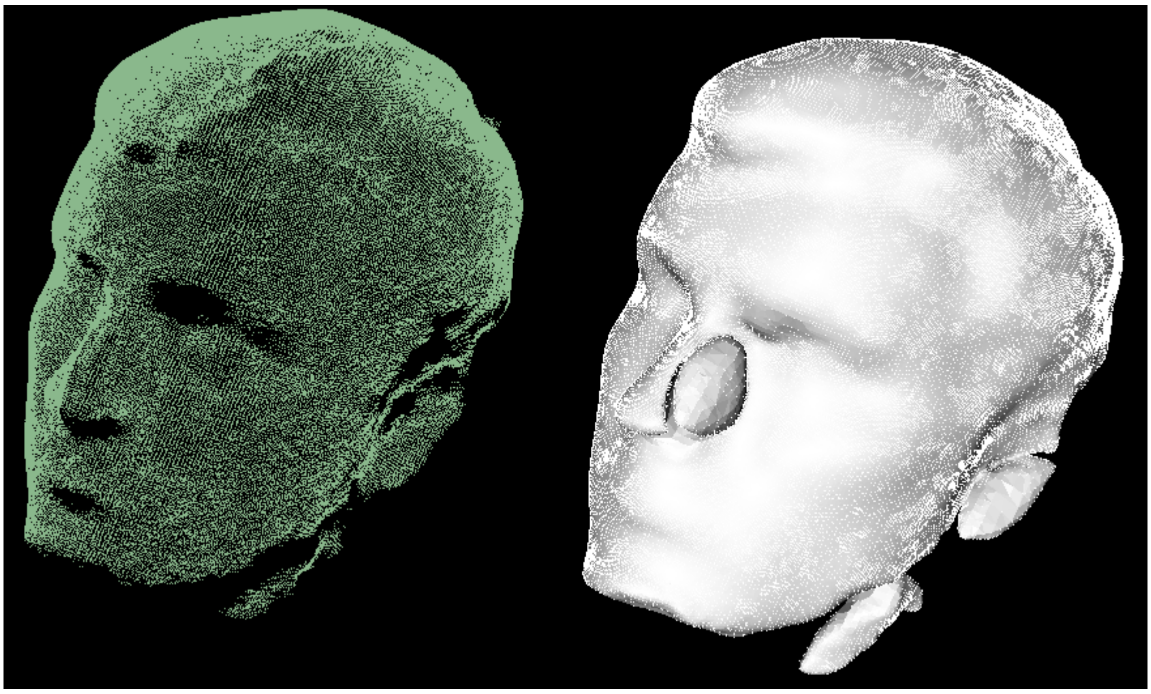


Figure 11: On the left, a registered point cloud of a face and on the right, the Poisson surface reconstruction.