Point Cloud Library - Toyota Code Sprint
Midterm Review
Point Cloud Smoothing and Surface Reconstruction Project

Alexandru-Eugen Ichim

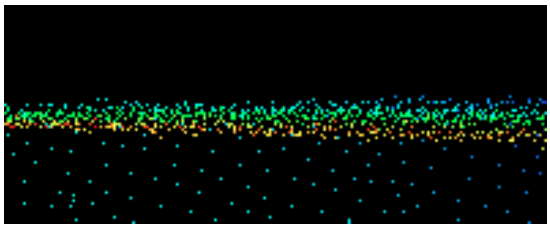
February 6, 2012

1 Probem Description

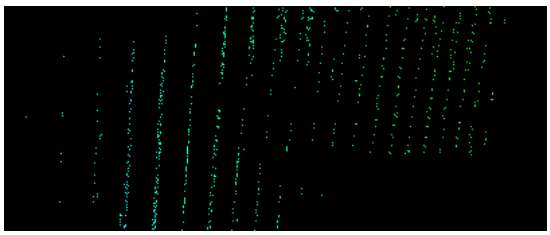
In the past months, immense advances have been made in 3D perception by the introduction of cheap 3D cameras developed by PrimeSense in the form of the Microsoft Kinect or the Asus Xtion Pro Live. The affordability of these sensors launched a revolution in terms of the amount of hobbyists, engineers and researchers interested in studying the possibilities opened.

Unfortunately, this affordability comes with the price of very poor data quality. As compared to their more expensive counterparts (e.g., time-of-flight cameras based on laser sensing), the RGB-D cameras present numerous issues that restrict their utilization in applications that need more precision. This is caused by artifacts such as:

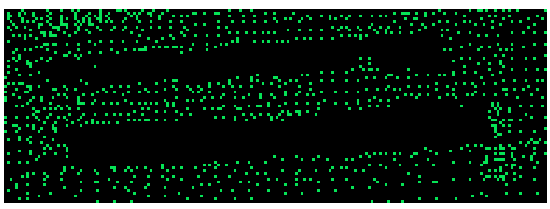
- high level of noise in both the depth and color images



- quantization artifacts



- missing pixels



- various color image distortions, specific to webcam sensors and optics

The goal of this project is to analyze the various methods available for improving the quality of the data delivered by the Kinect-like sensors. The approach we shall take is to first understand the behavior of these sensors and then look into previous work that has been done for point cloud smoothing, especially in the Computer Graphics community (who have been treating this problem years before the launch of the new generation of RGB-D cameras).

2 Gathering Datasets

As required by Toyota, we started recording a series of typical household scenes. We currently have collected 30 recordings using an Asus Xtion Pro camera. They are gathered in an SVN repository; one can easily download them by the following command:

```
svn co http://svn.pointclouds.org/data/Toyota
```

Those datasets are mainly meant to represent realistic situations that a personal robot might face in an undirected human environment. All of the scenes are recorded starting from a distance of about 3-4 meters from the main subject and getting close and rotating around it, in order to simulate the behavior of a robot and to capture most of the artifacts that the PrimeSense cameras present.

- Bed Sheets - Figure 1.
- Bottles - Figure 2.
- Door Handles - Figure 3.
- Glasses - Figure 4.
- Keyboards - Figure 5.
- Office Chairs - Figure 6.
- Wired Telephones - Figure 7.
- Shoes - Figure 8.
- Tupperware - Figure 9.
- Other - Figure 10.

3 Surface Reconstruction

After we have collected part of our datasets of interest, we proceed in testing our available smoothing algorithms. Please note that these tests use only real sensor data of scanned objects that are rather irregular, so we do not have any ground truth for our benchmarks. As such, we will limit ourselves just to a visual inspection of the results. This inspection will look mostly into sensor artifacts that we might have in the clouds after the algorithms were applied or artifacts caused by the algorithm itself (issues such as over-smoothing).

Figure 1: Bed Sheets datasets

Figure 2: Bottles datasets

Figure 3: Door Handles datasets

Figure 4: Glasses datasets

Figure 5: Keyboards datasets

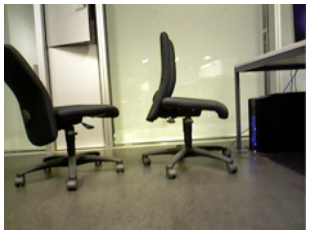


Figure 6: Office Chairs dataset

Figure 7: Wired Telephones dataset

Figure 8: Shoes datasets

Figure 9: Tupperware datasets

3.1 PCL Surface Maintainance

A target of the TOCS project was to do a thorough reorganization of the pcl::surface module, the one responsible for the point cloud smoothing and surface reconstruction algorithms. In this direction, we fixed bugs (more details on the bug tracker), cleaned up a lot of code, and also refined the architecture of the module. We have now structured it by adding three base classes which differentiate between algorithms with distinct purposes:

- **MeshConstruction** - reconstruction algorithms that always preserve the original input point cloud data and simply construct the mesh on top (i.e. vertex connectivity)
- **SurfaceReconstruction** - reconstruction methods that generate a new surface or create new vertices in locations different than the input point cloud
- **MeshProcessing** - methods that modify an already existent mesh structure and output a new mesh

3.2 VTK Smoothing Algorithms

We implemented PCL wrappers around the VTK mesh processing algorithms, and they are now fully documented and functional:

PCL Wrapper Class	VTK Class
MeshSubdivisionVTK	vtkLinearSubdivisionFilter vtkLoopSubdivisionFilter vtkButterflySubdivisionFilter
MeshSmoothingLaplacianVTK	vtkSmoothPolyDataFilter
MeshSmoothingWindowedSincVTK	vtkWindowedSincPolyDataFilter

In the developer's blog, we have presented a detailed description of the benchmarks we did. The results were not satisfactory on all the datasets. The **bed_sheets** dataset was successfully smoothed, but there were problems with more intricate structures such as the stacked **tupperware**.

For the VTK smoothing tests, we took the raw clouds, triangulated them using the OrganizedFastMesh triangulation with the TRIANGLE_ADAPTIVE_CUT option, and then fed this to the 3 smoothing algorithms from the VTK library.

The first one to be tested is *MeshSmoothingLaplacianVTK* with the default parameters recommended by VTK, but with an increase on the number of iterations from 20 to 100.

3.2.1 Bed Sheets Dataset

Here, the results are satisfactory, in both cases, the quantization artifacts are reduced, but still visible (as shown in Figure 11). The mesh after smoothing looks more natural, with a better surface curvature.

3.2.2 Bottles and Tupperware Datasets

In this case, the Laplacian smoothing does not work well anymore. The quantization and the high noise level is still present in the case of both the bottles and tupperware datasets. The main reason for this is the fact that the objects of interest were quite far away from the sensor and the quantization artifacts are quite accentuated (i.e., there are large gaps between the points belonging to the same object). See Figure 12.

The mesh subdivision schemes we have been provided by the VTK library are not of great use for our scenarios, as they just split up triangles in the mesh, inheriting from their artifacts. Furthermore, these schemes are highly dependent on the quality of the initial triangulation - which in our case is the simple OrganizedFastMesh - does not yield excellent results. They basically just resample point on the triangles present in the input mesh, without taking into consideration any more complex information about the vertex neighborhood. See Figure 13.

Another thing we tried was to combine the simple subdivision with the previous laplacian smoothing, and the results are visually decent, as shown in Figure 14. Again, we inherit the problems of the subdivision scheme (the holes caused by the incorrect triangulation).

3.3 Moving Least Squares

This is one of our most promising algorithms. We first did a benchmark using this algorithm and the results were rather good. The biggest impediment was the quantization effect of the Kinect, and thus the large spacing between points at larger distances from the camera. In order to circumvent this issue, we are currently testing and implementing a series of upsampling algorithms that would fill these holes. Please see the blog for more information and quantitative results.

3.3.1 Bed Sheets Dataset

The results seem satisfactory, in general. As we can see in Figure 15, MLS removes some of the quantization effects (note that the bed was at about 1.5-2m away from the camera), although the slices are still clearly visible. Due to the fact that the details in some wrinkles were lost using a 5 cm smoothing radius, we also tried a 3 cm radius, which seemed to reduce the over-smoothing effect.

Figure 16 shows that the usage of polynomial fitting in the MLS algorithm is useful for preserving sharp edges. One can see that the image in the middle is over-smoothed with the 5 cm radius, but the ridges are preserved in the third image.

3.3.2 Tupperware Dataset

On one hand, MovingLeastSquares seems to group points together and form visible 'long holes'. This is due to the heavy quantization errors introduced by the sensor - the table and the curtains in the back are at about 2.5-4m from the camera (see Figure 17).

On the other hand, it clearly improves the shape of the objects (Figure 18). The second figure shows a top-down view of the table. The tupperware seems much more smoother and grip-able, without loss of information.

3.3.3 Glasses Dataset

In the list of objects we are interested in, there are transparent glasses/mugs. Unfortunately, the PrimeSense technology proves incapable of recording any depth for the points corresponding to the glasses, as shown in Figure 19. There is nothing a surface reconstruction algorithm can do in order to recreate the points on the glasses, so we shall discard this dataset in our following benchmarks.

3.3.4 Bottles Dataset

As expected, the transparent parts of the plastic bottles have not been recorded by the depth sensor. The result is very satisfactory. MLS in its current implementation does not add any

points in the reconstruction, but one can notice the very good silhouette of the bottles, as compared to the very noisy input. See Figure 20.

3.3.5 Future Work

The Moving Least Squares implementation we currently have in PCL showed great potential in our tests. We are currently working on improving it by adding an upsampling step by using a variety of heuristics. Please watch the developer's blog for more news on this.

3.4 Mesh Construction Algorithms

Here we looked into the various algorithms we have for constructing meshes from a point cloud. All of them returned unsatisfactory results, as the input point cloud is not modified, only mesh connectivity is added on top of it.

3.4.1 Marching Cubes

The algorithm was first presented 25 years ago by Lorensen and Cline. In PCL, this is implemented in the MarchingCubes class with the variants MarchingCubesGreedy and MarchingCubesGreedyDot. It is greedy because of the way the voxelization is done. Starting from a point cloud, we create a voxel grid in which we mark voxels as occupied if a point is close enough to the center of a cell. Obviously, this allows us to create meshes with a variable number of vertices (i.e., subsample or upsample the input cloud). We are interested in the performance of this algorithm with the noisy Kinect data. Time-wise, the algorithm ran in about 2-3 seconds for a 640x480 cloud.

Figure 21 shows the results for various leaf sizes, and Figure 22 shows a close-up on the highest resolution cloud.

We conclude that the results are not satisfactory, as the upsampling does not inherit the properties of the underlying surface. Furthermore, there is no noise-removal mechanism and the blocking artifacts are disturbing.

3.4.2 Naive Algorithm for Organized Point Cloud Triangulation

This algorithm is implemented in the *OrganizedFastMesh* class in PCL. The idea behind is very simple: it takes each point in the inherent 2D grid of the Kinect clouds and triangulates it with its immediate neighbors in the grid. NaN points (points that were not captured by the sensor) will result in holes in the mesh. This is a mesh construction method and will output a mesh with exactly the same vertices as the input cloud. It does not take care of noise or NaN values in any way.

A screenshot of the output can be seen in Figure 23. Visually, the result is decent, considering that the processing time is extremely small - just a single pass through all the points of the clouds.

4 Tools - Virtual Scanner

Apart from the acquired datasets using a real sensor, we found it useful to have a virtual scanner. This would take in a mesh and would allow the user to place the camera in the scene and capture clouds with varying parameters (size, density), along with simulated PrimeSense camera artifacts such as quantization and depth noise (see Figure 24).

The quantization artifacts were simulated by the following approach. The depth of a pixel is defined by:

$$Z = f * b/d$$

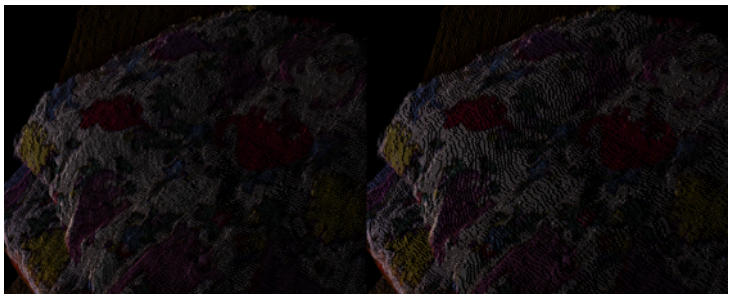
where f is the focal length in pixels, b the baseline, and d is the disparity measured in pixels. For the Kinect, the focal length was determined to be at 575 pixels, and the baseline is 7.5 cm. The Kinect quantizes the disparity by $\frac{1}{8}$-th of a pixel. For improved realism, we added Gaussian noise to the recordings before the quantization process.

A numerical example that proves this behavior:

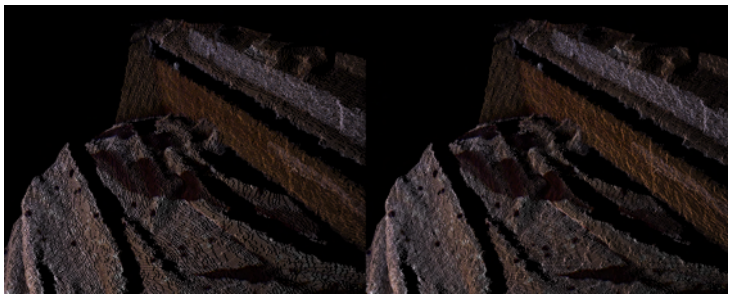
- consider a pixel with a disparity of $d_2 = 5px \Rightarrow Z_2 = 8.625m$
- the next disparity value is $d_1 = 5.125px \Rightarrow Z1 = 8.415m$
- the previous disparity value is $d_3 = 4.875px \Rightarrow Z3 = 8.8461m$
- $\Rightarrow$ the difference is of $21cm$ between the first two and increases to $22.1cm$ at the next quantized disparity value and will continue to increase at larger distances

I have written a user interface for the application, but Maurice Fallon, the other TOCS participant, informed me that he is working on a similar project and we are planning to integrate ideas from both implementations in a separate application.

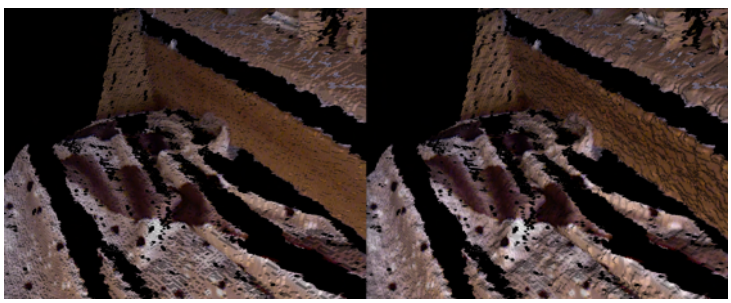
Figure 10: Other datasets



(a) set 1

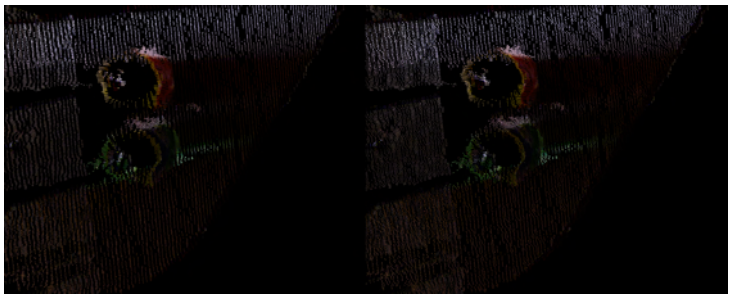


(b) set 2



(c) set 2 mesh

Figure 11: Bed Sheets smoothed using *MeshSmoothingLaplacianVTK*



(a) set 1

(b) set 2

Figure 12: Bottles and Tupperware smoothed using *MeshSmoothingLaplacianVTK*

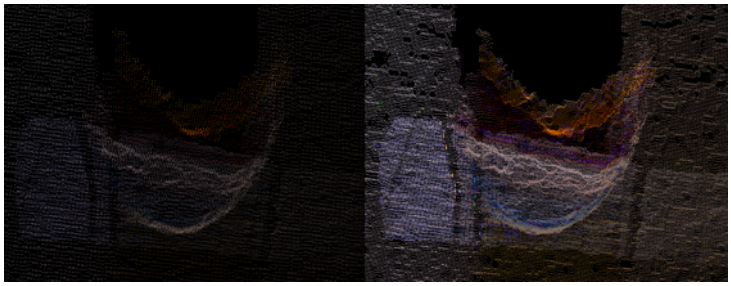


Figure 13: *MeshSubdivisionVTK* applied on a Kinect cloud

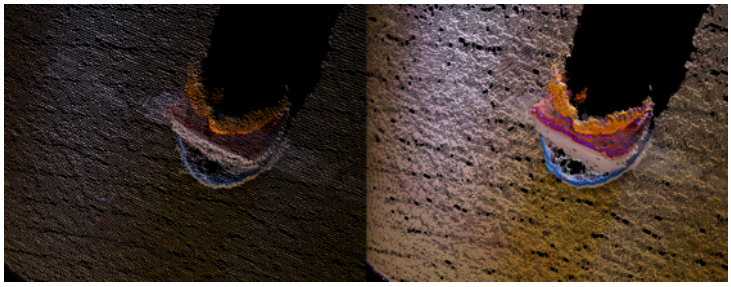


Figure 14: *MeshSubdivisionVTK* step and then *MeshSmoothingLaplacianVTK* on a Kinect cloud

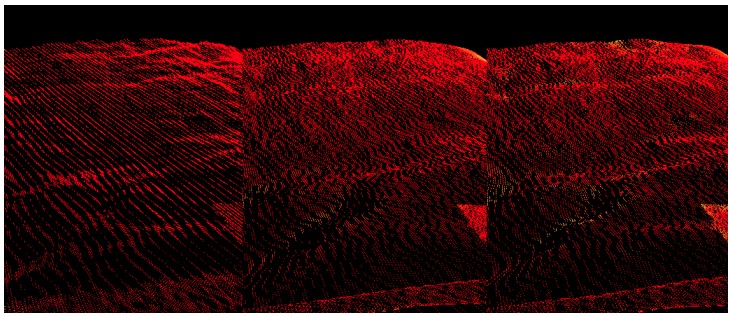


Figure 15: MLS applied on the Bed Sheets dataset; from left to right: original, MLS smoothed with $search\_radius = 0.05m$, MLS smoothed with $search\_radius = 0.03m$

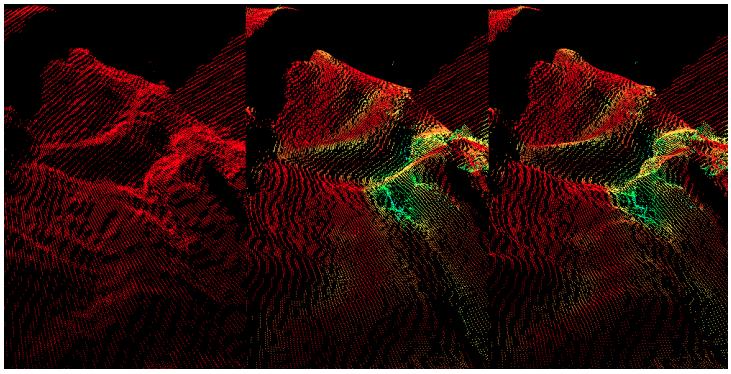


Figure 16: MLS applied on the Bed Sheets dataset; from left to right: original, MLS smoothed with $search\_radius = 0.05m$ and second order polynomial fitting, MLS smoothed with $search\_radius = 0.03m$ and second order polynomial fitting

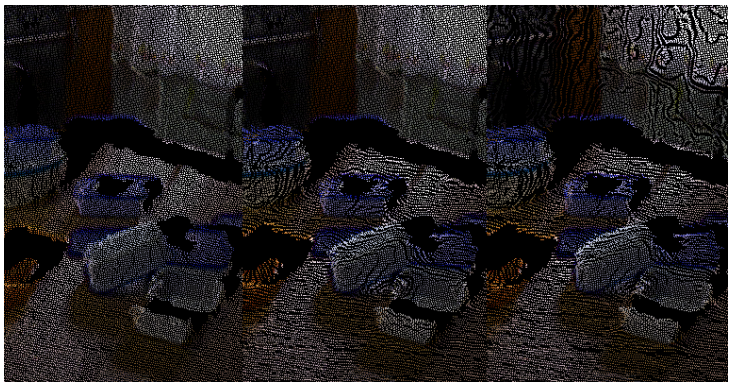


Figure 17: MLS applied on the Tupperware dataset; from left to right: original, MLS smoothed with $search\_radius = 0.03m$ and second order polynomial fitting, MLS smoothed with $search\_radius = 0.05m$ and second order polynomial fitting

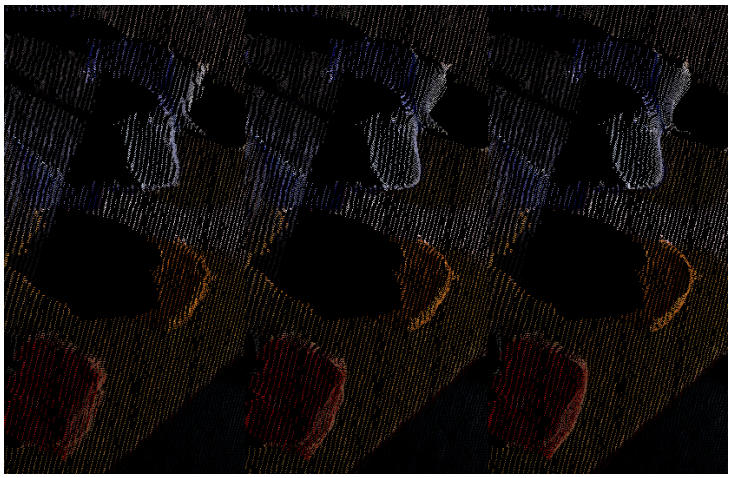


Figure 18: MLS applied on the Tupperware dataset; from left to right: original, MLS smoothed with $search\_radius = 0.03m$ and second order polynomial fitting, MLS smoothed with $search\_radius = 0.05m$ and second order polynomial fitting

Figure 19: Image showing the incapability of the Kinect to record transparent objects

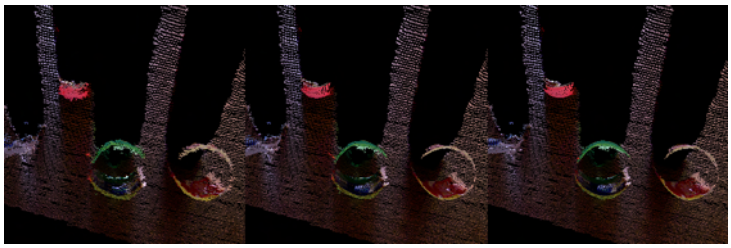


Figure 20: MLS applied on the Bottles dataset; from left to right: original, MLS smoothed with $search\_radius = 0.03m$ and second order polynomial fitting, MLS smoothed with $search\_radius = 0.05m$ and second order polynomial fitting

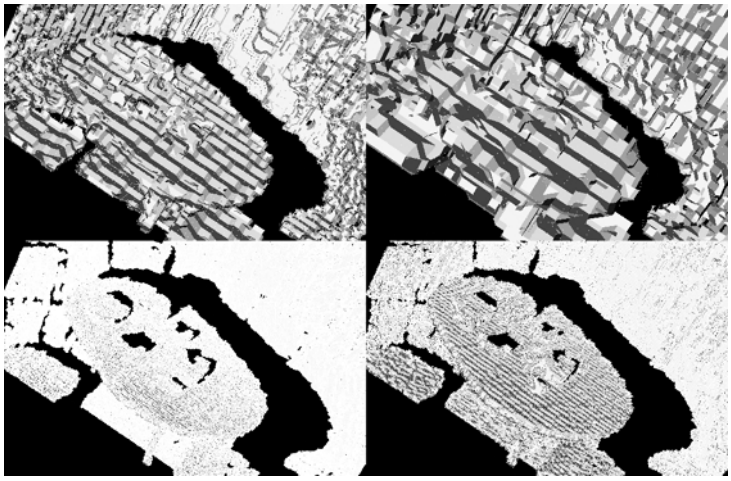


Figure 21: The Marching Cubes algorithm results: from left to right, bottom to top: leaf size of 0.5 cm, 1 cm, 3 cm, and 6 cm, respectively

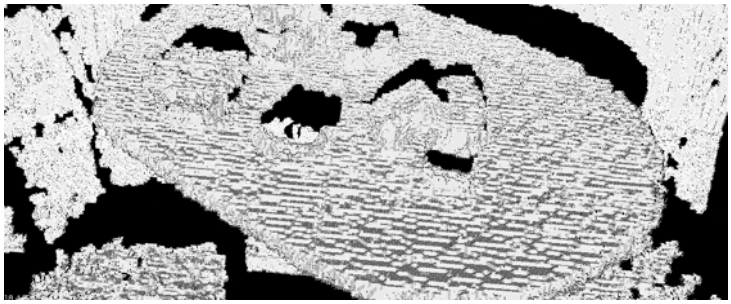


Figure 22: The Marching Cubes algorithm results: from left to right, bottom to top: leaf size of 0.5 cm, 1 cm, 3 cm, and 6 cm, respectively

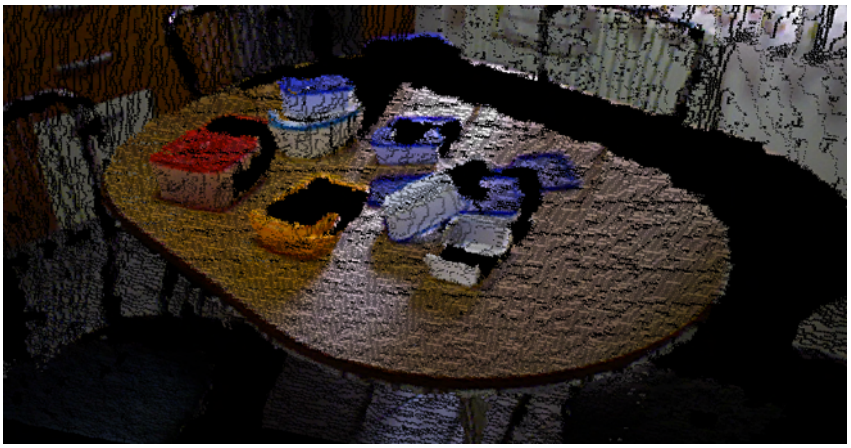


Figure 23: OrganizedFastMesh triangulation example

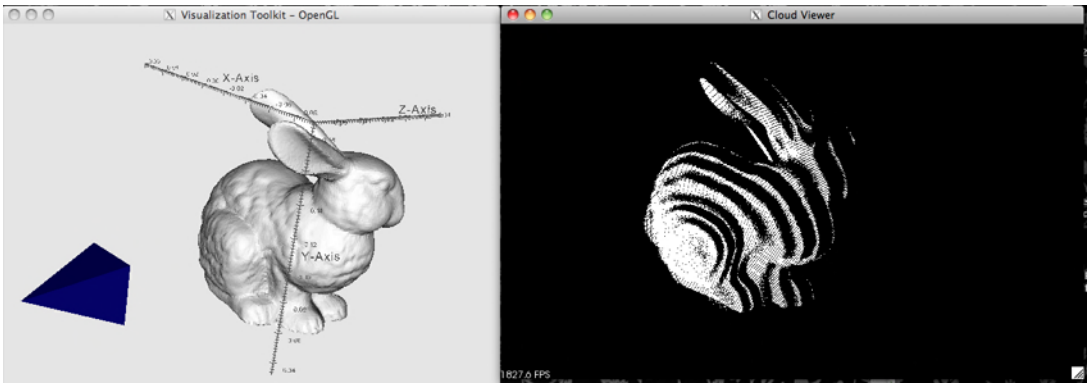


Figure 24: Virtual Scanner screenshot - tentative user interface