Point Cloud to Mesh using Poisson Surface Reconstruction

Ken Kobayashi
UC Berkeley
Berkeley, CA
takeshi_kobayashi@berkeley.edu

Zhaoyan Lin
UC Berkeley
Berkeley, CA
zhaoyan_lin@berkeley.edu

Yirong Zhen
UC Berkeley
Berkeley, CA
yirongzhen@berkeley.edu

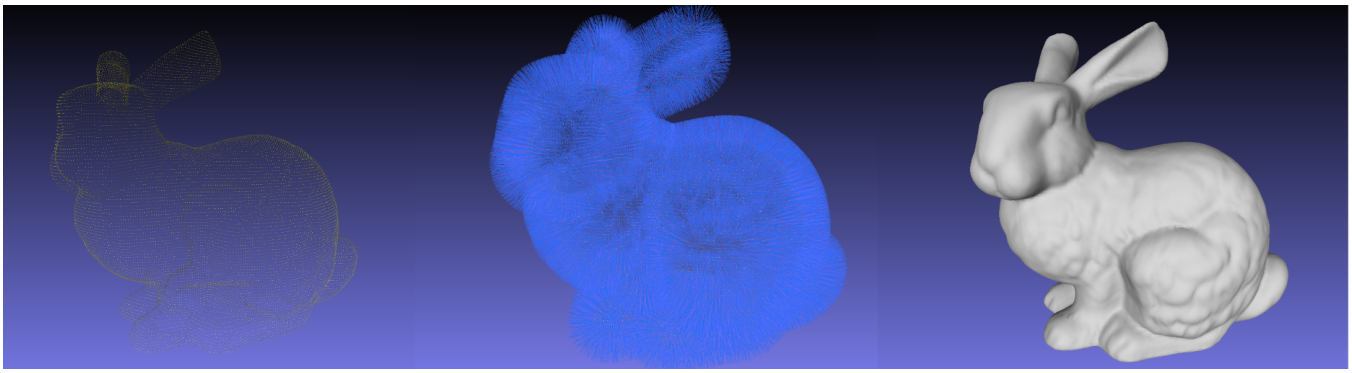

Figure 1: Point cloud, normals and reconstructed surface.

ABSTRACT

A growing number of sensors, such as LiDAR and Microsoft Kinect, become available today, with 3D point clouds being their raw output. Surface reconstruction allows recovering more features about the surface representation of objects by generating a mesh from a point cloud. In this project, we implement the key algorithm of the Poisson Surface Reconstruction method along with a pipeline of generating a mesh from point cloud data in the PLY format.

KEYWORDS

Poisson equation, surface reconstruction, point cloud, mesh, computer graphics

1 INTRODUCTION

3D scanners are now widely used to obtain virtual representations of 3D shapes in many domains. Such scanning often gives only local connectivity on the surface or a sample of vertices, like a point cloud. However, we are more interested in a mesh representation so that we can manipulate the geometry more easily. Meshes enable more applications, including smoothing, resampling, texture mapping, collision checking, and modeling in scenes.

Nonetheless, the reconstruction of surfaces from point clouds is an ill-posed problem because an infinite number of surfaces can pass through a given set of points. Moreover, the point sampling is often nonuniform. The positions and normals can be noisy due to sampling inaccuracy and scan mis-registration. Given these challenges, various surface reconstruction methods are developed, attempting to infer the topology of unknown surfaces, accurately fit the noisy data, and fill holes reasonably.

In this project, we aim to learn about how to deal with this challenging problem of reconstructing a mesh from a point cloud. The goal of our project is to take as input a point cloud with vertex positions but no normal vectors, estimate the normals, implement the Poisson Surface Reconstruction algorithm [4] to approximate an implicit surface equation and extract an isosurface to output as a mesh.

2 RELATED WORK

Surface Reconstruction Surface reconstruction refers to the problem of generating a mesh from a set of surface samples (in this project, point clouds). Surface reconstruction methods can be first categorized by the surface representation into two approaches: explicit and implicit.

As for explicit methods, the reconstructed surface interpolates most of the input samples and requires post-processing to smooth the surface and correct the topology. In our project, the poisson surface reconstruction method falls in the implicit function category. In an implicit approach, it defines a function with value less than zero outside the model and greater than zero inside. The surface is reconstructed by extracting the zero-level set.

The major difference between implicit and explicit methods is that the implicit algorithms need to be preceded by an iso-contouring technique because the outputs of these approaches are manifold surfaces and usually defined by a complex function [5]. That implicit surface needs to be triangulated by a follower method. In our project, the used approach is the Marching Cube algorithm.

Poisson Surface Reconstruction Poisson's equation is a partial differential equation with broad applications. Surface reconstruction is an inverse problem. The goal is to reconstruct a smooth surface based on a large number of points, a point cloud, where each point contains information about their location and an estimate of the local surface normal. Poisson's equation can be utilized to solve this problem with a technique called Poisson Surface Reconstruction first published in [4].

The goal of this technique is to reconstruct an implicit function f whose value is zero at the points and whose gradient at the points equals to the normal vectors. The set of points and the normal vectors are thus modeled as a continuous vector field. The implicit function f can found by integrating the vector field. Since not every vector field is the gradient of a function, the problem may or may not have a solution: the necessary and sufficient condition for a smooth vector field V to be the gradient of a function f is that the curl of V must be identically zero. In case this condition is difficult to impose, it is still possible to perform a least-squares fit to minimize the difference between V and the gradient of f.

In order to effectively apply Poisson's equation to the problem of surface reconstruction, it is necessary to find a good discretization of the vector field V. The basic approach is to bound the data with a finite difference grid. For a function valued at the nodes of such a grid, its gradient can be represented as valued on staggered grids, i.e. on grids whose nodes lie in between the nodes of the original grid. It is convenient to define three staggered grids, each shifted in one and only one direction corresponding to the components of the normal data. On each staggered grid we perform [trilinear interpolation] on the set of points. The interpolation weights are then used to distribute the magnitude of the associated component of ni onto the nodes of the particular staggered grid cell containing pi. In [4], the authors give a more accurate method of discretization using an adaptive finite difference grid, i.e. the cells of the grid are smaller (the grid is more finely divided) where there are more data points. They suggest implementing this technique with an adaptive octree.

Marching Cubes Marching Cubes is a popular algorithm for isosurface extraction originally proposed in [6] and further improved in [3]. It is used for creating a polygonal mesh from a voxel model. The basic idea is to first divide the volume into small cubes. Knowing an implicit surface equation and a targeted isovalue, if some voxels of a cube have values less than the targeted isovalue, and some have values greater than that, then the voxel must contribute some component of the isosurface. Triangular patches can be created to divide the cube into regions inside or outside the isosurface by determining which edges of the cube are intersected by the isosurface. The surface is extracted by connecting the patches from all cubes on the isosurface boundary.

3 METHOD OVERVIEW

3.1 Input

The input point clouds we used are provided the Stanford 3D Scanning Repository, like the Stanford Bunny. Besides, we manually made up some simple point clouds, including a single cube consisting of 8 vertices and some stacked cubes for experimenting our implementation. All the files are in the PLY format.

3.2 Tools

PCL [7] The Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing. We used it to manipulate the point clouds.

Eigen [2] Eigen is a header only library with very useful functions for solving linear algebra problems. We used the Conjugate Gradient Solver to solve the least squares problem before extraction of the iso-surface.

MeshLab [1] MeshLab is an open source system for processing 3D data. We use it as our main visualizer for PLY files.

3.3 Pipeline

Given a PLY file, we first parse it into a point cloud. We then estimate its normals and concatenate into an oriented point set stored in the Octree data structure using the PCL library. The Poisson Surface Reconstruction method is implemented to solve for an approximate indicator function of the inferred shape, whose gradient best matches the input normals. We use the Eigen library here to solve the least squares problem. The output is then iso-contoured using the marching cubes method, and the output of marching cubes is then stored into a PLY output file.

4 TECHNICAL APPROACH

4.1 PLY Parsing and Normal Estimations

Here we simply use the PCL library to read in the PLY file. The Stanford 3D model library contains point cloud files of different models, but we chose to use the bunny file which was one of the more simple models. After the PLY file is parsed, we need the normal vector at each point p for the Poisson reconstruction algorithm. In this step, we turn to the PCL library for such a functionality to save time. The PCL library provides the method for normal estimation. Firstly, we split the input points into smaller chunks using the spatial decomposition technique, a KD-tree. We then call the library function to perform the closest point search in that space. In PCL, we can opt for either determining a fixed number of k points in the neighborhood of point p, or all points which are found inside of a sphere of radius r centered at p. Since the scale of our point cloud can vary, we choose the first method. The normals are then calculated through eigendecomposition of the k-neighborhood point surface patch.

Next, we concatenate the normal vectors to the original point cloud. It gives us an oriented point cloud that is required by the Poisson reconstruction algorithm.

4.2 Octree

After realizing the lack of time, we decided to also leave the octree implementation to the PCL library. An octree or some sort of 3D space partitioning structure is needed for PSR because the algorithm takes advantage of global relationships. If the same thing was implemented without an octree, the algorithm would take forever to finish. The PSR algorithm also looks at each of the lowest level nodes in the tree to calculate the per node functions necessary to solve for the indicator functions.

4.3 Poisson Surface Reconstruction Implementation

Here, we followed the authors report very closely [4]. First, now that the vertices are partitioned into the octree nodes, for every node $o \in \mathcal{O}$, we define F_o to be a unit-integral, "node function"

centered about o and scaled to the size of o

$$F_o(q) \equiv F\left(\frac{q - o.c}{o.w}\right)\frac{1}{o.w^3} \tag{1}$$

Here, q is a vertex position, $o.c$ is the center of the octree node, and $o.w$ is the width of o. We now have to define the base function F, but we chose to use the function described by the authors which is:

$$F(x,y,z) \equiv (B(x)B(y)B(z))^{*n} \quad \text{with} \quad B(t) = \begin{cases} 1 & |t| < 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

This is simply the n^{th} convolution of a box filter with itself. This is essentially a smoothing filter we apply to every node. We also decided to use a value of $n = 3$ to follow the author's steps and try to reduce complications. The paper then tells us to define the vector field (at sub node precision) by using trilinear interpolation between the 8 nearest octree nodes on each sample position. The function is thus:

$$\vec{V}(q) \equiv \sum_{s \in S} \sum_{o \in \mathrm{Ngbr}_D(s)} \alpha_{o,s} F_o(q) s.\vec{N} \tag{3}$$

Where s is the sample vertex, $s.N$ is the normal of the sample, α is the trilinear interpolation weights, and $Ngbr_D$ is the 8 nearest nodes at tree depth D. Now that we have defined the vector field relative to the octree nodes, we move on to the least squares minimization problem. The report goes extensively on how the final form is reached, but for our purposes, these were the important components:

$$L_{o,o'} \equiv \left\langle \frac{\partial^2 F_o}{\partial x^2}, F_{o'} \right\rangle + \left\langle \frac{\partial^2 F_o}{\partial y^2}, F_{o'} \right\rangle + \left\langle \frac{\partial^2 F_o}{\partial z^2}, F_{o'} \right\rangle \tag{4}$$

$$v_o = \left\langle \nabla \cdot \vec{V}, F_o \right\rangle \tag{5}$$

$$\min_{x \in \mathbb{R}^{|\mathcal{O}|}} \|Lx - v\|^2 \tag{6}$$

We first had to compute the matrix L and vector v for the final solution. Once these components were constructed, we could use the Eigen Conjugate Gradient Solver to solve for x. We can then move onto calculating isovalues which closely approximate the position of the sample inputs. This is done by evaluating χ at the sample positions and using the average of the values for isosurface extraction.

$$\tilde{\chi} = \Sigma_o x_o F_o \tag{7}$$

$$\partial\tilde{M} \equiv \{q \in \mathbb{R}^3 | \tilde{\chi}(q) = \gamma\} \quad \text{with} \quad \gamma = \frac{1}{|S|}\sum_{s \in S} \tilde{\chi}(s \cdot p) \tag{8}$$

Once we have the isovalues, we can move on to using Marching Cubes to extract the isosurface.

4.4 Marching Cubes

Here we use prewritten code from the authors of PSR [4]. Their marching cubes code takes in isovalues to determine whether a point is inside the object or outside. Other than this, the code is the standard marching cubes algorithm, where it slides a cube over a 3D space and uses preset configurations, based on the points of the cube that are inside or outside the object, to construct triangles. We then take the output triangles and store them inside another PLY file using the PCL PLY writer.

5 RESULTS

In this section, we present the reconstruction results using the algorithm we implemented and analyze the results for a better understanding of the algorithm.

Our first set of experiments are done on the point cloud of a single cube. The input file is only consist of eight vertices. We intend to debug our implementation with the shape as simple as possible. In the example, we set the depth to be 3. However, the result as shown in Figure 2, it's far from good. Several faces of the cube are missing, while there are also curved surfaces along the edges. We suggest that it is the result of not enough samples and the property that faces around each vertex of the cube are orthogonal to each other. The input is too sparse for the surface features to be extracted.

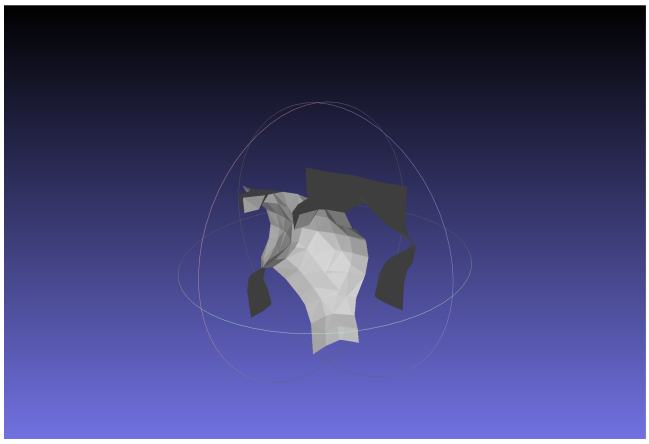

Figure 2: Reconstruction result of a cube.

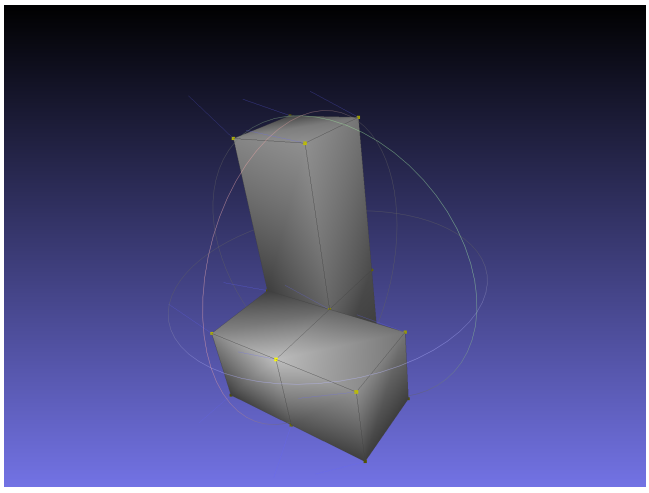

Figure 3: The stack of cubes.

We decide to move to a more complicated example of five stacked cubes in Figure 3. This point cloud is consists of 22 points. The depth used is still 3 given the simplicity. Similarly, the results are showing

missing faces. The reconstructed surface does not properly wrap the shape. The following two Figures 4, 5 show two angles of view of the stacked cubes. The point cloud is still too sparse to reconstruct the surface well.

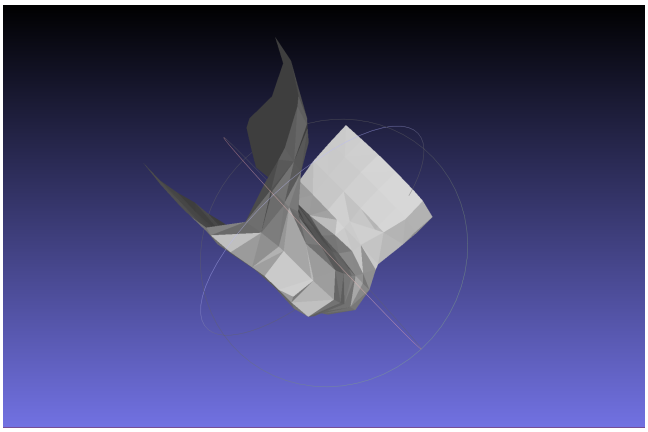

Figure 4: Reconstruction result of cubes.

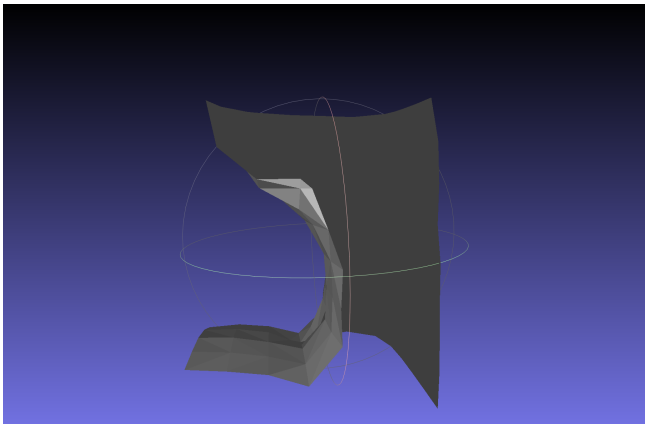

Figure 5: Reconstruction result of a cubes.

After that, we decide to test on the Stanford bunny, which has more than 30k points, using a tree depth of 10. This time, a closed surface is successfully constructed despite some defects in Figure 7. We compare it carefully with the ground truth in Figure 6. On the face of the bunny, the eyes have extra volumes connected to the ear part, looking like a cancer. Similar effects are seen on the check, the lower jaw, and the chest. Also, a platform appears under the bunny's feet, where there is supposed to be nothing. Aside the above mentioned parts, we observe that the details of the body are well reconstructed. We believe that there is an issue with our code pipeline that could be causing these issues. If we had more time, this would have been the next issue to investigate.

We also tested the bunny with different tree depths. We decrease it to be 6 in Figures 8. With a lower depth number, it is clear the details on the surface become rough. We can also see clear polygons

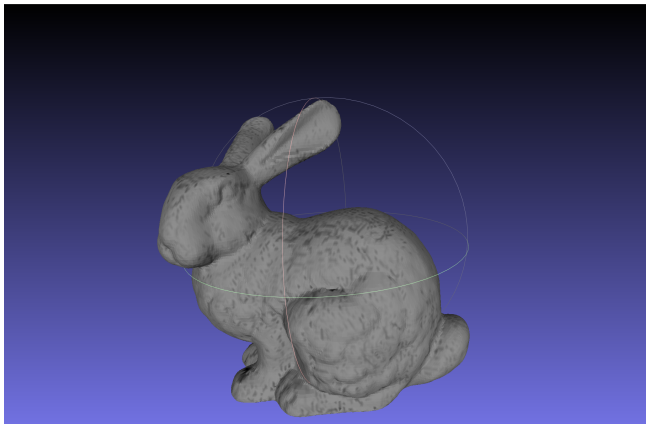

Figure 6: Ground truth.

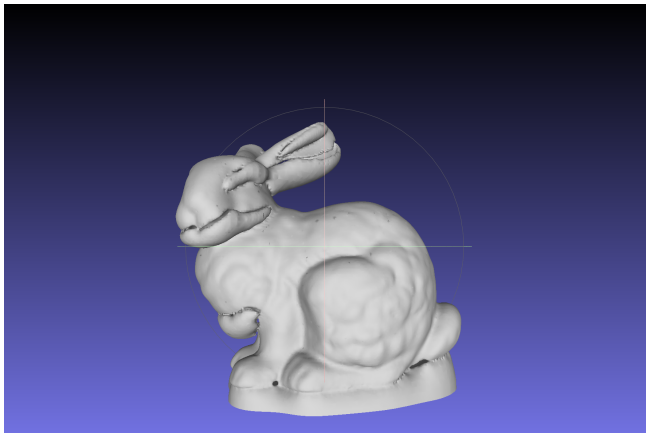

Figure 7: Reconstruction result of the bunny. Depth = 10

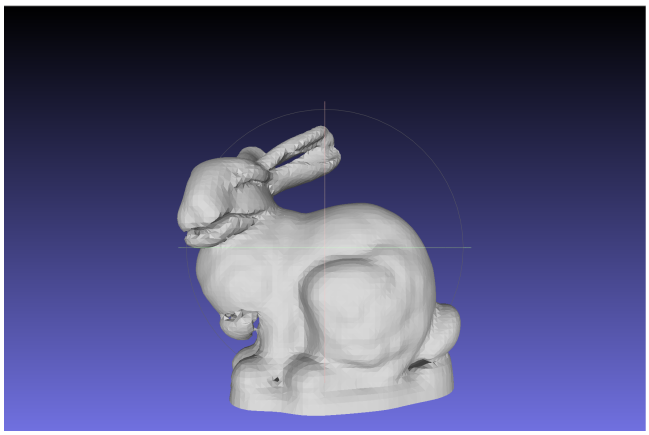

Figure 8: Reconstruction result of the bunny. Depth = 6.

in these results. This is a good sign that the algorithm implementation is somewhat working. As the tree depth decreases, then end tree nodes should be smoothing over more sample points per node,

so we essentially see a reduction in resolution. Therefore, a larger depth number is necessary when having dense point clouds, if we want more resolution in the features.

Here we also present a set of example reconstructing the surface of a sphere. It is worth mentioning that the scale of the sphere is significantly larger than the bunny. The original point cloud is in Figure 9 and the result of our implementation is in Figure 10. Poisson reconstruction is able to deal with it quite effectively. We also quickly implement the greedy projection triangulation method using the PCL library as shown in Figure 11. There are obvious holes in the reconstructed mesh. A better mesh required further processing. Hence, we show that the Poisson Surface Reconstruction method is a powerful reconstruction algorithm that generates sealed meshes in this case.

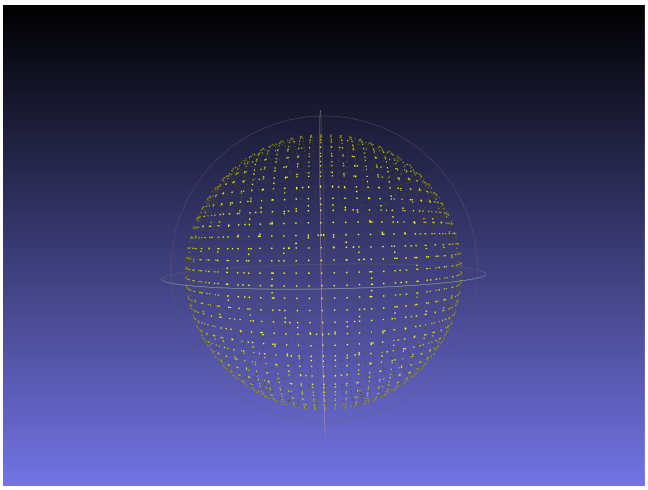

Figure 9: Point cloud of the sphere

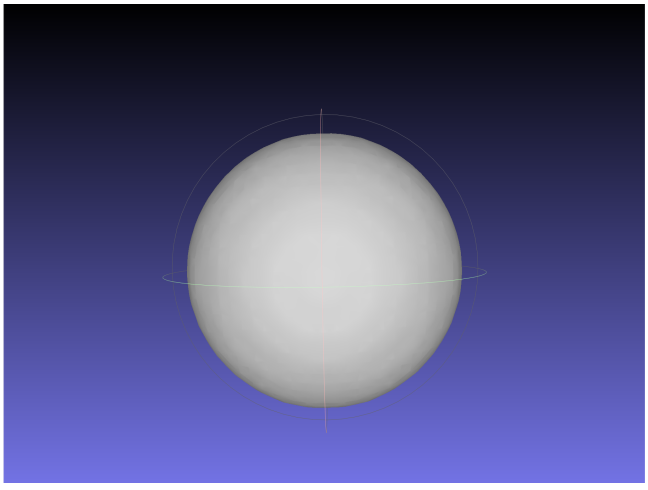

Figure 10: Reconstruction result of the sphere using Poisson

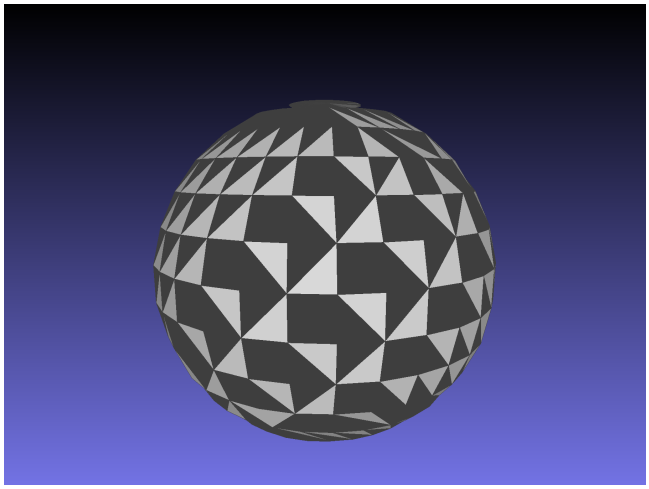

Figure 11: Reconstruction result of the sphere using greedy projection triangulation

6 CONCLUSION

In this project, we implement the Poisson Surface Reconstruction algorithm and propose the pipeline of solving the surface reconstruction problem. We take advantage of existing libraries and combine different techniques for different steps in our project. We took the time to understand the math behind the algorithm and tackle it by simplifying it into smaller stages. Our results have defects but are satisfactory results for us.

During the project, we have also learned how to research a problem, plan for a project and execute the project while changing the plan all the time. If we could have planned again, we would start understanding PSR earlier and have more time testing and improving our implementation. We have found that existing libraries were very helpful. Originally, we planned to implement parsing and visualizing the point cloud from scratch but turned to PCL to save time. Overall, we have learned lessons both in technical understanding and project management.

REFERENCES

[1] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*, Vittorio Scarano, Rosario De Chiara, and Ugo Erra (Eds.). The Eurographics Association. https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136

[2] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

[3] Wolfgang Heiden, T Goetze, and Jürgen Brickmann. 1993. Fast generation of molecular surfaces from 3D data fields with an enhanced âĂIJmarching cubeâĂİ algorithm. *Journal of Computational Chemistry* 14, 2 (1993), 246–250.

[4] Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)* 32, 3 (2013), 29.

[5] A Khatamian and Hamid Arabnia. 2016. Survey on 3D Surface Reconstruction. *Journal of Information Processing Systems* 12 (01 2016), 338–357. https://doi.org/10.3745/JIPS.01.0010

[6] William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM siggraph computer graphics*, Vol. 21. ACM, 163–169.

[7] Radu Bogdan Rusu and Steve Cousins. 2011. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China.