Implicit 3D Model Surface Mesh Reconstruction

Xuhui Wang

November 2021

Abstract

3D model surface reconstruction from point clouds, widely used in medical imaging, industry design, model visualization and a variety of engineering community, is a difficult challenge in geometry processing field, of which the main challenges include the robustness to noise, the reconstruction process without surface normals and so on. Especially, the reconstructed surface need to be topologically equivalent to the sampled surface and also geometrically close. In this project, to perform this challenging task, I introduce and implement a sequence of methods that firstly compute an implicit MLS function approximating a 3D point cloud with given (but possibly unnormalized) normals, sample the implicit function on a 3D volumetric grid using moving least square method, apply the marching tets algorithm to extract a triangle mesh of this zero level set, and finally experiment with various MLS reconstruction parameters.

Introduction

(Note that, in this report, the wordings that describe the algorithms are all from the course material by Dr. Schneider and I will give the appropriate citation at the end of each corresponding paragraph. On the other hand, all resulted figures, methods implementation, statistics, analysis, explanations and conclusions are all from my personal work.) Our main task is to construct an implicit function $f(x)$ defined on all $x \in R^3$ whose zero level set contains (or at least passes near) each input point. That is, for every point p_i in the point cloud, we want $f(p_i) = 0$. Furthermore, ∂f (the isosurface normal) evaluated at each point cloud location should approximate the point's normal provided as input. We construct f by interpolating a set of target values, d_i, at "constraint locations", c_i. The MLS interpolant is defined by minimization of the form

$$f(x) = \underset{\phi}{argmin} \sum w(c_i, x)(\phi(c_i) - d_i)^2$$

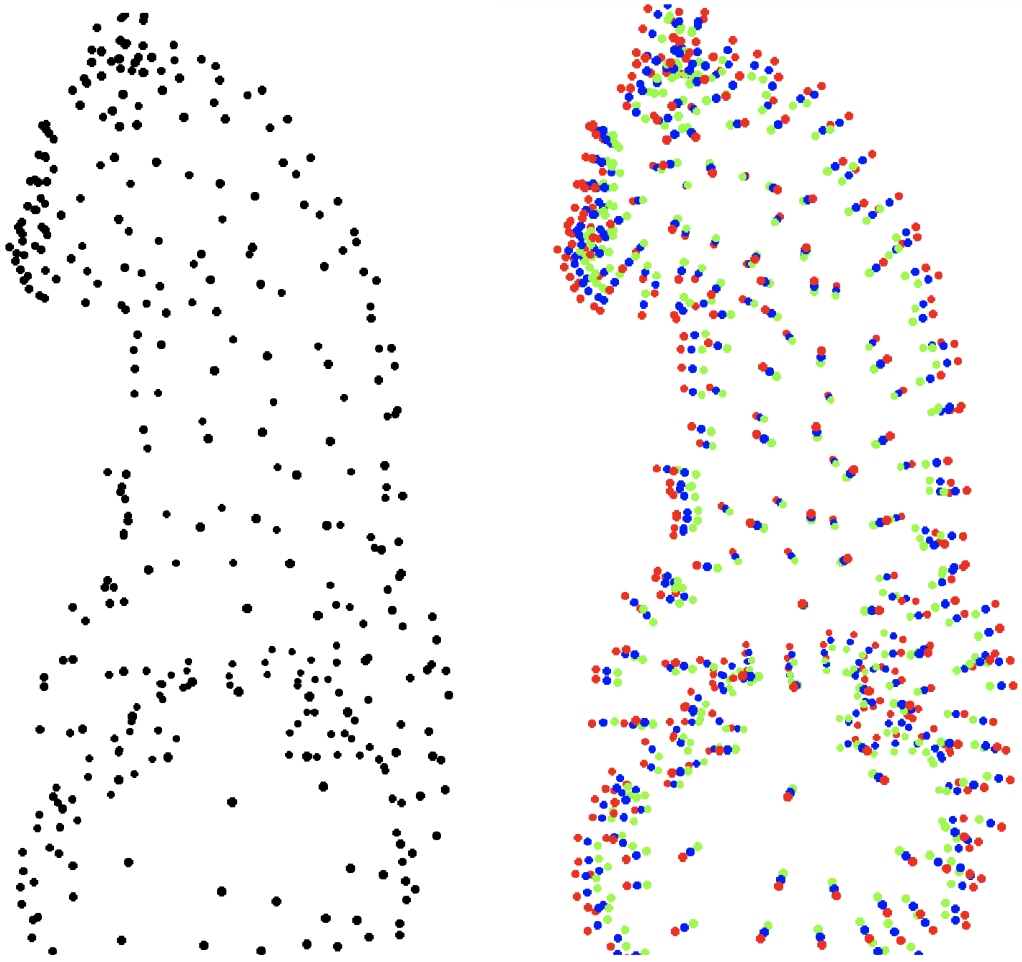
where $\phi(x)$ lies in the space of admissible function (e.g., multivariate polynomials up to some degree) and w is a weight function that prioritizes each constraint equation depending on the evaluation point, x.[1]

Setting up the Constraints

Our first step is to build the set of constraint equations by choosing constraint locations and values. Naturally, each point p_i in the input point cloud should contribute a constraint with target value $f(p_i) = d_i = 0$. But these constraints alone provide no information to distinguish the object's inside (where we want $f < 0$) from its outside (where we want $f > 0$). Even worse, the minimization is likely to find the trivial solution $f = 0$ (if it lies in the space of admissible functions). To address these problems, we introduce additional constraints incorporating information from the normals. [1]

For each point p_i in the point cloud, we first add a constraint of the form $f(p_i) = d_i = 0$. Then we fix an eps value, for instance $eps = 0.01\times$ the bounding box diagonal. For each point p_i, we compute $p_i+ = p_i + eps \times n_i$, where n_i is the normalized normal of p_i. Check that p_i is the closest point to p_i+. If not, halve eps and recompute p_i+ until this is the case. Then, we add another constraint equation: $f(p_i+) = eps$. After this, we repeat the same process for $-eps$, i.e., add equations of the form $f(p_i-) = -eps$. we also need to check each time that p_i is the closest point to p_i-. Finally, we append the tree vectors p_i, p_i+, and p_i- and corresponding f to a unique vector p and f. After these steps, we would have $3n$ equations for the implicit function $f(x)$. [1]

We use the point cloud of a cat object as the example here. Figure 1 shows the input point cloud for the cat mesh and inward/outward value constraints. The green, red and blue labels correspond to inside, outside, and on the surface respectively. The blue points in the right figure are the same as the black ones in left figure.



(a) input point cloud (b) inward/outward value constraints

Figure 1: The input point cloud for the cat mesh and inward/outward value constraints

MLS interpolation

We create a regular volumetric grid around the point cloud: compute the axis-aligned bounding box of the point cloud, enlarge it slightly, and divide it into uniform cells (tets). The grid resolution is configured by the global variable resolution which can be changed. We call the grid vertices x and the tets connecting them T.

MLS interpolation is used here to construct an implicit function satisfying the constraints as nearly as possible. We won't define the function with an explicit formula; instead, we characterize it as the linear combination of polynomial basis functions that best satisfies the constraints in some sense. At a given point x_i in x, we evaluate this function by finding the "optimal" basis function coefficients (which will vary from point to point!) and using these to combine the basis function values at x_i. [1]

Specifically, we evaluate the MLS function at every node x_i of a regular volumetric grid containing the input point cloud. For each grid node x_i of the grid, evaluate the implicit function $f(x_i)$, whose zero level set approximates the point cloud. We use the $Wendland$ weight function with radius configured by $wendlandRadius$ and degree $k = 0, 1, 2$ polynomial basis functions configured by $polyDegree$. Only use the constraint points

with nonzero weight (i.e., points p with $||x_i - p|| < wendlandRadius$). Note that if the number of constraint points within $wendlandRadius$ is less than twice the number of polynomial coefficients (i.e., 1 for $k = 0$, 4 for $k = 1$, and 10 for $k = 2$), we can assign a large positive (outside) value to the grid point. [1]

For this step, we store the field value $fx = f(x_i)$ in a numpy array, using the same ordering as in x. As shown in Figure 2, we render these values by coloring each grid point red/green depending if they are inside outside (i.e., $fx < 0$ or $fx > 0$).

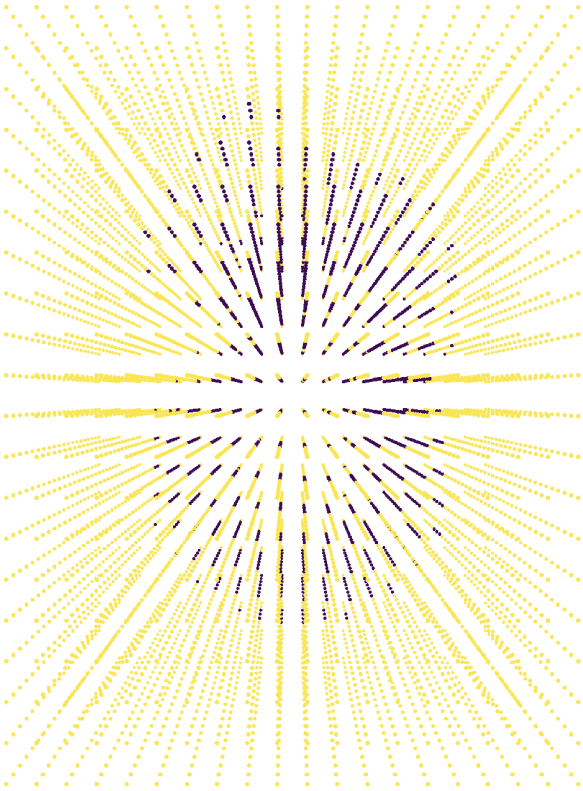


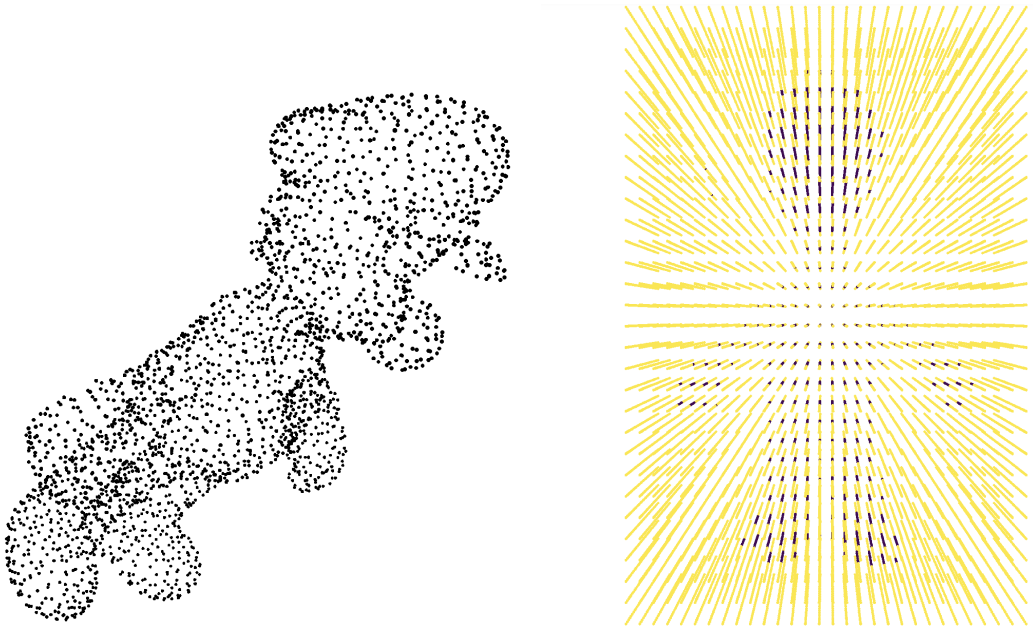
Figure 2: Plot of the grid points x colored according to being inside or outside the input cloud

Optimization

Note that to construct the MLS equations, we will perform queries to find, for a query point q, the closest input point to q (needed while constructing inside/outside offset points), and all input points within distance h of q (needed to select constraints with nonzero weight). Although a simple loop over all points could answer these queries, it would be slow for large point clouds. We improve the efficiency by implementing a simple spatial index (a uniform grid at some resolution). By this, we mean binning vertices into their enclosing grid cells and restricting the neighbor queries to visit only the grid cells that could possibly satisfy the query. [1]

Commonly, point clouds are not aligned with the canonical axes. Running reconstruction on an axis-aligned grid is wasteful in this case: many of the grid points will lie far outside the object. We also need to devise an automatic (and general) way to align the grid to the data and implement it. [1]

We use the point cloud of a Luigi object as an example here as shown in Figure 3. The processing time is way shorter than using brute force loop of computing.



(a) (b)

Figure 3: Plot of the grid pointsxcolored according to being inside or outside the input cloud

Extracting the Surface

After the above optimizations, we can now use marching tets to extract the zero isosurface from the grid. The extraction has already been implemented, and the surface is displayed. The implicit function obtained from MLS might be noisy, and the reconstructed mesh will contain several pieces. We filter out and keep the largest component. [1]

Combining with the previous parts, we display multiple output with different parameters. It could be observed that the surface becomes finer as the resolution increases and $Wendland$ function radius decreases.

Firstly, as shown in Figure 4, we plot the reconstructed Luigi surface with the parameters: resolution = (40,40,40), polydegree = 1, $Wendland$ function radius = 0.1 box diagonal.

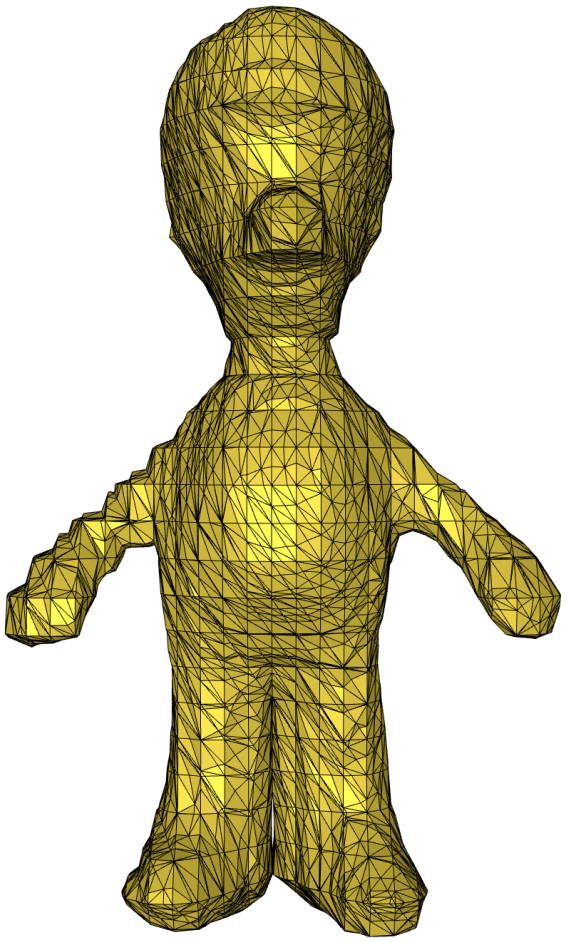


Figure 4: The reconstructed Luigi surface using marching tets and filtering

Then, as shown in Figure 5, we plot the reconstructed cat surface with the parameters: resolution = (40,45,40), polydegree = 1, $Wendland$ function radius = 0.1 box diagonal.

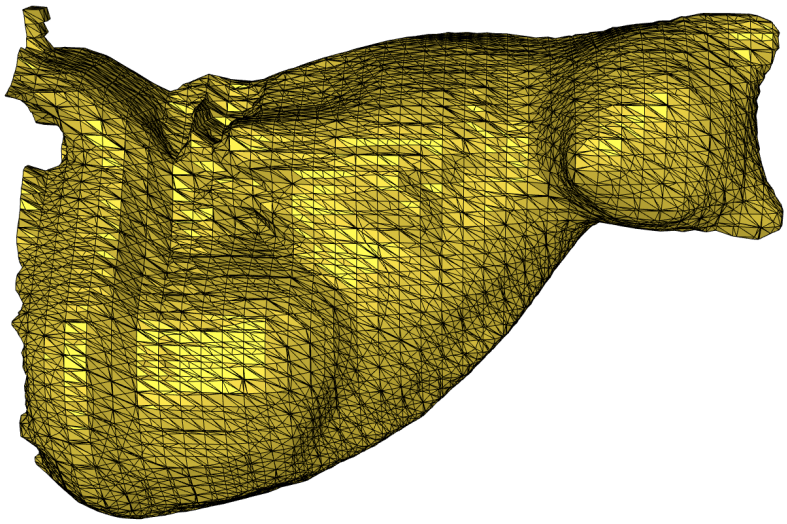


Figure 5: The reconstructed cat surface using marching tets and filtering

Then, as shown in Figure 6, we plot the reconstructed cat surface with the parameters: resolution = (20,35,20), polydegree = 0, *Wendland* function radius = 0.1 box diagonal.

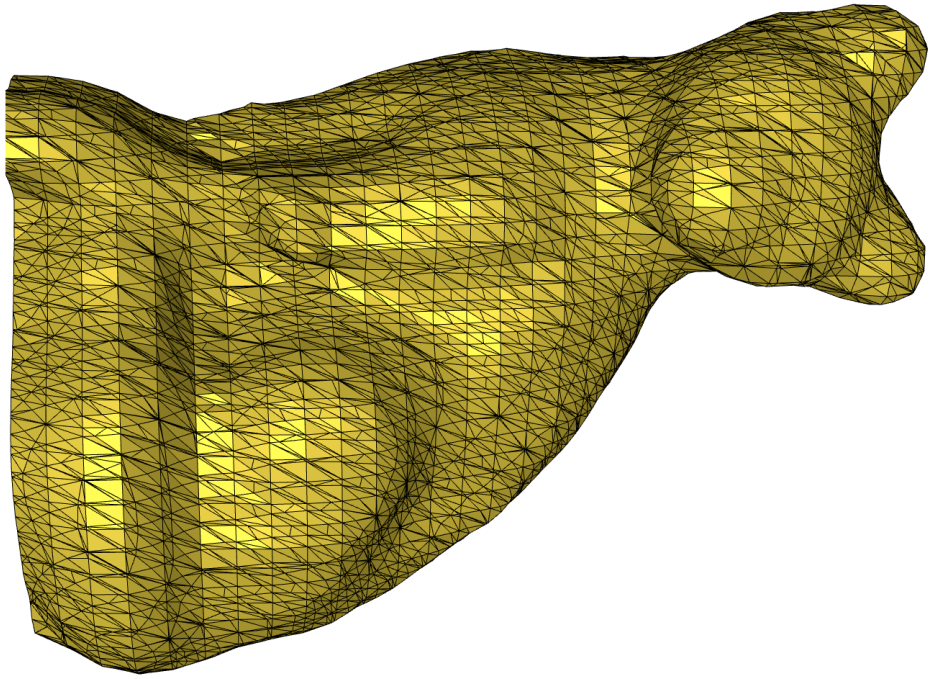


Figure 6: The reconstructed cat surface using marching tets and filtering

Then, as shown in Figure 7, we plot the reconstructed cat surface with the parameters: resolution = (20,30,35), polydegree = 2, *Wendland* function radius = 0.2 box diagonal.

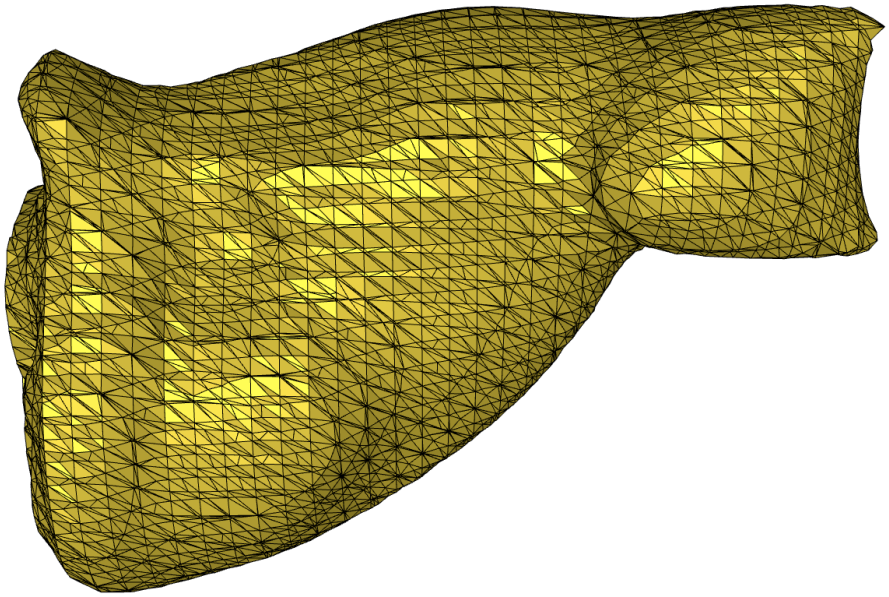


Figure 7: The reconstructed cat surface using marching tets and filtering

Then, as shown in Figure 8, we plot the reconstructed cat surface with the parameters: resolution = (25,30,25), polydegree = 1, *Wendland* function radius = 0.15 box diagonal.

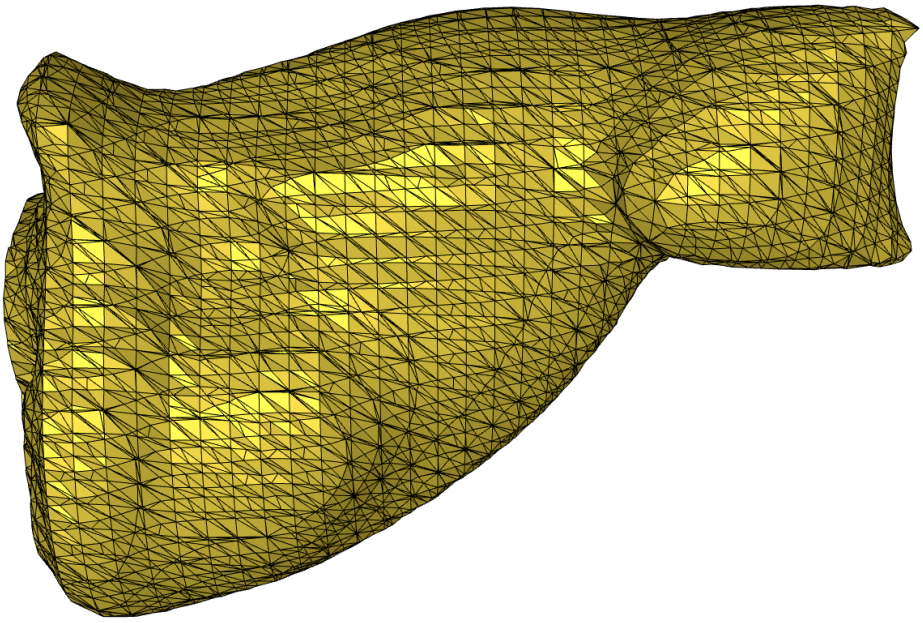


Figure 8: The reconstructed cat surface using marching tets and filtering

Conclusion

In this project, to perform implicit 3D model surface reconstruction, I introduced a sequence of methods that firstly compute an implicit MLS function approximating a 3D point cloud with given (but possibly

unnormalized) normals, sample the implicit function on a 3D volumetric grid using moving least square method, apply the marching tets algorithm to extract a triangle mesh of this zero level set, and finally experiment with various MLS reconstruction parameters. In addition, I used two different models as examples to visually demonstrate the effectiveness of the methods. The smooth and noise-free model surfaces resulted show that MLS and marching tets methods are efficient and effective to accomplish the tasks of computer model surface reconstruction in application.

References

[1] T. Schneider, *Csc486b: Geometric modelling course materials.*