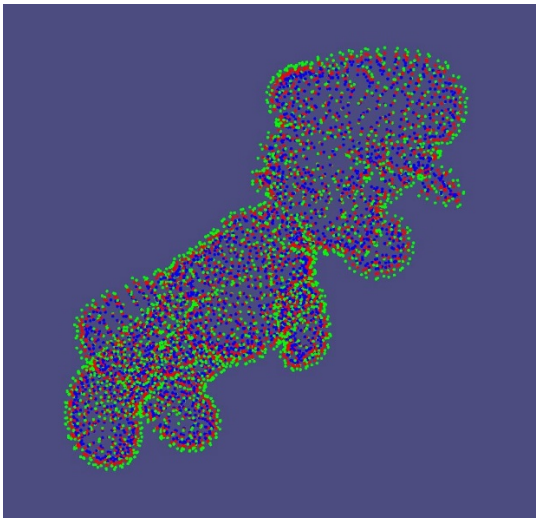
Assignment2- Report

Dayou Du (dd2645)

Github id: potato1996

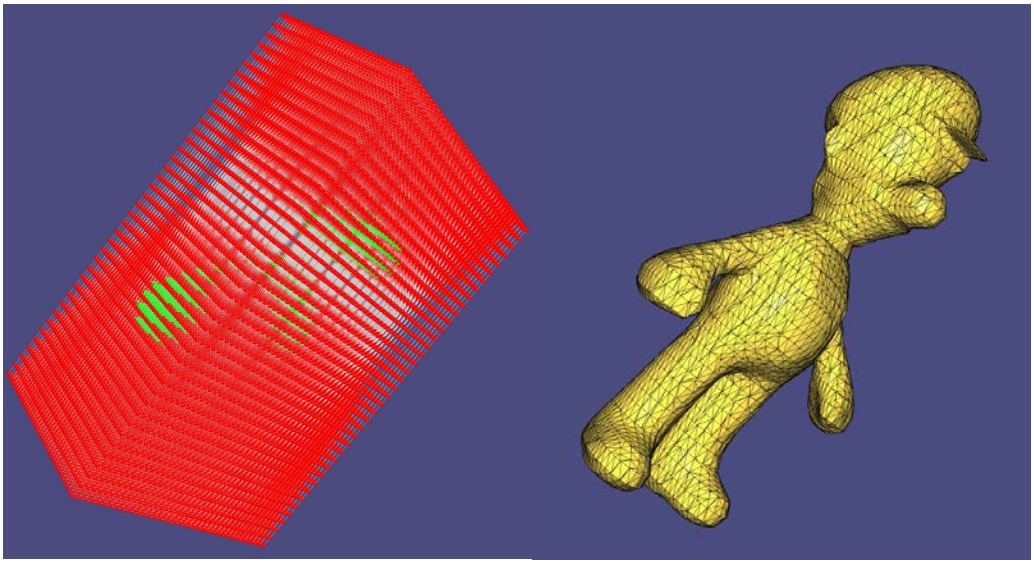
Section 1: Setting Up the Constrains

Here is a demo on model luigi.off. In my settings green, red, and blue points means outside, on, and inside the surface correspondingly.



Section 2: MLS Interpolation

Here is a demo with colored grid nodes and on model luigi.off (grid resolution 40x40). The red and green points correspond to outside and inside the surface. We can see that although the original point cloud is not axis-aligned, we solved this problem with PCA method and created an object-aligned grid.

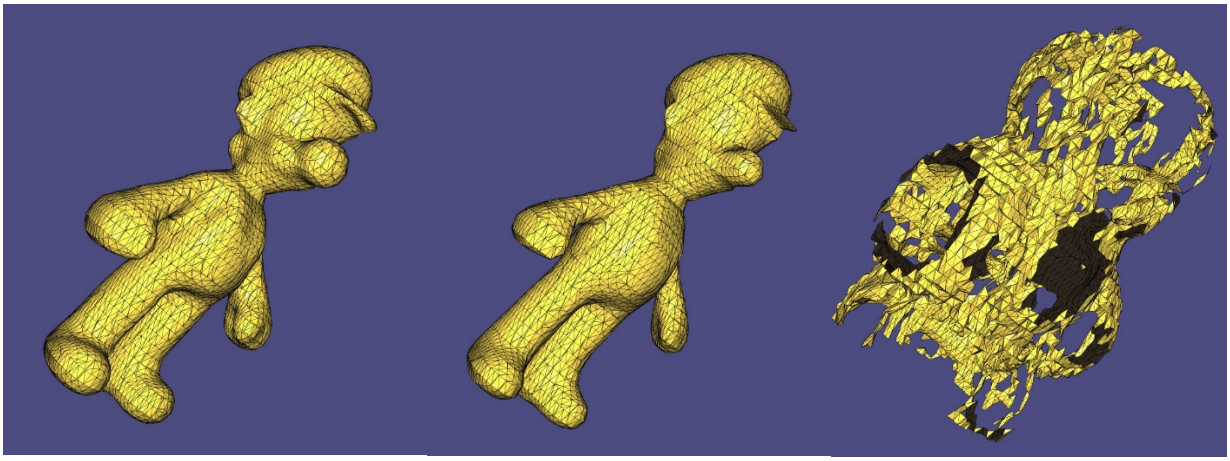


Section 3: Extracting the Surface.

First, we will show several sets of experiments with different parameter settings on luigi.off. Then we will put the reconstruction result of all the other models in the next section.

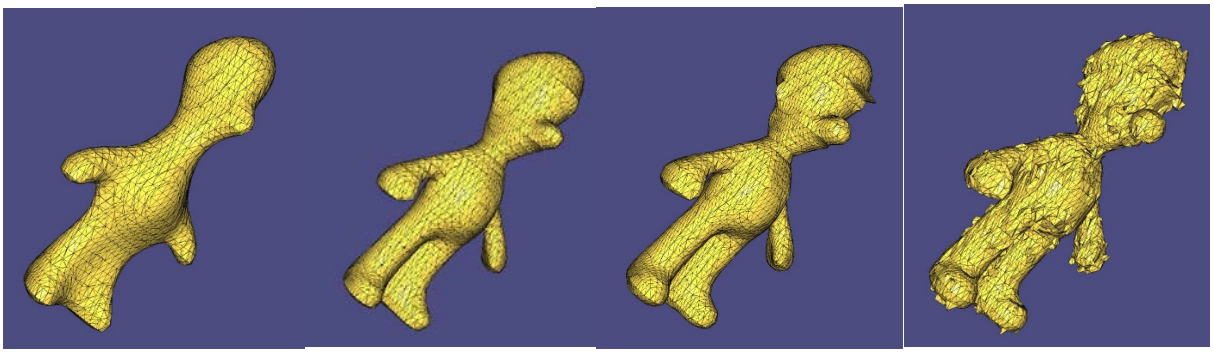
Experiment with different polyDegrees:

Figures below are the results with resolution 40x40, and polyDegree = 0, 1, 2 respectively. (wendlandRadius are the same) We can see that higher ordered polynomial function could give us weird results.



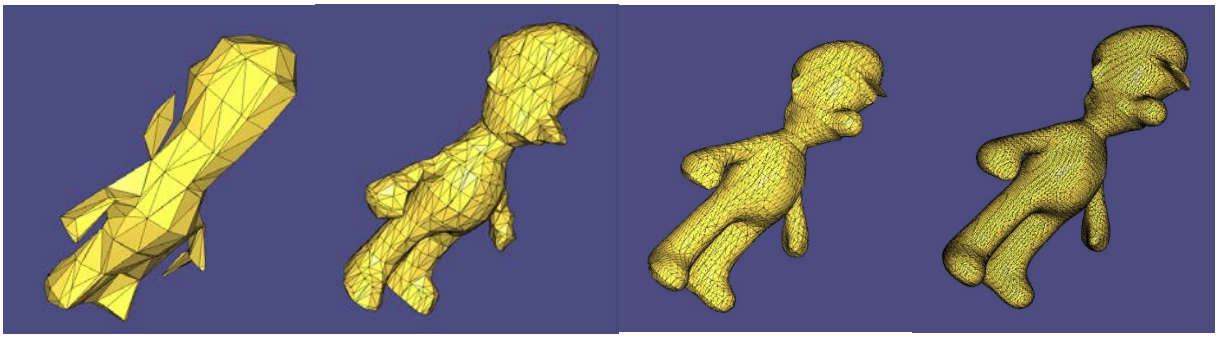
Experiment with different wendlandRadius:

Figures below are the results with resolution 40x40, polyDegree = 1. From left to right the wendlandRadiu is from large to small. We can see that with large wendlandRadiu, the local details will be "smoothed out". Meanwhile with a too small wendlandRadiu, there will be local noises.



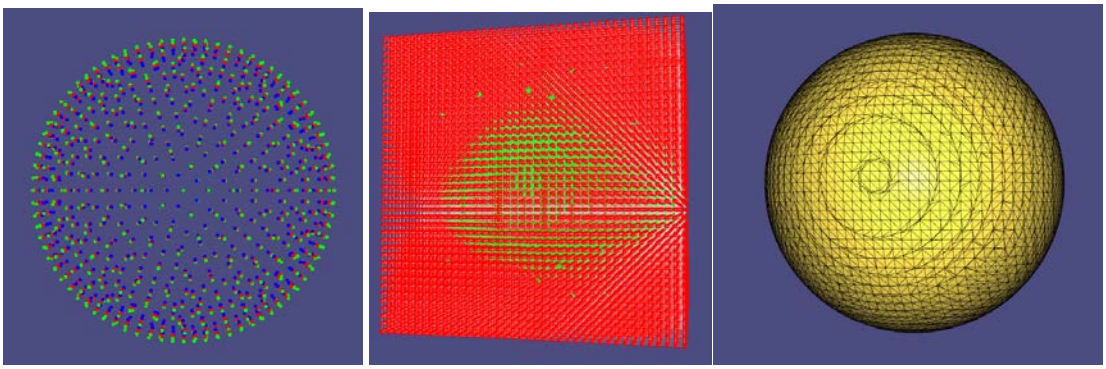
Experiment with different resolutions:

Figures below are the results with resolution = 10x10, 20x20, 40x40, 80x80, respectively.

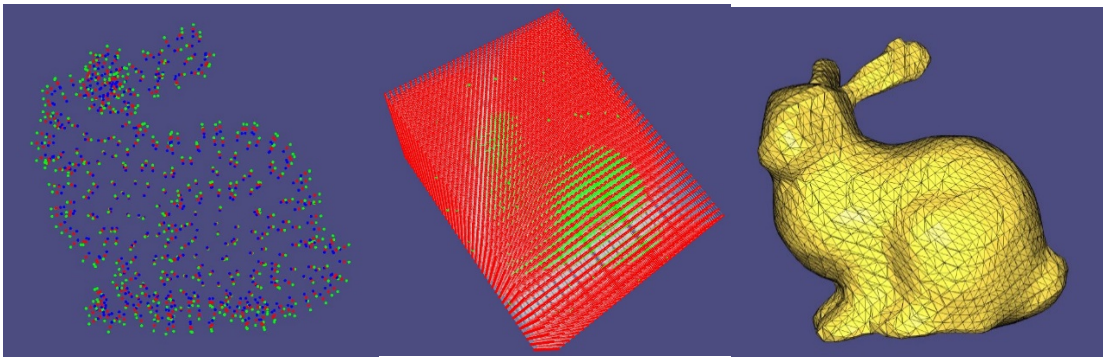


Reconstruction of Other Point Cloud Models

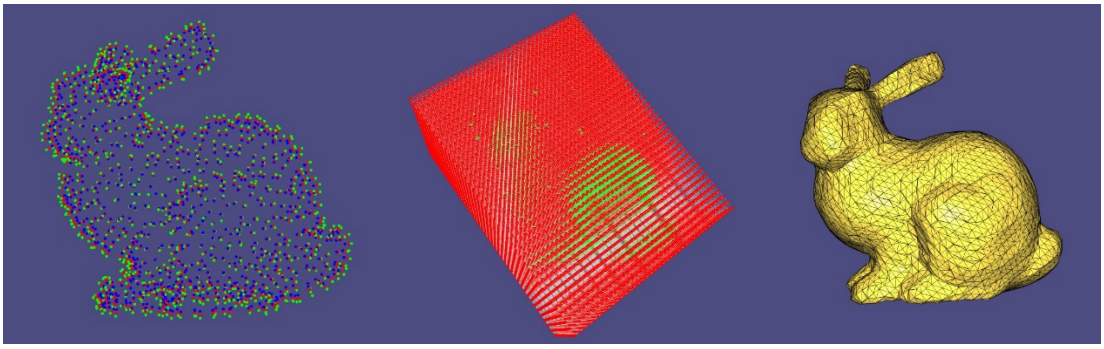
Sphere.off



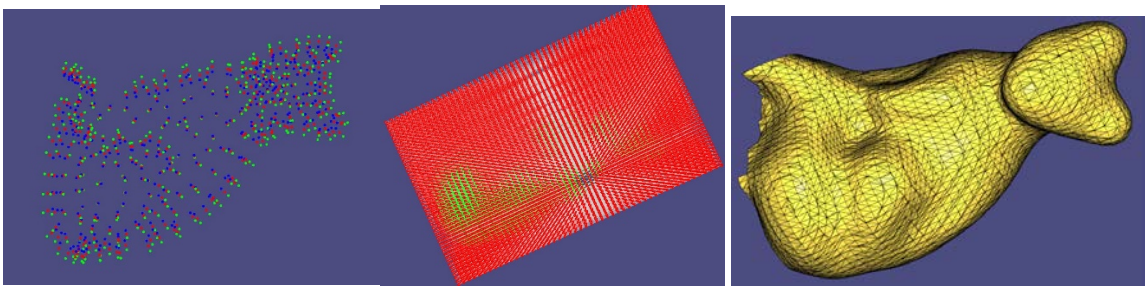
Bunny-500.off:



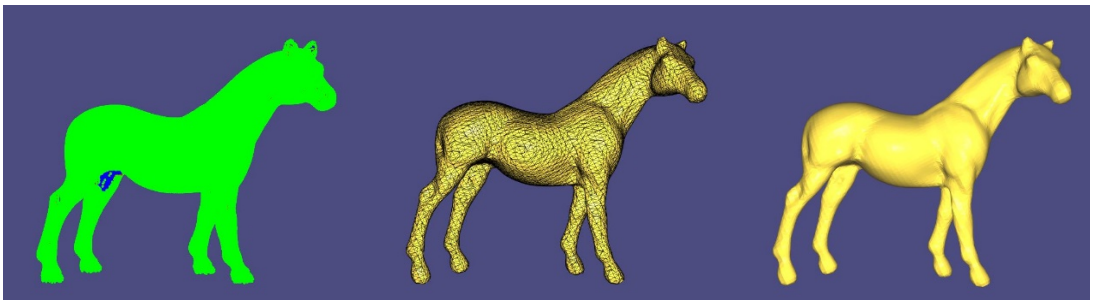
Bunny-1000.off



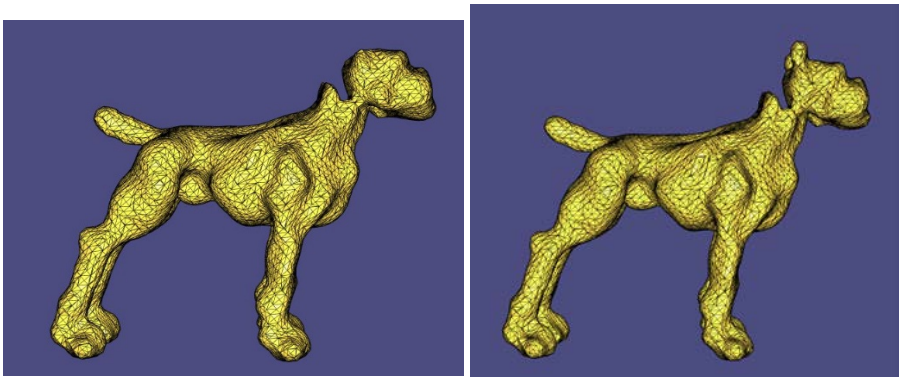
Cat.off



Horse.off



Hound.off:



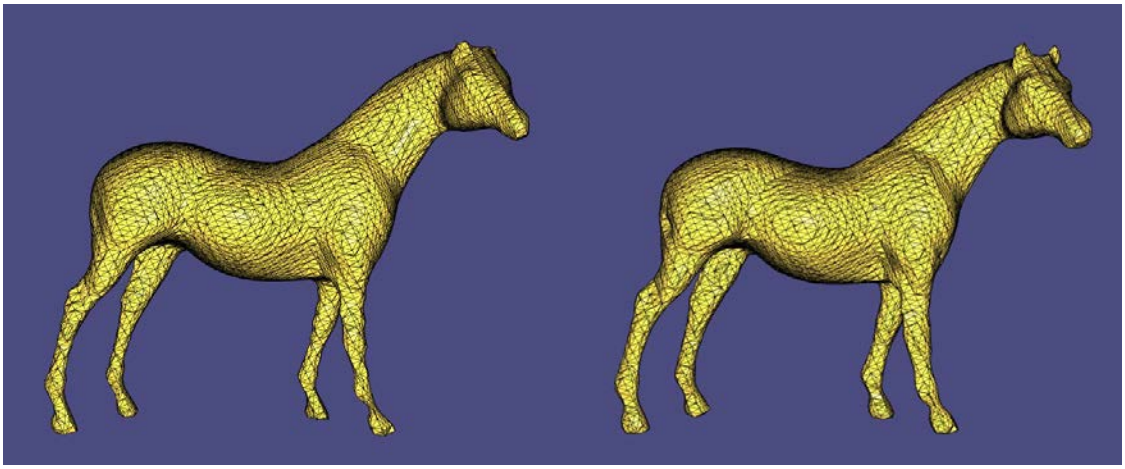
Optional Task 2: Better Normal Constrains

In this task, we basically follow the idea from paper *Interpolating and Approximating Implicit Surfaces from Polygon Soup*:

1. Instead of creating three constrain points from each original point in the point cloud, we use only one point as one constrain

2. For each grid point, we evaluate the signed distance from it to the tangent plane at each of the constrain point within wendlandRadiu. So that we replace d_i with $s_i(x) = (x - p_i).dot(n_i)$, where p_i is the position of the constrain point and n_i is the normal at the constrain point.

Here we do a brief comparison between this method and our original method. The left one is the reconstructed horse.off using our original method, and the right one is the method in Optional Task 2 (under the same wendlandRadiu, resolution = 60x60, polyDegree = 1):



We can see that on the small regions (e.g. The front legs of the horse), the method in Optional Task 2 performs much better than our original method. That's because in small spaces the created constrain points may not be that robust – i.e. A proper epsilon becomes tricky at those regions. This method can at least easy us from tuning the epsilons.

Optional Task 3: Screened Poisson Surface Reconstruction

For this task we downloaded the standalone implementation that author provides, and we tried the same horse model. The five figures below show the result with depth = 2, 5, 8, 10, 12 respectively.

Instead of fitting the scalar field directly like in MLS, the Poisson Surface Reconstruction applies Laplacian on the scalar field function and trying to fit the gradient field.

Generally, we can find out that the quality of the result gains with larger depth. Comparing to our MLS method, the generated surface looks “smoother”, i.e. less noisy and don’t have strange outliers. That’s probably because the Poisson reconstruction is a global solution (like RBF). So it can mitigate the influence of local noisy data points.