Assignment 2

1 Part 1: Surface Registration

1.1 Implementation

For this part of the assignment, RANSAC was implemented. The geomproc library was used to implement RANSAC. Some modifications to the library were made. The main modification was the creation of the 'best_matches' function inside alignment.py. That function builds off of the function 'best_match', by looping over all pairs of descriptors between both point clouds, and keeping track of the top 'num_matches' found for each descriptor in 'desc1'. The new function only increases the run time of 'best_match' by a factor of $log(n)$ where n is the number of top matches, indicated by the parameter 'num_matches'. The small run-time increase is given by the usage of the Sorted Containers python library. A sorted list of the current best 'num_matches' is efficiently maintained for each descriptor from 'desc1' using the library. For each time a new pair of descriptors is iterated over: If the distance between the descriptors is less than the maximum distance element in the current best 'num_matches' then, an element is removed from the list (if the list's length is equivalent to 'num_mathes') and an element is inserted to the list corresponding to the pair of elements and their distance. The sorted list is used as a black box, but under-the-hood works as a binary tree. The logarithmic time for insertion and removal of elements in the sorted list is indicated on the documentation website for Sorted Containers. Another modification to the geomproc library was the function 'get_point_cloud', which is used for getting the point cloud of a mesh (used exclusively in subsection 1.2.1). The last modification to the library was the 'add_noise' function in pcloud.py, for adding noise to the locations of points in a point cloud. I will include the entire modified geomproc library in my submission. The file I made for the RANSAC implementation is called 'assignmen_2_part_1.py'. It will be inserted into the main directory of the modified version of the geomproc library.

The RANSAC algorithm is implemented using geomproc and following the algorithm described in the slides of the course. Descriptors for points are computing using 'spin_images'. The top-n closest matching descriptors for each point are computed using the 'best_matches' function. When necessary, points from a point cloud are sampled using the 'sample' method in the geomproc library. Correspondences from the output of 'best_matches' are sampled using a 'sample_correspondences' function I made. A new sample is made for each main iteration of the RANSAC algorithm. The initial transformation for aligning the point clouds is derived using the sampled points from 'sample_correspondences'. The inliers of a transformation are determined using a 'threshold' for euclidean distance between proposed matches/correspondences of pairs of points. If the distance between two points in a proposed correspondence is less than the 'threshold', then they are inliers, and otherwise, are outliers. Subsequent transformations are made using the inliers of the previous transformation, and stop once the amount of inliers is no long decreasing, or if the amount of inliers is less than the amount of inliers in the current best transformation resulting from one iteration of the main RANSAC loop, or if the amount of inliers reaches 0. This entire process is repeated 'num_trials' times

in the main RANSAC loop. The transformation with the most inliers after the completion of all the trials is kept as the 'best transformation' and the associated models are saved as .obj files 'bunny_corr.obj' to show the best alignment and 'bunny_uncorr.obj' to show the point clouds prior to being aligned.

1.2 Results

Figure 1 shows the initial identical bunny meshes unaligned. The bunny mesh was chosen for running the RANSAC experiments because everyone loves cute bunnies. The blue bunny mesh is given by taking the red mesh and rotating it $\pi/3$ along the x axis, $\pi/2$ along the y axis, $\pi/4$ along the z axis and translating by $0.5, 0.2$ and 0.3 along the x, y, z axes respectively. The same transformation will be used for each set of parameters in all the subsubsections contained in this subsection. In each experimentation done in all the subsubsections of this subsection, each trial of the RANSAC algorithm started by sampling 10% of the total pairs of correspondences along with sampling from their top n best matches. The figures in this subsection showing the alignment of meshes using RANSAC include lines illustrating the correspondences found between the sampled points of the best alignment (AKA the alignment output by the RANSAC algorithm, along with the correspondences which gave the alignment).

In the following subsubsections, when simply stating the execution time of the RANSAC algorithm, the time it takes for the spin image descriptors to be computed and the run time of 'best_matches' is not included. The two previous mentioned metrics will be explicitly stated when it is relevant.

1.2.1 Aligning Identical Transformed Point Clouds

As an initial result and test to make sure the method was working, a bunny mesh was aligned to a transformed version of itself. An inlier threshold of 0.1 was used, along with sampling from the top 3 corresponding matches for each point. Finding the top 3 correspondences for each point took 33.78 seconds. Each trial of the RANSAC algorithm sampled 10% of the pairs of correspondences, as an initial sample. Figure 1 shows the initial identical meshes unaligned. Figure 2 shows the meshes aligned using RANSAC. As can be seen, the meshes are essentially perfectly overlapping. Another test where only the top $n = 1$ corresponding match is considered shows a perfect alighment, in figure 3. Finding the top 1 correspondence for each point took 33.62 seconds. Calculating the spin images for both experiments took 27.65 seconds. A spin image was calculated for each point in the entire point cloud, using all the points in the point cloud for casting votes.

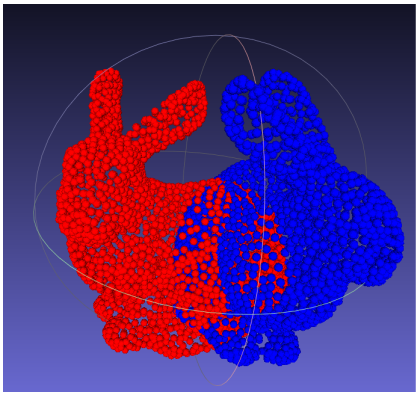

Figure 1: The original bunny (red), along with a transformed (rotated and translated) version of the bunny (blue).

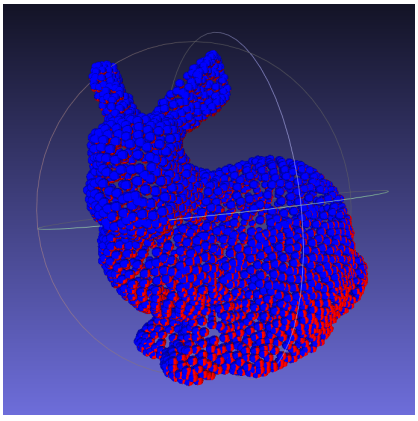

Figure 2: The original bunny (red) after being rotated and translated using RANSAC, to match the other bunny (blue). The top $n = 3$ matches for each point were used. 111 inliers were found in the final transformation. The algorithm took 0.51 seconds to execute.

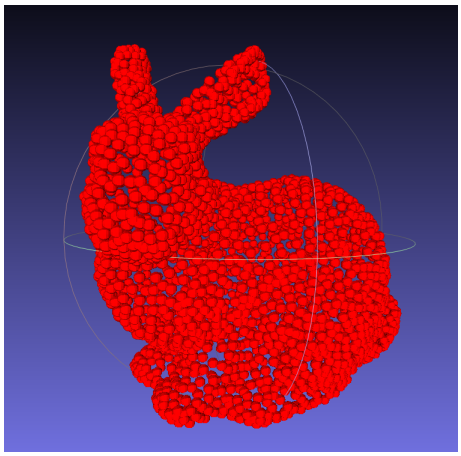

Figure 3: The original bunny (red) after being rotated and translated using RANSAC, to match the other bunny (blue). The top $n = 1$ match for each point was used. Unsurprisingly, the alignment is perfect, you can't seen any of the blue points (they are directly overlapped by the red points). 250 inliers were found in the final transformation. The algorithm took 0.52 seconds to execute.

1.2.2 Aligning Two Differently Sub-sampled and Transformed Point Clouds from the Same Point Cloud

To increase the difficulty of the problem, the bunny mesh was sampled twice, to give two similar meshes. The mesh has 2503 points vertices and 4968 faces in total. For both meshes, 1000 points were sampled from the surface of the mesh, for generating 1000 spin images along with 10000 samples for casting votes for creating each of the 1000 spin image descriptors. The 1000 sample points for each unaligned mesh can be seen in figure 4. Calculating the spin images took 39.33 seconds. For each iteration of the main loop of the RANSAC algorithm, 100 pairs of descriptors were sampled along with one of their top 3 corresponding matches. Finding the top 3 correspondences took 5.65 seconds. A threshold of 0.1 was used. The best transformation for one run of the RANSAC algorithm using these settings gave 29 inliers. The RANSAC algorithm took 0.23 seconds to complete. As can be seen in figure 5, the alignment was very accurate.

1.2.3 Aligning Two Differently Sub-sampled and Transformed Point Clouds from the Same Point Cloud, With Some Extra Noise

To make the problem more difficult, some noise was also added to the sampled point cloud. The noise was generated by translating each point along the x, y, z axes using a $3-$variable multivariate normal distribution, with 0.0025 in each diagonal entry of the covariance matrix, and 0 in every other entry. The mean of of the 3 variables was set to 0. The generated noise was added to each vector representation of each vertex. The same parameters for RANSAC

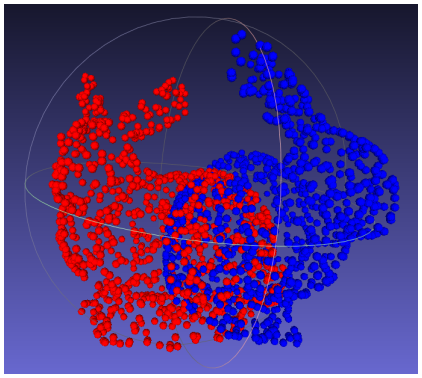

Figure 4: The sampled un-transformed bunny (red), along with a sampled and transformed (rotated and translated) version of the bunny (blue). Only the 1000 points that had spin images created are shown.

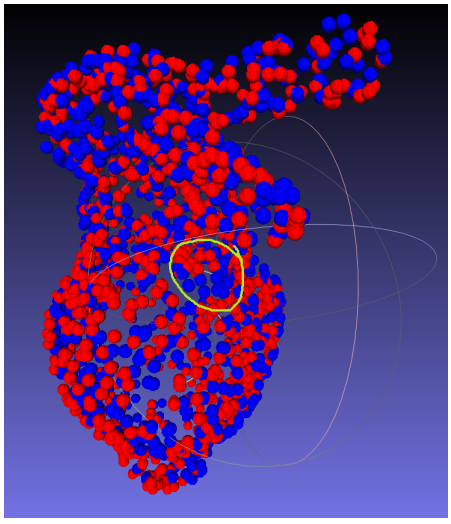

Figure 5: The sampled un-transformed bunny (red) after being rotated and translated using RANSAC, to match the sampled and transformed version of the bunny (blue). Upon inspecting the alignment, only a single spurious correspondence was found. Only the 1000 points that had spin images created are shown.

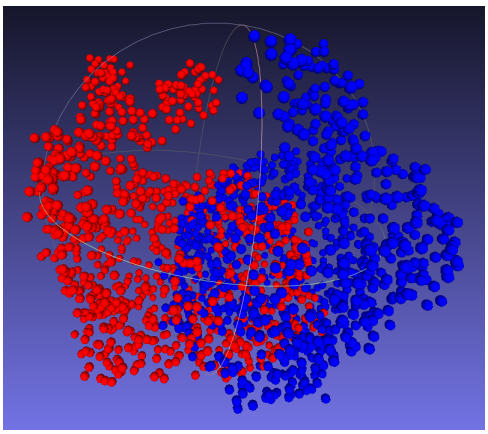

Figure 6: The sampled points with added noise of the un-transformed bunny (red), along with another sample of points from a transformed (rotated and translated) version of the bunny, with added noise after being sampled. Only the 1000 points that had spin images created are shown.

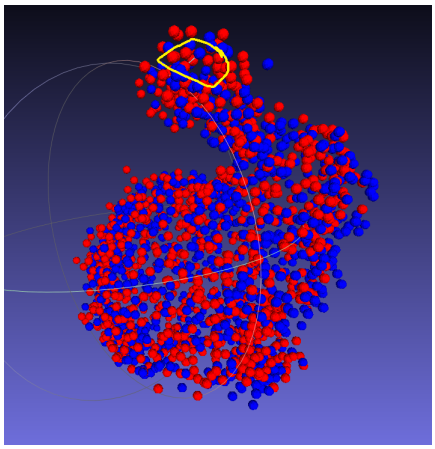

Figure 7: An alignment using RANSAC of two sets of noisy sampled points from the same bunny mesh. Upon inspecting the alignment, only a single spurious correspondence (outline in yellow) was found. Only the 1000 points that had spin images created are shown. The blue bunny appears to be leaning forward more than the red one.

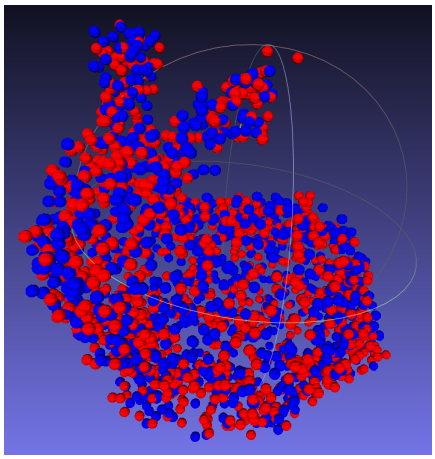

Figure 8: An alignment using RANSAC of two sets of noisy sampled points from the same bunny mesh. Upon inspecting the alignment, 3 spurious correspondences were found, but the alignment appears very accurate. Only the 1000 points that had spin images created are shown.

experimentation in 1.2.2 were used. The execution time of generating the spin images was 39.32 seconds. At first, an experiment sampling the 3 top n matches for each point was done, giving only 12 inliers, and a RANSAC run time of 0.23 seconds. Only 1 spurious correspondence was found in the final alignment, see figure 7. Another experiment sampling only the top $n = 1$ match for each point gave a slightly better alignment with 18 inliers, and a run time of 0.21 seconds. 3 spurious matches were found (not shown in the figure), see figure 8. Despite having more spurious matches, the alignment appeared to be better. The blue bunny was no longer tilted forward too much. This was likely due to the negative effects of the spurious matches cancelling each other out, and also being suppressed by a larger amount of correct matches.

1.2.4 Aligning Two Differently Sub-sampled and Transformed Point Clouds from the Same Point Cloud, With Some Extra Noise and Estimated Normals

Next, on top of some noise being added to the sampled point cloud, normals were estimated for each sampled point, using geomproc's 'estimate_normals' function. First the point clouds were sampled, using the same parameters as in 1.2.3, then 'estimate_normals' was called on each sampled point cloud, with the 4 nearest neighbors of each point used for estimating normals, along with no specified index of a point with a consistent normal. The first experiment with $n = 3$ top matches used ended up having only 2 inliers and taking 0.24 seconds to run, see figure 9. Another experiment after that gave a result with 4 inliers taking 0.23 seconds to run, see figure 10. The estimated normals appeared to not be consistent enough to generate strong correspondences. Another experiment with an $n = 1$ top match used ended

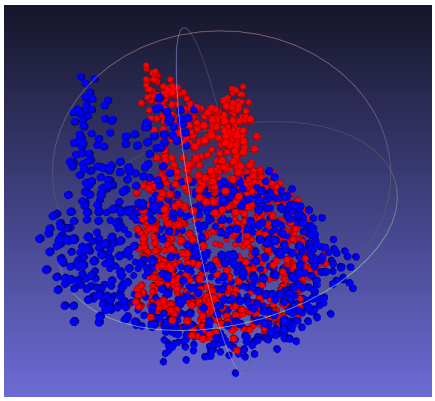

Figure 9: An alignment using RANSAC of two sets of noisy sampled points with estimated normals from the same bunny mesh. Only the 1000 points that had spin images created are shown. The alignment appears to be very inaccurate. The top $n = 3$ correspondences for each point were considered for sampling.

up having 4 inliers and taking 0.24 seconds to run, with an alignment that was just as poor as the previous mentioned experiment, see figure 11. No spurious correspondences were seen in these experiments.

1.3 Conclusion

The RANSAC algorithm was effective at aligning meshes, even under difficult constraints. Even though the artificially introduced noise gave a correspondence with many outliers, the meshes were still able to be aligned. When estimated normals were used, the meshes weren't able to align as well. Accurate normals appear to be very important, a normal that is slightly angled the wrong way could have a drastically different spin image descriptor due to points falling in the wrong bins.

Through random sampling, the RANSAC algorithm could quickly find a suitable amount of inliers to perform an accurate transformation. Experimenting with the top n matches for each point did not have much of an effect on the resulting alignment. It is expected that for meshes with many points (such as the bunny) increasing n above 1 is usually not necessary. The sampling process of RANSAC will eventually find a good set of matching descriptors given an ample amount correct matches exist. Increasing n would be more advantageous in a scenario where the top descriptor match for each point is not reliable, or is very similar in quality to the second best, third best, etc... matches, making a more exhaustive search more beneficial.

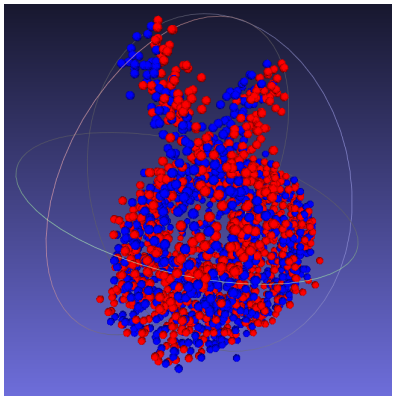

Figure 10: An alignment using RANSAC of two sets of noisy sampled points with estimated normals from the same bunny mesh. Only the 1000 points that had spin images created are shown. The alignment appears to be fairly accurate. The top $n = 3$ correspondences for each point were considered for sampling.

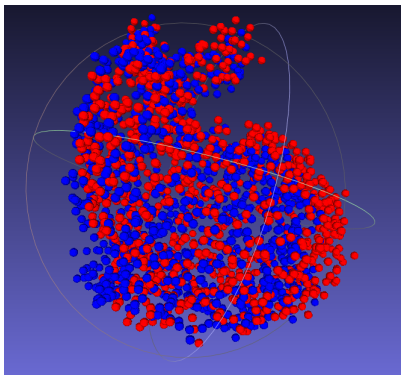

Figure 11: An alignment using RANSAC of two sets of noisy sampled points from the same bunny mesh. The alignment appears to be somewhat accurate. Only the 1000 points that had spin images created are shown. The top $n = 1$ correspondence for each point was considered for sampling

Assignment 2

Figure 12: A stock photo of a beautiful bunny. Bunnies are active animals which rarely stop moving for very long, making them difficult to get 3D scans of.

2 Part 2: Surface Reconstruction

2.1 Implementation

The test_rbf_vectorized.py code from the geomproc demos was used to test the RBF reconstruction method. The file I made using that code is called 'assignmen_2_part_2.py'. It will be inserted into the main directory of the modified version of the geomproc library. Marching cubes was used with a 64 by 64 grid, and the rbf reconstruction had an epsilon value of 0.01. The sphere mesh and the bunny mesh were both experimented on. The sphere mesh was chosen due to its simplicity, and as a comparison, the bunny mesh was chosen due to its complexity (and the fact that everyone loves adorable little bunnies, see figure 12).

2.2 Results

2.2.1 Sphere

The best results for reconstructing the sphere were with the triharmonic kernel. To me, this makes some intuitive sense, the triharmonic kernel gives higher weight to points further away. A sphere is a uniform shape, and so the weights on points can be evenly valued. Given an area on the sampled sphere is missing and must be reconstructed, an emphasis should be made on the overall formation of the shape, i.e. on the points far away from that point. With the biharmonic kernel, because areas of the circle were more sparsely sampled than others, this lead to irregular local changes in the function given by the rbf method when filling out missing areas of the circle mesh by relying to heavily on irregular local point clouds. The Wendland kernel did not perform quite as well as the triharmonic kernel, but still had a smoother surface compared to the biharmonic kernel, possibly given by the smoothness of the Wendland kernel when plotted on a graph.

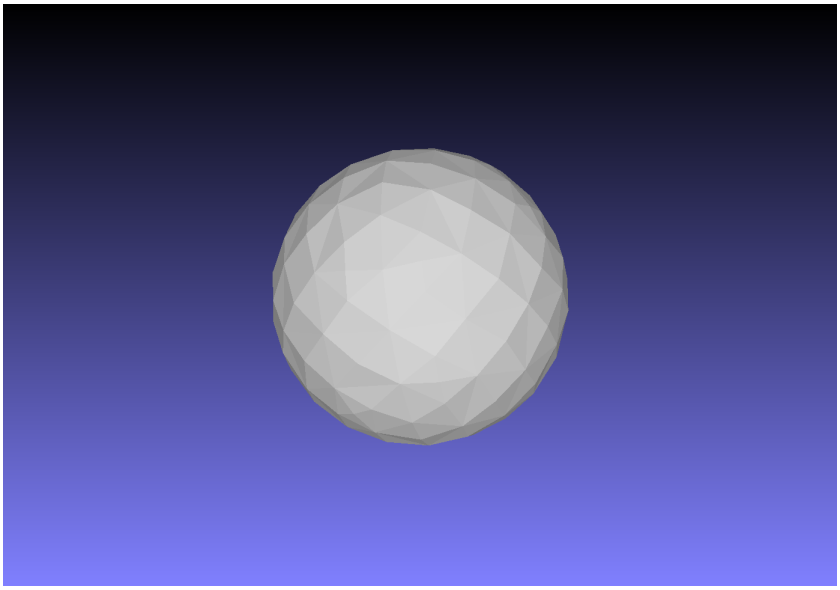

Figure 13: The original sphere mesh.

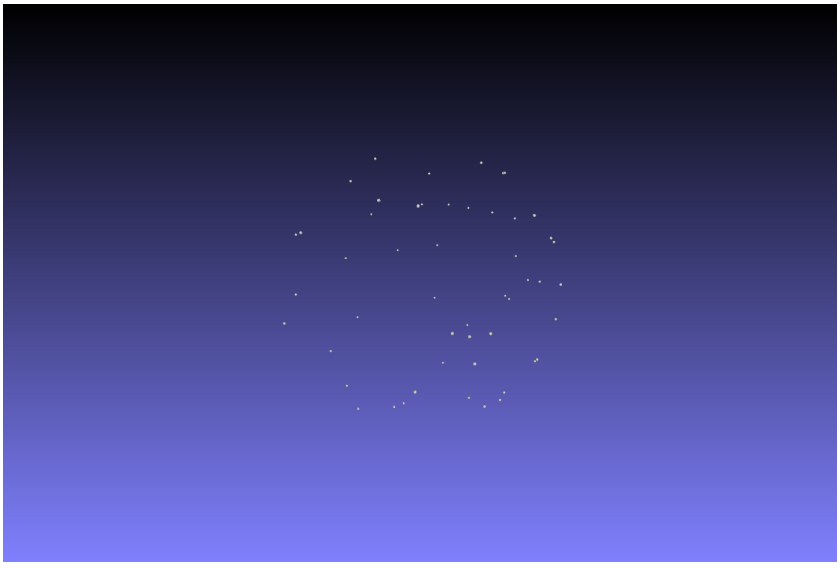

Figure 14: A point cloud sample of the sphere mesh. Points on the surface of the mesh were sampled, equal to 1/4 of the original meshes number of vertices.

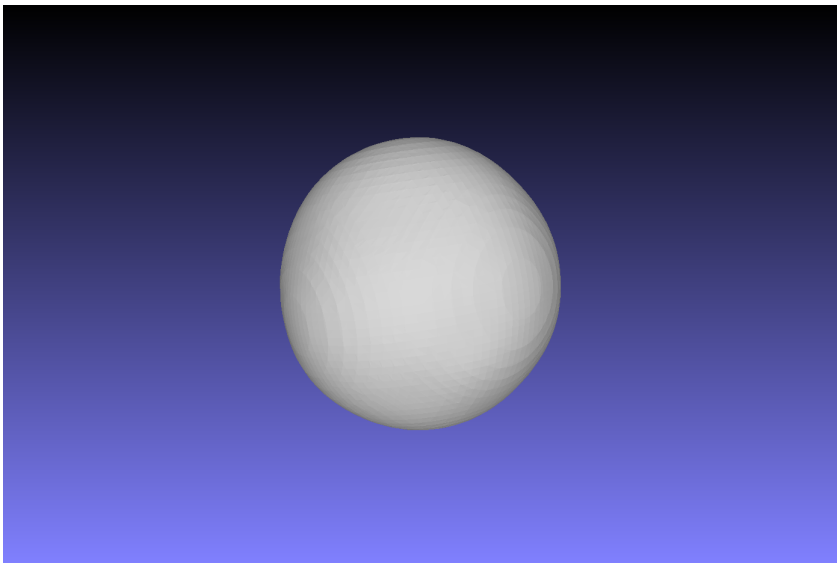

Figure 15: The sphere mesh reconstructed using the Wendland kernel with $h = 1$. The reconstruction using marching cubes took 136.90 seconds

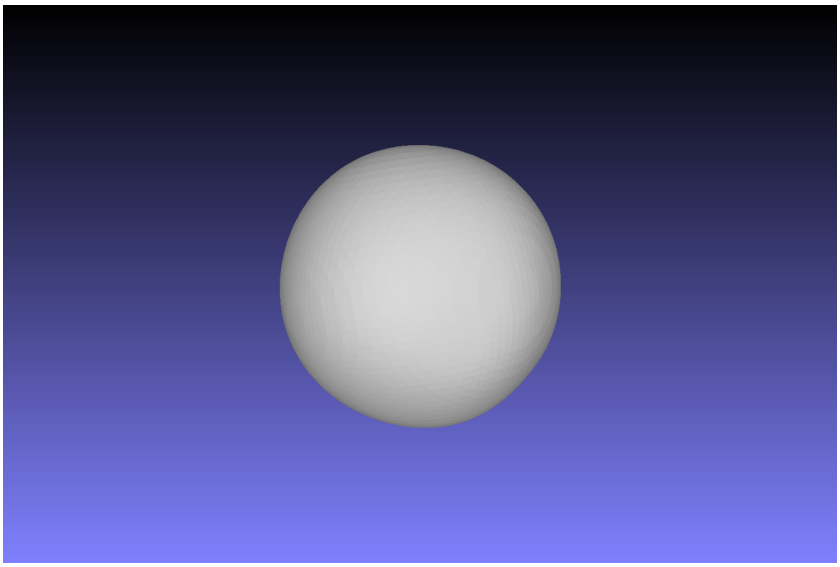

Figure 16: The sphere mesh reconstructed using the Wendland kernel with $h = 0.001$. The reconstruction using marching cubes took 136.34 seconds

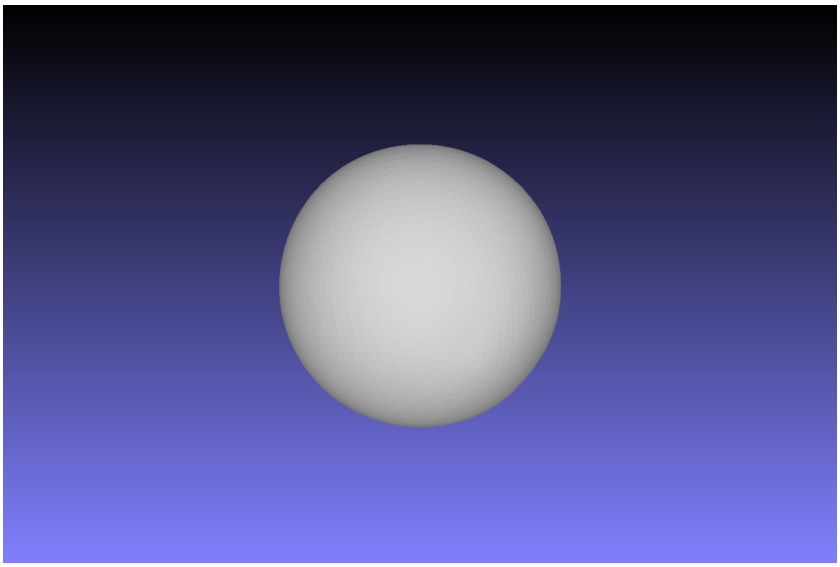

Figure 17: The sphere mesh reconstructed using the triharmonic kernel. The reconstruction using marching cubes took 72.64 seconds

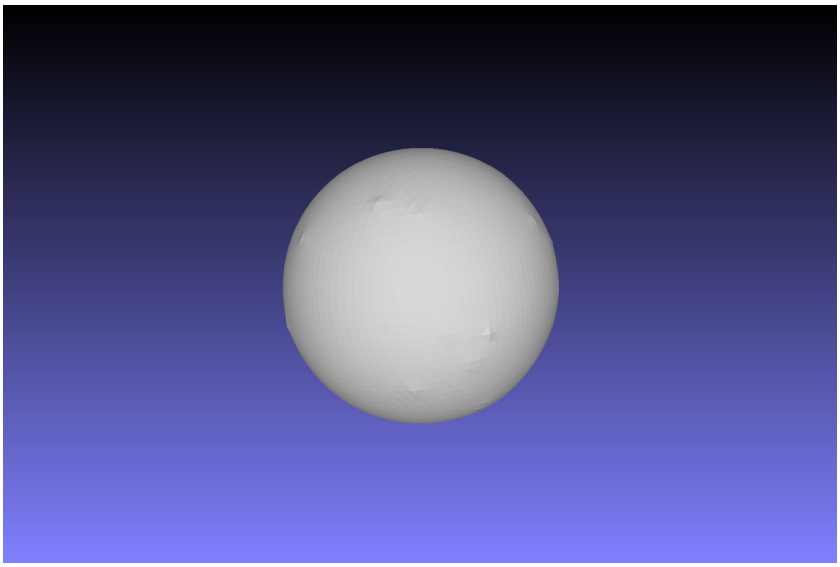

Figure 18: The sphere mesh reconstructed using the biharmonic kernel. The reconstruction using marching cubes took 64.95 seconds.

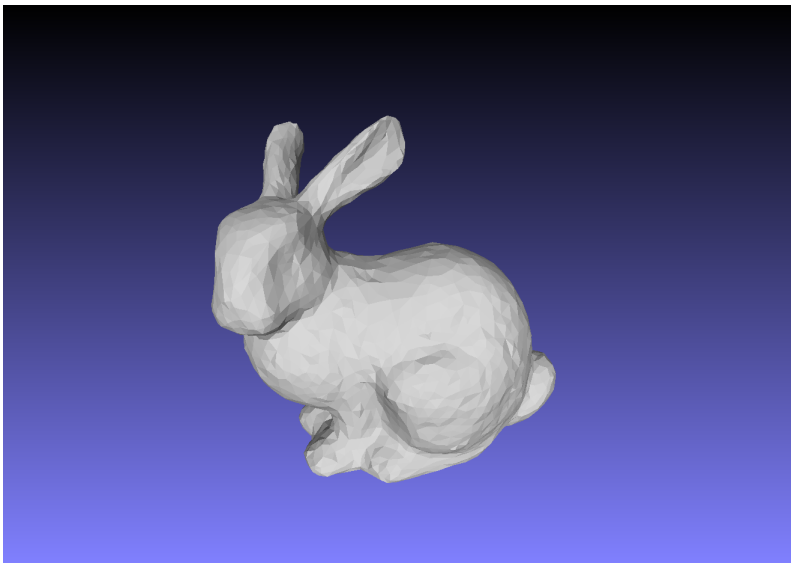

Figure 19: The original bunny mesh.

2.2.2 Bunny

Similar results seen with the sphere mesh were given with the bunny mesh. The bunny mesh was best reconstructed using the triharmonic kernel and Wendland kernel with $h = 0.001$. The biharmonic kernel reconstructed the bunny nicely, but had too much smoothing and loss of detail in the reconstruction. The Wendland kernel with $h = 1$ had an interesting result, by having a function which became negative again outside of the mesh.

2.3 Conclusion

The rbf reconstruction effectively reconstructed shapes from sparse point cloud samples. The parameters used in the reconstruction were sensitive to change, and the kernel chosen was the most significant influence when acquiring accurate results.

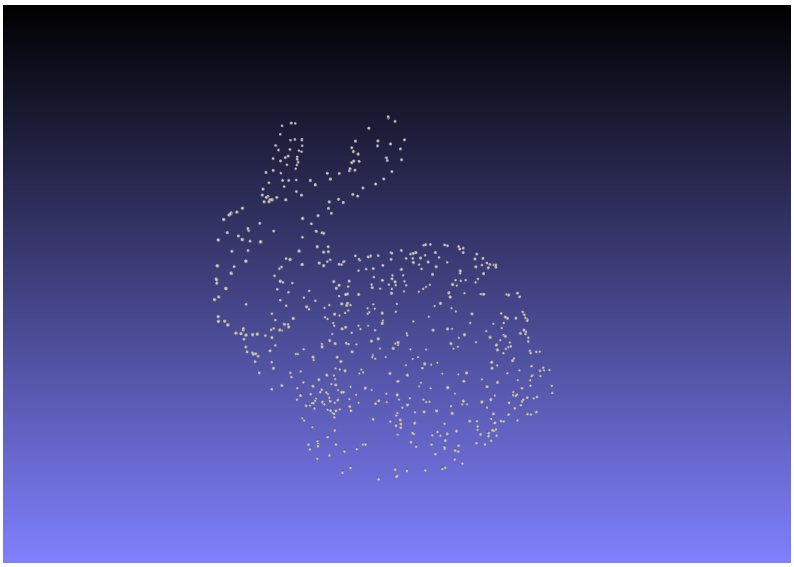

Figure 20: A point cloud sample of the bunny mesh. Points on the surface of the mesh were sampled, equal to 1/4 of the original meshes number of vertices.

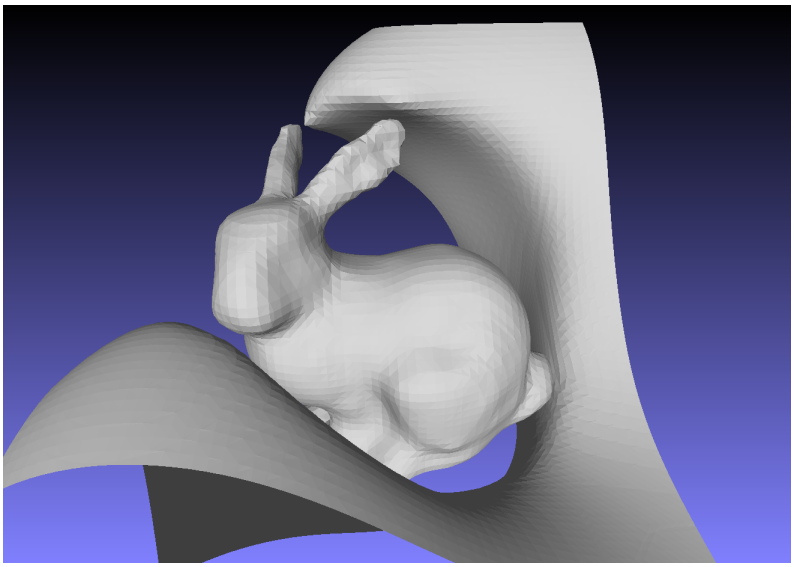

Figure 21: The bunny mesh reconstructed using the Wendland kernel with $h = 1$. The reconstruction using marching cubes took 303.94.

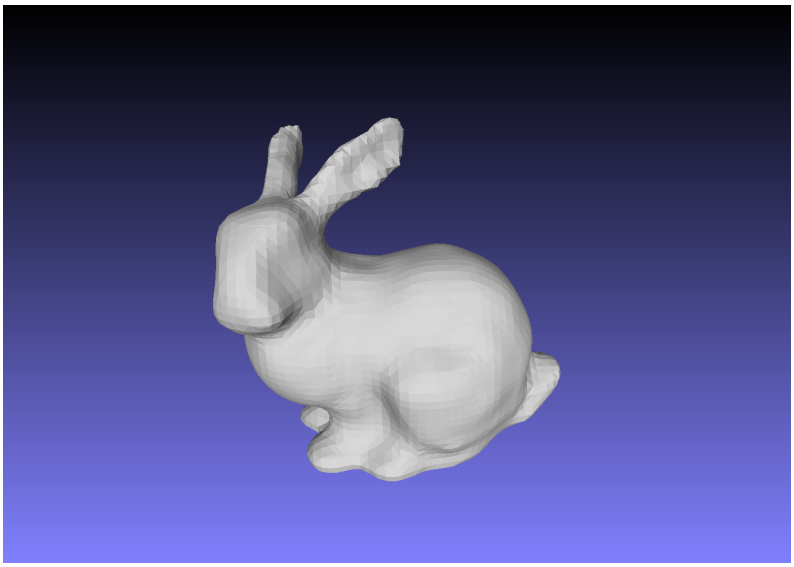

Figure 22: The bunny mesh reconstructed using the Wendland kernel with $h = 0.001$. The reconstruction using marching cubes took 294.87. seconds.

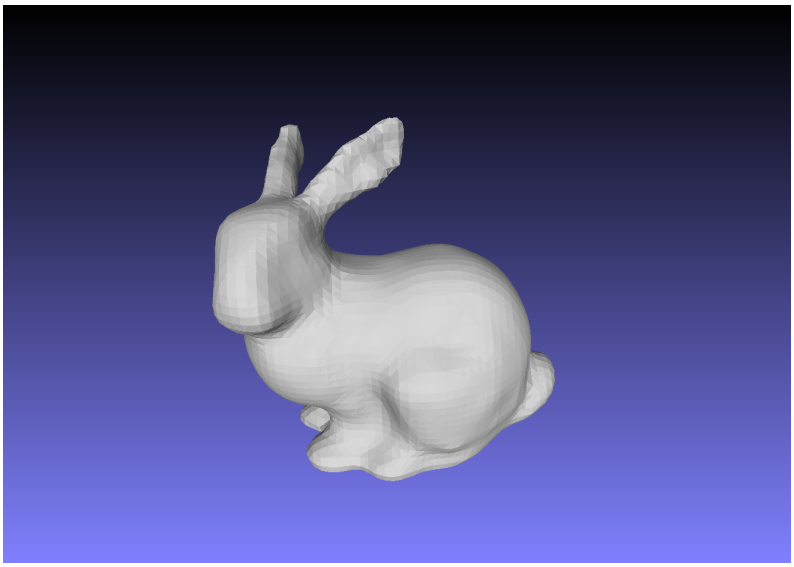

Figure 23: The bunny mesh reconstructed using the triharmonic kernel. The reconstruction using marching cubes took 169.99 seconds.

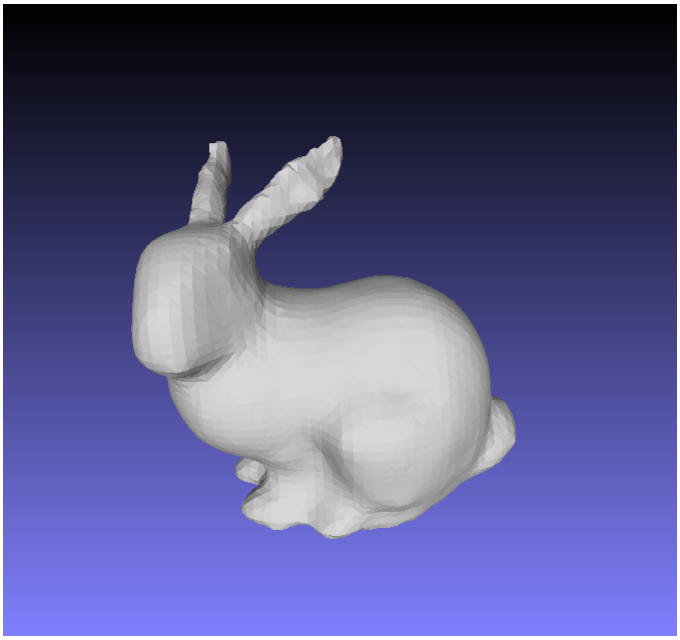

Figure 24: The bunny mesh reconstructed using the biharmonic kernel. The reconstruction using marching cubes took 108.14 seconds.