

Implementation of "Reconstruction and Representation of 3D Objects with Radial Basis Functions"

AZHARUDDIN

ACM Reference Format:

Azharuddin. 2022. Implementation of "Reconstruction and Representation of 3D Objects with Radial Basis Functions". 1, 1 (December 2022), 8 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Surface reconstruction from point cloud is a standard and well known problem in computational geometry. Input data is in the form of a set of points which are scanned by 3D range scanners. In this paper, polyharmonic radial basis functions (RBFs) are used to reconstruct the smooth surfaces from the point cloud data. RBFs are interpolation functions which interpolates the values which are the points on the surface. First, we fit the RBF to a subset of points (carefully chosen), because in case of point cloud data, we have large number of points, so keeping in mind the time taken and the memory capacity of the available machine, we have to select an appropriate number of points. Then we use the centre reduction method in which we calculate the absolute error at every point and if the error is more than the threshold we set, then we add more points to our existing point set and fit again the RBF to them. In the implementation, I have used the point cloud of bunny data which has nearly 40k points, point cloud of Arm which has nearly 10k points and finally point cloud of screwdriver which has nearly 12k points. We reconstruct the surface in each of the cases by center reduction algorithm which will be discussed in the "Approach" section. Let us have a look at the problem statement in mathematical form.

2 PROBLEM STATEMENT:

Given n distinct points $\{(x_i, y_i, z_i)\}_{i=1}^n$ on a surface M in \mathbb{R}^3 , find a surface M' that is a reasonable approximation to M . We model the surface implicitly with a function $g(x, y, z)$. We say that g implicitly defines M if the surface M consists of all the points (x, y, z) that satisfy the equation $g(x, y, z) = 0$. We will do this in 3 steps:

- Constructing a signed distance function
- Fitting an RBF to the resulting signed distance function of given points
- Extracting the surface as the zero level set of the calculated RBF.

3 APPROACH

We have a scattered data interpolation problem. The trivial solution always exists. In order to avoid it, we add some off surface points and give them some non zero value. We chose the signed distance function as our function g discussed

Author's address: Azharuddin.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.

Manuscript submitted to ACM

Manuscript submitted to ACM

above. Now we want to approximate this SDF g by the function $s(x)$. We are given a set of distinct nodes $X = \{x_i\}_{i=1}^n$ and a set of function values $\{f_i\}_{i=1}^n$ find an interpolant s such that $s(x_i) = f_i, i = 1, 2, \dots, n$. The interpolant is chosen from $BL^{(2)}(\mathbb{R}^3)$, the Beppo Levi space of distribution on \mathbb{R}^3 . In our implementation, we will use C++ and its libigl library. We have taken the off surface points along the surface normal. All our input data is in the form of the mesh from which we obtain vertices and find the per vertex normal using the inbuilt function in the library. Then we set off surface points to be along the normal and away from the surface by some distance. That distance is carefully chosen and in our implementation we have taken that distance to be the 1% of the diagonal of the bounding box. The interpolant is of the form

$$s(x) = p(x) + \sum_{i=1}^n \lambda_i (|x - x_i|) \quad (1)$$

where p is a linear polynomial and coefficients λ_i are real and the $|\cdot|$ is the well known Euclidean norm on \mathbb{R}^3 .

In the case of bunny, I have taken the values of the function to be 0.05. Then we have to solve a system of linear equations which will give us the coefficients λ_i and coefficient of linear polynomial. We can calculate $s(x)$ from it. Since we have selected an appropriate subset of points, we have to find the error $\epsilon_i = f_i - s(x_i)$ at each and every point, and if maximum of absolute value of the error is smaller than "fitting accuracy" then we stop otherwise we append new centers or points to the existing subset. This is called RBF center reduction. By this method we can approximate a large number of points to the desired accuracy using only fewer centers. In the bunny example, we calculate error at each point 40k points. I have added 200 points in every step. After adding more points, we again have to fit an RBF to these new and large subset of points. There will come a stage when the error will be less than our fitting accuracy or we have added sufficient number of points according to the computational capacity of the machine we are using. Then we have to stop. Now we have to extract the zero level set of this surface. We use marching cubes to find the iso surface. In libigl, we have the marching cubes function, where we have to specify the grid resolution and pass the interpolant values $s(x)$ we have calculated at each point. In case of bunny, I have set the grid resolution to be 50. The following code snippet shows how we use marching cubes in libigl:

```
MatrixXd SV;
MatrixXi SF;
cout << "Marching cubes" << endl;
igl::marching_cubes(S, GV, res(0), res(1), res(2), 0, SV, SF);
```

The output is a mesh with vertices SV and faces SF. We can render this mesh in libigl using the following code snippet:

```
igl::opengl::glfw::Viewer viewer;
viewer.data().set_mesh(SV, SF);
viewer.launch();
```

Now, I will share some results and observations which shows how center reduction works. We will also see the reconstructed shapes from various point cloud data. First Lets us have a look at how we are going to form the system and solve it.

4 BACKGROUND

We have the equation:

$$s(x) = p(x) + \sum_{i=1}^n \lambda_i (|x - x_i|) \quad (2)$$

We will form a linear system and solve it to find the coefficients. The linear polynomial has the form

$$p(x) = c_1 + c_2x + c_3y + c_4z \quad (3)$$

We have to solve the linear system

$$\begin{pmatrix} A & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \quad (4)$$

where

$$A_{i,j} = |x_i - x_j|, i, j = 1, 2, \dots, n \quad (5)$$

$$P_{i,j} = p_j(x_i), i = 1, 2, \dots, n, \text{ and } j = 1, 2, 3, 4. \quad (6)$$

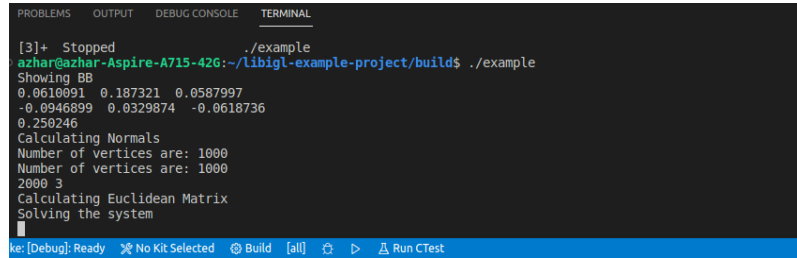
$p_1 = 1, p_2 = x, p_3 = y, \text{ and } p_4 = z$

Here we have chosen the basis function to be identity function according to the paper. We will find $s(x)$ as follows:

$$s(x) = c_1 + c_2x + c_3y + c_4z + \sum_{i=1}^n \lambda_i (|x - x_i|) \quad (7)$$

5 RESULTS

Our first example is bunny which has nearly 40k points. Initially we have selected 1000 points from the point clouds of bunny then we keep on adding 200 more centers until the maximum absolute error is smaller than the fitting accuracy. Number of vertices and maximum absolute error is shown in the figures.1, 2, 3. The final iteration gives us the result



```

[3]+ Stopped ./example
azhar@azhar-Aspire-A715-426:~/libigl-example-project/build$ ./example
Showing BB
0.0610091 0.187321 0.0587997
-0.0946899 0.0329874 -0.0618736
0.250246
Calculating Normals
Number of vertices are: 1000
Number of vertices are: 1000
2000 3
Calculating Euclidean Matrix
Solving the system

```

Fig. 1. Initial step Bunny

which can be seen in 4

Finally we display the reconstructed bunny in 5. Original can be seen in figure 6.

Our second example is Arm. Which has nearly 10k points. Same as bunny, first we selected 1000 points and fit the RBF to it then we keep on adding new centers until we get our desired accuracy or we reach an upper bound on the number of vertices. First and the last iterations observation for the Arm example is shown in figure 7 and 8. . And the arm is shown in figure 9. Original arm is shown in the figure 10.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
0.626986
0.633371
0.639787
0.646224
0.652672
Check
Maximum of error is : 0.265125
Number of vertices are: 1200
Number of vertices are: 1200
2400 3
Calculating Euclidean Matrix
Solving the system
ke: [Debug]: Ready  No Kit Selected  Build  [all]  Run CTest

```

Fig. 2. Bunny example

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
0.614345
0.620534
0.626748
0.632977
0.639211
Check
Maximum of error is : 0.270995
Number of vertices are: 1400
Number of vertices are: 1400
2800 3
Calculating Euclidean Matrix
Solving the system
ke: [Debug]: Ready  No Kit Selected  Build  [all]  Run CTest

```

Fig. 3. Bunny example

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
0.548806
0.553706
0.558623
0.56355
Check
Maximum of error is : 0.14842
Marching cubes
7577 3
15106 3
50 50 30
igl::opengl::glfw::Viewer usage:
[drag] Rotate scene
A,a Toggle animation (tight draw loop)
ke: [Debug]: Ready  No Kit Selected  Build  [all]  Run CTest

```

Fig. 4. Last iteration of bunny example

Our next example is of Screwdriver which has more than 10k points. First we select 1000 points and keep on adding new centers until we reach the threshold. Outputs are shown in figures 11 and 12. Original screwdriver mesh is shown in figure 13.

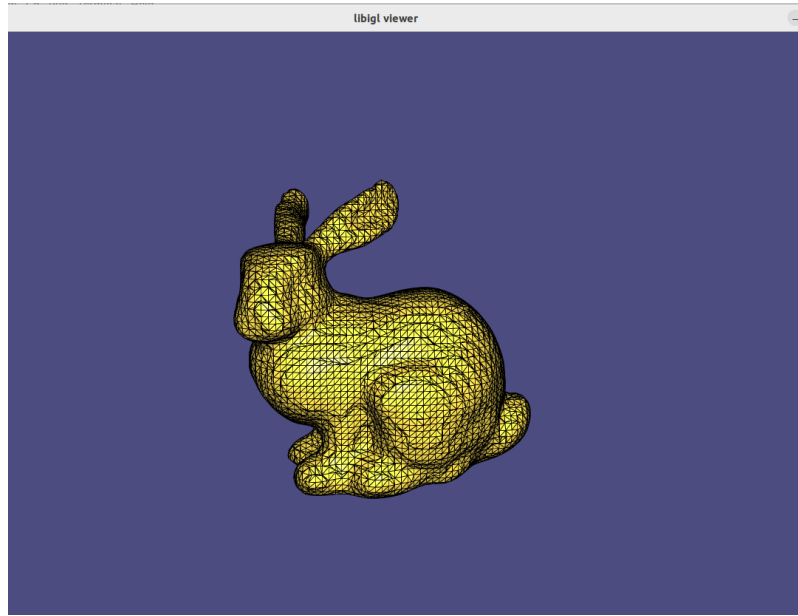


Fig. 5. Reconstructed Bunny from Point cloud data using center reduction

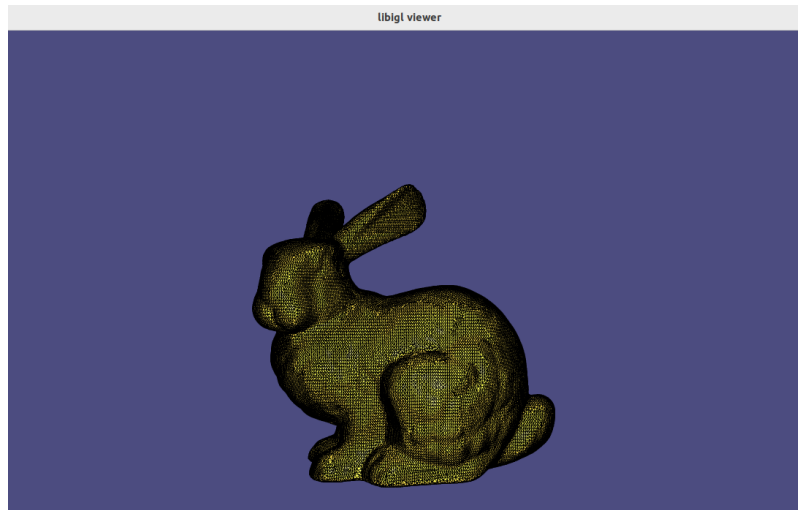


Fig. 6. Original Bunny

```
[100%] Built target example
azhar@azhar-Aspire-A715-42G:~/libigl-example-project/build$ ./example
Showing BB
0.910938 0.37088 0.171304
-0.910616 -0.357154 -0.248811
2.00614
Calculating Normals
Number of vertices are: 1000
Number of vertices are: 1000
2000 3
Calculating Euclidean Matrix
Solving the system
```

Fig. 7. Arm example

```
0.443262
0.44818
0.453047
0.457905
0.462792
Check
Maximum of error is : 0.199187
Number of vertices are: 1600
Number of vertices are: 1600
3200 3
Calculating Euclidean Matrix
Solving the system
```

Fig. 8. Arm example

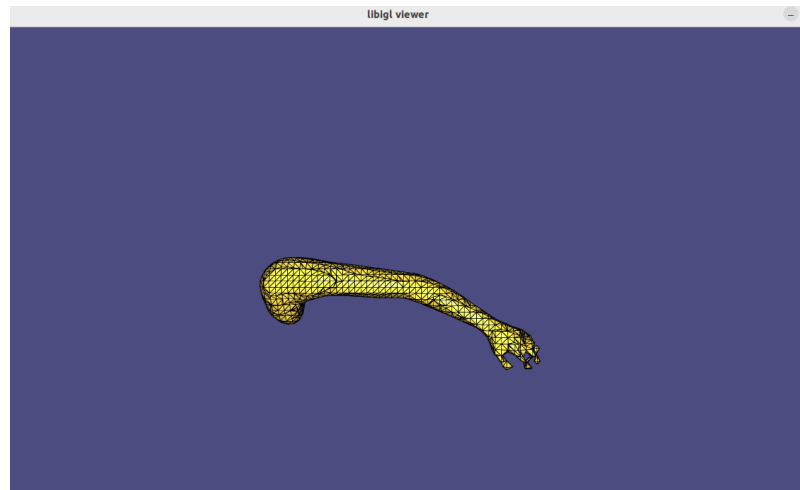


Fig. 9. Reconstructed Arm from Point cloud data using center reduction

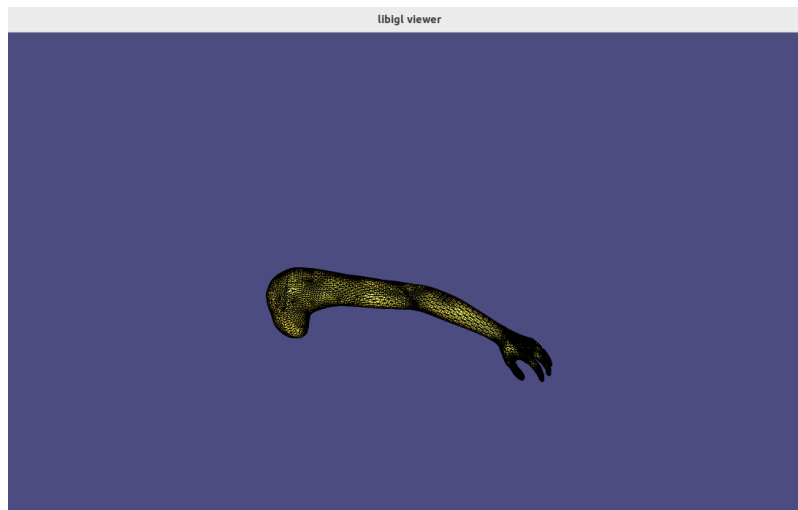


Fig. 10. Arm (original)

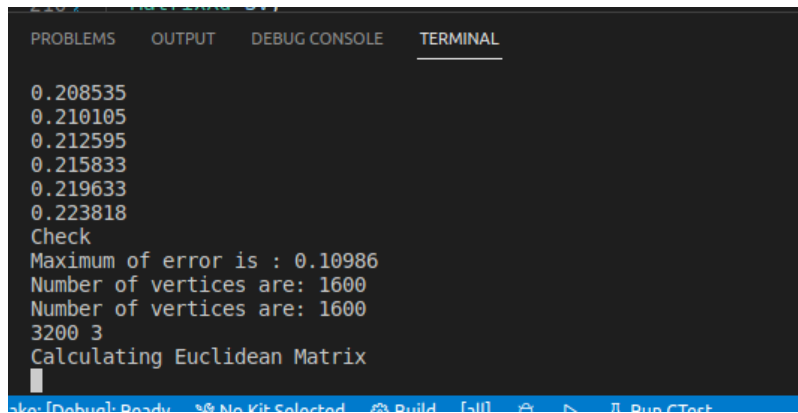


Fig. 11. Screwdriver example

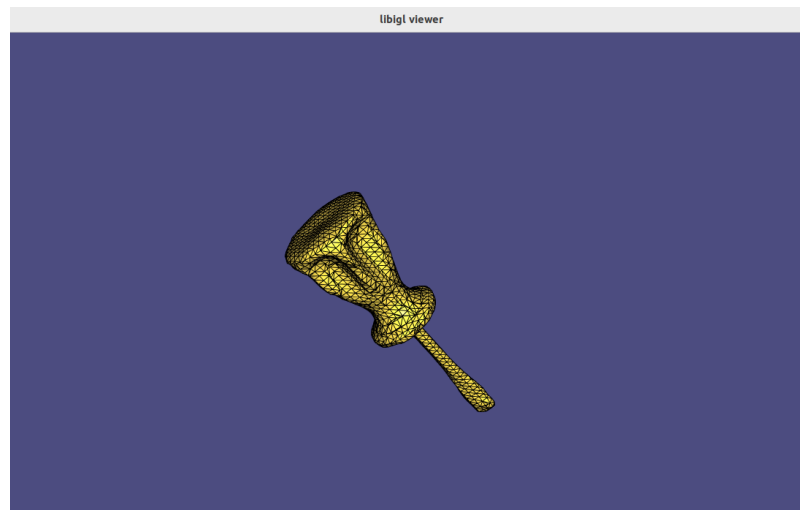


Fig. 12. Reconstructed Screwdriver from Point cloud data using center reduction

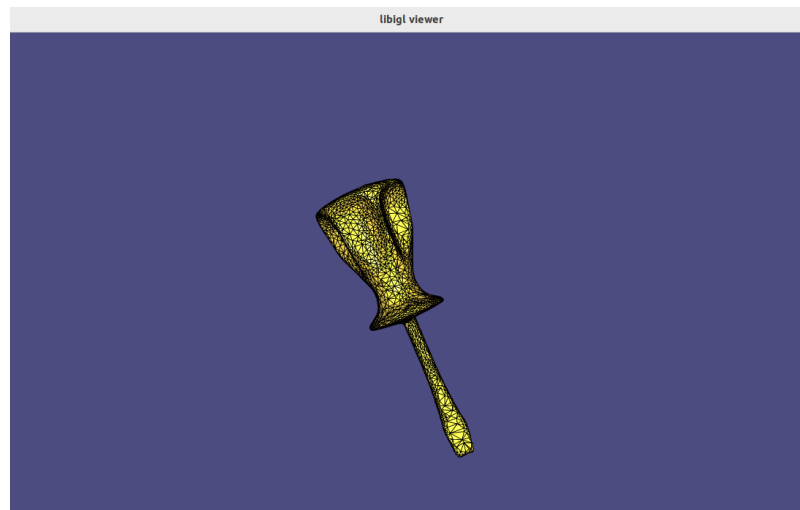


Fig. 13. Screwdriver (Original)