

# Normal Estimation for Point Clouds: A Comparison Study for a Voronoi Based Method

Tamal K. Dey    Gang Li    Jian Sun

The Ohio State University, Columbus OH, USA

---

## Abstract

*Many applications that process a point cloud data benefit from a reliable normal estimation step. Given a point cloud presumably sampled from an unknown surface, the problem is to estimate the normals of the surface at the data points. Two approaches, one based on numerical optimizations and another based on Voronoi diagrams are known for the problem. Variations of numerical approaches work well even when point clouds are contaminated with noise. Recently a variation of the Voronoi based method is proposed for noisy point clouds. The centrality of the normal estimation step in point cloud processing begs a thorough study of the two approaches so that one knows which approach is appropriate for what circumstances. This paper presents such results.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

---

## 1. Introduction

In many problems dealing with point cloud data, a normal estimation step precedes the main task. For example, in surface reconstruction, the quality of the approximation of the output surface depends on how well the estimated normals approximate the true normals of the sampled surface, see [AK04, ABCO\*01, BC00, DS05] for example. Similar correlation exists between estimated normals and point-based rendering of surfaces [AA03]. Being so central to point cloud processing, the normal estimation step deserves special attention on its own right. The problem is compounded by the fact that the input point cloud can be noisy. Therefore, we need a normal estimation method that remains robust against noise.

There are two dominant approaches for estimating normals from point clouds; one is *numerical* applying some optimization technique, the other is mostly *combinatorial* applying some Delaunay/Voronoi property. The numerical optimization based approach is known to work well under noise. However, it is not established how well the Voronoi based approach scales with noise. The original algorithm of Amenta and Bern [AB99] that uses the *poles* in the Voronoi diagrams works well with the data that are not noisy. It does not work in principle and in practice when data becomes noisy. Recently, starting with the work of [DG04],

Dey and his co-authors [DGS05, DS05] have suggested a Voronoi/Delaunay based method for estimating normals from noisy point cloud data. The centrality of the normal estimation step in point cloud processing begs a comparison of this technique with the competitive numerical based approaches. The purpose of this paper is to make this comparison study.

In the numerical approach, a widely used technique is to find a proper set of points in the local neighborhood for a point  $p$  and then compute a plane that best fits to these chosen points. The normal of the plane is taken as the estimated normal at  $p$ . This basic plane fitting method has been made more effective with sophisticated modifications. We consider two such variations, one by Pauly, Keiser, Kobbelt and Gross [PKKG03] and the other by Mitra, Nguyen and Guibas [MNG04] which have been shown to work well in practice.

## 2. Plane fitting methods

Hoppe et al. [HDD\*92] proposed an algorithm where the normal at each point is estimated as the normal to the fitting plane obtained by applying the total least square method to the  $k$  nearest neighbors of the point in the point cloud. Specifically for a point  $p$  and its  $k$  nearest neighbors  $\{p_i\}_{i=1}^k$ ,

they find the fitting plane  $n^T x = c$  for  $p$  by minimizing the error term  $e(n, c) = \sum_{i=1}^k (n^T p_i - c)^2$  under the constraint  $n^T n = 1$ . Notice that the normals computed by fitting planes are unoriented. They proposed an algorithm to orient the normals consistently.

Pauly et al. [PKKG03] and Mitra et al. [MNG04] improved the method in two different ways. Pauly et al. noticed that the fitting plane for a point  $p$  should respect the nearby points more than the distant points in the point cloud. Hence the neighboring points are assigned different weights based on their distances to  $p$ . The smaller the distance of a sample from  $p$ , the bigger the weight it has. In other words, they re-defined the error term as  $e(n, c) = \sum_{i=1}^k (n^T p_i - c)^2 \theta(\|p_i - p\|)$ , where  $\theta(\cdot)$  is a weighting function. In their implementation [PKKG03], the weighting function is taken as Gaussian, i.e.,  $\theta(\|p_i - p\|) = e^{-\frac{\|p_i - p\|^2}{h^2}}$ , where  $h^2$  is chosen to be one third the square distance between  $p$  and its  $k$ -th nearest neighbor. We call this method as weighted plane fitting method, or WPF in short.

Mitra, Nguyen and Guibas noticed that a proper selection of the value of  $k$  is crucial to obtain a good normal estimation. Using the same value of  $k$  at all points as in [HDD\*92] could give biased fitting especially at places where samples are arbitrarily dense. Hence, instead of using  $k$  nearest neighbors of the point, they consider the samples within a ball of certain radius  $r$ . Under the assumption that the noise has zero mean and standard deviation  $\sigma_n$ , they could get a bound on the angle between the estimated normal and the true normal with a probability almost one. An optimal radius  $r$  can be obtained by minimizing this bound, which has the following expression in three dimensional case provided the probability is  $1 - \epsilon$ :

$$r = \left( \frac{1}{\kappa} \left( c_1 \frac{\sigma_n}{\sqrt{\epsilon \rho}} + c_2 \sigma_n^2 \right) \right)^{\frac{1}{3}} \quad (2.1)$$

where  $\rho$  is the local sampling density,  $\kappa$  is the local curvature and  $c_1$  and  $c_2$  are some constants. The actual algorithm takes  $\sigma_n$  as user input and evaluates  $r$  in an iterative manner. Initially  $\rho$  and  $\kappa$  are evaluated based on the  $k$  ( $= 15$ ) nearest neighbors and then the radius  $r$  is obtained from equation 2.1. Once one gets the neighborhood size  $r$ ,  $\rho$  and  $\kappa$  are re-evaluated based on the samples within this neighborhood to get a better estimation of  $\rho$ ,  $\kappa$  and  $r$ . They claim that three iterations in general are enough to obtain good estimations for all the quantities. We call the above method as the adaptive plane fitting method, or APF in short.

### 3. Big Delaunay ball method

For a "noise-free" point set, Amenta et al. [AB99] proposed a Voronoi based method for estimating normals. For a given set of points  $P \subset \mathbb{R}^3$ , let  $\text{Vor}P$  and  $\text{Del}P$  denote the Voronoi diagram and its dual Delaunay triangulation of  $P$  respectively. Denote the Voronoi cell for a point  $p$  as  $V_p$ . Amenta

and Bern [AB99] showed that the line through  $p$  and the furthest Voronoi vertex in  $V_p$ , called its *pole*, can approximate the normal at  $p$  up to orientation. However this property does not hold for noisy samples. Dey and Goswami [DG04] extended the idea of poles to the noisy samples.

Call a ball *Delaunay* if its boundary circumscribes a Delaunay tetrahedron, or equivalently has a center at a Voronoi vertex  $v$  and has a radius  $\|v - p\|$  where  $v \in V_p$ . By definition, the Delaunay balls are maximally empty. The Delaunay balls with poles at their centers are called *polar balls*. The observation of Amenta and Bern can be interpreted in terms of the polar balls as follows. If  $p$  is a sample point on the boundary of a polar ball  $B$ , the segment joining  $p$  and the center of  $B$  estimates the normal direction at  $p$ . Dey and Goswami observed that, under some reasonable noise model, certain Delaunay balls remain relatively big and can play the role of polar balls. This suggests an algorithm for estimating the normals for the noisy point cloud. Redefine the *pole* for a point  $p \in P$  as the furthest vertex of its Voronoi cell whose dual Delaunay ball is big. Similar to the "noise-free" case, the normal line at  $p$  can be approximated by the line through  $p$  and its *pole*. We call this algorithm Big Delaunay Ball algorithm, or BDB in short.

#### 3.1. Algorithm

The key to the BDB algorithm is to identify the big Delaunay balls. The big Delaunay balls are identified by comparing their radii with the nearest neighbor distances of the incident samples. Specifically, for a point  $p \in P$ , let  $\lambda_p$  denote its average nearest distances to the five nearest neighbors of  $p$  in  $P$ . We call a Delaunay ball big if its radius is larger than  $c\lambda_p$  for at least one of its incident points  $p \in P$  where  $c$  is an user defined parameter. A small value for  $c$  makes the algorithm sensitive to the noise since the small Delaunay balls are identified as big. On the other hand, a large value for  $c$  makes less Delaunay balls marked as big. As a result, more points have no big Delaunay ball incident on them and hence no normal can be estimated for these points. We fix  $c = 2.5$  in our experiments which yields the best normal estimation for all the models.

After we obtain the estimation for the normal lines, We adopt the same method as Hoppe et. al [HDD\*92] to orient them consistently. The entire algorithm is described in Figure 1.

#### 3.2. Justification

The justification of the BDB algorithm is given by a claim in [DS05]. To understand the claim, one needs the definition of local feature size  $\text{lfs}(\cdot)$  for a smooth surface  $\Sigma$ . For any point  $x \in \Sigma$ ,  $\text{lfs}(x)$  is defined to be the distance of  $x$  to the medial axis of  $\Sigma$  [AB99]. Assume that  $P$  is a noisy sample of  $\Sigma$  where it satisfies the locally uniform  $\epsilon$ -sampling conditions. We refer the readers to [DS05] for a definition of this

```

BDB( $P, c$ )
  Compute Del  $P$ 
  for each point  $p \in P$ 
    compute  $\lambda_p$ 
    for each Delaunay ball incident on  $p$ 
      if its radius  $r > c\lambda_p$  then mark it as big
    endfor
  endfor
  for each point  $p$  incident to a big Delaunay ball
    find the furthest Voronoi vertex  $c$  in  $V_p$ 
    compute the normal line as the line through  $p$  and  $c$ 
  endfor
  orient the normals consistently.
    
```

**Figure 1:** Algorithm BDB.

sampling condition. Roughly, this sampling condition means that each point of  $\Sigma$  has a sample point within a small factor (given by  $\epsilon$ ) of the local feature size and also the sample points cannot cluster together arbitrarily.

**Claim 1 (DS05)** Let  $p \in P$  be incident to a Delaunay ball with the center  $c$  and radius  $r$ . Let  $r > \frac{1}{5} \text{fs}(\tilde{p})$  where  $\tilde{p}$  is the closest point of  $p$  in  $\Sigma$ . Then, the acute angle between the normal line at  $\tilde{p}$  to  $\Sigma$  and the line through  $p$  and  $c$  is  $O(\epsilon)$  for a sufficiently small  $\epsilon > 0$ .

Notice that some sample points may have no big Delaunay ball incident on them. Hence no normal can be estimated for these points. However Claim 2 proved in [DG04] shows that there are sufficiently many big Delaunay balls and hence the sample points to estimate the normals of the surface almost everywhere.

**Claim 2 (DG04)** For each point  $x \in \Sigma$ , there is a Delaunay ball containing a medial axis point inside and a sample point on the boundary within  $O(\epsilon)$  distance from  $x$ .

Claim 1 and Claim 2 justify the BDB method.

## 4. Comparison

In this section, we compare the big Delaunay ball (BDB) method with the WPF and APF methods. Since BDB method does not estimate the normal for all points in the point cloud, we only compare the estimated normals for those points at which the BDB method estimates the normals. Notice that, the normal estimation only at a subset of the input points is not a serious restriction for the BDB method as the normals can be interpolated such as with Gaussian interpolation [AK04, DGS05] to obtain normals at other points.

### 4.1. Experimental setup

For comparing the methods, ideally we need to measure the deviation of the estimated normals from the “true” surface normals. But, for point cloud data, often we do not know

the sampled surface. We compensate for this shortcoming by computing a set of *referential* normals as described below.

For experiments with noise we obtain noisy data by adding noise to the original point cloud data. The x, y and z components of the noise are independent and uniformly distributed. Their amplitudes (noise level) are controlled by a factor as described later. For referential normals, first we compute a surface from the original data (presumably no noise) by a surface reconstruction software called COCONE [COC]. Then, the average of the normals of the triangles incident to a vertex  $p$  in this surface is taken as the referential normal for  $p$ .

The other surfaces we consider are some parametric algebraic surfaces. The true normals to these surfaces are numerically computed. The normal at each point is computed as the cross product of two tangential vectors.

We define the error of an estimated normal at a point  $p$  as the angle (in radians) between the referential normal and the estimated normal. Obviously the smaller the error, the better the normal estimation is.

### 4.2. Noisy data

In our experiment, we choose the noise level both with respect to a global and a local scale.

For the global scale, we take the amplitude of noise to be a factor of the largest side of an axis parallel bounding box of the point cloud. Since this global yardstick is large, the factor needs to be small so that the point cloud after perturbations remain reasonable for reconstruction. The factors we experiment with are 0, 0.005, 0.01 and 0.02. The global scale for perturbations is perhaps more close to reality. In choosing the factor with respect to a local scale, we use the average distance of a point  $p$  to its five nearest neighbors. The point  $p$  is perturbed with a factor of this distance. Four factors 0, 0.5, 1 and 2 are considered for the experiments. In the APF method, we increase the value of parameter  $\sigma_n$  accordingly as the noise level increases.

We use three data sets, TORUS, BIGHAND and MAX-PLANCK for perturbations with noise. Just to give an idea of the perturbations, see the rendered point clouds of BIGHAND in Figure 4 with different noise levels.

Table 1 and Table 2 show the errors of different normal estimation methods over these three point clouds with different noise levels. They list the error values, standard deviations and timings. We make several observations from these experimental data. Also, we plot the average errors in Figure 2.

First of all, when the noise level is low, all three methods estimate normals well as indicated by small mean error and small standard deviation. The WPF and BDB methods perform almost comparably. In general, WPF method gives the best estimation when the noise level is low. As the noise

Model Name	# pts	Noise Level	Mean Error			Standard Deviation			Timing(sec)		
			BDB	WPF	APF	BDB	WPF	APF	BDB	WPF	APF
TORUS	3200	0	0.112	0.014	0.080	0.051	0.005	0.069	2.99	0.61	3.20
		0.005	0.167	0.114	0.203	0.124	0.061	0.125	3.00	0.65	2.98
		0.01	0.258	0.256	0.477	0.209	0.172	0.342	1.99	0.67	3.12
		0.02	0.355	0.622	0.798	0.284	0.396	0.403	1.87	0.68	3.23
BIGHAND	38218	0	0.053	0.019	0.032	0.069	0.048	0.073	44.20	6.55	17.65
		0.005	0.169	0.094	0.135	0.157	0.119	0.152	44.37	6.18	17.47
		0.01	0.244	0.196	0.508	0.211	0.181	0.379	36.69	6.25	17.80
		0.02	0.311	0.455	0.797	0.275	0.324	0.419	35.13	6.67	17.19
MAX-PLANCK	49089	0	0.056	0.028	0.034	0.062	0.036	0.045	45.99	8.14	21.57
		0.005	0.204	0.125	0.202	0.188	0.108	0.205	44.88	8.04	21.49
		0.01	0.374	0.307	0.759	0.336	0.262	0.410	44.18	8.16	21.53
		0.02	0.593	0.664	0.835	0.444	0.398	0.411	45.11	8.12	21.93

**Table 1:** Normal estimation errors (in radians) under noise levels determined with respect to the global scale.

Model Name	# pts	Noise Level	Mean Error			Standard Deviation			Timing(sec)		
			BDB	WPF	APF	BDB	WPF	APF	BDB	WPF	APF
TORUS	3200	0	0.112	0.014	0.080	0.051	0.005	0.069	3.01	0.60	3.20
		0.5	0.219	0.150	0.259	0.152	0.083	0.162	1.99	0.63	3.16
		1	0.340	0.445	0.753	0.252	0.293	0.414	1.78	0.66	3.18
		2	0.495	1.023	1.129	0.353	0.374	0.326	1.87	0.65	3.19
BIGHAND	38218	0	0.053	0.019	0.032	0.069	0.048	0.073	44.08	6.34	17.37
		0.5	0.175	0.092	0.139	0.141	0.077	0.129	35.19	6.32	17.35
		1	0.241	0.198	0.451	0.206	0.143	0.336	33.26	6.50	17.17
		2	0.328	0.524	0.829	0.286	0.348	0.416	34.14	6.66	17.71
MAX-PLANCK	49089	0	0.056	0.028	0.034	0.062	0.036	0.045	45.92	8.26	21.37
		0.5	0.160	0.092	0.132	0.120	0.060	0.095	38.58	8.17	20.99
		1	0.237	0.197	0.442	0.198	0.123	0.333	39.57	8.45	21.59
		2	0.309	0.573	0.822	0.281	0.372	0.417	37.26	8.33	22.08

**Table 2:** Normal estimation errors (in radians) under noise levels determined with respect to the local scale.

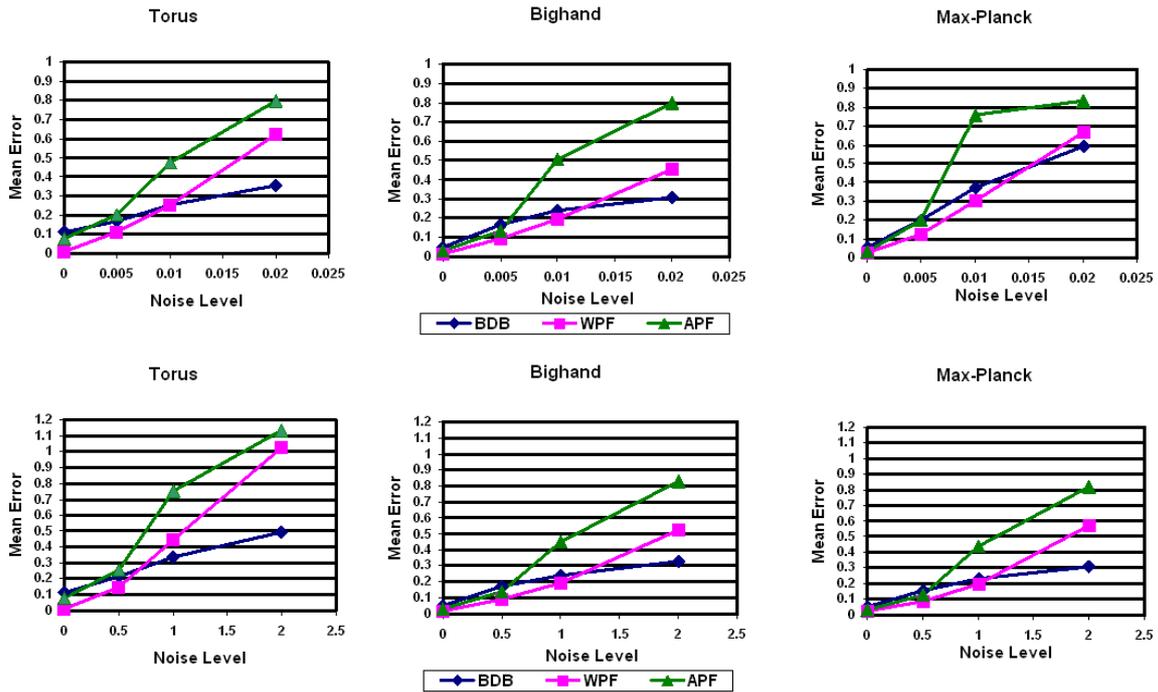
level increases, BDB method gives relatively better performance. In general both WPF and BDB perform better than the APF method, the difference being more pronounced for larger noise levels.

### 4.3. Special cases

Other than the noisy point clouds, we experiment with the normal estimation methods on a couple of special point clouds. In one case the point cloud samples the surface very unevenly and in the other, the point cloud samples a "thin"

surface, i.e., the surface has high curvature at some areas. These special point clouds are obtained by sampling some parametric surfaces in a special way.

We obtain the first type of special point cloud TORUS by sampling a torus with dense samples along the equator line, as the left most picture of the first row in Figure 3 shows. Also, we sample a part of a single sheet hyperboloid where the points line up along some curves as shown in Figure 3. This point cloud HYPERBOL also serves as an example where the surface has boundaries.



**Figure 2:** Normal estimation comparison under noise levels with respect to the global scale(upper row), and the local scale(bottom row).

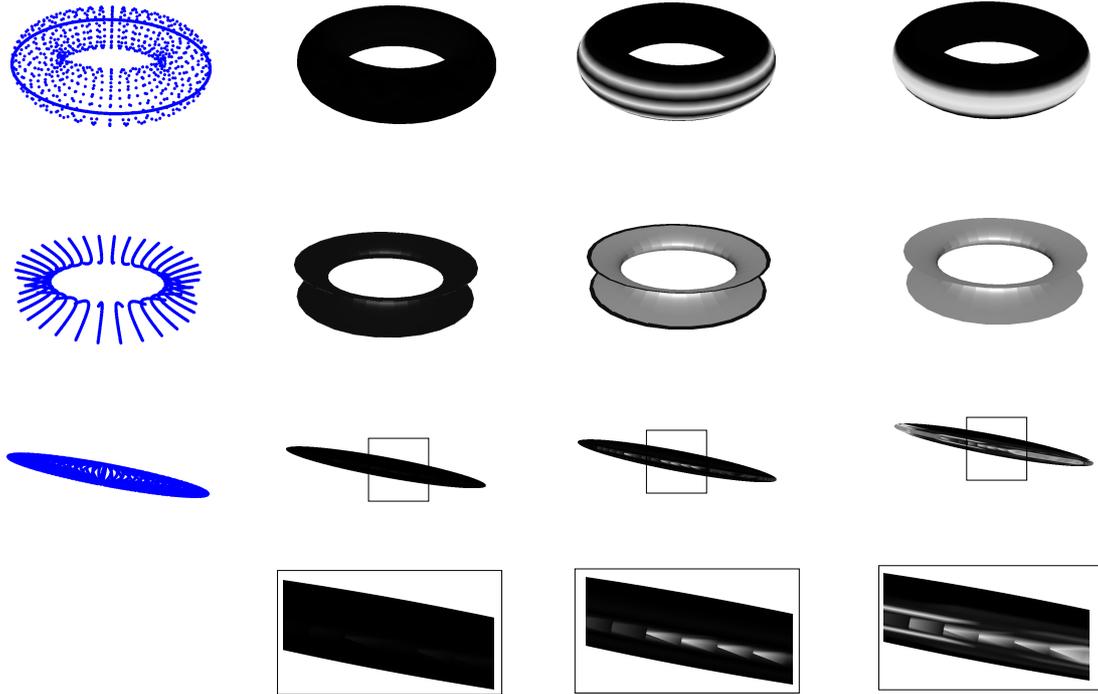
For both of TORUS and HYPERBOL, BDB method works better than the other two methods as Figure 3 shows. The reason can be explained as follows. All three methods use some  $k$ -nearest neighbors to estimate the normals for some value of  $k$ . While WPF and APF use these neighbors to fit a plane, BDB use them to estimate the local sampling density. When points line up along a curve on the surface, APF and WPF try to fit a plane through the points along the curve since  $k$  nearest neighbors lie along that curve. Consequently, this plane deviates from the true tangent plane considerably. Of course, if  $k$  is chosen large enough the problem goes away. But, this becomes a serious issue for the users to supply an appropriate  $k$  and also a single  $k$  may not be appropriate for all places on the surface. Bad estimation of normals along the equator line of TORUS and almost all over HYPERBOL by APF and WPF are results of this pronounced dependency on  $k$ . We kept  $k = 40$  (the default value in PointShop3D) for WPF method and  $k = 15$  (the suggested value in [MNG04]) for APF method. The BDB method, on the other hand, does not depend on the choice of  $k$  so sensitively. It only needs to estimate the average distance to its local neighbors for some  $k$  neighbors. We kept  $k = 5$  in the experiments. The arrangement of points along some specific directions (which is not rare in scanning processes) is not so harmful for the BDB method. In summary, BDB does not rely on local information as much as APF and WPF do.

In another special class of point clouds, we sample a very thin ellipsoid; see the left most picture of the third row in Figure 3. For a point at the high curvature region, the neighboring points from the point cloud does not lie close to a plane and hence the fitting plane computed by WPF method or APF method could not approximate the tangent plane properly at those points. Of course, this problem can be attributed to the poor sampling density at the high curvature regions which is not again uncommon in the scanning processes. However, BDB method is not so sensitive to this relatively poor sampling as Figure 3 shows.

In the above examples, we make some exaggeration about the specialty of the point cloud for the illustration purpose. However these special cases do occur in the real data up to a certain degree.

#### 4.4. Timings

Tables 1 and 2 show the timings for the three methods. Clearly, the BDB method is the slowest. The main reason is that it employs a Delaunay triangulation procedure to the entire point cloud while the other two methods can operate very locally. The very reason of globality for which BDB works better than the other two in special cases makes it slower. However, we must mention that we used CGAL 2.3 [CGA] for computing the Delaunay triangulations. Recent



**Figure 3:** The first column shows the point clouds of TORUS, HYPERBOL and ELLIPSOID. We take the normal estimation error as grey scale color for each point and render the surfaces with Gouraud shading. The darker the surface, the better the normal estimation. The column two, three and four show the normal estimation results of BDB, WPF and APF methods respectively.

versions of CGAL are faster and we plan to test the timings on these versions in future. On reasonable state-of-the-art PCs, the Delaunay triangulation takes time in the order of minutes and hours for point cloud data in the range of several hundred thousands and millions respectively [DGH01]. Therefore, timings do not become a prohibitive issue for the point clouds up to this range.

#### 4.5. Summary

We summarize our observations from the experiments as follows. When the noise level is low and the point cloud samples the surface more or less evenly, all the three methods perform almost equally well though WPF gives the best results. When the noise level is relatively high and the sampling is skewed along some curves or is not dense enough for thin parts, BDB works the best. In general, if the size of the point cloud is no more than a few million points ( $\sim 5$  million points), BDB is safer to use if one does not have specific knowledge about the quality of the input. Otherwise, WPF or APF should be preferred.

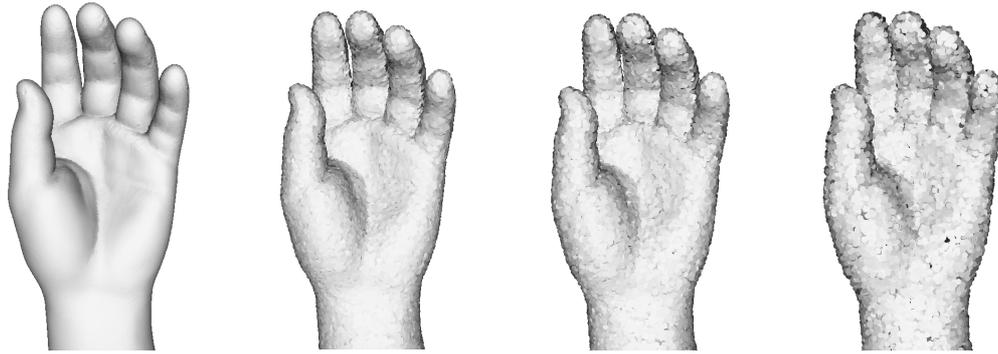
#### 5. Applications

In earlier work, normal estimations with numerical techniques have been used for some applica-

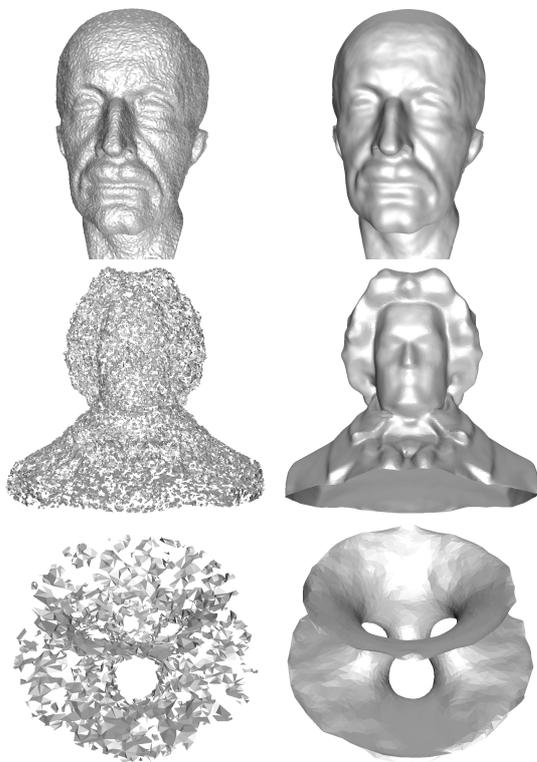
tions [AA03, PKKG03]. In this section we show some example applications where BDB method can be used effectively.

The first one is the point cloud rendering. We feed the point cloud together with the estimated normals to PointShop 3D and use the so call "OpenGL preview" technique to render the point cloud directly. Figure 4 shows the rendering results for BIGHAND with different noise levels.

Dey and Sun [DS05] define a smooth MLS (moving least squares) surface called adaptive MLS (AMLS) based on the set of points with normals possibly containing noise. All sample points can be projected onto an approximation of this surface using a Newton projection method. Once all sample points are projected, one can use any of the existing reconstruction algorithms to reconstruct the surface. Here we use the COCONE software [COC] which can reconstruct surfaces with or without boundary. In figure 5, we show the results of three different point clouds: MAX-PLANCK, HYPERSHEET and LUDWIG. The left column shows the reconstruction results before the point cloud gets smoothed and the right column shows the reconstruction results after smoothing.



**Figure 4:** First hand from left is the rendering result for the original smooth point cloud. The 2nd, 3th and 4th hands (from left to right) are the rendering results for the point clouds with noise levels (local scale) 0.5, 1 and 2 respectively.



**Figure 5:** Smooth reconstruction from noisy point clouds.

#### Acknowledgements.

We acknowledge the support of Army Research Office, USA under the grant DAAD19-02-1-0347 and NSF, USA under grants DMS-0310642 and CCR-0430735.

#### References

- [AA03] ADAMSON A., ALEXA M.: Ray tracing point set surfaces. In *Proceedings of Shape Modeling International* (2003), pp. 272–279.
- [AB99] AMENTA N., BERN M.: Surface reconstruction by voronoi filtering. *Discr. Comput. Geom.* 22 (1999), 481–504.
- [ABCO\*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C.: Point set surfaces. In *Proc. IEEE Visualization* (2001), pp. 21–28.
- [AK04] AMENTA N., KIL Y. J.: Defining point-set surfaces. In *Proceedings of ACM SIGGRAPH 2004* (Aug. 2004), ACM Press, pp. 264–270.
- [BC00] BOISSONNAT J. D., CAZALS F.: Smooth surface reconstruction via natural neighbor interpolation of distance functions. In *Proc. 16th. Annu. Sympos. Comput. Geom.* (2000), pp. 223–232.
- [CGA] CGAL: Cgal library. [www.cgal.org](http://www.cgal.org).
- [COC] COCONE: [www.cse.ohio-state.edu/~tamaldey](http://www.cse.ohio-state.edu/~tamaldey). *The Ohio State University*.
- [DG04] DEY T. K., GOSWAMI S.: Provable surface reconstruction from noisy samples. In *Proc. 20th Annu. Sympos. Comput. Geom.* (2004), pp. 330 – 339.
- [DGH01] DEY T. K., GIESEN J., HUDSON J.: Delaunay based shape reconstruction from large data. In *Proc. IEEE Sympos. Parallel and Large Data Visualization and Graphics* (2001), pp. 19 – 27.
- [DGS05] DEY T. K., GOSWAMI S., SUN J.: Extremal surface based projections converge and reconstruct with isotopy. *Technical Report OSU-CISRC-05-TR25*, also available from authors' web-pages (April 2005).
- [DS05] DEY T. K., SUN J.: An adaptive mls surface for reconstruction with guarantees. *Technical Report OSU-CISRC-05-TR26*, also available from authors' web-pages (April 2005).

- [HDD\*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *Proceedings of ACM SIGGRAPH 1992* (1992), vol. 26, pp. 71–78.
- [MNG04] MITRA N. J., NGUYEN A., GUIBAS L.: Estimating surface normals in noisy point cloud data. In *Internat. J. Comput. Geom. & Applications* (2004), p. to appear.
- [PKKG03] PAULY M., KEISER R., KOBELT L., GROSS M.: Shape modeling with point-sampled geometry. In *Proceedings of ACM SIGGRAPH 2003* (2003), ACM Press, pp. 641–650.