A linear method to consistently orient normals of a 3D point cloud

Craig Gotsman · Kai Hormann

Abstract

Correctly and consistently orienting a set of normal vectors associated with a point cloud sampled from a surface in 3D is a difficult procedure necessary for further downstream processing of sampled 3D geometry, such as surface reconstruction and registration. It is difficult because correct orientation cannot be achieved without global considerations of the entire point cloud. We present an algorithm to orient a given set of normals of a 3D point cloud of size N, whose main computational component is the least-squares solution of an $O(N)$ linear system, mostly sparse, derived from the classical Stokes' theorem. We show experimentally that our method can successfully orient sets of normals computed locally from point clouds containing a moderate amount of noise, representing also 3D surfaces with non-smooth features (such as corners and edges), in a fraction of the time required by state-of-the-art methods.

Citation Info

Conference
SIGGRAPH

Location
Denver, July 2024

Publisher
ACM

Pages
Article 55, 10 pages

DOI
10.1145/3641519.365742

1 Introduction

3D reconstruction from point clouds has been the subject of intense research for at least three decades, the primary objective being to construct a surface from a set of 3D samples of that surface, typically the output of a scanning process. Many variants of the problem exist, depending on what additional information is available. For example, if each point sample is accompanied by the surface normal vector at that point, which sometimes can also be measured by the scanning device, the problem becomes easier, as this gives additional information on the plane tangent to the surface at that point, and the direction pointing "outside" the volume enclosed by the surface.

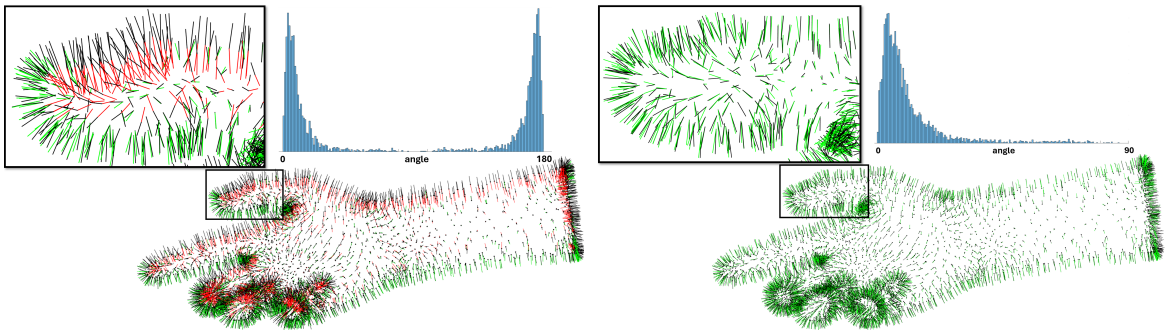


Figure 1: We advocate that normal computation for a 3D point cloud can be achieved by computing the normal at each point based on local information only, resulting in random "inside"/"outside" orientations, and then consistently orienting them to all be "outside" by a global procedure which involves just the solution of a linear system. We visualize this on the Hand point cloud (sampled from the Hand triangle mesh) as follows: black lines are the "ground truth" normals computed from the original triangle mesh, green and red lines are the locally-computed normals, where normals in green are correctly oriented and those in red are incorrectly oriented. (Left) Unoriented locally-computed normals and histogram of the distribution of angles between these normals and ground truth normals in the range $[0, 180°]$; approximately half of them are incorrectly oriented. (Right) Same locally-computed normals correctly oriented after running our simple global procedure. These oriented local normals are a very good approximation to the ground truth normals, as can be seen in the distribution of angles between these normals and the ground truth in the range $[0, 90°]$, which is concentrated in the interval $[0, 15°]$. We conclude that there is really no need to compute the normals themselves globally.

As the normal vector at a point depends primarily on the *local* behaviour of the surface at that point, its direction as perpendicular to the tangent plane is relatively easy to estimate. This is typically done by approximating the tangent plane based on the point and a small number of its closest neighbors in the cloud. Principal Component Analysis (PCA), numerically performed using singular value decomposition (SVD), can detect the two principal vectors spanning this small subset, defining the tangent plane. The third vector is then the direction of the normal to this plane [5]. While easy, this method cannot tell the correct "orientation" (i.e., the sign) of the normal vector, namely whether it points "inwards" or "outwards" at that point. This characteristic is *global*, in the sense that *all* normal vectors need to be considered and oriented consistently throughout the point cloud in order to determine the correct orientation of each individual normal. More sophisticated local techniques for estimating surface normals in the presence of noise [14, 3] or those based on higher order differential properties [1] exist, but these are also local, and cannot reveal the correct orientations. The interested reader is referred to Sanchez et al. [16] for a comprehensive survey of local normal estimation techniques.

This paper, as some before, assumes that sufficiently accurate estimation of normals up to their orientation is, in practice, a *solved problem*, and attempts to solve *only* the remaining global orientation problem. This is demonstrated also in Figure 1. Thus, we present a method to consistently orient *given* normal vectors of a point cloud, which may have arbitrary (i.e., incorrect) orientations. This is done by generating and solving a set of (overdetermined) linear equations for the signs of the given vectors. The equations are derived from the well-known vector analytic theorem due to Stokes, involving the integral of the inner product of certain 3D functions with the normal vector over the entire surface. Approximating these integrals by a discrete sum generates the linear equations. Consequently, requiring just the (least squares) solution of a typically sparse linear system, our method is orders of magnitude faster than other methods, which tend to resort to the solution of large global non-linear optimization problems.

2 Related work

Estimating normals for a point cloud is a well-understood problem and there exist many local methods to do this. However, the local methods typically fail to orient the normals correctly, as this requires global information to guarantee consistency throughout the point cloud (i.e., all normals point "outwards"), thus need some extra effort. Normal orientation methods may be classified into two approaches: propagation methods and global methods. Propagation methods start at some point in the cloud and "greedily" traverse the cloud while orienting the normals associated with each point in turn in a serial manner so that all normals seen so far are aligned. Starting with the early work of Hoppe et al. [5], which does not do too well on challenging inputs, more sophisticated propagation techniques have been developed over the years [9, 17, 6], culminating in the method of Metzer er al. [13]. which combines local propagation within small "patches" of points with a final semi-global step based on a "dipole" electric field induced by all previously oriented patches. Global methods are more elaborate, typically solving a global optimization problem, thus require more computational effort than propagation methods, especially on large inputs. The most recent "Globally Consistent Normal Orientation" (GCNO) method of Xu et al. [18] does precisely this, relying on the fact that a consistent orientation of the normals leads to an accurate computation of so-called *winding numbers* of the shape, essentially defining well the partition of space into the interior and exterior of the 3D object. The winding number of a point $c=(c_x,c_y,c_z)$ relative to a closed 2-manifold $S\subset\mathbb{R}^3$ enclosing a volume V, namely $S=\partial V$, is defined as

$$w(c)=\frac{1}{4\pi}\iint_{p\in S}\frac{(c-p)\cdot\hat{n}(p)}{\|c-p\|^3}\,\mathrm{d}s(p)=\begin{cases}1, & c\in V,\\ 0.5, & c\in S,\\ 0, & c\notin V,\end{cases}\tag{1}$$

where $\hat{n}(p)$ is the unit normal vector at $p\in S$, "$\cdot$" is the inner product operator, and $\mathrm{d}s(p)$ is the (area of the) surface element at p. The observant reader familiar with physics will note that the integrand $\frac{(c-p)\cdot\hat{n}(p)}{4\pi\|c-p\|^3}$ is just the (scalar) electric potential at p induced by a dipole at c polarized in the direction of the normal $\hat{n}$, and the (vector) dipole itself, $\frac{c-p}{4\pi\|c-p\|^3}$, is the gradient of the fundamental solution of the 3D Laplace equation centred at c [4],

$$\Phi(p)=\frac{1}{4\pi\|c-p\|},$$

which is harmonic everywhere except at the pole c, thus the dipole has vanishing divergence in any region not containing c. Equation (1) is sometimes called the Gauss formula [11, 10] or Gauss's Law [2] (related to the divergence theorem) and may be approximated in the discrete setting as a finite sum over the N points p_i having 3D coordinates (x_i, y_i, z_i) and their normals $\hat{n}_i$ sampled from the surface,

$$w(c) \approx \frac{1}{4\pi} \sum_{i=1}^{N} A_i \frac{(c - p_i) \cdot \hat{n}_i}{\|c - p_i\|^3},$$

where A_i is the surface area associated with p_i. The Gauss formula is frequently used to determine whether a given point is inside or outside a closed surface in 3D, and Lu et al. [11] and Lin et al. [10] have used this formula, especially the case of c on S, for surface reconstruction purposes. We provide these details because, as we shall see later, our approach is conceptually related to these methods in that it relies heavily on the properties of the divergence of certain 3D functions, especially those which are divergence-free (i.e., have vanishing divergence).

3 Applying Stokes' theorem

Our method will make extensive use of the well-known Stokes' theorem in vector analysis [2]. Assume a 2-manifold $S \subset \mathbb{R}^3$ with a boundary B and a function $G : \mathbb{R}^3 \to \mathbb{R}^3$ possessing continuous first-order partial derivatives, namely, $G = (G_x(x, y, z), G_y(x, y, z), G_z(x, y, z))$. Then its curl through S is equal to its contour integral over B,

$$\iint_S (\nabla \times G) \cdot \mathrm{d}S = \oint_B G \cdot \mathrm{d}B. \tag{2}$$

The direction of positive circulation of B and the direction of positive flux through S are related by a right-hand-rule, namely, the fingers circulate along B and the thumb points in the direction of the positive flux.

As a matter of notation, note that the differential $\mathrm{d}S$ in (2) is equivalent to $\hat{n}(p)\mathrm{d}s(p)$ in (1). Note also that the surface S should be continuous, but not necessarily smooth, as long as the two integrals in (2) exist. A more explicit version of (2) is the following. Denote $F = \nabla \times G$, namely,

$$F = \nabla \times G = \left(\frac{\partial G_z}{\partial y} - \frac{\partial G_y}{\partial z}, \frac{\partial G_x}{\partial z} - \frac{\partial G_z}{\partial x}, \frac{\partial G_y}{\partial x} - \frac{\partial G_x}{\partial y} \right),$$

then

$$\iint_S \left(F_x \, \mathrm{d}y \, \mathrm{d}z + F_y \, \mathrm{d}z \, \mathrm{d}x + F_z \, \mathrm{d}x \, \mathrm{d}y \right) = \oint_B \left(G_x \, \mathrm{d}x + G_y \, \mathrm{d}y + G_z \, \mathrm{d}z \right). \tag{3}$$

In the discrete case, the surface integral in (2) can be approximated by a sum over the N samples $p_i = (x_i, y_i, z_i)$ of the surface S, where each inner product of $F = \nabla \times G$ evaluated at that point with the unit normal $\hat{n}$ at the point is weighted by the area A_i that the sample represents,

$$\iint_S F \cdot \mathrm{d}S \approx \sum_{i=1}^{N} A_i \big(F(x_i, y_i, z_i) \cdot \hat{n}_i \big).$$

Similarly, the contour integral in (2) can also be approximated by a discrete sum. We now elaborate on two special cases of Stokes' theorem that we will use in our approach.

3.1 The homogeneous case

The first special case of (2) arises when the surface S is closed, namely, has no boundary. Then (2) reduces to

$$\iint_S (\nabla \times G) \cdot \mathrm{d}S = 0.$$

The observant reader will notice that this is also a consequence of the divergence theorem [2], due to the identity

$$\nabla \cdot (\nabla \times G) = 0, \tag{4}$$

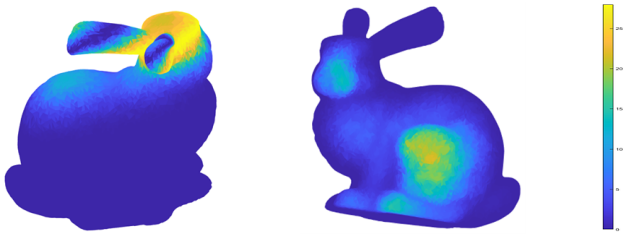


Figure 2: Norm of a compactly supported 3D function F^c of type (7) on the Bunny model bounded by a unit cube for (left) $c=(0.9,1.25,1.5)$ with density 4689/10000 and (right) $c=(-0.5,1.0,-1.0)$ with density 4073/10000.

namely, $\nabla\times G$ is always divergence-free for any 3D function G [2].

Now, if we are given a set of N unoriented normals $\hat{n}_i$ for the samples $p_i=(x_i,y_i,z_i)$ and we aim to orient them correctly, then we can take a suitable 3D function G, derive $F=\nabla\times G$, and seek signs $s_i\in\{+1,-1\}$, such that

$$\sum_{i=1}^{N} A_i\big(F(x_i,y_i,z_i)\cdot\hat{n}_i\big)s_i=0. \tag{5}$$

This is a *single* homogeneous linear equation in the N unknowns s_i, and we will need many more to solve for a reliable solution. Fortunately, any number of such equations can be generated by using different functions G, although care must be exercised to not use functions which are linearly dependent on each other, as these will not add any new information (i.e., independent equations). For example, using $G^1(x,y,z)=(y,-z,x)$, $G^2(x,y,z)=(-y^2,z^2,x^2)$ and $G^3(x,y,z)=(3y^2-y,-3z^2+z,-3x^2-x)$ will create only two independent equations of the type (5).

A simple way to generate an essentially unlimited number of linearly independent functions, via a parameter $c=(c_x,c_y,c_z)\in\mathbb{R}^3$, which we call the *center* of the function, is to consider

$$G^c(x,y,z)=\left(\frac{1}{r},\frac{1}{r},\frac{1}{r}\right),\qquad r(x,y,z)=\big\|(x,y,z)-(c_x,c_y,c_z)\big\|,$$

so that

$$F^c(x,y,z)=\nabla\times G^c(x,y,z)=\left(\frac{(y-c_y)(z-c_z)}{r^3},\frac{(z-c_z)(x-c_x)}{r^3},\frac{(x-c_x)(y-c_y)}{r^3}\right). \tag{6}$$

The proof that $\{F^{c_j}:j=1,\dots,m\}$ is a linearly independent set of functions for all m is elaborate because of the multivariate nature of the functions. We will just remark that this would be obvious in the 1D case, where $G^c(x)=\frac{1}{|x-c|}$, by examining the Wronskian [4] of $\left\{F^{c_j}=\frac{\mathrm{d}G^{c_j}}{\mathrm{d}x}:j=1,\dots,m\right\}$, that is, the determinant of the matrix formed by these m functions and their first $m-1$ derivatives. It is well-known that a set of functions is linearly independent if its Wronskian is not the constant zero function. Since the Wronskian in this case is just the determinant of a Vandermonde matrix multiplied to the left and to the right by diagonal matrices, it is indeed a non-zero function, if and only if all the c_j are distinct. Note that F^{c_j} is supported on all of $\mathbb{R}^3$, which makes for a dense set of equation coefficients.

Since sparse linear systems are more desirable than dense ones, it is advantageous to use functions F with compact support. This can be done using the smooth cubic B-Spline $B:\mathbb{R}\to\mathbb{R}$, which is supported on $[-2,2]$,

$$B(t)=\tfrac{1}{12}\big(|t+2|^3-4|t+1|^3+6|t|^3-4|t-1|^3+|t-2|^3\big),$$

and defining, for example,

$$\begin{aligned} G^c(x,y,z)&=\big(B(y-c_y),B(z-c_z),B(x-c_x)\big),\\ F^c(x,y,z)&=\nabla\times G^c(x,y,z). \end{aligned} \tag{7}$$

By taking m distinct centres $\{c_1,\dots,c_m\}$, the resulting linear system is then

$$Ms=0, \tag{8}$$

where the $m\times N$ matrix M is defined as

$$M_{j,i}=A_i\big(F^{c_j}(x_i,y_i,z_i)\cdot\hat{n}_i\big).$$

Depending on the position of c_j relative to the point cloud, using the resulting $F^{c_j}(x,y,z)$ will give a *sparse* equation for s_i, since the values of $F^{c_j}(x_i,y_i,z_i)$ will vanish on many of the points in the cloud. This equation will, albeit, still be homogeneous. See Figure 2 for some examples of these functions.

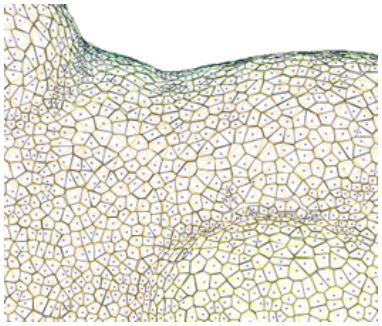
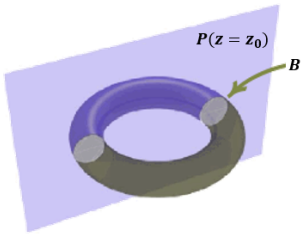


Figure 3: Voronoi polygons associated with the 10000-point Bunny cloud. The polygons are in black (triangulated in yellow). Note that they do not form a seamless surface, as expected.

3.2 The non-homogeneous case

To solve for $s=(s_1,\ldots,s_N)$ requires constructing equations of the type (5) with appropriate functions F, either dense or sparse. As we will see later, these are augmented by a sparse set of N homogeneous regularization equations based on a Laplacian operator derived from the point cloud, so the number of equations is larger than N, and should be solved for s in the least squares sense. However, the fact that the system is homogeneous complicates matters, as it permits the trivial solution $s=(0,\ldots,0)$, and eliminating this requires solving an eigensystem. It would be much better if we were able to generate some non-homogeneous equations, namely those with a non-vanishing right-hand side. We propose to do this by returning to the general version of Stokes' theorem (2) with a non-vanishing right-hand side. This requires operating on a surface with a boundary. To achieve this, we *cut* through the surface with a *cut-plane* P (see inset) and identify the boundary B of the resulting open manifold(s). Assume without loss of generality that P is a an x-y-parallel plane $z=z_0$. Then (3) becomes



$$\iint\limits_{S \text{ above } P} F\cdot \mathrm{d}S = \oint\limits_B G_x(x,y,z_0)\mathrm{d}x + \oint\limits_B G_y(x,y,z_0)\mathrm{d}y,$$

with a similar equation for the portion of S below the cut-plane P.

In the discrete case, if B is approximated by the (counter-clockwise) 2D polygon $Q=(q^1,\ldots,q^k)$ with k vertices, the two complementary non-homogeneous equations are

$$\sum_{p_i \text{ above } P} A_i\big(F(x_i,y_i,z_i)\cdot\hat{n}_i\big)s_i = \sum_{i=1}^{k}\frac{1}{2}\big(G_x(q^i)+G_x(q^{i+1})\big)(q_x^{i+1}-q_x^i)+\sum_{i=1}^{k}\frac{1}{2}\big(G_y(q^i)+G_y(q^{i+1})\big)(q_y^{i+1}-q_y^i) \quad (9)$$

and

$$\sum_{p_i \text{ below } P} A_i\big(F(x_i,y_i,z_i)\cdot\hat{n}_i\big)s_i = -\sum_{i=1}^{k}\frac{1}{2}\big(G_x(q^i)+G_x(q^{i+1})\big)(q_x^{i+1}-q_x^i)-\sum_{i=1}^{k}\frac{1}{2}\big(G_y(q^i)+G_y(q^{i+1})\big)(q_y^{i+1}-q_y^i), \quad (10)$$

where the vertex q^{k+1} is taken to be identical to q^1. Similar equations may be written for cut-planes which are x-z-parallel or y-z-parallel, or even for cut-planes at arbitrary orientations.

4 Implementation details

Our normal orientation method consists of setting up a sufficient number of homogeneous and non-homogeneous linear equations for the variables s_i associated with the unoriented normals and then solving them in the least squares sense. In this section we elaborate on some of the details of implementing this in practice.

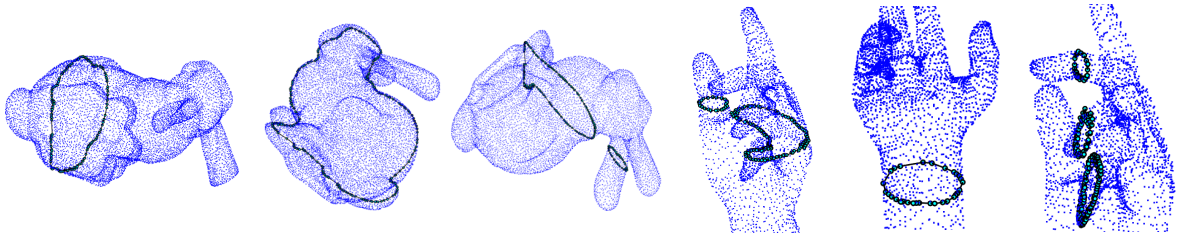


Figure 4: Polygonal boundaries of the cross-sections of the Bunny and Hand models at three different planes. Note that some have multiple contours.

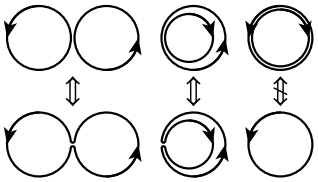
4.1 Estimating areas

To set up the equations in (5), (9), and (10), we first need to determine the value of A_i, the surface area associated with the point p_i having coordinates (x_i, y_i, z_i). A straightforward way to do this is to approximate it with the area of the *Voronoi polygon* associated with this point in the Voronoi diagram constructed from the point and a fixed number (say 12) of its nearest neighbors, when projected to the approximate tangent plane at that point, when computed as usual using PCA. Figure 3 shows these Voronoi polygons for a point cloud of 10000 points sampled from the Stanford Bunny model. Note that these Voronoi polygons themselves may be viewed as a coarse approximation to the sampled surface, but are far from sufficient for a good reconstruction, if that is the objective.

4.2 Identifying cross sections

To set up the equations in (9) and (10), we have to be able to extract a 2D planar boundary of the cross-section of the point cloud when cut with a plane. We do this by identifying all cloud points within a very small distance of the plane (which is a parameter), projecting them to the plane, and applying a Travelling Salesman Problem (TSP) algorithm [12] to each of the connected components of a k-nn graph (we found that $k = 6$ is a good choice) of this subset to order them into a set of closed 2D contours. It is desirable that the contours be significant, namely contain a large enough number of points. In practice, it is wise to discard cut-planes which result in a small number of contour points.

Note that the boundary may consist of multiple polygons, or nested polygons, which does not change anything in the theory, except that the right-hand-side of the linear systems (9) and (10) is computed as the sum of contour integrals over all the contours, when they are all oriented consistently. A consistent orientation is achieved when the outermost contours are counter-clockwise with respect to the cut-plane normal in the direction in which the relevant portion of the point cloud resides, and inner contours have alternating orientations as they nest deeper. Distinct contours that are so close that they "merge" into one in our processing do not pose a problem, since the sum of the contour integrals of two touching contours is the same as a single contour integral of the combined contour (see inset), except if two nested contours are everywhere so close to each other that they are detected as a single contour. The latter may occur for thin-shelled objects, if the sample distance is above the thickness of the shell, and we did not encounter this case in our experiments. See Figure 4 for some boundaries extracted from some cut-planes of the Bunny and Hand point clouds.



4.3 Choosing F

To generate functions F to use in equations (5), (9), and (10), a number of techniques may be employed. The key requirement, as seen in (4), is that F has vanishing divergence in the volume enclosed by the manifold S, that is, $\nabla \cdot F = 0$. The first technique is the method described in Section 3 and which we use in practice: take F to be $\nabla \times G$ for a reasonable G. This also has the advantage of being able to use G for the contour integral in (3) in the non-homogeneous case.

A second possibility is taking $F = \nabla\Phi = \left(\frac{\partial\Phi}{\partial x}, \frac{\partial\Phi}{\partial y}, \frac{\partial\Phi}{\partial z}\right)$, where $\Phi\colon \mathbb{R}^3 \to \mathbb{R}$ is harmonic in $\mathbb{R}^3$. Then, $\nabla \cdot F = \nabla^2\Phi = 0$, by definition. An example of a parametric family of such functions is the 3D potential, featuring

in (1), parameterized by its "center" c,

$$\Phi^c(x,y,z) = -\frac{1}{r}, \qquad r(x,y,z) = \left\|(x,y,z)-(c_x,c_y,c_z)\right\|, \qquad F^c(x,y,z) = \left(\frac{x-c_x}{r^3}, \frac{y-c_y}{r^3}, \frac{z-c_z}{r^3}\right).$$

Note that these functions are harmonic at all points except c, which is the reason why the integral (1) vanishes only if $c \notin V$, as only then $\nabla \cdot F$ vanishes in all of V.

A third, but more complicated, possibility [15] arises from considering two well-behaved scalar functions $u(x,y,z)$ and $v(x,y,z)$ and defining

$$F = \left(\frac{\partial u}{\partial y}\frac{\partial v}{\partial z} - \frac{\partial u}{\partial z}\frac{\partial v}{\partial y}, \frac{\partial u}{\partial z}\frac{\partial v}{\partial x} - \frac{\partial u}{\partial x}\frac{\partial v}{\partial z}, \frac{\partial u}{\partial x}\frac{\partial v}{\partial y} - \frac{\partial u}{\partial y}\frac{\partial v}{\partial x}\right).$$

Then it is easy to see that $\nabla \cdot F = 0$.

The second and third options for generating F may be used primarily for the homogeneous case, since then no contour integrals are required.

4.4 Setting up the equations

To orient a cloud of N 3D points, we need to set up a non-homogeneous linear system having rank at least N, certainly containing at least N equations. Rank $N-1$ is obtained from a Laplacian matrix used for regularization (see below), so not many more are required in theory to obtain full rank. However, more equations are needed to force a consistent orientation of the normals.

We found that in practice, for "easy" inputs, namely point clouds with relatively simple and smooth shapes, which are sampled uniformly and regularly, it suffices to take few homogeneous equations of the type (5), using the family of parametric compactly-supported functions described in (7). The parametric centres are taken as random points within the sphere enclosing the point cloud. A few more pairs of non-homogeneous equations of the type (9) and (10) are taken using the parametric family of functions described in (6). These equations are generated using centres for very few cross-sections, sometimes even just one cross-section suffices.

For more challenging inputs, especially those with irregular sampling patterns and density, also those containing sharp edges and corners, many more equations are needed, sometimes up to three cut-planes along each of the three major axes, each generating $O(N)$ non-homogeneous equations of type (9) and (10), along with $O(N)$ homogeneous equations of type (5) (one per center).

These linear equations are augmented with a regularization component based on the (combinatorial) Laplacian matrix L of the k-nearest neighbor graph of the point cloud. Typically we take $k = 10$. This additional homogeneous set of $3N$ equations forces some degree of continuity on the three components of the oriented vectors $s_i \hat{n}_i$,

$$Rs = 0,$$

where

$$R = \begin{pmatrix} \mathrm{diag}(\hat{n}_x) \\ \mathrm{diag}(\hat{n}_y) \\ \mathrm{diag}(\hat{n}_z) \end{pmatrix} L,$$

with $\mathrm{diag}(v)$ denoting the diagonal matrix whose diagonal entries are taken from the vector v, and $\hat{n}_x, \hat{n}_y, \hat{n}_z$ denoting the vectors of the x-, y-, and z-components of the normals $\hat{n}_1, \ldots, \hat{n}_N$.

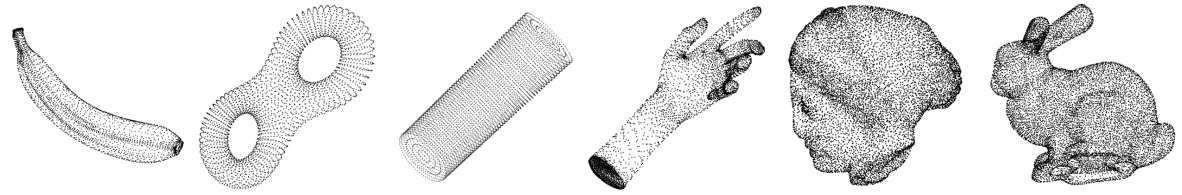
The Laplacian equations are weighted by the average area per point, that is, by $\mathrm{mean}(A_i)$. The complete set of $O(N)$ linear equations, which may be very sparse, is then solved for the vector s by least squares and the desired orientation signs are obtained as $\mathrm{sign}(s_i)$.

4.5 Filtering the solution

In practice, the discretization of the surface integral (2) to the discrete setting, and the transition from a continuous value of the solution s_i to the discrete set $\{+1,-1\}$ inevitably introduces noise into the solution, which could result in a "sprinkling" of incorrect orientations in the solution. Although we observed that there is typically a healthy concentration of the values of the s_i around the desirable values of $\{+1,-1\}$, there may be some values of s_i close to zero, thus risking a mis-classification of their sign. Fortunately, this "noise" can be filtered out by requiring that the orientation of each resulting normal agrees with the orientation of

Table 1: Experimental results for our algorithm vs. the GCNO algorithm on the inputs of Figure 5.

		our algorithm						GCNO algorithm			
dataset	points	cut-planes	non-homogeneous equations per cut-plane	homogeneous equations	solve runtime (secs)	incorrect orientations after solve	incorrect orientations after filter	solver iterations	solve runtime (secs)	incorrect orientations after solve	incorrect orientations after filter
Banana	2984	1	100	50	0.2	1	0	15	280	9	0
Eight	3070	1	200	100	0.6	24	0	13	446	1	0
Cylinder	4802	1	10	0	0.4	1	0	11	660	3	2
Hand	5111	6	1500	2000	67.4	54	2	27	950	82	27
Venus	8268	1	40	20	0.7	15	0	20	1140	18	1
Bunny	10000	2	1000	500	38.5	30	0	23	2340	1	0



Banana Eight Cylinder Hand Venus Bunny

Figure 5: Datasets used to test our algorithm.

the average normal of its nearest neighbors. This is forced iteratively in a small number of passes over the cloud and will typically eliminate all such noise. Note that this simple procedure can combat only a very small amount of random noise and is far from sufficient on its own to consistently orient close to half of the original input set of randomly oriented normals.

5 Experimental results

We have implemented the algorithm described in this paper in non-optimized MATLAB code and run it on a number of representative inputs using a Windows 11th Gen Intel i7 machine containing 4 cores with 32 GB RAM. We have compared the orientations that it produces and the corresponding runtimes with the state-of-the-art GCNO algorithm of Xu et al. [18], which is compiled C++ code solving a global non-linear optimization problem, kindly published by the authors at https://github.com/Xrvitd/GCNO.

We feed our algorithm unoriented normals which are obtained from MATLAB's *pcnormals* routine for point clouds, which essentially computes them from the tangent plane derived from the nearest neighbors of each point. Our algorithm takes as parameters the number of cut-planes (alternating along the x, y, z directions), the number of centres per cut-plane for generating pairs of non-homogeneous equations (9) and (10), and the number of centres for generating the sparse homogeneous equations (5). Cut-planes with the same orientation are equally spaced (in parallel) throughout the bounding box of the point cloud. Cut-planes yielding a very small number of contour points (e.g., when they pass between model components or just graze the surface) are discarded.

For each run, we counted the number of incorrectly oriented normals generated by the algorithm, as compared to the ground truth, and the number of such normals after the noise filtering phase. The ground truth was computed from the original triangle mesh version of the data set, where the triangles are consistently oriented and a vertex normal is defined as the average normal of the faces incident on the vertex. A normal is considered oriented correctly, if and only if the angle between it and the ground truth normal is less than 90 degrees, or, equivalently, their scalar product is positive. In practice, approximately half of the initial (locally-computed) normals are correctly oriented.

The main computational bottleneck of the algorithm is the solution of the linear system. We used MATLAB's iterative *lsqr* routine, which is an adaptation of the conjugate gradient method, as this is considered the most efficient when the system is sparse. Iteration is terminated at the earliest among achieving relative tolerance of 10^{-6} or 2500 iterations.

Figure 5 shows renderings of the models we experimented with. These are originally triangle meshes (from which the ground truth normal orientation were computed), but only the point set was used as input to the normal orientation algorithm.

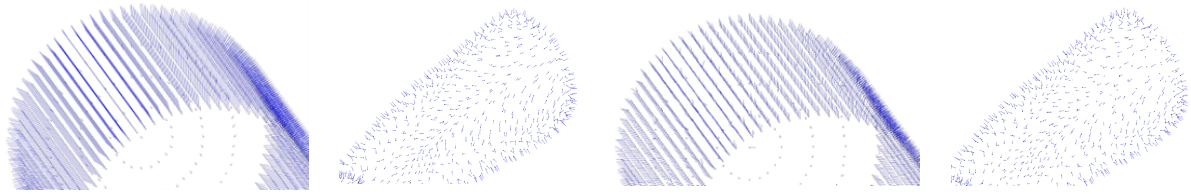


Figure 6: (Left) Output of our algorithm vs. (right) output of GCNO on the Cylinder and the Bunny model. Note how, while many of the normals computed by GCNO point outwards from the Cylinder surface (so are correctly aligned with the ground truth), they have still not converged to their correct directions.

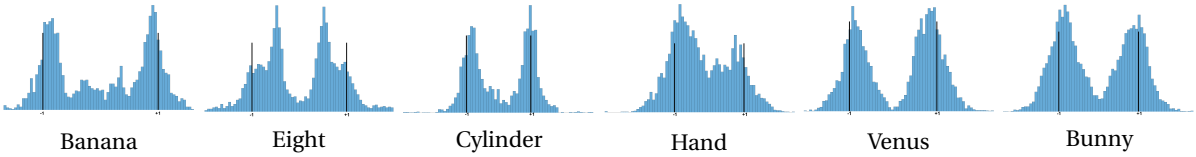


Figure 7: Distributions of the values in the solution vectors on the inputs of Figure 5. Note the good concentration around the desired values ± 1 (marked as black lines).

Table 1 shows the results of running our algorithm and the GCNO algorithm on these inputs. GCNO employs an iterative optimization solver which is initialized with random normals and terminates after a max number of iterations or a tolerance on the gradient. For a fair comparison between the two methods, we monitored the progress of GCNO and stopped it when the output seemed to be close to the ground truth, and then ran that output through the same post-processing noise filtering routine we used on our outputs. The table shows that our algorithm obtains almost perfect results, yet the solver runs in orders of magnitude less time than that of the GNCO solver, especially on the easier models. For example, the Cylinder model, which contains 4802 points, is relatively easy, the only difficult part being the creases around the two bases. Our algorithm correctly orients the locally-computed normals of this model in 0.3 seconds, whereas GNCO requires 660 seconds to compute its output. As evident in Figure 6, while all but two of the resulting GCNO normals point outwards from the Cylinder, many of them still have not converged to the correct direction (radial to the Cylinder's main axis). This is true also for portions of the Bunny model, most notably the ear, which is challenging due to the presence of high curvature.

Figure 7 shows the distribution of the entries of our solution vector s. Note the good clustering around the desired values of ± 1.

5.1 The correct number of equations

It is not that obvious how to determine *a priori* the correct number of equations of types (5), (9), and (10) required beyond the $3N$ sparse regularization equations due to the point cloud Laplacian. Certainly it does not hurt to use more than what is needed, although this could be wasteful in runtime. To illustrate this tradeoff, we ran our algorithm with different numbers of equations on the 10000 point Bunny model with locally estimated normals, as summarized in Table 2. We started with one cut-plane with 500 non-homogeneous equations and 250 homogeneous equations. This did not orient all the normals correctly, and increasing the number of equations with this single cut-plane did not seem to make much difference, while increasing runtime. A difference was made when we added another cut-plane, resulting in a correct solution with a solve time of 38.5 seconds. Certainly adding more equations to this also yielded perfect results, albeit at an unnecessarily longer runtime. We note that the base 30000 regularization equations are extremely sparse, thus have minimal effect on the runtime. Most of the runtime is due to the dense non-homogeneous equations.

Some of the difficulty in orienting the normals correctly is due to the noise in the local estimates of the normals and the areas associated with the points, especially in poorly sampled regions of the model. To gauge this sensitivity, we ran our algorithm on the Bunny, when using as input randomly oriented ground truth normals (computed from the triangle mesh). As expected, the algorithm performed better on this input, correctly orienting these normals in 15.1 seconds at our first attempt with just one cut-plane with 500 non-homogeneous equations and 250 homogeneous equations. These results are also documented in Table 2.

Table 2: Experimental results of running our algorithm on the Bunny dataset using different numbers of equations. Normals are either estimated locally or the ground truth. The bold values mark the configuration used for the Bunny in Table 1.

				locally estimated normals			ground truth normals		
cut-planes	non-homogeneous equations per cut-plane	homogeneous equations	total number of equations (excl. Laplacian)	solve runtime (secs)	incorrect orientations after solve	incorrect orientations after filter	solve runtime (secs)	incorrect orientations after solve	incorrect orientations after filter
1	500	250	750	17.2	58	10	15.1	22	0
1	1000	500	1500	37.1	49	8	35.0	14	0
1	1000	1000	2000	38.3	50	7	36.5	10	0
1	2000	1000	3000	43.5	35	6	40.0	0	0
2	**1000**	**500**	**2500**	**38.5**	**30**	**0**	33.7	0	0
2	2000	1000	5000	45.6	25	0	40.6	0	0
2	2000	5000	9000	64.7	7	0	58.9	0	0

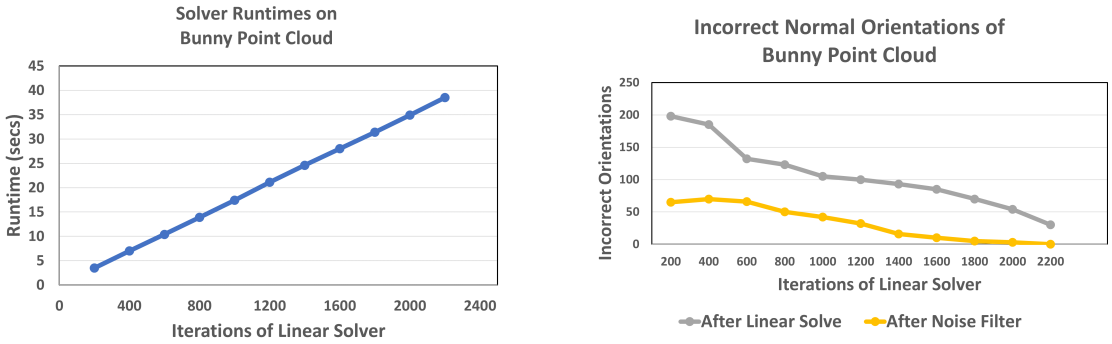


Figure 8: The effect of limiting the number of *lsqr* linear solver iterations on the Bunny point cloud: (left) solver runtimes as a function of the number of iterations; (right) number of incorrect normal orientations after the solve and after the subsequent noise filter as a function of the same.

5.2 Solver accuracy

Another way to reduce runtime, but without reducing the number of equations, is to limit the number of iterations performed by the linear solver. Since the solution is anyway "snapped" to its sign, it is probably not that important to seek a very accurate solution to the equations. Our implementation limits the solver to 2500 iterations (and in most cases it stops earlier due to the alternate tolerance termination parameter), but this could presumably be reduced to less without causing much damage.

Figure 8 shows the effect of limiting the linear solver to a number of iterations ranging between 200 and 2200 on the Bunny point cloud, both in terms of the solver runtime and the number of incorrect normal orientations after the solve and after the subsequent noise filter. While it requires 2200 solver iterations (38.5 seconds) to perfectly orient the 10000 normals of this input, the number of incorrect orientations is a mere 16 after 1400 iterations (24.6 seconds), which in most applications can absolutely be tolerated.

5.3 Sensitivity to noise

To measure our algorithm's sensitivity to noise, we ran the same datasets after adding random noise of 1% of the unit cube bounding box to each of the point cloud coordinates. This revealed reasonably robust behaviour. For example, on the noisy Cylinder model, all normals were correctly oriented using one cut-plane with 50 centres (100 non-homogeneous equations) and 50 homogeneous equations, running in 0.6 seconds. This compares to one cut-plane with 10 centres and no homogeneous equations, running in 0.4 seconds, for the clean model. On the noisy Venus model, all normals were correctly oriented using three cut-planes with 20 centres (40 non-homogeneous equations per cut-plane) and 20 homogeneous equations, running in 0.9 seconds. This compares to one cut-plane with 20 centres and 20 homogeneous equations, running in 0.7 seconds, for the clean input.

5.4 Surface reconstruction

Some works on normal orientation have demonstrated the quality of their results by performing surface reconstruction from the point cloud and the oriented normals that they have computed, typically using a variant of the Poisson reconstruction algorithm [7, 8]. A high quality reconstruction is considered validation of the normal computation and orientation output. We believe that, while surface reconstruction is one way to evaluate the normal generation routine, it is somewhat subjective, and prefer, as we have done, to just examine the raw output, as it may be used in more than just surface reconstruction in downstream processing.

6 Discussion

We have described an algorithm for consistently orienting a given set of normals of a 3D point cloud. Our main contribution is to formulate a set of linear equations for the orientation signs. Although we solve for "relaxed" continuous values, snapping the values to their signs yields very good results. The number of equations required to generate a reliable and robust solution seems to be, as expected, dependent mostly on the shape "complexity", namely, the number of protrusions, corners, and edges, and the sampling density and regularity. It is much less dependent on the total number of samples. In easy cases, a single cut-plane suffices. For more complex shapes, multiple cut-planes per each of the three axes are required to capture the shape.

We use MATLAB's built-in linear least squares *lsqr* iterative solver in our implementation, as this is efficient for sparse linear systems such as those we generate. This solver is limited, though, to linear systems of up to 50000 equations or so. In a more efficient implementation, it should be possible to employ much faster linear solvers, which are a well-understood "commodity" in the scientific computing community, especially for sparse systems and parallel computing environments, resulting in a very fast procedure, even for extremely large datasets.

We solve for N orientation signs for a set of given normal vectors of a point cloud, the latter typically coming from a standard local procedure. It may be possible to go further than we do, using the same linear machinery to solve for the normals themselves, as opposed to just their signs. As with the GCNO algorithm, this would allow to compute normals from scratch for a given set of 3D points: instead of solving for just N signs, we would solve for $3N$ values, the three components of each of the N normal vectors. Note that these are three independent values per normal, since the length of the normal would encode the area A_i associated with the i-th point. However, we do not believe the quality of normals computed in a global manner this way would be better than those computed locally and then consistently oriented, which, as we have demonstrated, is feasible and gives excellent results.

Lastly, we note that our method can be applied also to normals of 3D point clouds sampled from manifolds which are not closed, namely have inherent boundaries, as Stokes' theorem applies to this scenario as well, as long as the boundaries and their correct orientations are given or can be detected automatically in the point cloud. In this case, the equations must take these boundaries into account as an additional contour integral in the right-hand side of the equations, similar to (9) and (10), thus all equations become non-homogeneous. It is also best that cut-planes not intersect these boundaries, so that the boundaries impact the right-hand side of only (9) or (10).

References

[1] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Computer Aided Geometric Design*, 22(2):121–146, Feb. 2005.

[2] S. J. Colley. *Vector Calculus*. Pearson, Boston, 4th edition, 2012. ISBN 978-0-321-78065-2.

[3] C. Duan, S. Chen, and J. Kovacevic. Weighted multi-projection: 3D point cloud denoising with tangent planes. In D. Florencio, A. Reibman, and L. Swindlehurst, editors, *Proceedings of the 6th Global Conference on Signal and Information Processing*, GlobalSIP 2018, pages 725–729. IEEE, Piscataway, Nov. 2018.

[4] L. C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, 2nd edition, 2010. ISBN 978-1-4704-6942-9.

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *ACM SIGGRAPH Computer Graphics*, 26(2):71–78, July 1992.

[6] J. Jakob, C. Buchenau, and M. Guthe. Parallel globally consistent normal orientation of raw unorganized point clouds. *Computer Graphics Forum*, 38(5):163–173, Aug. 2019.

[7] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In A. Sheffer and K. Polthier, editors, *Proceedings of the 4th Symposium on Geometry Processing*, SGP '06, pages 61–70. Eurographics, Aire-la-Ville, June 2006.

[8] M. Kazhdan and H. Hoppe. Screened Poisson surface reconstruction. *ACM Transactions on Graphics*, 32(3), July 2013.

[9] S. König and S. Gumhold. Consistent propagation of normal orientations in point clouds. In M. Magnor, B. Rosenhahn, and H. Theisel, editors, *Proceedings of Vision, Modeling, and Visualization 2009*, VMV '09, pages 83–92. Otto-von-Guericke-Universität, Magdeburg, Nov. 2009.

[10] S. Lin, D. Xiao, Z. Shi, and B. Wang. Surface reconstruction from point clouds without normals by parametrizing the Gauss formula. *ACM Transactions on Graphics*, 42(2), Apr. 2023.

[11] W. Lu, Z. Shi, J. Sun, and B. Wang. Surface reconstruction based on the modified Gauss formula. *ACM Transactions on Graphics*, 38(1), Feb. 2019.

[12] J. Lundgren. TSPSEARCH — Heuristic method for Traveling Salesman Problem (TSP). MATLAB Central File Exchange, Apr. 2019. [Online; accessed 29-November-2023].

[13] G. Metzer, R. Hanocka, D. Zorin, R. Giryes, D. Panozzo, and D. Cohen-Or. Orienting point clouds with dipole propagation. *ACM Transactions on Graphics*, 40(4), Aug. 2021.

[14] N. J. Mitra, A. Nguyen, and L. Guibas. Estimating surface normals in noisy point cloud data. *International Journal of Computational Geometry & Applications*, 14(4 & 5):261–276, Oct. 2004.

[15] P. J. Olver. Conservation laws and null divergences. *Mathematical Proceedings of the Cambridge Philosophical Society*, 94(3):529–540, Nov. 1983.

[16] J. Sanchez, F. Denis, D. Coeurjolly, F. Dupont, L. Trassoudaine, and P. Checchin. Robust normal vector estimation in 3d point clouds through iterative principal component analysis. *ISPRS Journal of Photogrammetry and Remote Sensing*, 163:18–35, May 2020.

[17] N. Schertler, B. Savchynskyy, and S. Gumhold. Towards globally optimal normal orientations for large point clouds. *Computer Graphics Forum*, 36(1):197–208, Jan. 2017.

[18] R. Xu, Z. Dou, N. Wang, S. Xin, S. Chen, M. Jiang, X. Guo, W. Wang, and C. Tu. Globally consistent normal orientation for point clouds by regularizing the winding-number field. *ACM Transactions on Graphics*, 42(4), July 2023. The GCNO source code is available at https://github.com/Xrvitd/GCNO.