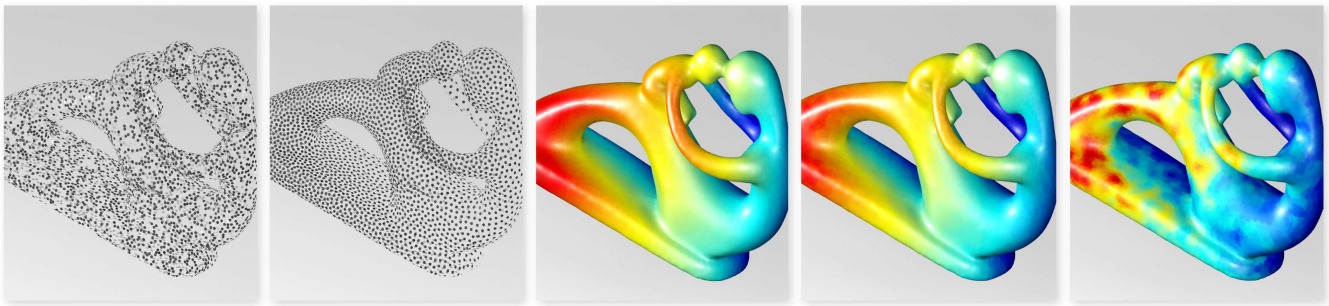
Uniformization and density adaptation for point cloud data via graph Laplacian

Chuanjiang Luo Xiaoyin Ge Yusu Wang

The Ohio State University



(a) input (b) our result (c) our output density (d) ground truth density (e) PCD-Lloyd's density

Figure 1: *Given an arbitrarily distributed input point set sampled from a hidden surface, our method can produce an output point set with both good local uniformity and a global density distribution precisely adapted to a prescribed target density function. (a) shows our input point set. (b) shows our result adapted to a prescribed target density distribution as shown in (d). The density distribution obtained by our output is shown in (c). (e) shows the density distribution obtained by a PCD-Lloyd's relaxation method but with an input that is already very close to the target distribution (see Section 3 for details). Both results are generated with a 5K sparse point set, and a 50K dense point set (representing the hidden surface).*

Abstract

Point clouds data is one of the most common types of input for geometric processing applications. In this paper we study the point cloud density adaptation problem that underlies many preprocessing tasks of points data. Specifically, given a (sparse) set of points Q sampling an unknown surface and a target density function, the goal is to adapt Q to match the target distribution. We propose a simple and robust framework that is effective at achieving both local uniformity and precise global density distribution control. Our approach relies on the Gaussian-weighted graph Laplacian and works purely in the points setting. While it is well-known that graph Laplacian is related to mean-curvature flow and thus has denoising ability, our algorithm uses certain information encoded in the graph Laplacian that is orthogonal to the mean-curvature flow. Furthermore, by leveraging the natural scale parameter contained in the Gaussian kernel and combining it with a simulated annealing idea, our algorithm moves points in a multi-scale manner. The resulting algorithm does not require the input points to have a good initial distribution (neither uniform nor close to the target density distribution), which is required by most previous refinement-based methods. We demonstrate the simplicity and effectiveness of our algorithm with point clouds sampled from different underlying surfaces with various geometric and topological properties.

Keywords: point-based geometry, re-sampling, density control, uniformity, graph Laplacian

1 Introduction

Point cloud data is one of the most common types of input data for geometric processing applications. With the rapid advancement of data acquisition technology, large amount of point cloud data are now routinely generated to represent geometric objects. In recent years, there has been significant interest on performing geometric modeling and processing tasks directly from point-based representations. However, the raw point clouds generated by a 3D scanner are usually noisy, over-dense and/or have non-uniform distribution. Therefore, it is often desirable to simplify, smooth, or resample these point clouds to improve their quality before feeding them to subsequent geometric procession tasks (e.g., surface reconstruction, point cloud based rendering and feature detection). Among these aforementioned processes, point cloud uniformization and downsampling have been well studied in the past decades, e.g., [Lipman et al. 2007; Balzer et al. 2009; Öztireli et al. 2010; Chen et al. 2013]. However, starting with a noisy and poorly distributed point set, how to make its distribution conform to a user defined target density distribution is still an open question.

In this paper, we study this *density adaptation* problem that underlies many pre-processing tasks of a point cloud data. Given a set of points Q (usually sparse) and P (usually denser) both sampled from an unknown surface M, and a target density function $\mathrm{g} : \mathrm{M} \to \mathbb{R}$, the goal is to move Q to a new set of points $\widehat{Q}$ that matches the target density function g on M, where the surface M is represented by the denser point set P. We propose a simple and effective framework for this problem. Our approach leverages the behavior of the Gaussian-weighted graph Laplacian, and by combining it with a simulated annealing idea, it works in a multi-scale manner. It does not require the point set Q to be a subset of P, and the distribution of Q can be very different from the target distribution g.

Related work. Given a set of points, projecting them onto an implicitly defined surface (i.e., moving least square) is a widely used approach to obtain point-set surface representations; e.g, [Levin 2004; Alexa et al. 2003; Amenta and Kil 2004]. Various further processing to "consolidate" ([Alexa et al. 2003]) these points are proposed, including the simplification [Pauly et al. 2002], anisotropic

smoothing [Lange and Polthier 2005] and geometry-aware resampling [Öztireli et al. 2010; Chen et al. 2013]. Recently, [Lipman et al. 2007] introduced the Locally Optimal Projection (LOP) operator that moves a set of points Q to obtain a nice distribution for the hidden surface approximated by another set of input points P via "repelling-force". An improved weighted-LOP operator was proposed in [Huang et al. 2009]. This, combined with a novel normal estimation and propagation algorithm, gives rise to a robust framework that can output a set of clean and evenly distributed points associated with accurate normal information.

Except for [Lipman et al. 2007; Huang et al. 2009], which targets an even (uniform and well-separated) distribution, the work aforementioned typically focus on denoising, and there is no clear requirement nor control over the density distributions of the output points. In this paper, we consider an orthogonal problem that aims at producing a set of points adapted to an arbitrarily user-defined target distribution.

Another category of density adaption methods were developed with the assumption that the underlying *surface mesh* is known [Aurenhammer 1987; Du et al. 2002; Balzer et al. 2009; Li et al. 2010]. These approaches often leverage a Voronoi-like structure to help enforce local uniformity, and then use a Lloyd's relaxation type process [Lloyd 1982] to iteratively adjust point locations till they converge. In particular, adaptive sampling to match a target density on surfaces can be achieved by using weighted Voronoi diagram or constraint centroidal Voronoi tessellation (CCVT) [Aurenhammer 1987; Du et al. 2002; Balzer et al. 2009; Li et al. 2010]. While a good local uniformity can be achieved, the adjustments of points location in these methods tend to be local. Hence they often require that the input point set has a good initial distribution already close to the target density. Our method has no such requirement, and can be applied for arbitrary point set input.

Our contributions We present a simple and effective density adaptation algorithm for points data. The main contributions are as follows. We believe that the simplicity and effectiveness of our algorithm will help advocate its practical usage as a preprocessing step to improve points quality.

1. Our algorithm can work in a pure points data setting and requires only the construction of the (Gaussian-weighted) graph Laplace operator, therefore is easy to implement. We make the connection between our approach with the standard Kernel Density Estimation (KDE), and this connection itself is of independent interest.
2. By leveraging some properties of the graph Laplacian as well as a simulated annealing idea, our algorithm can achieve both good local uniformity / well-separateness among points, and accurate output density control. It does not require input points to be in a good initial distribution close to the target distribution (see Figure 1).
3. Our algorithm is simple, effective, and robust to noise, small holes as well as boundaries present in input points.

2 Density Adaptation Algorithm

Assume we have an underlying surface M embedded in $\mathbb{R}^3$. Our input is (i) a set of n points $P = \{p_1, \ldots, p_n\}$ sampled, not necessarily uniformly, from on and around M; (ii) another set of m points Q, and (iii) a target density function $\mathsf{g} : \mathrm{M} \to \mathbb{R}^+$ (the form that g is given is discussed in Section 2.3). The density adaptation problem aims to move Q to a new set of points $\widehat{Q}$ that matches the target density function g. If the target g is the uniform distribution, we also call this process *the uniformization of* Q. The role of P is to serve as a point-set approximation of the underlying surface M: Its distribution can be very different from the target distribution. Furthermore, Q may or may not be a subset of P (it is also possible that $Q = P$), and can be far sparser than P.

In Section 2.1 we describe how to move Q to obtain a uniformly distributed set of points $\widehat{Q}$, to illustrate the main ideas and components. We then explain in Section 2.3 how to achieve a non-uniform distribution.

Gaussian-weighted graph Laplacian. Given a set of points $Q = \{q_1, \ldots, q_m\}$ sampled from a surface (2-manifold) M, the Gaussian-weighted (GW) graph Laplace operator L_t^Q is a linear operator such that given any function $f : \mathrm{M} \to \mathbb{R}$ with values available at points in Q, we have:

$$\mathrm{L}_t^Q f(q_i) = \frac{1}{mt^2} \sum_{j=1}^{m} e^{-\frac{\|q_i - q_j\|^2}{t}} (f(q_i) - f(q_j)). \qquad (1)$$

Although the above formulation involves all pairs of points q_i and q_j, due to the exponential decaying effect of the Gaussian kernel, we only need to consider those q_j within $O(\sqrt{t})$ distance to q_i when estimating Eqn (1). Hence in practice, the construction of the GW-Laplace operator L_t^Q requires only the proximity graph of Q.

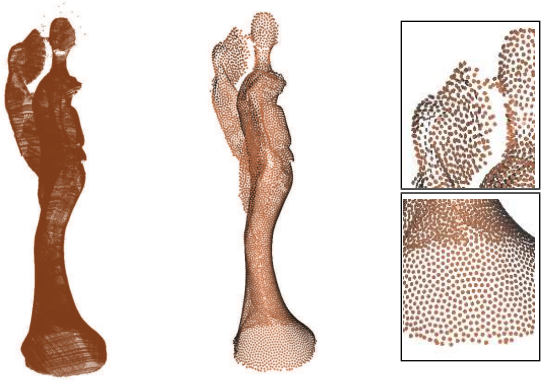


Figure 2: *Left: input 170K raw scan data; Middle: 10K downsampled and then uniformized; Right: zoom-in details of boundaries.*

There has been much theoretical study of this graph Laplacian and its variants. In the limit as the number of points m tends to ∞ and the parameter t tends to 0 at appropriate rates, and if the points Q are uniformly sampled from the manifold M, then L_t^Q converges to the Laplace-Beltrami (also called the manifold Laplacian) $\triangle_{\mathrm{M}}$ for M [Belkin 2003]. In the case when Q is not sampled from a uniform distribution, but rather, from a distribution $\mathsf{p} : M \to \mathbb{R}^+$, it turns out that [Hein et al. 2007], for any function $f : \mathrm{M} \to \mathbb{R}$:

$$\mathrm{L}_t^Q f(x) \rightsquigarrow \frac{\pi}{4} \mathsf{p}(x) \triangle_{\mathrm{M}} + \frac{\pi}{2} \langle \nabla \mathsf{p}(x), \nabla f(x) \rangle \qquad (2)$$

where ∇ denotes the gradient operator.

2.1 Uniformization

Consider the coordinate functions X, Y, and Z, which returns the $\mathbf{x}$-, $\mathbf{y}$-, and $\mathbf{z}$-coordinate for a point $x \in \mathbb{R}^3$, respectively. For any vector $\vec{\mathbf{v}}$, note that $\langle \vec{\mathbf{v}}, \nabla \mathsf{X}(x) \rangle = \vec{\mathbf{v}}.\mathbf{x}$ (i.e, the $\mathbf{x}$-component of vector $\vec{\mathbf{v}}$). Similarly, $\langle \vec{\mathbf{v}}, \nabla \mathsf{Y}(x) \rangle = \vec{\mathbf{v}}.\mathbf{y}$, and $\langle \vec{\mathbf{v}}, \nabla \mathsf{Z}(x) \rangle = \vec{\mathbf{v}}.\mathbf{z}$. Let $\mathsf{p} : \mathrm{M} \to \mathbb{R}$ denote the density function that the current set of points Q is sampled from. Now apply L_t^Q to the three coordinate functions $\overrightarrow{\mathsf{P}} = [\mathsf{X}\ \mathsf{Y}\ \mathsf{Z}]^t$. By Eqn (2) we then derive:

$$\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(x) \rightsquigarrow \frac{\pi}{4} \mathsf{p}(x) \triangle_{\mathrm{M}} \overrightarrow{\mathsf{P}}(x) + \frac{\pi}{2} \nabla \mathsf{p}(x). \qquad (3)$$

It is known that $\triangle_{\mathrm{M}} \overrightarrow{\mathsf{P}}(x)$ specifies the so-called mean-curvature flow: This vector lies in the normal direction $\vec{\mathbf{n}}(x)$ of M at x, and its magnitude is the mean-curvature of M at x. This motivates the Laplacian-based iterative denoising approaches for surfaces [Desbrun et al. 1999; Lange and Polthier 2005], by repeatedly projecting a point in the direction of $\triangle_{\mathrm{M}} \overrightarrow{\mathsf{P}}(x)$, and is also behind the mean-shift types approaches for high-dimensional data analysis.

We instead focus on the second term $\nabla \mathsf{p}(x)$, which has not been used much so far in applications. Being the gradient of a function on M, $\nabla \mathsf{p}(x)$ lies in the tangent plane $T_{\mathrm{M}}(x)$ of M at x, and is thus orthogonal to the first term. In other words, $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(x)$ can be written as two orthogonal terms as in Eqn (3), with the first one being the projection in the normal direction, and the second one being the projection in the tangent space. If Q is uniformly distributed, then the corresponding density function p is a constant function, implying $\nabla \mathsf{p}(x) = 0$ for all $x \in \mathrm{M}$. Hence to uniformize Q, we aim to nullify $\nabla \mathsf{p}(x)$. Specifically, our algorithm, shown in Algorithm 1, will repeatedly move points in Q in the direction of the tangent projection of $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(x)$ (which approximates $\nabla \mathsf{p}(x)$) till this projection becomes zero for all points in Q – at which moment, Q reaches a uniform distribution. (Another interpretation of this tangent component via kernel density estimator is given in the Appendix A.)

input : Two sets of points P and Q
output: New point set Q forming a uniform distribution of the underlying surface M approximated by P
begin
  for $(t = 256t_0; t \geq 0.5t_0; t = t/2)$
    for $(k = 0; k < \mathrm{IterNum}; k$ ++)
      Step 1: Compute the tangent projection $\vec{\mathsf{V}}_{\|}(Q)$ of $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(Q)$ for all points in Q ;
      Step 2: Move Q in direction $\vec{\mathsf{V}}_{\|}(Q)$ by step-size $\mu(k)$;
      Step 3: Project Q to the underlying surface approximated by P ;
    end for
  end for
end

Algorithm 1: Uniformization algorithm

Before we explain the algorithm, we remark that as mentioned earlier, the role of the point set P is to provide an approximation of the underlying surface. In our algorithm, we only use P to estimate tangent spaces. In particular, given a point q, we approximate the tangent plane $T_M(q)$ at q as follows: let $p \in P$ be the nearest neighbor of q in P. $T_M(q)$ is taken as the best plane fitting the k-nearest neighbors of p in P in the least square sense. The default value of k is 15. We also note that there is a parameter t in the Gaussian kernel of L_t^Q: Intuitively, only points within a region of radius $\Theta(\sqrt{t})$ around q will influence $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$. The parameter t provides a natural way to decide the scale-level, and as we will see, other parameters in our algorithm often depend on this "neighborhood size" $\sqrt{t}$.

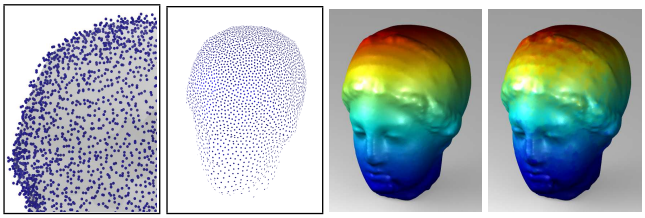
We now describe the main steps of Algorithm 1.

(Step 1): We compute $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$ at each point $q \in Q$. Let $\vec{\mathsf{V}}_{\|}(q)$ denote the projection of $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$ on the estimated tangent plane $T_{\mathrm{M}}(q)$. In Algorithm 1, we use $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(Q)$ (resp. $\vec{\mathsf{V}}_{\|}(Q)$) to represent the set of $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$ (resp. the set of $\vec{\mathsf{V}}_{\|}(q)$) for all $q \in Q$.

(Step 2): To nullify the tangent component, we move q in the direction of $\vec{\mathsf{V}}_{\|}(q)$. Intuitively, $\vec{\mathsf{V}}_{\|}(q)$ approximates the gradient $\nabla \mathsf{p}(q)$ of the current density function p, and is pointing from a denser to a sparser region of Q. By moving points in this direction, we spread them away from denser regions. The displacement vector $\vec{\mathsf{V}}_{\|}(q)$ is zero *for all* $q \in Q$ only when Q achieves a uniform distribution.

Specifically, we move q to $\tilde{q} = q + \mu(k)\vec{\mathsf{V}}_{\|}(q)$, where $\mu(k)$ is a step-size used in the k-th iteration of the inner **for**-loop. We now discuss how to set this step-size. In a sense, we are performing a gradient-descent type of approach to minimize the difference between the current density and the target density. We use the following *non-adaptive annealing* strategy to balance the simplicity of our algorithm and to avoid getting stuck in local minima [Robbins and Monro 1951]. We set $\mu(k) = \mu(0)/(1 + \lfloor \frac{k}{T} \rfloor)$, where the initial step size $\mu(0)$ is $\sqrt{t}/2$ where t is the parameter used in the graph Laplacian L_t^Q. The step size is constant for the first T iterations, allowing points to find the general location in that scale, before annealing (decreasing) it at a slow pace. T is a parameter that needs to be empirically set. In all our experiments, we set $T = 2$, which provides a good tradeoff for both speed and accuracy.

(Step 3): After q is moved to $\tilde{q} = q + \mu(k)\vec{\mathsf{V}}_{\|}(q)$, we need to project $\tilde{q}$ back to the underlying surface. This is achieved by simply projecting $\tilde{q}$ to the estimated tangent space $T_{\mathrm{M}}(q)$. This step contains a least-square fitting (to approximate the tangent plane) and a projection, which makes our uniformization algorithm robust w.r.t. reasonable amount of noise. See Figure 3 for an example.



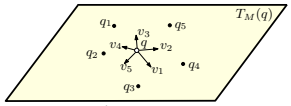
(a) noisy input (b) our output (c) target density (d) our density

Figure 3: *(a): Input are 5K points Q over 50K dense points P: uniform noise of magnitude 1.5% of the diameter is added at each point. (b): Output points adapted to the target density function shown in (c). Note that these points are also smoothed by our algorithm. (d): Density map of our output points. The correlation of output density and target density is 0.99 and L_2-error is 0.015.*

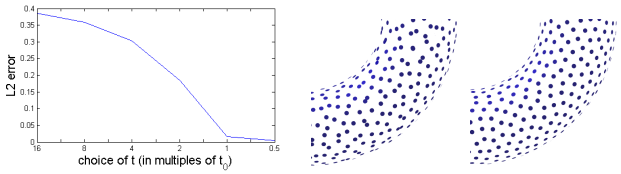
Choice of t. As discussed earlier, the parameter t involved in $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$ roughly specifies the neighborhood size (scale) we consider around each point q. At the beginning when the distribution of Q is still far from the target distribution, we use a large t value so that the displacement of each q is guided by the global distribution of points in Q at this point. As the distribution of Q becomes closer to the target, we use smaller and smaller t to perform local refinement. Specifically, we adjust t as follows. We start with $t = 256t_0$, where t_0 is the square of the average distance between a point to its nearest neighbor in the input point set Q. We perform $\mathrm{IterNum} = 24$ iterations of the non-adaptive annealing step (inner **for**-loop in Algorithm 1). We then reduce t to its half and repeat until $t = 0.5t_0$. We observe that this strategy, combined with the simple non-adaptive annealing approach described above, is both efficient and effective at recovering from local minima even when the initial distribution is far from the target. An example is shown in Figure 1 and will be explained in more detail in Section 3.

Well-separateness. Interestingly, it turns out that by reducing t to be smaller than t_0, we can obtain *well-separateness* among points where points are even spaced with no points get too close. Intuitively, when t is smaller than t_0, only the immediate neighbors of q will contribute when computing $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$ (analogous to one-ring neighbors of q in a mesh); and these points need to form a somewhat regular pattern around q for $\vec{\mathsf{V}}_{\|}(q)$ to be zero.

See the right figure for an illustration where all points are already projected onto the tangent plane $T_M(q)$: For small t, only the immediate neighbors $q_1, \ldots, q_5$ of q affect $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$. By Eqn (1), each q_i will contribute a vector v_i which is some scaling of $q - q_i$, and $\vec{\mathsf{V}}_{\|}(q) \approx \sum_{i=1}^{5} v_i$. Intuitively, an evenly distributed Q tends to form to ensure that this sum is zero for all points q. See Figure 4 for the effect of reducing t. In particular, note that we already achieve a good uniform distribution when $t = t_0$ (as reflected by the low L_2-error between the output density and uniform distribution). However, by further reducing t to be $0.5t_0$, while L_2 error of the output density distribution changes little, we can see in Figure 4 (b) and (c) that the points become more evenly spaced (well-separated).



Initial distribution. The two control variables, t in the graph Laplace operator and the step size $\mu(k)$ differentiate our algorithm from Lloyd's relaxation type local refinement methods (description of some variants can be found in the Supplement). Both t and $\mu(k)$ play a role analogous to the "temperature" in the simulated annealing method. At the beginning we will use a higher temperature which allows us to use more global information and move faster. Gradually the temperature decreases, allowing us to use more local information and move slower for local refinement. This is the reason our method can achieve an accurate target density even when the input points distribution is far from the target. See Figure 1 and Section 3 for examples and for comparison.



(a) L_2 error v.s. choice of t (b) $t = t_0$ (c) $t = 0.5t_0$

Figure 4: *(a) plots the L_2-error of the density of intermediate point set Q as the parameter t decreases in Algorithm 1. The points obtained when $t = t_0$ and $t = 0.5t_0$ are shown in (b) and (c): note that at $t = t_0$, while the distribution is already close to uniform (small L_2-error), points are* not *yet well-separated.*

2.2 Boundaries and Holes

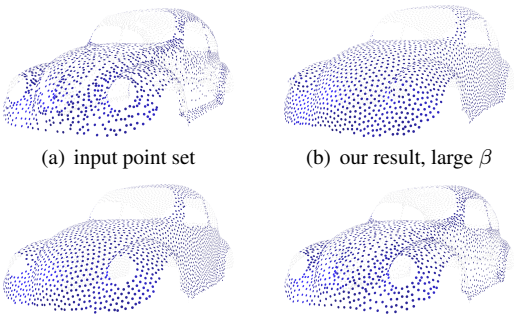
It turns out that L_t^Q behaves in a fundamentally different manner for points around boundaries of a surface M [Belkin et al. 2012]. Specifically, in contrast to Eqn (2), for a point $x \in \partial M$, we now have:

$$\mathrm{L}_t^Q f(x) \rightsquigarrow \frac{1}{\sqrt{t}} \cdot \frac{\pi^{1/2}}{2} \mathsf{p}(x) \partial_{\vec{\mathbf{n}}} f(x) + o(\frac{1}{\sqrt{t}}), \qquad (4)$$

where $\vec{\mathbf{n}}$ is the outward normal at x: namely, $\vec{\mathbf{n}}$ lies in the tangent plane $T_{\mathrm{M}}(x)$ at x and is normal to the boundary curve ∂M. Terms from Eqn (2) are now hidden in the second term of Eqn (4). If we plug in the coordinate functions $\overrightarrow{\mathsf{P}}$ as f, the first term in Eqn (4) becomes $\frac{\pi^{1/2}}{2\sqrt{t}} \mathsf{p}(x) \vec{\mathbf{n}}$, which lies in the tangent plane $T_{\mathrm{M}}(x)$. In other words, for any point $p \in Q$, the tangent projection $\vec{\mathsf{V}}_{\|}(q)$ of $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$ is now dominated by this first term (due to its $\frac{1}{\sqrt{t}}$ scaling factor) instead of by the gradient of the current density function $\nabla \mathsf{p}$. Our algorithm, by moving points in the direction of $\vec{\mathsf{V}}_{\|}(q)$, extends points across the boundary in the tangent plane in the outward normal direction $\vec{\mathbf{n}}$. Such extension can close small holes (missing data): see Figure 5 (b) where the small regions of missing data around the center of the front cover of the car are now filled. See also an example in the Supplement.

If the boundary is large (and thus may be real boundary), then such extending effect is no longer desirable. One may think that a straightforward way to handle this is as follows: For a point q, if its current position is far from its nearest neighbor $\tilde{p}$ in the dense point P, say, $d = \|q - \tilde{p}\| \geq \alpha$ for a threshold α, then we consider it to be too far from the boundary and can just project it back to $\tilde{p}$ (instead of projecting it onto the tangent plane at $\tilde{p}$) . However, this approach has two shortcomings: (1) while holes of size $O(\alpha)$ can be filled, points around boundaries will also be extended by about $O(\alpha)$ distance (since points within $O(\alpha)$ distance to the true boundary will not be projected back); and (2) there tends to be accumulation of points along the boundaries (due to the projection of points back to their nearest neighbors in P), thus points distribution around boundaries are not uniform. We need a strategy that can seal small holes while *not* extending the real boundaries.

To this end, we take the following multi-scale strategy: Instead of using a fixed threshold α to decide whether to project a point back to the boundary or not, we choose a multi-scale threshold $\alpha = 3\sqrt{t}$ depending on the current scale. That is, we project a point q back to its nearest neighbor $\tilde{p} \in P$ only if $d = \|q - \tilde{p}\| \geq 3\sqrt{t}$. Since at any time, there are points within $O(\sqrt{t})$-distance outside the boundary, the true boundary is now seen as interior at the current scale t. Furthermore, since the algorithm projects outside points back to the boundary gradually as the scale t decreases, there is little accumulation formed at the boundaries because the uniformization algorithm (Steps 1–3) can now spread out those points projected back sufficiently fast. Hence our algorithm achieves good distribution even around the boundaries: See Figure 5.



(a) input point set (b) our result, large β

(c) our result, small β (d) PCD-Lloyd's result

Figure 5: *(a) Input points. (b) and (c) show our outputs, with large β small holes are filled while large boundaries are preserved in (b); while with small β the headlights are untouched in (c). (d) shows the output of the PCD-Lloyds method (described in Section 4) which achieves a less uniform distribution. Furthermore, there is no flexibility in controlling whether a hole can be filled or not.*

One last issue is that we do not wish to, as t (thus the scale) decreases, project a point from an already filled hole back to the boundary: this could happen when the radius of an already filled hole is bigger than $3\sqrt{t}$ as t becomes small. This can be addressed by leveraging the behavior of graph Laplacian around boundaries. In particular, according to Eqn (4), $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$ has a significantly larger magnitude at the boundary than in the interior due to the factor $\frac{1}{\sqrt{t}}$. In comparison, a point in the middle of an already filled hole is now an interior point, and thus has a much lower magnitude of $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$ (as its scaling factor is only $O(1)$, see Eqn (2)). So instead of using only the condition $d > 3\sqrt{t}$, we now project a point q back to $\tilde{p}$ if $(d > 3\sqrt{t})$ *and* $(|\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)| > \lambda)$, where λ is twice the

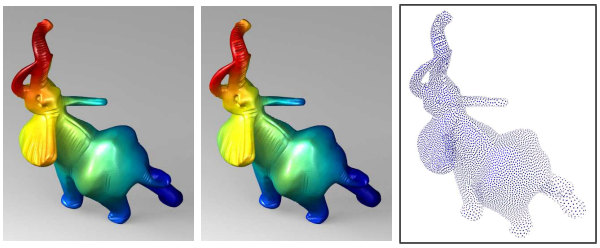
median magnitude of all $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(x)$. By projecting back only points with large magnitude of $\mathrm{L}_t^Q \overrightarrow{\mathsf{P}}(q)$, points from filled holes will not be projected back and a sealed hole will not be broken again as the scale t decreases. Finally, to have a more direct control on the size of the holes sealed, we use a parameter β and project a point q back only if

$$(d > 3\sqrt{t}) \ \textit{and} \ [(|\mathrm{L}_t^Q f(x)| > \lambda) \ \textit{or} \ (d > \beta)].$$

This will leave holes with radius larger than β un-filled. In Figure 5 (b) and (c), by adjusting the only parameter β, we can choose whether to fill the holes for the headlights of the car or not. From Figure 5, we also note that our algorithm achieves a much better uniform distribution than the Lloyd's method (see e.g, the front and the side of the car). Furthermore, it is important to note that the Lloyds-type or iterative CCVT-type methods do not provide control on what can be filled, while our algorithm can provide this flexibility.

We remark that an alternative possible approach for this task could be to first perform certain region growing procedure on the point set P to fill the hole. The graph Laplace L_t^P w.r.t. the dense point set P (instead of Q) can be potentially used to perform region growing. However, this approach, if it works, will be computationally more expensive, as it operates on the dense point set P, while our current approach only operates on the sparse set Q.

2.3 Non-uniform density adaptation



(a) target density (b) our result density (c) our result point set

Figure 6: *(a) Target density and (b) our output density function as achieved by the 10K output points shown in (c).*

The algorithm to obtain a non-uniform target distribution $\mathsf{g} : \mathrm{M} \to \mathbb{R}$ for points in Q follows the same framework as the case of uniformization. The only modification is that, at each iteration, we now wish to nullify $\nabla\mathsf{p}(q) - \nabla\mathsf{g}(q)$ for each point q. Hence instead of moving a point q in the direction of $\vec{\mathsf{V}}_{\|}(q)$, we now move it in the direction of $\frac{2}{\pi}\vec{\mathsf{V}}_{\|}(q) - \nabla\mathsf{g}(q)$ where $\frac{2}{\pi}\vec{\mathsf{V}}_{\|}(q)$ approximates $\nabla\mathsf{p}(q)$. Assume that we can obtain $\mathsf{g}(x)$ for any query point x. To estimate $\nabla\mathsf{g}(q)$ at each $q \in Q$, we locally fit a quadratic function in the tangent plane $T_{\mathrm{M}}(q)$ using points within $\sqrt{t}$ distance to q.

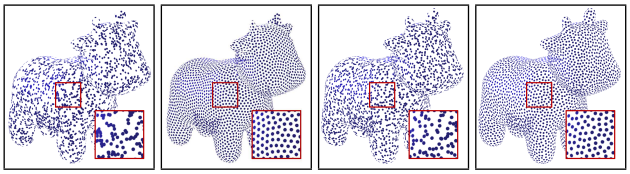
The target density function $\mathsf{g} : \mathrm{M} \to \mathbb{R}$ needs to satisfy the property that $\int_{\mathrm{M}} \mathsf{g}(x)dx = 1$. However, very often, given that M is not known, we are given an empirical target density function $\tilde{\mathsf{g}} : \mathrm{M} \to \mathbb{R}$ which is a scaled version of g; that is, $\tilde{\mathsf{g}}(x) = c \cdot \mathsf{g}(x)$ for some constant c. (For example, $\tilde{\mathsf{g}}(x) = x.\mathbf{x}^2 + x.\mathbf{y}^2 + x.\mathbf{z}^2$.) In such a case, we need to estimate the constant $c = \tilde{\mathsf{g}}/\mathsf{g}$.

The integral $\int_{\mathrm{M}} \mathsf{g}(x)dx$ can be approximated using points in Q by $\sum_{i=1}^m \mathsf{g}(q_i)A_i$, where A_i is the area of the underlying surface M represented by each sample point q_i. Following the integral estimation in [Luo et al. 2009], we estimate A_i as follows: project the local neighbors $Q_i \subset Q$ (points from Q within $\sqrt{t}$ distance from q_i) onto the tangent plane $T_{\mathrm{M}}(q)$. Compute the Voronoi diagram of the projection of these local neighbors Q_i in $T_{\mathrm{M}}(q)$ and take the area of the Voronoi region associated to q_i as A_i. This Voronoi diagram is only computed locally for a small number of points, and is fast in practice. Now we have:

$$\sum_{i=1}^m \mathsf{g}(q_i)A_i = 1 \Rightarrow \frac{1}{c}\sum_{i=1}^m \tilde{\mathsf{g}}(q_i)A_i = 1 \Rightarrow c = \sum_{i=1}^m \tilde{\mathsf{g}}(q_i)A_i.$$

Note that the estimation of c needs to be done only once. Some examples of obtaining non-uniform distribution are in Figure 1, 3, 6, 9, and Table 1.

3 Experiments



(a) (b) (c) (d)

Figure 7: *(a) Input 5K points Q. (b) Our output when $P = Q$. (c) shows the output of [Huang et al. 2009] when P has 10K and (d) shows its output when P has 40K points. (The output of [Huang et al.] when $P = Q$ is almost identical to input points.)*

Several examples of obtaining uniform and non-uniform target distributions can be found in Figure 1, 2, 3, 5, and 6. More results and applications will be shown in this section.

Comparison with weighted LOP. For the uniformization case, we compare our results with the weighted LOP operator approach in [Huang et al. 2009]. We note that a main advantage of our algorithm[1] is that we obtain good results even when the point set P (to approximation the hidden surface) is rather sparse, including when $P = Q$; see Figure 7. The algorithm of [Huang et al. 2009] typically requires that the ratio of $\frac{|P|}{|Q|}$ is large (say ≥ 10). The Lloyd's relaxation style methods (to be described shortly) also requires the presence of a dense point set P and perform slightly worse than the weighted LOP operator, so we omit the results due to limited space.

Comparison with weighted PCD-Lloyds and PCD-CCVT. Furthermore, note that the weighted LOP operator approach cannot adapt Q to an arbitrary density distribution. Hence for the case of non-uniform target distribution, we compare our results with both a weighted Lloyd's relaxation method (weighted Voronoi diagram) adapted for handling point clouds data, and a point-clouds (PCD) version of the Constrained Centroidal Voronoi Tessellation (CCVT). Both of weighted Lloyd's relaxation and CCVT methods have traditionally been developed for meshed surfaces. Here, we adapt them to PCD-versions to handle point clouds data. Lloyd's relaxation is a Voronoi diagram based relaxation method, and can be easily adapted to handle points data by approximating Voronoi cells using points from the dense set P. To handle non-uniform target distribution, we assign input points with weights and use the power diagram to replace the Voronoi diagram [Aurenhammer 1987]. We refer to this method as *the weighted PCD-Lloyd's method* (or PCD-Lloyd for short); see the Supplement for the detailed description. Constrained Centroidal Voronoi Tessellation (CCVT) can be used to produce a sample set with user defined density distribution [Du et al. 2002]. Popular algorithms to obtain CCVT are also relaxation-based processes to iteratively adjust point locations. Here we use a simple PCD adaptation of such iterative

[1] An important part of [Huang et al. 2009] is robust normal estimation and propagation. Here we only focus on its uniformization effect.

Size of Q	data	$\|P\| = 25K$				$\|P\| = 50K$			
		Our Alg.	Init. Input	power diagram	CCVT	Our Alg.	Init. Input	power diagram	CCVT Output
5K	fertility	0.998 / 0.023	0.881 / 0.149	0.888 / 0.145	0.881 / 0.150	0.998 / 0.020	0.908 / 0.167	0.929 / 0.147	0.926 / 0.148
	cow	0.998 / 0.023	0.887 / 0.135	0.895 / 0.130	0.893 / 0.132	0.998 / 0.022	0.920 / 0.113	0.934 / 0.104	0.923 / 0.105
	elephant	0.988 / 0.058	0.932 / 0.115	0.923 / 0.123	0.921 / 0.125	0.990 / 0.044	0.954 / 0.095	0.954 / 0.095	0.944 / 0.105
	kitten	0.999 / 0.016	0.921 / 0.132	0.930 / 0.124	0.927 / 0.126	0.999 / 0.016	0.951 / 0.102	0.963 / 0.089	0.962 / 0.090
	venus	0.999 / 0.013	0.885 / 0.143	0.896 / 0.137	0.895 / 0.138	0.999 / 0.014	0.920 / 0.118	0.933 / 0.109	0.932 / 0.110
10K	fertility	0.998 / 0.025	0.751 / 0.219	0.759 / 0.216	0.752 / 0.220	0.998 / 0.024	0.710 / 0.214	0.768 / 0.188	0.745 / 0.189
	cow	0.998 / 0.024	0.850 / 0.161	0.855 / 0.160	0.852 / 0.162	0.998 / 0.024	0.866 / 0.147	0.881 / 0.139	0.880 / 0.140
	elephant	0.988 / 0.076	0.882 / 0.151	0.876 / 0.155	0.874 / 0.156	0.990 / 0.065	0.922 / 0.122	0.914 / 0.129	0.912 / 0.130
	kitten	0.999 / 0.017	0.890 / 0.152	0.901 / 0.145	0.898 / 0.147	0.999 / 0.017	0.922 / 0.128	0.934 / 0.119	0.932 / 0.121
	venus	0.999 / 0.013	0.826 / 0.172	0.836 / 0.167	0.833 / 0.169	0.999 / 0.013	0.880 / 0.143	0.891 / 0.137	0.889 / 0.138
15K	fertility	0.997 / 0.028	0.647 / 0.297	0.656 / 0.295	0.651 / 0.298	0.998 / 0.025	0.796 / 0.330	0.805 / 0.327	0.803 / 0.328
	cow	0.998 / 0.024	0.712 / 0.218	0.725 / 0.214	0.722 / 0.215	0.998 / 0.022	0.845 / 0.161	0.858 / 0.155	0.856 / 0.156
	elephant	0.983 / 0.093	0.820 / 0.185	0.820 / 0.186	0.817 / 0.187	0.987 / 0.081	0.911 / 0.131	0.904 / 0.136	0.903 / 0.137
	kitten	0.999 / 0.018	0.783 / 0.206	0.796 / 0.202	0.792 / 0.203	0.999 / 0.016	0.896 / 0.148	0.908 / 0.140	0.906 / 0.141
	venus	0.999 / 0.013	0.653 / 0.240	0.663 / 0.237	0.659 / 0.238	0.999 / 0.013	0.845 / 0.163	0.855 / 0.159	0.853 / 0.160

Table 1: *Correlation/L_2 error comparisons. The "Init. Input" is the input fed to the PCD-Lloyds and PCD-CCVT methods, and it is already close to target distribution; while the input to our algorithm is rather far from target distribution, typically has a correlation less than 0.1.*

CCVT framework as described in the Supplement, and we refer to it as the *PCD-CCVT method.*

Both the Lloyd's and relaxation-based CCVT approaches are local methods, where the Voronoi centers/generators are repeatedly updated locally. Hence, while being very simple and effective at local refinement, they are somewhat deficient at making large adjustments to point locations, tend to get stuck in local minima, and require a good input distribution already close to the target distribution. To improve their performance, we first obtain an adaptive subsample from the dense points P using a strategy similar to the importance sampling approaches (adaptive dart throwing) used in blue-noise sampling, to generate a set of points Q that is already close to the target distribution: we feed this Q to both the weighted PCD-Lloyd's and the PCD-CCVT methods.

In Table 1, we compare the accuracy of the output density distribution of our method with both PCD-Lloyd's and PCD-CCVT methods, where we measure the quality of the output points by (i) the statistical correlation of the density function $\tilde{\mathrm{g}}$ of output points with the target density g, and (ii) the L_2-error, which is simply $\frac{\|\mathrm{g}-\tilde{\mathrm{g}}\|}{\|\mathrm{g}\|}$ by treating g and $\tilde{\mathrm{g}}$ as two m-vectors with $m = |Q|$. Our algorithm is able to achieve much more accurate target distribution, especially when the ratio between $\frac{|P|}{|Q|}$ gets smaller. Typically, the improvement of both PCD-Lloyd's and PCD-CCVT methods over its input distribution is not significant, although it does help to improve local regularity / well-separateness of points (which is not reflected in the L_2-error and the correlation). If we feed the same input (as the input to our algorithm) to both methods, their outputs are far from target distributions, and thus these results are not reported.

Another example is shown in Figure 1, which is a hard case since points have to move through the many tunnels (e.g, arms and the connection between heads) to adjust their distribution. The distribution of input points fed to the weighted PCD-Lloyd's method is already close to the target distribution, with a correlation of 0.91 with the target distribution. However, the weighted PCD-Lloyd's method fails to improve the point distribution much, where its output distribution, shown in (e), has a correlation of 0.93 with the target distribution. Our algorithm, on the other hand, takes in as input a quite different distribution (shown in (a)) far from target distribution with a correlation -0.04 with the target distribution, yet still achieving 0.99 correlation in the output distribution (shown in (d)). (If we feed our input to the weighted PCD-Lloyd's method directly, its output has only a correlation of 0.22 with the target distribution.)

We remark however, that the PCD-Lloyd's and PCD-CCVT methods typically converge in fewer iterations than our algorithm. For example, the PCD-Lloyd's typically converges within 30 iterations for most test data in Table 1, while our algorithm will run $24 \times 10 = 240$ (as shown in Algorithm 1) iterations. Our algorithm requires more iterations both because it takes a multi-scale approach to avoid getting stuck in local minimal and because it achieves a much more accurate target distribution.

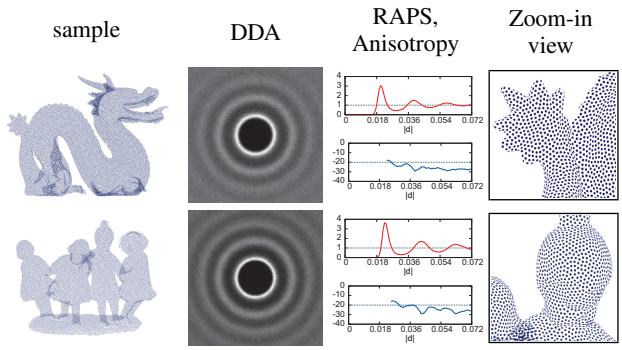
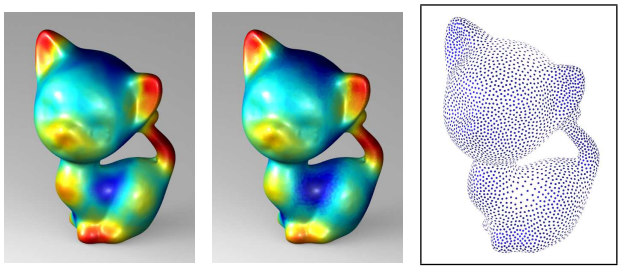


Figure 8: *The DDA power-spectrum, radial-average power spectrum (RAPS) and the anisotropy plot. Each output has 40K points, and the dense point cloud P has 100K points. Reducing the number of points in Q maintains similar DDA power spectrum, although small features in the model start to be smoothed. The RAPS and anisotropy plot are widely used in evaluating point distributions, especially in the context of blue noise sampling. Though we do not claim the blue noise property, we still include them here for completion.*

Well-separateness. In Figure 8, we show the evenness of our output points using the so-called DDA (Differential Domain Analysis) power-spectrum [Wei and Wang 2011]. Intuitively, the DDA spectrum shows the kernel density estimates (smooth histogram) of pairwise distances between sample points. White means high value and black means low value. A white ring of certain radius imply that many pairs of sample points share this pairwise distance value, say r. The black disk in the center implies the absence of the sample pairwise distances which smaller than r. We note that the power-spectrum of our output shows a clear white ring with dark interior, indicating the evenness of the points we produce.

Geometry-aware sampling. In Figure 9, we show an example where our density adaption algorithm is used to obtain a geometry-aware sub-sampling of input points. In particular, our target density function g here is a smoothed version of the mean-curvature. To obtain the $\mathrm{g}(x)$ value at a point q, we simply take the normal projection of the graph Laplacian $\mathrm{L}_t^Q \overrightarrow{\mathrm{P}}(q)$: recall Eqn (3), this normal

projection (the first term in Eqn (3)) is proportional to the mean-curvature flow. To reduce the variation of the density function, we average this approximated mean-curvature value at q with those of its neighbors to smooth the density function. It is interesting to note is that the geometry-aware sub-sampling can be achieved using the same graph Laplacian framework as used for density adaption.



(a) target density (b) our result density (c) our result point set

Figure 9: *(a) (smoothed) mean curvature as target density, (b) our output density map, and (c) the output samples. Here* $|P| = 50k, |Q| = 5k$*. The correlation between the target and our output distribution is 0.9875, and* L_2*-error of our output distribution is 0.0271.*

Time complexity. Assuming that a k-nearest neighbor search in n points takes $O(\log n)$ time, our algorithm runs in $O(I|Q|(\log|P|+\log|Q|))$, where I is the number of iterations of the inner **for**-loop in Algorithm 1. Our current implementation, over $|P| = 25K$ points, takes 31, 81 and 169 seconds for $|Q|$ = 5K, 10K, and 15K, respectively. Over $|P|$ = 50K, it takes 33, 84, and 172 seconds for $|Q| =$ 5K, 10K, and 15K, respectively. Its dependency on P is only $\log|P|$. Hence the time increase is small when P gets larger.

Limitations. Given the averaging nature of the graph Laplacian, our algorithm has limitations in reconstructing density functions that have sharp changes. It will also be interesting to see how to preserve sharp features of the hidden surface as we adjust the point locations during the density adaptation algorithm. To this end, the framework of bilateral blue noise sampling in [Chen et al. 2013] could be useful, which we aim to investigate.

References

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2003. Computing and rendering point set surfaces. *IEEE Trans. Vis. Comput. Graphics 9*, 1, 3–15.

AMENTA, N., AND KIL, Y. J. 2004. Defining point-set surfaces. *ACM Trans. Graph. 23*, 3, 264–270.

AURENHAMMER, F. 1987. Power diagrams: Properties, algorithms and applications. *SIAM J. Comput. 16*, 1 (Feb.), 78–96.

BALZER, M., SCHLÖMER, T., AND DEUSSEN, O. 2009. Capacity-constrained point distributions: a variant of Lloyd's method. *SIGGRAPH '09, ACM Trans. Graph. 28*, 3, 86:1–86:8.

BELKIN, M., QUE, Q., WANG, Y., AND ZHOU, X. 2012. Toward understanding complex data: graph laplacians on manifolds with singularities and boundaries. In *Conf. Learning Theory (COLT)*, 36.1–36.26. JMLR – Proceedings Track 23.

BELKIN, M. 2003. *Problems of Learning on Manifolds*. PhD thesis, The University of Chicago.

BOTEV, Z. I., GROTOWSKI, J. F., AND KROESE, D. P. 2010. Kernel density estimation via diffusion. *Ann. Statist. 38*, 5, 2916–2957.

CHEN, J., GE, X., WEI, L.-Y., WANG, B., WANG, Y., WANG, H., FEI, Y., QIAN, K.-L., YONG, J.-H., AND WANG, W. 2013. Bilateral blue noise sampling. *ACM Trans. Graph. 32*, 6 (Nov.), 216:1–216:11.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *26th Ann. Conf. Computer Graphics and Interactive Techniques*, SIGGRAPH '99, 317–324.

DU, Q., GUNZBURGER, M. D., AND JU, L. 2002. Constrained centroidal voronoi tessellations for surfaces. *SIAM J. Sci. Comput. 24*, 5 (May), 1488–1506.

HEIN, M., AUDIBERT, J.-Y., AND LUXBURG, U. V. 2007. Graph laplacians and their convergence on random neighborhood graphs. *J. Mach. Learn. Res. (JMLR) 8*, 1325–1368.

HUANG, H., LI, D., ZHANG, H., ASCHER, U., AND COHEN-OR, D. 2009. Consolidation of unorganized point clouds for surface reconstruction. *ACM Trans. Graph. 28*, 5, 176:1–176:7.

LANGE, C., AND POLTHIER, K. 2005. Anisotropic smoothing of point sets. *Computer Aided Geometric Design 22*, 7, 680 – 692.

LEVIN, D. 2004. Mesh-independent surface interpolation. In *Geometric Modeling for Scientific Visualization*, G. Brunnett, B. Hamann, H. Mller, and L. Linsen, Eds., Mathematics and Visualization. Springer Berlin Heidelberg, 37–49.

LI, H., WEI, L.-Y., SANDER, P. V., AND FU, C.-W. 2010. Anisotropic blue noise sampling. *SIGGRAPH Asia '10, ACM Trans. Graph. 29*, 6, 167:1–167:12.

LIPMAN, Y., COHEN-OR, D., LEVIN, D., AND TAL-EZER, H. 2007. Parameterization-free projection for geometry reconstruction. *ACM Trans. Graph. 26*, 3 (July).

LIU, Y., WANG, W., LÉVY, B., SUN, F., YAN, D.-M., LU, L., AND YANG, C. 2009. On centroidal voronoi tessellation—energy smoothness and fast computation. *ACM Trans. Graph. 28*, 4 (Sept.), 101:1–101:17.

LLOYD, S. P. 1982. Least square quantization in pcm. *IEEE Transactions on Information Theory 28*, 2, 129–137.

LUO, C., SUN, J., AND WANG, Y. 2009. Integral estimation from point cloud in d-dimensional space: A geometric view. In *Proc. 25th ACM Sympos. on Comput. Geom.*, 116–124.

MARRON, J. S., AND WAND, M. P. 1992. Exact mean integrated squared error. *The Annals of Statistics 20*, 2, pp. 712–736.

ÖZTIRELI, A. C., ALEXA, M., AND GROSS, M. 2010. Spectral sampling of manifolds. *ACM Trans. Graph. 29*, 6, 168:1–168:8.

PAULY, M., GROSS, M., AND KOBBELT, L. P. 2002. Efficient simplification of point-sampled surfaces. In *Proceedings of the conference on Visualization '02*, VIS '02, 163–170.

ROBBINS, H., AND MONRO, S. 1951. A stochastic approximation method. *Annals of Mathematical Statistics 22*, 400–407.

WEI, L.-Y., AND WANG, R. 2011. Differential domain analysis for non-uniform sampling. *ACM Trans. Graph. 30*, 4, 50:1–50:10.

A Connection to Kernel Density Estimation

It turns out that there is an alternative way to interpret the tangent projection $\vec{\mathsf{V}}_{\|}(q)$ of the graph Laplacian $\mathsf{L}_t^Q\overrightarrow{\mathsf{P}}(q)$ performed on the coordinates functions $\overrightarrow{\mathsf{P}} = [\mathsf{X}, \mathsf{Y}, \mathsf{Z}]$ via the standard kernel density estimator (KDE). Specifically, suppose current points $Q = \{q_1, \ldots, q_m\}$ are i.i.d. sampled with respect to the density function $\mathsf{p} : \mathrm{M} \to \mathbb{R}$. Now consider the following popular *Gaussian kernel density estimator* [Marron and Wand 1992] that gives rise to an

empirical density function $\tilde{g} : \mathbb{R}^3 \to \mathbb{R}$ (to approximate p) with support being the ambient space (which is $\mathbb{R}^3$ in our case): at any $\mathbf{x} \in \mathbb{R}^3$, we have [2]:

$$\tilde{g}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} K_t(\mathbf{x}, q_i), \quad \text{where} \quad K_t(\mathbf{x}, q_i) = \frac{1}{\pi t} e^{-\frac{\|\mathbf{x}-q_i\|^2}{t}}.$$

The ambient gradient of this empirical density function (w.r.t. the coordinates of the ambient space $\mathbb{R}^3$) is:

$$\begin{aligned} \nabla\tilde{g}(\mathbf{x}) &= \frac{2}{mt}\sum_{i=1}^{n}(q_i - \mathbf{x})K_t(\mathbf{x}, q_i) = \\ &\frac{2}{mt}\cdot\frac{1}{\pi t^2}\sum_{i=1}^{m}[e^{-\frac{\|\mathbf{x}-p_i\|^2}{t}}(q_i - \mathbf{x})] = \frac{2}{\pi}\mathrm{L}_t^Q\overrightarrow{\mathsf{P}}(\mathbf{x}). \end{aligned} \tag{5}$$

Now if we restrict this empirical density function to the hidden surface M, denoted by $\tilde{g}' := \tilde{g}|_{\mathrm{M}}$, then we have that the gradient of $\tilde{g}'$ at a point $\mathbf{x} \in \mathrm{M}$ is simply the projection of $\nabla\tilde{g}(\mathbf{x})$ onto the tangent space T_x at $\mathbf{x}$. This projection, is exactly $\frac{2}{\pi}\vec{\mathsf{V}}_{\|}(\mathbf{x})$, which matches what we have (the second term) in Eqn (3). In other words, the density gradient we obtained can also be viewed as the gradient of the empirical Gaussian kernel density estimator restricted to the manifold case (instead of in the ambient space).

While the connection between the Gaussian kernel density estimator and heat diffusion process has been investigated in the Euclidean space [Botev et al. 2010], here we build the connection with the graph Laplacian, and also the connection for the manifold case. We believe that this connection itself is interesting. Eqn (5), combined with the theoretical studies on the Gaussian-weighted graph Laplacian, helps to provide better understanding of the ambient gradient of the kernel density estimator $\tilde{g}$. For example, by Eqn (5) and Eqn (3), it now becomes evident that the normal component of $\nabla\tilde{g}$ is in fact proportional to the mean-curvature flow at a point x in the interior of a manifold. As another example, suppose the input domain M is a singular manifold, which consists of a set of (potentially intersecting) manifolds with boundaries. Recent work in [Belkin et al. 2012] shows that $\mathrm{L}_t^Q f$ behaves drastically different around the singular set, which includes the boundary of each manifold patch, non-smooth folds in each manifold patch, as well as intersections of multiple pieces of manifolds. In particular, for a point x around the singular set, Eqn (2) and (3) no longer hold; instead, an equation with a form similar to Eqn 4 holds. Such singular behavior will now carry over to the gradient of the KDE density estimator. The projection of $\mathrm{L}_t^Q\overrightarrow{\mathsf{P}}(\mathbf{x})$ in the tangent space at $\mathbf{x}$ (if it can be defined) will no longer reflect the gradient of the true density function. Hence to handle domains with singularities, different strategies are necessary for points around singularities if true density is desirable.

[2] The general form of the Gaussian kernel below should be $K_t(x, y) = \frac{1}{(\pi t)^{d/2}} e^{-\frac{\|\mathbf{x}-q_i\|^2}{t}}$. We choose the dimension of the Gaussian kernel to be the intrinsic dimension $d = 2$.