Moving Level-of-Detail Surfaces

CORENTIN MERCIER, LTCI, Télécom Paris, Institut Polytechnique de Paris, France
THIBAULT LESCOAT, LTCI, Télécom Paris, Institut Polytechnique de Paris, France
PIERRE ROUSSILLON, LTCI, Télécom Paris, Institut Polytechnique de Paris, France
TAMY BOUBEKEUR, Adobe Research, France
JEAN-MARC THIERY, Adobe Research, France

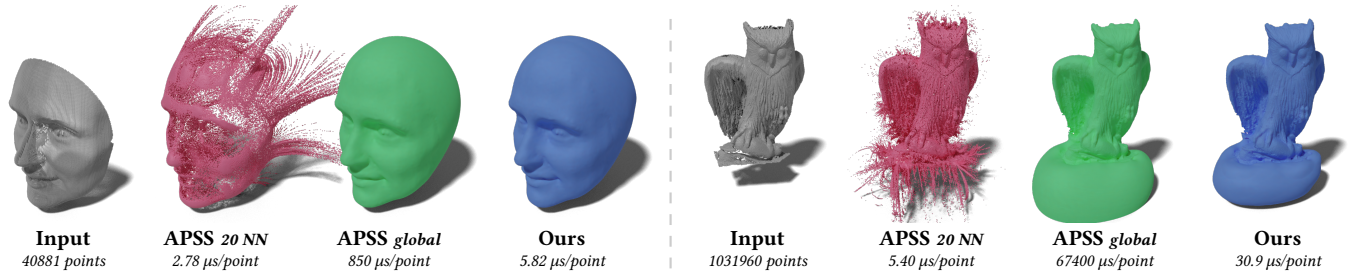


Fig. 1. Modeling Algebraic Point Set Surfaces using a fixed number of input nearest points results in unacceptable approximations far from densely sampled regions. Our smooth approximation copes naturally with complex inputs featuring large missing parts and competes with global optimization approaches while allowing for pointwise smooth projection and filtering.

We present a simple, fast, and smooth scheme to approximate Algebraic Point Set Surfaces using non-compact kernels, which is particularly suited for filtering and reconstructing point sets presenting large missing parts. Our key idea is to consider a moving level-of-detail of the input point set which is adaptive w.r.t. to the evaluation location, just such as the samples weights are output sensitive in the traditional moving least squares scheme. We also introduce an adaptive progressive octree refinement scheme, driven by the resulting implicit surface, to properly capture the modeled geometry even far away from the input samples. Similarly to typical compactly-supported approximations, our operator runs in logarithmic time while defining high quality surfaces even on challenging inputs for which only global optimizations achieve reasonable results. We demonstrate our technique on a variety of point sets featuring geometric noise as well as large holes.

CCS Concepts: • **Computing methodologies** → **Point-based models**.

Additional Key Words and Phrases: Point-based modeling, MLS projections, point set surfaces

ACM Reference Format:
Corentin Mercier, Thibault Lescoat, Pierre Roussillon, Tamy Boubekeur, and Jean-Marc Thiery. 2022. Moving Level-of-Detail Surfaces. *ACM Trans. Graph.* 1, 1, Article 130 (May 2022), 10 pages. https://doi.org/10.1145/8888888.7777777

Authors' addresses: Corentin MercierLTCI, Télécom Paris, Institut Polytechnique de Paris, France, corentin.mercier@grosmi.net; Thibault LescoatLTCI, Télécom Paris, Institut Polytechnique de Paris, France, thibault@lescoat.fr; Pierre RoussillonLTCI, Télécom Paris, Institut Polytechnique de Paris, France, roussillon.pierre@gmail.com; Tamy BoubekeurAdobe Research, France, boubek@adobe.com; Jean-Marc ThieryAdobe Research, France, jthiery@adobe.com.

0730-0301/2022/5-ART130 $15.00
https://doi.org/10.1145/8888888.7777777

1 INTRODUCTION

Moving Least Squares (MLS) projection operators [Levin 1998, 2003] offer an elegant unified framework to model Point Set Surfaces (PSS) from unorganized point clouds [Alexa et al. 2001], with immediate application to point-based filtering and mesh reconstruction. At their core, MLS operators take the form of iterative projection procedures from $\mathbb{R}^3$ to $\mathbb{R}^3$, whose stationary set is the PSS. At each iteration, a geometric primitive, e.g., a plane is optimized to fit the input point set under a weighting kernel centered at the evaluation position, before projecting this point onto the primitive. In terms of quality, the resulting PSS is guaranteed to be smooth as long as smooth global kernels are used.

Unfortunately, such kernels induce untractable computations, even on inputs of moderate size, as the projection of a single point has a $O(N)$ complexity, N being the number of input samples. Consequently, it is common to localize the support of the MLS projection by gathering only the k nearest neighbors (kNN) – this typically breaks smoothness and creates holes (see Fig. 1).

Our objective is to design a point-wise operator that retains the simplicity of MLS but scales efficiently to non-compact kernels and reaches a robustness which is competitive with global optimizations [Huang et al. 2019; Kazhdan and Hoppe 2013] when it comes to large missing parts in the input point set. To do so, we propose a new MLS approximation scheme, introducing a *moving level-of-detail*, devised for Algebraic Point Set Surfaces (APSS) [Guennebaud et al. 2008]. In particular, we make the following contributions:

- we present a new hierarchical scheme for the efficient approximate convolution of any nD point data set with a smoothly-varying non-compact kernel running in $O(\log(N))$;

- we introduce an efficient world-space sampling scheme for precise, fast, and memory-efficient sampling and meshing.

We demonstrate its use for point-based filtering and mesh reconstruction of point sets presenting a significant amount of geometric noise as well as large missing parts.

2 RELATED WORK

In this section, we discuss the most relevant works to our technique, and refer the reader to recent surveys [Berger et al. 2013, 2017] for a more general overview.

Point Set Surfaces. The Moving Least Square (MLS) operator [Levin 1998] is a flexible scattering data approximation model, particularly useful to model surfaces from unorganized point clouds [Alexa et al. 2001; Amenta and Kil 2004; Levin 2003]. This operator usually comes in the form of a local projection of the ambient space presenting good denoising capabilities [Levin 2003; Reuter et al. 2005] and fast enough to quickly reconstruct or render point-based models [Alexa et al. 2001; Guennebaud et al. 2008]. While its original definition involved a local least squares fit of a plane followed by the optimization of a polynomial patch over the plane, its more recent instances have simplified the process, by directly fitting simple geometric primitives locally, using them as the projection support. Hence, the various flavors of MLS projections differ mostly in the type of primitives that is locally fitted, such as planes [Levin 2003], spheres [Guennebaud et al. 2008; Guennebaud and Gross 2007] or parabolic-cylinders [Ridel et al. 2015]. While the projection procedure may also account for per-sample Hermite data [Alexa and Adamson 2009], a number of methods have been specially designed to reproduce sharp features and salient mesostructures [Guillemot et al. 2012; Öztireli et al. 2009; Reuter et al. 2005]. The underlying surface defined by the MLS operator can be improved using local features [Dey and Sun 2005] or gradient fields [Chen et al. 2013], and even supports non-linear filtering such as morphological opening and closing [Calderon and Boubekeur 2014]. In this paper, we focus on *algebraic point set surfaces* (APSS) [Guennebaud et al. 2008; Guennebaud and Gross 2007], which fit local algebraic spheres, making the operator both robust and accurate while staying fast. However, tractable performances are reached only when localizing the support of the operator, i.e., fitting the algebraic sphere over only a local subset of the input point cloud – typically the k nearest neighbors (NN) – which forbids large hole filling and poorly performs with poorly sampled objects.

Multi-level Partition of Unity (MPU). Given an input oriented point set, the MPU method [Ohtake et al. 2003] aims at building an implicit surface from locally-estimated quadratic functions, using an octree-based partitioning. During the top-down octree construction, the subset of points belonging to a cell is approximated by a quadric, subdividing recursively the cell when the residual error is above a fixed tolerance. This results in a set of local implicit surfaces, smoothly blended together using a partition of unity to form the whole object. While primitives are locally estimated from the input point set, those are attached to the cells and not to individual query points as in PSS methods. Consequently, MPU does not output a simple projection operator, and further polygonalization is required even for visualization purposes. Xiao [2011] replaces the local polynomial with the local algebraic spheres of APSS, located at the centers of the nodes. However, this method inherits the aforementioned limitations of MPU and somehow departs from MLS techniques in its nature, as the resulting projection is performed over the global implicit function.

Global Surface Reconstruction. The geometry processing literature is rich with methods reconstructing a surface globally from an input point set, with or without normals. Among them, the *Poisson Surface Reconstruction* [Kazhdan et al. 2006] and its variants are a popular choice in many 3D capture pipelines. These methods interpret the input normals as a sampling of the gradient of the unknown solid's indicator function. A 3D vector field is then reconstructed from this sampling and a Poisson equation is solved to obtain a scalar function out of which a surface is contoured. This method was further refined with the addition of a screening term [Kazhdan and Hoppe 2013], taming the smoothing often occuring with the original method. Another competing method, with results of equivalent quality, was proposed by Calakli et al. [2011]. Their approach is similar to the Poisson surface reconstruction, but instead of forcing the implicit function to approximate the indicator function of the volume, they enforce low Hessian energy of the function and obtain smooth reconstructions as a result. More recently, deep learning was introduced to global surface reconstruction by Hanocka et al. [2020], building a self-prior inferred by a neural network to iteratively shrink-wrap the convex hull of the input sampling. Other approaches include the normal-free method of Mullen et al. [2010] or more recently the works of Barill et al. [2018] and Lu et al.[2018]. We refer the reader to the survey by Berger et al. [2017] for an extensive overview of this vast domain.

Finally, our approach can be seen as a Fast Multipole Method (FMM) [Rokhlin 1985]. [Carr et al. 2001] is a popular global method that uses FMM at its core to accelerate their global solver. This method falls outside the scope of MLS surface methods: (i) their fairness parameter is a penalty term balancing sample fitting with minimal Hessian square norm whereas ours is a traditional fairing kernel; (ii) they output a volumetric scalar function whereas we output a projection operator; (iii) exploring their fairness parameter space requires recomputing a global (volumetric) system each time from scratch, often requiring several minutes of computation, whereas our technique allows updating each sample projection independently in real time.

3 MOVING LEVEL-OF-DETAIL SURFACES

We consider as input an oriented point set $\mathcal{P} = \{\mathbf{p_i}, \mathbf{n_i}, \sigma_i\}_i$ (position, normal, estimated area), and a smooth kernel $\mathcal{H} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^+$.

For any point $\mathbf{q}$ in 3D space, APSS optimizes an algebraic sphere $S_q : \mathbb{R}^3 \to \mathbb{R}$ (a scalar field defined everywhere in $\mathbb{R}^3$, whose 0-set is either a 3D sphere or an oriented 3D plane for 0-curvature spheres) to $\mathcal{P}$, and projects $\mathbf{q}$ onto its 0-set. The weight of each point $\mathbf{p_i}$ of $\mathcal{P}$, describing its influence over $\mathbf{q}$, is computed using $\mathcal{H}$ as $w_i = \mathcal{H}(\mathbf{q}, \mathbf{p_i})$. Using this, an algebraic sphere S_q is fitted at $\mathbf{q}$:

$$S_q(\mathbf{X}) := (u_0, \mathbf{u_{123}}, u_4) \cdot (1, \mathbf{X}, \|\mathbf{X}\|^2), \tag{1}$$

with $\mathbf{u_{123}} \in \mathbb{R}^3$ and $u_0, u_4 \in \mathbb{R}$. S_q should fit the weighted point set $(\mathbf{w}, \mathcal{P})$ both in terms of positions (i. e., $S_q(\mathbf{p_i}) \simeq 0$) and normals (i. e., $\nabla S_q(\mathbf{p_i}) \simeq \mathbf{n_i}$, with $\nabla S_q(\mathbf{X}) = (\mathbf{u_{123}} + 2u_4\mathbf{X})$). Note that Eq. 1 captures adequately both "oriented 3D spheres" as well as oriented 3D planes. A sphere with center $c \in \mathbb{R}^3$ and radius $r \in \mathbb{R}$ is given by $\{X \text{ s.t. } \|X-c\|^2 - r^2 = 0\}$, which is homogeneous to Eq. 1 (resulting in $u_0 = \|c\|^2 - r^2$, $u_{123} = -2c$, $u_4 = 1$, and a non-normalized gradient $\nabla = 2(X-c)$). Similarly, a plane passing through $c \in \mathbb{R}^3$ with normal $n \in \mathbb{R}^3$ is given by $\{X \text{ s.t. } (X-c)\cdot n = 0\}$ (resulting in $u_0 = c \cdot n$, $u_{123} = n$, $u_4 = 0$, and a normalized gradient $\nabla = n$).

Following Guennebaud et al. [2008], we use *algebraic spheres* instead of *geometric spheres* as the former degenerates gracefully to planes when $u_4 \to 0$ – as pointed above, u_4 captures the curvature of the algebraic sphere under appropriate normalization constraints – and we first align the gradients $\{\nabla S_q(\mathbf{p_i})\}$ onto the normals $\{\mathbf{n_i}\}$, before adjusting the level-set to fit the positions:

$$(\mathbf{u_{123}}, u_4) = \text{argmin} \sum_i w_i\sigma_i \|\nabla S_q(\mathbf{p_i}) - \mathbf{n_i}\|^2$$
$$u_0 = \text{argmin} \sum_i w_i\sigma_i S_q(\mathbf{p_i})^2 \text{ with fixed } (\mathbf{u_{123}}, u_4)$$

As pointed out in [Guennebaud et al. 2008] (Section 3.3), this procedure provides increased robustness compared to, e.g., fitting both gradients and points at the same time.

This results in the following closed form expression for S_q:

$$\begin{cases} u_4 & = \frac{1}{2}\frac{\sum_i w_i\sigma_i \mathbf{p_i}\cdot\mathbf{n_i} - \sum_i w_i\sigma_i \mathbf{p_i}\cdot\sum_i w_i\sigma_i \mathbf{n_i}/\sum_i w_i\sigma_i}{\sum_i w_i\sigma_i \|\mathbf{p_i}\|^2 - \|\sum_i w_i\sigma_i \mathbf{p_i}\|^2/\sum_i w_i\sigma_i} \\ \mathbf{u_{123}} & = \left(\sum_i w_i\sigma_i \mathbf{n_i} - 2u_4 \sum_i w_i\sigma_i \mathbf{p_i}\right)/\sum_i w_i\sigma_i \\ u_0 & = -\left(\sum_i w_i\sigma_i \mathbf{p_i}\cdot\mathbf{u_{123}} + u_4 \sum_i w_i\sigma_i \|\mathbf{p_i}\|^2\right)/\sum_i w_i\sigma_i. \end{cases} \quad (2)$$

With S_q in hand, one can obtain the projection $\bar{\mathbf{q}}$ by projecting $\mathbf{q}$ onto S_q, and define the normal $\mathbf{n}_{\bar{\mathbf{q}}}$ by taking the gradient of S_q at $\bar{\mathbf{q}}$:

$$\begin{cases} \bar{\mathbf{q}} = \text{Project}(\mathbf{q}, S_q) \\ \mathbf{n}_{\bar{\mathbf{q}}} = \nabla S_q(\bar{\mathbf{q}}) \;/\; \|\nabla S_q(\bar{\mathbf{q}})\| \end{cases} \quad (3)$$

Finally, one can extract a surface by contouring the implicit function $f(\mathbf{q}) = (\mathbf{q} - \bar{\mathbf{q}}) \cdot \mathbf{n}_{\bar{\mathbf{q}}}$, e. g., by using the Marching Cubes technique [Lorensen and Cline 1987] on a uniform grid or the Dual Contouring technique [Ju et al. 2002] on an adaptive octree.

Estimated areas. Our formulation is slightly different from the one of Guennebaud et al. [2008] in the sense that we make use of estimated local areas $\{\sigma_i\}$ to best approximate a spatial integral over $\mathcal{P}$ (without changing the kernel), whereas they use an estimated local density to parameterize the kernel locally (i. e., each input point influences the space around it differently). In essence, both strategies are designed to cope with variable density in the input point set and avoid *over-fitting* highly-sampled regions. This mild change has important practical consequences which make our approach possible. Since all weights w_i are of the form $w_i(\eta) = \mathcal{H}(\mathbf{p_i}, \eta)$, **i)** we can relate reliably the weight and distance functions for monotonic radial filter kernels $\mathcal{H}$, and **ii)** as we discuss in Sections 3.1 and 3.2, we can factorize the weights that are similar for samples located roughly in the same region when seen from far away. Both properties are instrumental to the design of our algorithm and make it efficient.

3.1 Smooth approximation

Looking at Eq. 2, we can factorize the weights for points that similarly influence $\mathbf{q}$: $\forall i \in I, w_i \simeq w \implies \sum_{i\in I} w_i\sigma_i\mathbf{p_i} \simeq w\sum_{i\in I}\sigma_i\mathbf{p_i}$.

This trivial observation points to a simple algorithm for approximating the algebraic sphere S_q fitting the weighted point set $(\mathbf{w}, \mathcal{P})$. In a preprocess, we organize $\mathcal{P}$ into an octree, storing in each node η the following *9D statistics* that aggregate the geometry of its contained input points $\{i \in I_\eta\}$:

$$\delta_{\text{node}}(\eta) = \begin{cases} \sigma = \sum_i \sigma_i \in \mathbb{R} & \mathbf{p}_\alpha = \sum_i \sigma_i\mathbf{p_i} \in \mathbb{R}^3 \\ \mathbf{n}_\alpha = \sum_i \sigma_i\mathbf{n_i} \in \mathbb{R}^3 & p_\beta = \sum_i \sigma_i\|\mathbf{p_i}\|^2 \in \mathbb{R} \\ pn_\beta = \sum_i \sigma_i\mathbf{p_i}\cdot\mathbf{n_i} \in \mathbb{R} \end{cases} \quad (4)$$

When computing S_q at query point $\mathbf{q}$, we traverse the octree, starting from its root and descending into it iteratively, and aggregate the statistics of a subset of nodes – our *moving level-of-detail*. Following common strategies [Barill et al. 2018; Greengard and Rokhlin 1987], we estimate this subset by stopping the descent when $\mathbf{q}$ is sufficiently far from the nodes. Finally, we approximate S_q using the statistics and the weights we associate with each node.

Weight estimation. To associate a weight w_η with a given node η, we simply consider the average position of the points inside η ($\mathbf{p}_\eta = \mathbf{p}_\alpha(\eta)/\sigma(\eta)$) and compute

$$w_\eta = \mathcal{H}(\mathbf{q}, \mathbf{p}_\eta) \quad (5)$$

More complex strategies involving the point distribution inside η (e. g., using Gaussian Mixture Models) could be used, but it would require integrating $\mathcal{H}$ against those, which is not feasible for arbitrary kernels. We found in our experiments that our simple strategy offered good results, even for challenging inputs.

3.2 Algorithm

A simple traversal strategy could consist in stopping at a node if the distance to its bounding sphere is larger than a given predetermined threshold, and rely on its statistics to represent its entire sub-tree. Similar binary decisions are for example used in [Barill et al. 2018] to estimate the winding number of a surface around query points, which is harmless in this case as they output an integer value, with the approximation below the quantization error.

In our case, however, this simple strategy cannot be used, as it results in non-smooth statistics, and therefore in a non-smooth surface (e.g., the top of the face in inset). We describe next a hierarchical scheme well-suited to the blending of any nD data, that we use to smoothly blend the statistics.

Smooth hierarchical partition of unity. By scaling the nodes' bounding spheres by the same factor $\lambda > 1$ (see Fig. 2, left), we construct *protection spheres* $\{\mathcal{S}(\eta)\}$ for all nodes $\{\eta\}$, that strictly contain their children and their protection spheres. We rely on these protection spheres to design our hierarchical partition of unity.

When traversing the octree, we stop in a node if the query point $\mathbf{q}$ is *outside* the protection sphere $\mathcal{S}(\eta)$ of the node. Otherwise, if $\mathbf{q}$ is inside $\mathcal{S}(\eta)$, we blend the node and its children sub-trees $Ch(\eta)$ statistics based on the respective distance to the protection spheres. In this more complex case, we return the following statistics for the

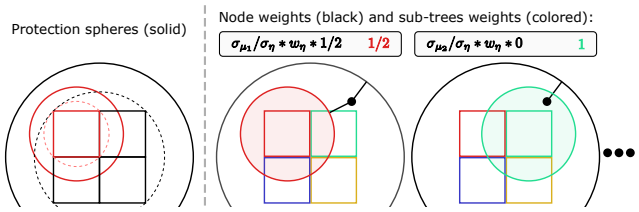


Fig. 2. Node/subtrees local partition of unity illustrated in 2D. $\{\mu_i\}$ (colored) denote the subtrees of node η (black). **First:** Bounding spheres (dotted) and protection spheres (solid: scaled bounding spheres) of the parent node (black) and one of its child nodes (red). **Others:** The respective distances to the protection spheres induce a hierarchical partition of unity.

entire sub-tree:

$$\delta_{\text{tree}}(\eta, \mathbf{q}) = \sum_{\mu \in Ch(\eta)} \delta_{\text{tree}}(\mu, \mathbf{q})(1 - \gamma_\mu(\mathbf{q})) + \sum_{\mu \in Ch(\eta)} \delta_{\text{node}}(\eta)\ \gamma_\mu(\mathbf{q})\ \frac{\sigma(\mu)}{\sigma(\eta)}\ \mathcal{H}(\mathbf{q}, \mathbf{p}_\eta), \tag{6}$$

$\gamma_\mu(\mathbf{q})$ describing the two-sphere local blending illustrated in Fig. 2. With $d(\mathbf{q}, S) = \|\mathbf{q} - \mathbf{c}\| - r$ the signed distance from the point $\mathbf{q}$ to a sphere S of center $\mathbf{c}$ and radius r, we can write:

$$\gamma_\mu(\mathbf{q}) = \Omega(d(\mathbf{q}, \mathcal{S}(\mu)), d(\mathbf{q}, \mathcal{S}(\eta))) \tag{7}$$

$$\text{with}\quad \Omega(x, y) = \begin{cases} 0 & \text{if } x, y \le 0 \text{ (i. e., } \mathbf{q} \in \mathcal{S}_\mu) \\ 1 & \text{if } x, y \ge 0 \text{ (i. e., } \mathbf{q} \notin \mathcal{S}_\eta) \\ F_\Omega(x/(x-y)) & \text{otherwise: } y < 0 < x \end{cases}$$

$$F_\Omega(u) = e^{(-e^{(1/(u-1))}/u^2)}$$

Note that this scheme induces an infinitely differentiable local partition of unity between the interior of the child's sphere $\mathcal{S}_\mu$ and the exterior of the parent's sphere $\mathcal{S}_\eta$, because $F_\Omega : [0,1] \to [0,1]$ is a strictly increasing function with null derivatives (of all orders) at 0 and 1 (see inset). Proof of this claim is given in Appendix A. Note that we also used $F_\Omega(u) = (1 - \cos(\pi u))/2$, which only results in a C^1 surface but we did not observe visual artifacts in practice.

We summarize in Alg. 1 the pseudo-code of our technique and we describe in Appendix B a detailed GPU implementation.

ALGORITHM 1: Computation of $(\bar{\mathbf{q}}, \mathbf{n}_{\bar{\mathbf{q}}}) = \text{Project}(\text{Point } \mathbf{q})$

```
δ = δ_tree( root , q) // see Alg. 2
compute u_0, u_123, u_4 from δ // see Eq. 2
q̄ = Project(q, algebraic sphere (u_0, u_123, u_4)) :
if (u_4 == 0) then
    // q̄: project q on plane u_0 + u_123 · X = 0
else
    c = −u_123/(2u_4)
    r = sqrt(max(0, ||c||^2 − u_0/u_4))
    // q̄: project q on sphere of center c and radius r
n = 2u_4 q̄ + u_123
n_q̄ = n/||n||
```

ALGORITHM 2: Computation of $\delta_{\text{tree}}(\text{Node } \eta,\ \text{Point } \mathbf{q})$

```
// returns the cumulative statistics for fitting an algebraic sphere to
   the subtree rooted at node η
if η is leaf then
    // accumulate statistics over points in the leaf
    return Σ_{i∈η} H(q, p_i) * δ_node(i)
else
    if q ∉ S(η) then
        // q is sufficiently far from the node
        return H(q, p_η) * δ_node(η)
    else
        // blend statistics between η and its children μ (Eq.6)
        return Σ_{μ∈Ch(η)} (1 − γ_μ(q)) * δ_tree(μ, q) +
          Σ_{μ∈Ch(η)} σ_μ/σ_η * H(q, p_η) * γ_μ(q) * δ_node(η)
```

Analysis. We stress that in Eq. 6, one has to distinguish the *node statistics* δ_{node} (computed following Eq. 4 and stored in the node) from the *tree statistics* δ_{tree} that require a recursive evaluation. The factor $\sigma(\mu)/\sigma(\eta)$ appears in Eq. 6 to account for that, while η has to be blended with its more detailed representation (all of its children), there are multiple independent local partitions of unity, one *per-child*, and we expect each of them to represent a proportion $\sigma(\mu)/\sigma(\eta)$ of the data. For example, if all points are located inside the same sub-tree (i. e., 7 of the 8 children are empty), the summation is only performed over this single sub-tree, and the partition of unity is truly performed between the node and its only child.

Note also that Eq. 6 describes the case when $\mathbf{q}$ is outside the protection sphere of η adequately as well, since in this case for each child μ we have $\gamma_\mu(\mathbf{q}) = 1$, and since $\sum_\mu \sigma(\mu)/\sigma(\eta) = 1$, we obtain $\delta_{\text{tree}}(\eta, \mathbf{q}) = \delta_{\text{node}}(\eta)\mathcal{H}(\mathbf{q}, \mathbf{p}_\eta)$.

3.3 Practical projection strategies

So far, from an input query point $\mathbf{q}$ we can compute an algebraic sphere S_q and project $\mathbf{q}$ on it. To retrieve the actual surface, one needs to advect $\mathbf{q}$ along the following vector field:

$$\mathbf{f}(\mathbf{0}) = \mathbf{q} \quad ; \quad \frac{\partial \mathbf{f}}{\partial t}(t) = \text{Project}(\mathbf{f}(\mathbf{t})) - \mathbf{f}(\mathbf{t}) \tag{8}$$

A common strategy is to iteratively replace $\mathbf{q}$ with $\text{Project}(\mathbf{q})$ (i.e., setting $\Delta t = 1$ in Eq. 8) for a fixed number of steps or until the displacement δf is small enough. Sadly, this scheme only performs well near the surface, mainly because weights $\{w_i(\mathbf{q})\}$ are almost constant for $\|\mathbf{q}\| \gg D$ (i.e., $\mathcal{P}$ is seen as a single sphere from points sufficiently far away). While projecting near the surface is still useful, e.g., for denoising, to fully exploit non-compact kernels we need to handle points from afar. We achieve this via an upper bound on $\|\delta f\|$ (in our case a tenth of the bounding box diagonal of $\mathcal{P}$).

3.4 Practical non-compact kernels

While most related works advocate using Gaussian kernels, we observe that they are inadequate for incomplete data. Indeed, the *tail* of the Gaussian vanishes quickly, and it is often impossible to find parameters providing the desired intent (see Fig. 3: one cannot tune a Gaussian allowing for smooth surface completion while retaining high-frequency geometric details). We thus promote

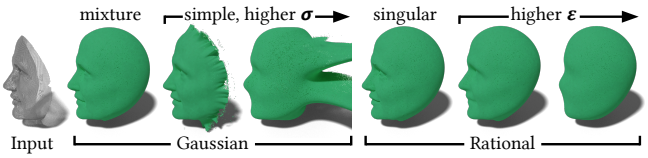


Fig. 3. As they vanish too quickly, Gaussian kernels are not suitable when approximating a surface with missing parts. Instead, Gaussian mixtures or rational kernels are more robust to holes. Whereas all these kernels are *approximating*, rational kernels can also be *interpolating*.

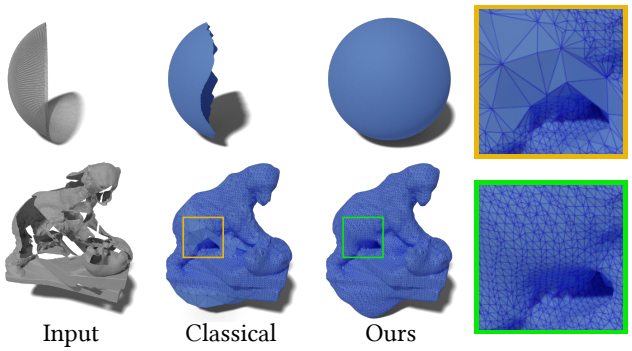


Fig. 4. The classical way to derive an octree from a point cloud focuses only on the samples locations and completely misses the holes. When meshing the surface on such an octree [Ju et al. 2002], this yields a coarse or invalid surface. Instead, we propose a new way to generate an octree that will encompass the full surface.

heavy tail kernels, such as Gaussian mixtures or rational kernels:

$$\mathcal{H}^{\mathrm{GM}}_{\{\sigma_i\}}(\mathbf{q},\mathbf{p}) = \sum_i \sigma_i^{-3} e^{\frac{-\|\mathbf{q}-\mathbf{p}\|^2}{2\sigma_i^2}} \quad ; \quad \mathcal{H}^{\mathrm{Rat}}_{\epsilon,k}(\mathbf{q},\mathbf{p}) = (\|\mathbf{q}-\mathbf{p}\|^2+\epsilon)^{\frac{-k}{2}} \quad (9)$$

Setting $\epsilon = 0$ when using $\mathcal{H}^{\mathrm{Rat}}_{\epsilon,k}$ results in an interpolating surface, as $\mathcal{H}^{\mathrm{Rat}}_{0,k}$ becomes singular on the input samples. Interestingly, one can navigate continuously between interpolating ($\epsilon = 0$) and approximating ($\epsilon > 0$) surfaces. A simple yet effective strategy to set appropriate parameters for $\mathcal{H}^{\mathrm{GM}}$ is to take $\sigma_i = a^i \sigma_0$ with $a > 1$.

4 SURFACE EXTRACTION

Densely sampling the surface is key for multiple applications, such as rendering it using *splats*. Perhaps more importantly, it is necessary for a fast and memory-bounded mesh extraction of the surface. Such surface extraction is commonly done using Dual Contouring [Ju et al. 2002] on an adaptive grid (an octree in 3D). However, the usual way of preparing the octree – locating leaf nodes where input points lie – is *input-sensitive* (densifying the octree around the input pointset only) rather than *output-sensitive* (densifying the octree around the expected output surface) [Zhao et al. 2021]; using the resulting octree for contouring will yield artifacts such as those seen in Fig. 4. As we target point-based modeling of surfaces arbitrarily far away from input points, we need to adapt the preparation of the octree.

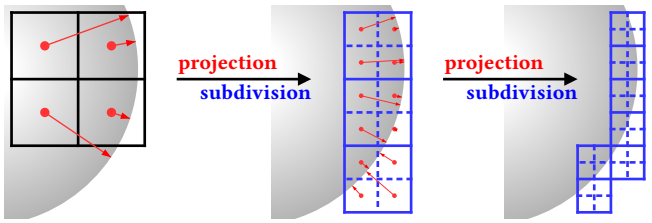


Fig. 5. We refine an octree by alternating **projection** of the nodes centers (and we keep only nodes containing the projections), and **subdivision** of all nodes. This refinement process is repeated until a target depth is reached.

Thus we propose a simple, yet effective strategy to densely sample the surface inside an adaptive octree that we construct in an iterative manner, following these two simple observations:

- an octree cell in which a 3D point is projected contains most probably part of the algebraic surface: we can refine the sampling inside this cell in a hierarchical manner by simply subdividing it and further investigating its children.
- an octree cell containing part of the surface is likely to have neighboring cells containing parts of the surface, as the implicit functions we construct are smooth (this observation has for example motivated past research on implicit function tracking [Bloomenthal 1988]).

As the surface might expand arbitrarily far outside the bounding box of $\mathcal{P}$, we need to be flexible in the generation of the octree; in particular, we allow nodes to exist anywhere in space, and index nodes via (d, x, y, z), with d the depth of a node and (x, y, z) its index in the grid at depth d. The initial root is $(0, 0, 0, 0)$, and spatially represents the bounding box of $\mathcal{P}$. With this definition, our aim is to generate nodes at depth D following the surface, corresponding to potentially multiple nodes at depth 0. Our key idea is to project *nodes*, by projecting their center and retrieving the node corresponding to the resulting position. We propose the following refinement process: starting from nodes $\mathcal{N}_0$ at depth 0 (initially, only $(0, 0, 0, 0)$), we obtain the nodes $\mathcal{N}_d$ by alternating node projection and node subdivision (Fig. 5). Some nodes in $\mathcal{N}_d$ might not have an ancestor in $\mathcal{N}_0$ (they are outside the starting nodes), notably in the case of large holes in $\mathcal{P}$. Hence, we add these unprocessed ancestors at depth 0 to $\mathcal{N}_0$ and refine them; this process is repeated until no new nodes need to be added to $\mathcal{N}_0$. Since we want the *final* octree to have depth D, we need to adjust the refinement depth d. Initially, $d = D$, but we adjust it following $\mathcal{N}_0$, as $d = D - \lceil \log_2 \max(|\mathcal{N}_0|_x, |\mathcal{N}_0|_y, |\mathcal{N}_0|_z) \rceil$, $|\mathcal{N}_0|_j$ denoting the extent of $\mathcal{N}_0$ along the j axis, taking here the input bounding box as unit system. This also means that we traverse fewer and fewer nodes when extending $\mathcal{N}_0$, limiting the memory and performance costs. Finally, we re-index the resulting nodes to form an octree (only 1 root) of depth D that closely follows the surface. To create a mesh, we use Dual-Contouring [Ju et al. 2002] on either this octree (our experimental setup) or an octree built from its leaves after projection on the MLoD Surface.

5 EVALUATION

We make our implementation publicly available here: https://github.com/CorentinMercier/MlodSurfaces.

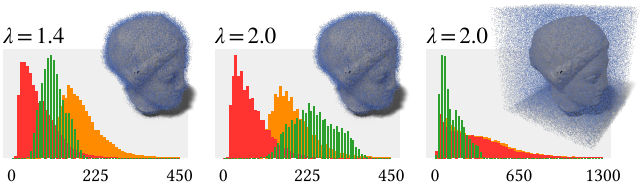


Fig. 6. We project 100k points on $\mathcal{P}$ and show the histograms of the number of nodes traversed per projection. As expected, more nodes are traversed when λ increases. Contrary to gathering the 20NN or the 500NN, our method does not suffer from extremely long traversals. Both our octree and the kd-tree here have a maximum depth of 12.

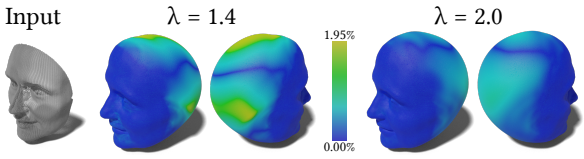


Fig. 7. Our method is accurate w.r.t. global APSS, as shown here by the distance between them, relative to the bounding box diagonal.

Table 1. Average time (μs) to project one point on the CPU. See additional material for a more complete table.

Model	Points	20 NN	100 NN	MLoDS	*Global*
Face	40881	2.78	5.79	5.82	852
Gargoyle	95435	1.11	4.56	12.7	3.00×10^3
Igea	134346	1.07	4.47	13.0	3.98×10^3
African Statue	220317	2.05	8.28	32.1	1.20×10^4
Bearded Man	499500	1.04	3.60	10.1	9.22×10^3
Eisbar	781865	5.75	14.6	24.6	4.41×10^4
Owl	1031960	5.40	13.2	30.9	6.75×10^4
Rhinoceros	1410356	6.28	18.1	32.9	8.70×10^4
Xyzrgb Dragon	3609455	5.06	7.21	8.65	1.70×10^4
Lucy	14027872	10.9	36.2	146	1.85×10^6

5.1 Accuracy and efficiency

As shown in Fig. 7, our method offers a ***smooth*** approximation of APSS that is controlled by λ: increasing λ provides more accurate results, but with an increased computational cost.

By looking at the tree traversal behaviors between our method and APSS with 20 NNs (Fig. 6), we can see that a NN search is fast only for query points near the input samples. For distant queries, such a search can traverse a very large number of nodes due to branching, whereas our method terminates quickly, as nodes are then deemed "uniform enough" to end the traversal.

Moreover, we observe that the number of traversed nodes depends mostly on the sphere scaling factor λ. Additional experiments (for λ up to 4) suggest that the number of traversed nodes follows a complexity of $O(\lambda^2)$. We find that $\lambda = 2$ gives good results in all our experiments, with no perceptible improvements in the output quality when increasing λ, although this is dependent on the type of kernel that is used.

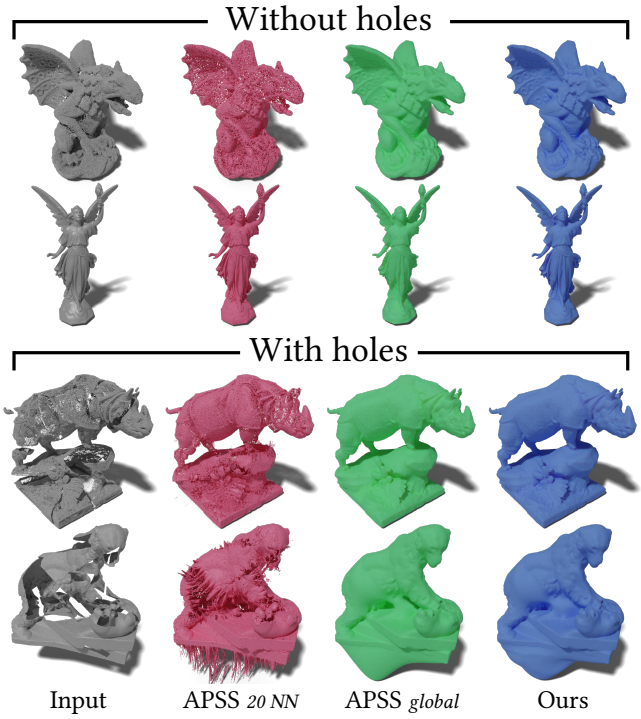


Fig. 8. Compared to standard APSS using 20 NNs, our approach is able to reconstruct a surface on clean data as well as on missing data. Our results are similar to the ones obtained with APSS global (cf. Fig. 7), with a faster computation process. The model of Lucy (line 2) is at a reduced resolution for *APSS global* due to its complexity (cf. Tab 1).

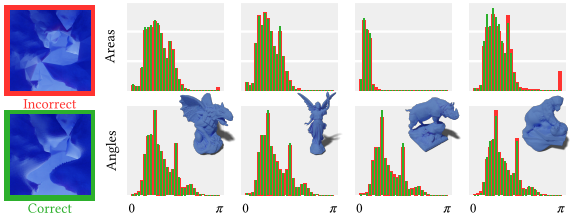
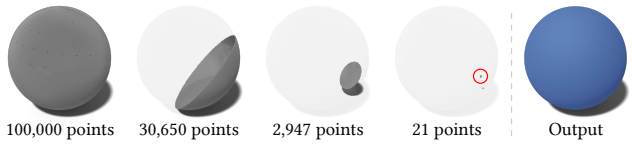


Fig. 9. We show the histograms of triangle areas (top row) and angles (bottom row) for the triangle meshes extracted using our octree (green) and a standard octree (red). The 4 shapes are the ones shown in Fig. 8. We show on the left close-ups of the surface meshes inside the Bear (Eisbar). Note that the last entry for each area histogram captures all remaining areas.

As expected, we observe in practice a complexity of $O(\log(|\mathcal{P}|))$ for a fixed value of λ (Tab. 1).

5.2 Resilience to missing data

We analyze our approach on synthetic point sets and real-life scans featuring uneven sampling and large missing parts, notably by artificially removing parts of the samples on some examples. Our operator remains faithful to global APSS, while the NN approach suffers from obvious instabilities (Figs. 1 and 8). Regarding contouring, our octree refinement process (Section 4) allows us to remain



100,000 points	30,650 points	2,947 points	21 points	Output	
Depth 6					
572.5 *ms*	225.4 *ms*	321.1 *ms*	980.3 *ms*		← *traverse*
351.3 *ms*	55.6 *ms*	200.1 *ms*	189.1 *ms*		← *regularize*
40,387 *nodes*	3,245 *nodes*	23,417 *nodes*	18,690 *nodes*		
Depth 8					
8331.1 *ms*	2034.7 *ms*	4168.4 *ms*	6295.8 *ms*		← *traverse*
6710.8 *ms*	406.4 *ms*	3391.7 *ms*	1994.7 *ms*		← *regularize*
633,963 *nodes*	48,967 *nodes*	369,551 *nodes*	233,532 *nodes*		

Fig. 10. Even for very small portions of the unit sphere, our octree generation algorithm still enables a faithful point cloud upsampling. The traversal time corresponds to our project and divide exploration strategy, while the regularization time includes grading and regularizing the octree.

largely insensitive to input sampling (see Fig. 10). In particular, there is a 1 to 1, 000, 000 variation factor of the input bounding box volume between the shown inputs, although they all yield the same output. This validates that our approach efficiently investigates the 3D space. This is critical, as we cannot in practice make any assumption on the extent of the surface.

We show in Fig. 9 histograms of angles and triangle areas for the models shown in Fig. 8 built with our adaptive octree and a usual octree, both using our adaptive octree root. The angles are mostly distributed around 45 and 90 degrees, which is expected from a dual contouring approach (that extracts quads that are later triangulated). Our approach avoids extremely large triangles on inputs featuring missing data (Bear and Rhino). We do not focus on mesh quality in our work, and existing *remeshing* approaches allow for efficient and fast mesh improvement, provided that an efficient *projection operator* allows reprojecting vertices on the surface [Botsch and Kobbelt 2004]. These approaches preserve the input topology, and require therefore *correct topology* of the input. While we do not guarantee this (which is a research domain on its own), we mostly avoid the artifacts obtained when contouring an implicit surface on an octree prepared using input samples featuring missing data (too coarse in some regions). This point is illustrated in the insets of Fig. 9 (left), which show the mesh connectivity of the Bear Model (Eisbar) using both approaches.

5.3 Robustness to noise

To study how resilient to noise our method is, we artificially degraded artifact-free inputs by introducing noise both on the positions and on the normals of the samples, as well as holes (Fig. 11). Our experiments suggest that our method is more sensitive to position noise than to normal noise, especially in the presence of large holes in the data. This sensitivity remains rather modest, however, even far away from the input samples, contrary to what has been previously reported in the literature for APSS [Berger et al. 2013]. This difference can be explained simply: the nearest neighbor approach only considers a few samples, making it *sensitive* to noisy samples, whereas we rely on *all* samples to fit the algebraic spheres.

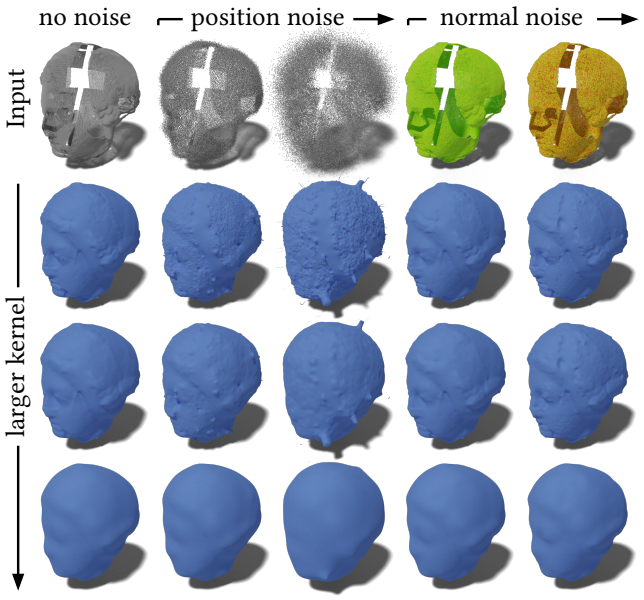


Fig. 11. Results on inputs degraded with a Gaussian noise applied on the positions and normals.

5.4 Application to surface mesh reconstruction

While surface reconstruction is not the main focus of our paper, we briefly compare our meshing results with Screened Poisson Reconstruction (SP) [Kazhdan and Hoppe 2013], Multi-level Partition of Unity (MPU) [Ohtake et al. 2003], and the machine learning based approach Point2mesh [Hanocka et al. 2020] (Fig. 12). Overall, SP yields accurate results, and we achieve similar quality in general. MPU often results in holes or outliers in the output, which is especially visible in Fig.12 on the last two lines. Point2mesh [Hanocka et al. 2020] presents notable artifacts on all reconstructions, despite long computation times. Overall, surface reconstruction methods sometimes struggle on incomplete data and are rather sensitive to input sampling, as shown on Fig. 12 (see more in supplementary material) and pointed out by recent work [Zhao et al. 2021].

5.5 Application to point set filtering

Finally, we showcase our operator for point set *filtering*, an application for which the compared global methods, whose input parameters are precision-related only, are not suitable (Fig. 13).

For completeness, we also compare to the standard kNN-based APSS approach and demonstrate that ultimately, to allow for arbitrary filtering, it is essential to consider the entire input.

Such filtering is key for applications such as multi-resolution deformations [Nader et al. 2014], or point multi-resolution descriptors for data analysis [Mellado et al. 2012]. We expect that our ability to efficiently filter point sets will impact both applications.

6 LIMITATIONS, AND FUTURE WORK

Our approach can be further developed in several ways.

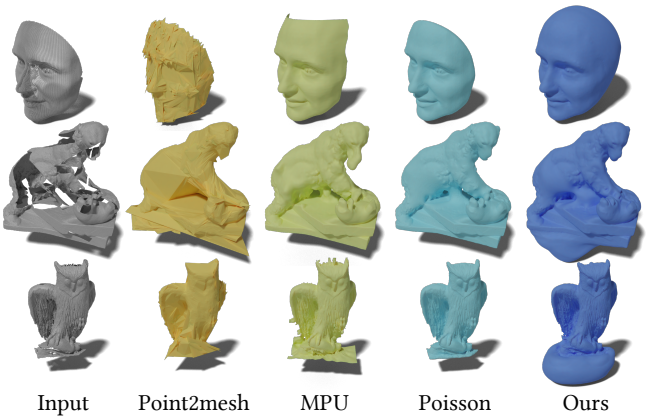


Fig. 12. Comparison of our surface reconstruction with Point2mesh, MPU, and Poisson.

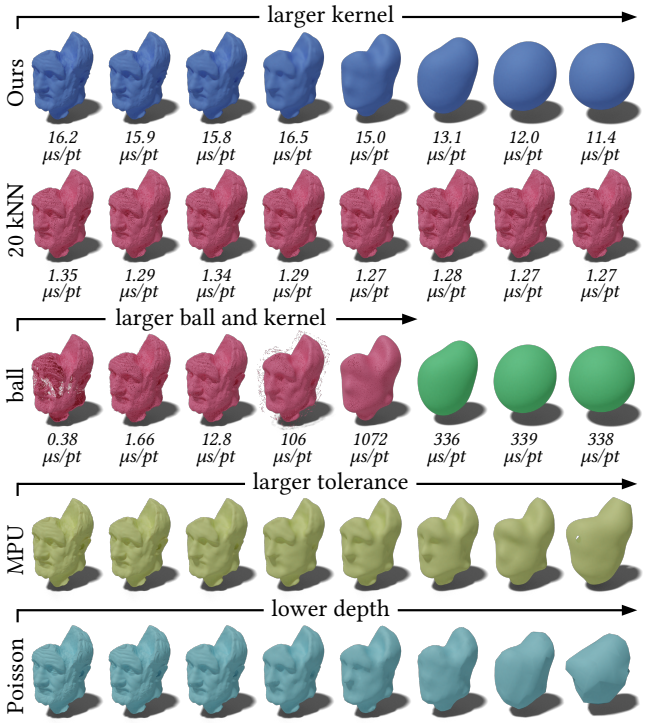


Fig. 13. Our approach allows for efficient large-scale *continuous* filtering of 3D point sets.

First, our traversal criterion only relies on the geometry of the protection spheres and the location of the evaluation point. This criterion could be refined by taking the distribution of the points inside the nodes into account.

Second, while we have studied how large support kernels can be used for the filtering of point sets and their robust surface extraction (in particular, far away from the input samples), our methodology could be used for the parametric design of smooth skeletons. Several existing techniques extract the medial axis of point sets using growing spheres fitting the data [Rebain et al. 2019] or clustering input samples advected inside the surface [Cao et al. 2010; Huang et al. 2013], and we think that our technique can be used for the smooth, parametric fitting of maximally-inscribed spheres as well, thus allowing for interactive shape modeling applications such as skeleton design and interactive quad mesh modeling ([Ji et al. 2010]).

Finally, our approach does not support sharp features or geometric priors, whereas [Guennebaud and Gross 2007] extend their original APSS formulation to handle *creases* and *boundaries* for instance. This sometimes prevents obtaining flat sharp surfaces where might be expected (see the bottom of the Owl and the Bear models in Fig. 8 and 12 for example).

7 CONCLUSION

We have presented *moving level-of-detail Surfaces*, a smooth scheme for efficiently approximating the convolution of multi-dimensional data against a smooth spatial kernel, and used it to compute efficient approximations of Algebraic Point Set Surfaces. We have demonstrated that the use of ad-hoc kernels that are simple to manipulate, such as Gaussian Mixtures and Rational Singular kernels, allows for intuitive continuous point set filtering as well as high-quality surface extraction of input point sets featuring large missing data, which are typical of real-life scanned data. Our approach competes with recent global surface extraction techniques while retaining all advantages of MLS local techniques such as the ability to project any 3D point on the surface in a smooth manner independently. Being entirely agnostic to the input weighting kernel, our approach opens a route to real-time point-based modeling by formulating the problem through the design of parametric kernels modulating the point set surface.

REFERENCES

Marc Alexa and Anders Adamson. 2009. Interpolatory point set surfaces - convexity and Hermite data. *ToG* 28, 2 (2009), 20.

M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. 2001. Point set surfaces. In *Proceedings Visualization, 2001. VIS '01.* 21–29, 537.

Nina Amenta and Yong Joo Kil. 2004. Defining point-set surfaces. In *ToG*, Vol. 23. ACM, 264–270.

Gavin Barill, Neil G Dickson, Ryan Schmidt, David IW Levin, and Alec Jacobson. 2018. Fast winding numbers for soups and clouds. *ToG* 37, 4 (2018), 43.

Matthew Berger, Joshua A Levine, Luis Gustavo Nonato, Gabriel Taubin, and Claudio T Silva. 2013. A benchmark for surface reconstruction. *ToG* 32, 2 (2013), 20.

Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. 2017. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 301–329.

Jules Bloomenthal. 1988. Polygonization of implicit surfaces. *Computer Aided Geometric Design* 5, 4 (1988), 341–355.

Mario Botsch and Leif Kobbelt. 2004. A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 185–192.

Fatih Calakli and Gabriel Taubin. 2011. SSD: Smooth signed distance surface reconstruction. *Computer Graphics Forum* 30, 7 (2011), 1993–2002.

Stéphane Calderon and Tamy Boubekeur. 2014. Point Morphology. *ToG (Proc. SIGGRAPH 2014)* 33, 4, Article 45 (2014), 45:1–45:13 pages.

Junjie Cao, Andrea Tagliasacchi, Matt Olson, Hao Zhang, and Zhixun Su. 2010. Point Cloud Skeletons via Laplacian-Based Contraction. In *Proc. of IEEE Conference on Shape Modeling and Applications'10*. 187–197.

J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. 2001. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In *Proc. SIGGRAPH (SIGGRAPH '01)*. 67–76.

Jiazhou Chen, Gael Guennebaud, Pascal Barla, and Xavier Granier. 2013. Non-oriented MLS Gradient Fields. *Computer Graphics Forum* 32, 8 (Aug. 2013), 98–109.
Tamal K Dey and Jian Sun. 2005. An Adaptive MLS Surface for Reconstruction with Guarantees.. In *Symposium on Geometry processing*. 43–52.
Leslie Greengard and Vladimir Rokhlin. 1987. A fast algorithm for particle simulations. *Journal of computational physics* 73, 2 (1987), 325–348.
Gaël Guennebaud, Marcel Germann, and Markus Gross. 2008. Dynamic Sampling and Rendering of Algebraic Point Set Surfaces. *Computer Graphics Forum* 27, 2 (2008), 653–662.
Gaël Guennebaud and Markus Gross. 2007. Algebraic Point Set Surfaces. In *ACM SIGGRAPH 2007 Papers (SIGGRAPH '07)*. New York, NY, USA.
Thierry Guillemot, Andrès Almansa, and Tamy Boubekeur. 2012. Non Local Point Set Surfaces. In *Proceedings of 3DIMPVT*.
Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. 2020. Point2Mesh: A Self-Prior for Deformable Meshes. *ToG* 39, 4, Article 126 (July 2020), 12 pages.
Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. 2013. L1-medial skeleton of point cloud. *ToG* 32, 4 (2013), 65–1.
Zhiyang Huang, Nathan Carr, and Tao Ju. 2019. Variational Implicit Point Set Surfaces. *ToG* 38, 4 (july 2019), 124:1–124:13.
Zhongping Ji, Ligang Liu, and Yigang Wang. 2010. B-Mesh: A Modeling System for Base Meshes of 3D Articulated Shapes. *Computer Graphics Forum* 29 (09 2010), 2169–2177.
Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. In *ToG*, Vol. 21. ACM, 339–346.
Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Vol. 7.
Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ToG* 32, 3 (2013), 29.
David Levin. 1998. *The approximation power of moving least-squares.* Technical Report. Math. Comp.
David Levin. 2003. Mesh-Independent Surface Interpolation. *Geometric Modeling for Scientific Visualization* 3 (01 2003).
William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM siggraph computer graphics*, Vol. 21. ACM, 163–169.
Wenjia Lu, Zuoqiang Shi, Jian Sun, and Bin Wang. 2018. Surface Reconstruction Based on the Modified Gauss Formula. *ToG* 38, 1 (2018), 2.
Nicolas Mellado, Gaël Guennebaud, Pascal Barla, Patrick Reuter, and Christophe Schlick. 2012. Growing Least Squares for the Analysis of Manifolds in Scale-Space. *Comp. Graph. Forum* 31, 5 (2012), 1691–1701.
Guy M Morton. 1966. A computer oriented geodetic data base and a new technique in file sequencing. (1966).
Patrick Mullen, Fernando De Goes, Mathieu Desbrun, David Cohen-Steiner, and Pierre Alliez. 2010. Signing the unsigned: Robust surface reconstruction from raw pointsets. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1733–1741.
Georges Nader, Gael Guennebaud, and Nicolas Mellado. 2014. Adaptive Multi-scale Analysis for Point-based Surface Editing. *Computer Graphics Forum* 33, 7 (2014), 171–179.
Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. 2003. *Multi-level partition of unity implicits.* Vol. 22.
A Cengiz Öztireli, Gael Guennebaud, and Markus Gross. 2009. Feature preserving point set surfaces based on non-linear kernel regression. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 493–501.
Daniel Rebain, Baptiste Angles, Julien Valentin, Nicholas Vining, Jiju Peethambaran, Shahram Izadi, and Andrea Tagliasacchi. 2019. LSMAT least squares medial axis transform. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 5–18.
Patrick Reuter, Pierre Joyot, Jean Trunzler, Tamy Boubekeur, and Christophe Schlick. 2005. Point set surfaces with sharp features. (03 2005).
Brett Ridel, Gael Guennebaud, Patrick Reuter, and Xavier Granier. 2015. Parabolic-cylindrical moving least squares surfaces. *Computers and Graphics* 51 (June 2015), 60–66.
Vladimir Rokhlin. 1985. Rapid solution of integral equations of classical potential theory. *Journal of computational physics* 60, 2 (1985), 187–207.
Chun-Xia Xiao. 2011. Multi-Level Partition of Unity Algebraic Point Set Surfaces. *J. Comput. Sci. Technol.* 26, 2 (March 2011), 229–238.
Tong Zhao, Pierre Alliez, Tamy Boubekeur, Laurent Busé, and Jean-Marc Thiery. 2021. Progressive Discrete Domains for Implicit Surface Reconstruction. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 143–156.

A SMOOTHNESS OF THE LOCAL PARTITION OF UNITY

This is the proof that the function $F_\Omega(u) = e^{(-e^{(1/(u-1))}/u^2)}$ from Equation 6 is a strictly increasing function with null derivatives at all orders at 0 and 1.

Lemma A.1. *F_Ω admits null derivatives of any order in 0^+ and 1^-.*

Proof. As $F_\Omega(u) = e^{\left(\frac{-e^{\frac{1}{(u-1)}}}{u^2}\right)}$, $F'_\Omega(u) = F_\Omega(u)\frac{2e^{\frac{1}{(u-1)}} - u\frac{e^{\frac{1}{(u-1)}}}{(u-1)^2}}{u^3}$.

It is clear, by recurrence, that derivatives of all orders will be composed of products of $F_\Omega(u)/u^n$ and $\frac{e^{(1/(u-1))}}{(u-1)^m}$, $(n,m) \in \mathbb{N}^2$.

In $u = 0^+$, $F_\Omega(u)$ dominates ($e^{-\infty}$ tends quickly to 0) over $1/u^n$, and thus $F_\Omega^{(k)}(0^+) = 0,\ \forall k$.

In $u = 1^-$, $e^{(1/(u-1))}$ dominates ($e^{-\infty}$ tends quickly to 0) over $1/(u-1)^m$, and thus $F_\Omega^{(k)}(1^-) = 0,\ \forall k$. □

Lemma A.2. *$\gamma_\mu(q)$ is smooth everywhere.*

Proof. For a point $\mathbf{q}$ outside S_μ and inside S_η, we have that: $\gamma_\mu(\mathbf{q}) = \Omega(d_\mu(\mathbf{q}), d_\eta(\mathbf{q}))$, with $\Omega(x,y) = F_\Omega(x/(x-y))$, $d_\mu(\mathbf{q})$ (resp. $d_\eta(\mathbf{q})$) denoting the signed distance from q to the sphere S_μ (resp. S_η). It follows that

$$\partial\gamma_\mu(\mathbf{q}) = F'_\Omega\left(\frac{d_\mu(\mathbf{q})}{d_\mu(\mathbf{q}) - d_\eta(\mathbf{q})}\right) \left[\frac{\partial d_\mu(\mathbf{q})(d_\mu(\mathbf{q}) - d_\eta(\mathbf{q})) - d_\mu(\mathbf{q})(\partial d_\mu(\mathbf{q}) - \partial d_\eta(\mathbf{q}))}{(d_\mu(\mathbf{q}) - d_\eta(\mathbf{q}))^2}\right]$$

By recurrence, it is straightforward to see that derivatives of any order of $\gamma_\mu(\mathbf{q})$ are products of $F_\Omega^{(k)}$, which go to 0 as $\mathbf{q}$ tends to S_μ ($F_\Omega^{(k)}(0^+) = 0\ \forall k > 0$) or when $\mathbf{q}$ tends to S_η ($F_\Omega^{(k)}(1^-) = 0\ \forall k > 0$). The derivatives vanish therefore quickly enough to match the null derivatives of $\gamma_\mu(\mathbf{q})$ at the boundary of the domain ($\gamma_\mu(\mathbf{q})$ is constant for $\mathbf{q} \in S_\mu$ and for $\mathbf{q} \notin S_\eta$). Additionally, it is clear that the derivatives of the term on the right side are all well-posed: The only point q where this scheme could result in a non-differentiable partition of unity is at the center of $\mathcal{S}_\eta$ (where $d_\eta(\mathbf{q})$ is continuous only), but by construction this point is strictly inside $\mathcal{S}_\mu$ and thus outside the domain of utility of F_Ω ($d_\mu(\mathbf{q}) < 0$ and $\gamma_\mu(\mathbf{q}) = 0$ here).

Finally, note that this derivation would remain valid for any other function F_Ω exhibiting the mentioned properties. □

B GPU IMPLEMENTATION

We present here our CUDA implementation of the octree traversal as well as the projection operator.

B.1 Octree traversal

The octree is traversed on the GPU in an unrolled recursion using a while loop. Traversal is done both ways as values computed from the bottom of the tree are needed for the evaluation of the statistics. Each thread computes the statistics corresponding to one point to project, and then compute the projection of the point. This is presented in Alg. 3. An example of a traversal is shown on a binary tree (instead of an octree for a clearer explanation) in Fig. 14. Each node and arrow color corresponds to a line in Alg. 3. When reaching a stopping condition (a leaf, a point outside an external bounding box, or in-between two bounding boxes), necessary elements of the tree are evaluated and then results are added at every level until coming back to the main node, which will contain the correct

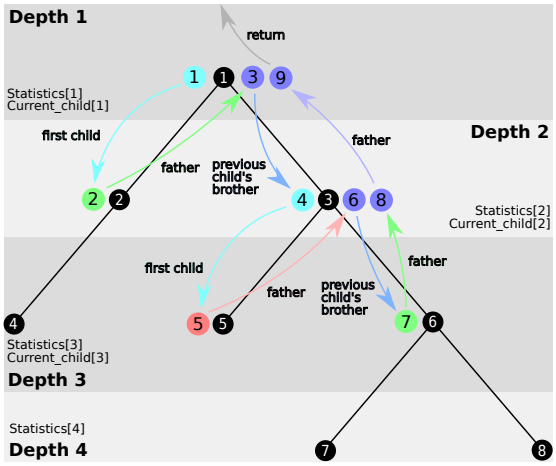


Fig. 14. Example of a tree traversal. Nodes are traversed following the colored arrows. Colors on arrows and nodes correspond to the color of the pseudo-code in Alg.3.

evaluation for the point being projected. This means that each thread needs to store the statistics at each possible depth.

ALGORITHM 3: Stopping conditions and branching of the GPU traversal.

```
Node node = root
Point point = point to project
while true do
    if (node is a leaf) then
        Compute leaf stats
        if (node is the root) then
            return
        else
            node = father
    else
        if (point is far enough from node) then
            Compute node stats
            if (node is the root) then
                return
            else
                node = father
        else
            if (stats from a child μ is known) then
                Blend child μ's and node stats
                if (child has a brother) then
                    node = child's brother
                else
                    if (node is the root) then
                        return
                    else
                        node = father
            else
                node = node's first child
Project points according to the statistics at root depth
```

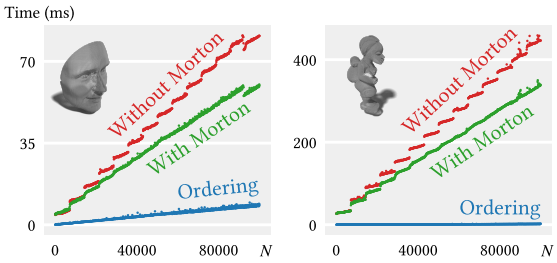


Fig. 15. Projecting N points, with and without ordering points following the Morton order. Even though ordering points using the Morton code is not free, doing so enable us to project points consistently faster.

B.2 Performances and memory coherency

In order to be able to do the traversal, the cumulated statistics and the last child traversed have to be known at each depth (except the child for the leaves). These values are illustrated in Fig.14 with *Statistics* and *Current_child* respectively.

The cumulative 9D statistics of one level are encoded with 9 32-bit floats, requiring 288 bits of local memory per depth. Maximum depth is fixed at 12, which seems to be a good compromise in terms of precision and speed. This means that a single thread needs enough registers for 12 times the statistics and 11 integers for the children. These statistics are necessary at each level. Indeed, the example described in Fig. 14 needs to store the statistics at depth 1 and 2 when going through nodes 3, 5, and 6 as the statistics of node 2 were previously computed and added into Statistics[1].

The octree is stored in global memory in breadth-first order, even if traversal occurs in depth-first order. This is done so that as lots of points are evaluated at the same time, the first threads at a given depth will be loading from global memory values from neighboring nodes that will be likely to be used by other threads. To improve memory latency – which is the bottleneck in our implementation – the kernel parameters and the first depth levels are stored in shared memory. It does not result in a major improvement but we still project points a little faster by about 0.7%.

When projecting many points in parallel, we first sort them using the Morton code [Morton 1966], to favor traversal coherency among concurrent threads. Figure 15 shows the average time on 10 iterations of the GPU projection using the Morton order and not using it. We notice a great improvement using the Morton order, even when including the time to compute it on the CPU. However, when projecting a reduced number of points (less than 10000 points), it is a little faster not to order the points, which seems natural as almost all points will be projected at the same time, so their order does not matter.