Author's Draft. Final version published in COMPUTER GRAPHICS Forum (2019) doi:10.1111cgf.13597. COMPUTER GRAPHICS *forum*

Filtered Quadrics for High Speed Geometry Smoothing and Clustering

Hélène Legrand Jean-Marc Thiery Tamy Boubekeur

LTCI, Telecom ParisTech, Paris-Saclay University

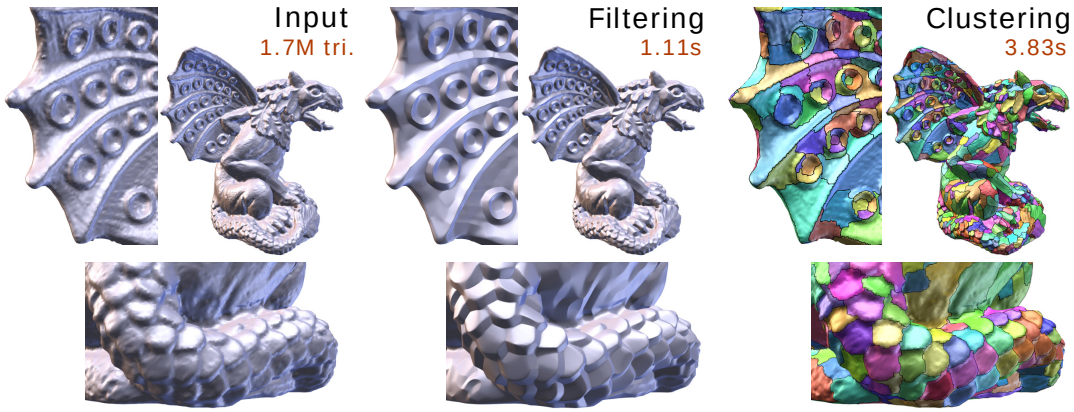


Figure 1: Enriching surface meshes with per-vertex filtered quadrics provides a compact local model of the geometry, enabling efficient feature-preserving smoothing and fast clustering. ($\sigma_s = 3.0$, $\sigma_r = 1.0$, $\sigma_d = 6.0$, $\sigma_q = 1.0$, $\theta = 0.95$. 1119 clusters.)

Abstract

Modern 3D capture pipelines produce dense surface meshes at high speed, which challenge geometric operators to process such massive data on-the-fly. In particular, aiming at instantaneous feature-preserving smoothing and clustering disqualifies global variational optimizers and one usually relies on high performance parallel kernels based on simple measures performed on the positions and normal vectors associated with the surface vertices. Although these operators are effective on small supports, they fail at properly capturing larger scale surface structures. To cope with this problem, we propose to enrich the surface representation with filtered quadrics, a compact and discriminating range space to guide processing. Compared to normal-based approaches, this additional vertex attribute significantly improves feature preservation for fast bilateral filtering and mode-seeking clustering, while exhibiting a linear memory cost in the number of vertices and retaining the simplicity of convolutional filters. In particular, the overall performance of our approach stems from its natural compatibility with modern fine-grained parallel computing architectures such as graphics processor units (GPU). As a result, filtered quadrics offer a superior ability to handle a broad spectrum of frequencies and preserve large salient structures, delivering meshes on-the-fly for interactive and streaming applications, as well as quickly processing large data collections, instrumental in learning-based geometry analysis.

CCS Concepts

•***Computing methodologies*** → ***Mesh models; Mesh geometry models;***

1. Introduction

The fast processing of 3D surface meshes is a growing demand for large data collection analysis, on-the-fly geometry capture and live use or interactive shape editing, among other applications. Under performance constraints, global optimization, numerical solvers and many of the traditional tools of geometry processing may not be available and clearly fall out of this application spectrum when dealing with multi-million triangle meshes.

In particular, most existing filtering and clustering methods report experiments taking seconds to minutes on models made of a few (sometimes tens of) thousands of polygons. In the current era of massive geometry generation, both in polygon count and instances, high speed filtering and clustering methods are becoming

highly instrumental to the graphics chain. This is particularly true in two classes of emerging application scenarios: large collection processing, for further analysis (e.g., learning) and live 3D capture and broadcast for which the available processing time is bounded by the target streaming update delay.

Indeed, for basic processing primitives such as filtering and clustering, the time constraint forces algorithms to rely on weak geometric measures such as euclidean distance between samples and tangent space similarity. One of the reasons such simple measures are used in real-time or interactive processing pipelines is related to the surface mesh format: position and normal vectors are always maintained in the mesh processing chain, and are almost the only quantities on which the geometric operators can rely.

We argue that *error quadrics*, traditionally used as temporary measures for surface simplification, can act as additional vertex attributes when they are filtered, to enrich the surface description with a knowledge of larger scale structures, so that fast processing operators can still behave as usual – gathering and convolving/averaging/comparing/etc a local set of neighboring samples – while maintaining high performance and scalability.

We propose a new geometric filter designed to run at high speed on large input 3D surfaces while being able to preserve from small to large features (Sec. 3), generating very smooth surface regions under a user-controlled threshold. Our approach is based on an enrichment process which establishes error quadrics as a new type of range attribute, characterizing locally the surface geometry, at each vertex. This extra attribute (i) has a fixed memory footprint, whatever the considered scale, (ii) can be used to guide surface filtering thanks to its intrinsic optimizer and (iii) copes well with parallel kernel evaluation on fine-grained architectures such as GPUs. As a result, we obtain a non-iterative local filter that runs at a similar speed to simple bilateral mesh filters, but with results which are similar to the output of high-quality variational methods that perform global iterative optimizations and run orders of magnitude slower.

While, so far, practitioners typically perform geometric filtering in a preprocess, store the resulting mesh, and eventually load it later for further use in different applications, our method renders possible a more flexible workflow, with the parameters of the operator being the only data that need to be stored/loaded/preserved, since the actual filtered/clustered geometry is generated from the raw data on demand, in split-seconds. Not only this enables the aforementioned scenarios, but this also helps delaying the decision on the filtering scale to the final application, a single model being subject to various levels of filtering depending on the final application constraints, e.g., optimization stability, feature line extraction, signature computation, etc. This typically avoids maintaining multiple versions of the model, as only the raw input is then required.

Moreover, the ability of our quadric surface fields to strongly regularize geometry under a given similarity threshold, while enforcing very sharp structures between smooth patches in the filter output, makes it a good candidate for fast mode seeking. We explore such an application in a new parallel clustering algorithm (Sec. 4) that extends the quick shift method, and produces clusters of similar quality than offline methods such as superfacets [SPDF14].

Both our contributions are designed for efficient fine-grain parallel execution on GPU (Sec. 5) and are very simple to implement. This natural compatibility with modern graphics architectures makes our methods extremely fast to execute, which is their main appeal. We evaluate our work on scanned meshes exhibiting several millions of primitives and the typical defects of acquired models, such as non-manifoldness and holes (Sec. 6). Considering modern 3D capture pipelines, we observe that their reconstructed output meshes present an amount of noise which typically corresponds to what our filter can handle, while at the same time, exhibit a polygon count that requires the speed and scalability that our method provides.

2. Background

Quadric Error Metric The Quadric Error Metric (QEM) was introduced by Garland and Heckbert in their seminal article dealing with triangle mesh decimation [GH97], to obtain high quality simplification of dense meshes quickly. This metric describes the integral over a given 2D-domain of the squared distance from a 3D point p to a set of tangent planes $\{(q,n)\}$. Noting that this squared distance can be expressed as $(p^T,1)Q_{(q,n)}(p^T,1)^T$, with

$$Q_{(q,n)} = \left[\begin{array}{c|c} nn^T & -n \\ \hline -n^T & (n\cdot q)^2 \end{array}\right],$$

the quadric associated with a given vertex v (its barycentric cell) can be obtained by averaging quadrics of its adjacent triangles $t_j \in T_1(v)$ with area a_j, center c_j and normal n_j:

$$Q_v := \sum_{t_j \in T_1(v)} \frac{a_{t_j}}{3} Q_{(c_j,n_j)}. \tag{1}$$

Interestingly, the minimizer $p^* = argmin_p (p^T,1)Q(p^T,1)^T$ of a quadric Q is given by $p^* = -A^{-1}b$, where

$$Q =: \left[\begin{array}{c|c} A & b \\ \hline b^T & c \end{array}\right].$$

By minimizing the quadric of a given region, we obtain a 3D point best fitting it in this sense. As the quadric of the union of two regions is simply given by the sum of their quadrics due to integrals additivity, Garland and Heckbert devise an efficient mesh simplification algorithm, where edges of minimal associated error are greedily decimated. This framework allows obtaining a multiresolution description of the input [Hop96] that can be traversed in real-time, and is general enough to accommodate other kinds of simple approximating primitives, e.g., such as spheres [TGB13]. It is also generally the first choice for simplifying *mesh animations* [DR05, KG05, LS09, ZZW10, TGBE16], as the memory and time complexities scale linearly in the length of the animation.

Bilateral Filtering The simplest way to denoise an image or a mesh is to use a low-pass filter, which was introduced for discrete surfaces by Taubin [Tau95]. Desbrun et al. [DMSB99] observe that this Laplacian smoothing operator can be seen as an integration over time of the heat equation, and present a method adapted to

irregular meshes. One drawback of this type of approach is that they do not distinguish between noise and high-frequency features, and both are smoothed indiscriminately. Several methods aiming at preserving these features have been developed, including algorithms based on anisotropic diffusion [DMSB00, CDR00], and a number of techniques using a normal filtering step before updating vertex positions accordingly [OBS02, YOB02, YOB03, SRML07, ZFAT11]. In the following, we focus on approaches related to the bilateral filter.

The bilateral filter for images was introduced by Tomasi et al. [TM98]. In this method, samples are filtered by averaging their neighbors values using two weighting terms: the first uses the spatial information, i.e., the distance between the sample and its neighbors, and the second takes into account the difference in signal, or range. In other words, each pixel becomes a weighted average of its similar neighbors. Adapting this technique to 3D surfaces is not straightforward, since for images, the signal and the position of a sample are separate, while for meshes the signal itself is part of the spatial information. Several methods were proposed to apply a bilateral filter to vertex positions [JD03, FDCO03], and later to surface normals for denoising [ZFAT11]. We focus here on the formulation of Jones et al. [JD03].

In this work, a local estimation of the shape is used to smooth the surface. A vertex position is computed by averaging predictions, obtained from the triangles in the neighborhood. For a vertex v and a triangle t_i of its neighborhood η, this prediction $P_i(v)$ is defined as the projection of v on the tangent plane of t_i. This tangent plane is defined by a center c_i and a smoothed normal n_i:

$$P_i(v) = v + n_i((c_i - v) \cdot n_i).$$

The filtered vertex v' is computed as an average of these predictions, weighted by a spatial term, which is function of the distance to the neighbor $\|v - c_i\|$, and a range term, which is function of the distance to the prediction $\|v - P(v)\|$ (figure 2):

$$v' = \frac{1}{w(v)} \sum_{t_i \in \eta} P_{t_i}(v) a_{t_i} G_{\sigma_s}(\|v - c_i\|) G_{\sigma_r}(\|v - P_{t_i}(v)\|),$$

where a_t is triangle t's area, and $w(v)$ is a normalization factor:

$$w(v) = \sum_{t_i \in \eta} a_{t_i} G_{\sigma_s}(\|v - c_i\|) G_{\sigma_r}(\|v - P_{t_i}(v)\|).$$

G_{σ_s} and G_{σ_r} are the Gaussian kernels used for the spatial term and the range term respectively. The parameters σ_s and σ_r control the amount of smoothing of the surface. To make the predictions more robust to noise, the normals are smoothed in a preprocessing step, with a Gaussian kernel of standard deviation $\frac{\sigma_s}{2}$.

Its ability to preserve sharp features, its local nature, and its simplicity, make the bilateral filter a reference in terms of smoothing and denoising. This led to a number of acceleration methods, such as bilateral grids [CPD07], Gaussian kd-trees [AGDL09], or a separable approximation of the filter [VB11].

Still inspired by image denoising, the non-local means filter [BCM05] was the starting point of several adaptations to 3D surfaces [YBS06, GAB12]. In this family of techniques, the aim is to use, among all the vertices, those with the neighborhood closest to the vertex to filter, independently of their position. He et al. [HS13] use a L_0 minimization in order to denoise, while favoring piecewise flat shapes. This technique is therefore well adapted for CAD models, but may not be suitable for more organic shapes. Fan et al. [FYP10] locally fit quadric surfaces, and cluster the surface into piecewise smooth subneighborhoods. The normals and curvature tensors are then filtered within these neighborhoods, before denoising the vertex positions with a second order bilateral filter. A mean shift filtering is proposed by Solomon et al. [SCBW14], as an iterative application of a generalized version of the bilateral filter on meshes. More recently, Wang et al. [WFL*15] propose to filter small scale geometric details by applying iteratively a joint bilateral filter to the normals [KCLU07], following the idea developed by Zhang et al. for images with the rolling guidance filter [ZSXJ14]. In this work the authors do not target denoising but geometric texture filtering, and sharp features are not recovered in the case of very noisy models. A bilateral filtering step on the normals is also used by Zhang et al. [ZDZ*15] as a first step in their iterative method. The question is approached from a different angle by Wang et al. [WLT16], who use neural networks to learn the relation between noisy geometry and its ground truth version. Yadav et al. [YRP17] propose an iterative denoising method that successively processes normals and vertex positions, and introduce the concept of element-based normal voting tensor for smoothing. Most of these methods target offline high-quality denoising and are not usable in practice for interactive application scenarios.

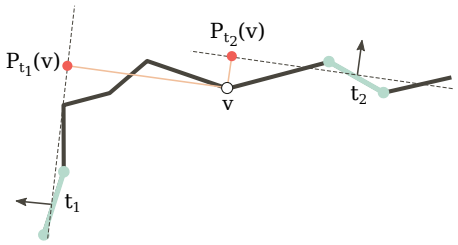


Figure 2: Predictions for the bilateral filter (smoothed normals). The prediction given by the neighbor t_2 on the same side of the sharp feature is closer to v than the prediction given by t_1 on the other side. It will therefore have more weight.

Nociar et al. examined the use of the QEM in the context of denoising [NF10]. After a first step of bilateral filtering on the normals, a quadric is associated with each vertex, as the sum of the quadrics of the incident triangles, computed with the smoothed normals. Each vertex is then iteratively moved, by minimizing its quadric, and updating it with the new positions. Vieira et al. [VNMC10] propose a simple QEM-based algorithm, also relying on a quadric per vertex. This quadric is computed as the sum of the fundamental quadrics in a first or second order neighborhood, weighted by the distance to the neighbor. The filtering is then applied in a single step, where each vertex is moved to the point minimizing its quadric. This simple algorithm preserves sharp features at the scale of the neighborhood, and enhances them to a certain extent. It also shows the QEM's ability to locally describe the surface in a robust manner, by recovering some details in very noisy inputs. However, this method is limited to small neighborhoods (second order in practice), and gives no control on the scale of the preserved

features. Moreover, the neighborhood definition makes the results sensitive to the tessellation and regularity of the input mesh.

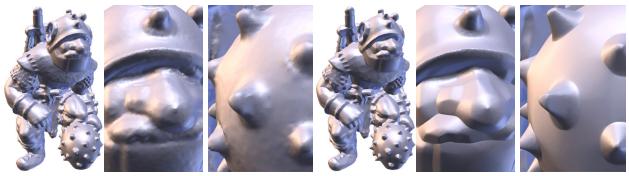
Mode seeking and clustering 3D partitioning techniques usually fall in one of two main categories. One is a more semantic approach, aiming at segmenting the object into meaningful parts. We will focus here on the other category, where the goal is to extract small regions based on geometric criteria.

Garland et al. [GWH01] propose an algorithm producing a hierarchy of regions based on a metric derived from the QEM. The faces are iteratively contracted pairwise, creating mostly planar regions. Variational Shape Approximation (VSA) [CSAD04] is a k-means-type algorithm that fits planar proxies to the geometry. Each iteration consists of a region growing phase based on the $L^{2,1}$ metric, and the computation of a proxy by region until convergence.

The superpixel concept was introduced by Ren et al. [RM03], based on the observation that pixels are the consequence of the discrete representation of images, and are not necessarily a natural primitive in many cases. Moreover, even at low resolutions, their number can make them difficult to handle. To reduce the number of elements to process, the pixels are grouped together locally, in a way that is consistent with the structures in the image. This pre-processing step should be as simple, fast, and scalable as possible. Several techniques were proposed to generate them, using in particular the Mean Shift algorithm [CM02], k-means [ASS*12], or the Quick Shift algorithm [VS08]. Superpixels have been used to accelerate and improve the results of many image processing algorithms, such as object detection or segmentation [HZR06, MPW*09, ZSL*13]. Simari et al. [SPDF14] build upon the SLIC superpixel method [ASS*12] to compute superfacets efficiently on large 3D meshes, with a k-means approach.

The Mean Shift algorithm is a general mode seeking and partitioning method handling N-dimensional data. Intuitively, the idea is to move each sample iteratively to maximize an underlying density function defined locally, usually as a sum over all the samples weighted by a kernel centered on the current sample. Every sample is moved iteratively in the direction of the gradient, towards regions with a higher density until convergence. The resulting points are the modes, and a partition of the initial data can be obtained by associating each point with its corresponding mode. Yamauchi et al. [YLL*05] use a Mean Shift on 6D points, considering the triangles centroid and their normals. The normals are then updated with the normals obtained for the modes. The second phase of the algorithm alternates between a region growing step based on the computed normals, and the updating of the centers.

The Medoid Shift [SKK07] is an algorithm similar in concept to the Mean Shift, where the possible movement of a sample is constrained to the position of another sample. This modification has the advantage of making the Medoid Shift non-iterative: by connecting each data point with a "next" point, one obtains the full paths to the modes. The need to choose a stopping criterion also disappears. Vedaldi et al. [VS08] observe that using the gradient is not the only option to converge towards modes. They introduce the Quick Shift algorithm, that connects every point to its closest neighbor with a higher density estimate. This method connects the data points into a single tree, from which a partition can be obtained by cutting the branches longer than a threshold. The modes are the roots of the resulting sub-trees. This algorithm was used notably in the context of superpixels [VS08], for curvature-based mesh segmentation [LKK15], and is very well suited to the GPU [FS10]. We propose to adapt this algorithm to use *filtered quadrics*, in order to produce a superpixel-like clustering of 3D surfaces.



input mesh $\sigma_s = 4, \sigma_r = 0.4$

Figure 3: **Quadric filtering** of the Grog model (1M tri.), removing small and medium details while preserving large features in 0.46s.

3. Quadric Filter

Given a surface mesh $M = \{V, T\}$ as input, with $V = \{v_i\}$ a set of vertices in $\mathbb{R}^3$ and $T = \{t_i\}$ a set of triangles indexing V, our approach is based on two observations. First, error quadrics offer a natural geometric range space to enrich the surface, as 2 neighboring vertices of the same geometric nature i.e., surrounded by similar local structures, are likely to have very close tangent plane distributions and therefore error quadrics. Second, such an additional attribute not only helps measuring similarity between surface vertices, but also guiding their specific smoothing process. This second observation is key in the general design of our filtering algorithm while the first one dictates how we enrich the input. In a nutshell, our algorithm first tailors a target quadric field on the input surface before conforming the actual geometry to it (see Fig. 3 for an example) and works as follow:

1. generate a **quadric field**, by computing a local error quadric for each vertex at a prescribed scale;
2. perform an **anisotropic diffusion** on this field on the surface to obtain an *enriched* guiding field that keeps important salient structures while strongly filtering any detail smaller than a prescribed threshold;
3. use the field as a **guide** for updating each input surface vertex, accounting for both its associated filtered quadric and the surface structure.

Each step of this algorithm takes the form of a small set of parallel kernels, designed for efficient GPU execution. The whole filter is controlled by three parameters:

- σ_b, which specifies the finest level of detail that each quadric should capture,
- σ_s, which specifies the "smoothness" of the filter i.e., the spatial support of the quadric diffusion process,
- σ_r, which specifies the "featureness" of the filter i.e., the range support of the quadric diffusion process.

Quadric field. Similarly to Vieira et al. [VNMC10], we start by computing one base quadric Q_v per input vertex v, as the area

weighted sum of the incident triangles quadrics. We then estimate a per-vertex quadric Q'_v which accounts for the local neighborhood of each vertex v:

$$Q'_v = \frac{1}{w(v)} \sum_{p \in \eta_{\sigma_b}} Q_p a_p G_{\sigma_b}(\|v-p\|)$$

with a_p the area associated with p, computed as the sum of the areas of the incident triangles, η_{σ_b} the spatial neighborhood with a radius depending on σ_b (we use $2\sigma_b$) and $w(v)$ the normalization term i.e., sum of the weights:

$$w(v) = \sum_{p \in \eta_{\sigma_b}} a_p G_{\sigma_b}(\|v-p\|)$$

This per-vertex quadric provides a good characterization of the shape at the scale of η_{σ_b}, in the form of an optimization space in which lies a good "average" geometric sample, which differs from their use as scalar field approximations of the shape, such as done by Fan et al. [FYP10] for instance. The key idea of the second step of our filter is to modulate this optimization space to later better search for a surface embedding that is both smoother and (large) feature-preserving.

Quadric diffusion. The key concept in our approach is to guide the surface filtering process using the quadric field as a control mechanism. Consequently, we aim at modifying this field in a way that later reflects in a feature-preserving surface smoothing, while still coping with high performance and natural parallel execution. To do so, we propose to compute an *anisotropic diffusion* of the quadrics in the form of a bilateral filter applied on them. On the contrary to bilateral mesh filtering which relies on a geometric prediction (tangent projection), our filter acts on an object where spatial (surface) and range (quadrics) domains are clearly separated. We account for both spatial proximity between the quadrics host vertices and range similarity of the quadric geometry. More precisely, for each vertex v, we compute its filtered quadric Q_v^{bl} as:

$$Q_v^{bl} = \frac{1}{w(v)} \sum_{p \in \eta_v} a_p G_{\sigma_s}(\|v-p\|) G_{\sigma_r}(\sqrt{Q'_p(v)}) Q'_p$$

The range extent σ_r acts on the similarity measure that we model with our error quadrics. We propose to estimate whether v is close to the optimizer associated to a neighbor p by simply evaluating the quadric of p at location v i.e.,

$$Q'_p(v) := (v^T|1) Q'_p \begin{pmatrix} v \\ 1 \end{pmatrix}.$$

Fig. 4 illustrates how quadrics act as range terms for bilateral filtering: points located on the same side of a sharp feature account more in the averaging than points located across the feature, thus better preserving it in the filtering.

Guided surface motion. So far, the actual surface geometry has not been modified and only the quadric field has been processed. With our per-vertex bilateral quadrics in hand, we can now guide the filtering process and optimize for each vertex in parallel. More precisely, our filter can run in two modes. In constrained mode, for each vertex v we search for the filtered position v', which minimizes the filtered quadric Q_v^{bl} along a filtered normal direction n, which is

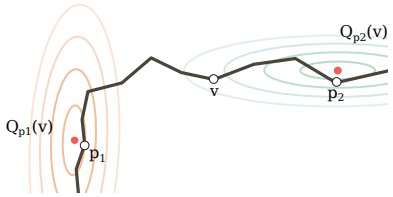


Figure 4: 2D representation of quadrics isocontours. The quadric error modeled by Q_{p_1} at v is stronger than Q_{p_2}. The neighbor p_1, which is on the other side of the sharp edge w.r.t. v, has therefore less influence than p_2, which is on the same side.

the result of normal smoothing process based on a Gaussian kernel of standard deviation σ_n. This boils down to computing:

$$v' = v + \lambda n \text{ , with}$$

$$\lambda = \frac{-(v^T A n + b^T n)}{n^T A n} \quad \text{and} \quad Q_v^{bl} =: \left[\begin{array}{c|c} A & b \\ \hline b^T & c \end{array}\right].$$

On the contrary to the actual minimizer of the quadric, this normal-constrained strategy preserves better the triangles shape and avoids artifacts such as pinching and triangle flips that may come from distant vertices having similar quadrics. In unconstrained mode, the vertex is simply put at the location of the filtered quadric minimizer as described previously. This mode enhances better strong features at the cost of triangle shape regularity (see Fig. 5). This is the main mode we use in the experiments reported in this paper.

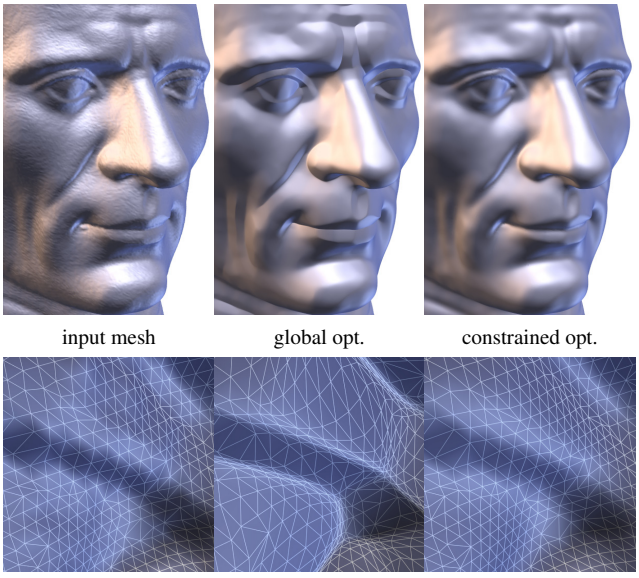
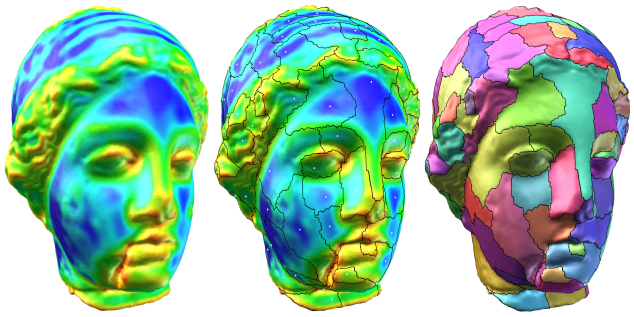


Figure 5: Optimal versus normal-constrained vertex placement (same parameters $\sigma_s = 3$ and $\sigma_r = 1$). The global minimizer makes the geometry move towards the features, while the constrained minimizer better preserves triangle shapes.



score modes and regions final partition

Figure 6: **Representativity score**, from blue for a high score to red for a low score. Middle : clusters frontiers (in black) and modes (white dots). Right : resulting partition. $\sigma_s = 2.0$, $\sigma_r = 1.0$, $\sigma_d = 6.0$, $\sigma_q = 1.0$ and $\theta = 0.87$. 156 clusters.

4. Quadric Clusters

Beyond geometric filtering, our bilateral quadric field can be instrumental for several applications that both require ignoring features that are smaller than a prescribed scale and preserving large ones. In particular, instantaneous surface clustering into regions of similar shape share similar objectives with feature-preserving filtering. Inspired by super-pixels in image/video processing, such methods aim at decomposing dense meshes into small clusters at which level various computations can be factorized, with large salient structures delimiting cluster boundaries. For instance, the *superfacets* of Simari et al. [SPDF14] provide such a decomposition and we propose to address this problem under high-speed constraints. Our key idea is to exploit our quadric field to group together vertices of similar geometric signature, forming what we call *quadric modes*. To achieve such a clustering in a fine-grained parallel fashion, we reformulate a fast error-driven mode extraction algorithm with our enriched surface attribute in a 2-step clustering where we:

1. compute a representativity score for each vertex, capturing how well its filtered quadric is in accordance with the neighboring geometry,
2. search modes based on this score using an evolution of the *Quick Shift* partitioning method.

Representativity score. Most mode seeking algorithms start by computing a Parzen density estimation P at each point x:

$$P(x) = \frac{1}{N}\sum_{i=1}^{N} K(x - x_i),$$

where K is typically a Gaussian kernel. All data points are then progressively moved toward the maxima of this density and modes are extracted at convergence locii. Considering our enriched 3D surfaces, we ideally seek for a small subset of V, whose associated quadrics properly capture the entire surface. The Quick Shift algorithm [VS08] then appears as an adequate partitioning strategy since, given a vertex v, it only requires finding the closest vertex v_i s.t. $P(v_i) > P(v)$. We base our clustering on this algorithm but substitute the Parzen density with a representativity score S_v that estimates how well Q_v^{bl} captures v's spatial neighborhood $\eta_{\sigma_{d_1}}$:

$$S_v = \frac{1}{\sum a_i} \sum_{v_i \in \eta_{\sigma_d}} a_i G_{\sigma_d}(\|v - v_i\|)\, G_{\sigma_q}\left(\sqrt{Q_v^{bl}(v_i)}\right),$$

where a_i is the area of the neighborhood of v_i, $Q_v^{bl}(v_i)$ is the evaluation of the filtered quadric at v_i, G_{σ_s} is a Gaussian kernel of standard deviation σ_s, and σ_d (resp. σ_q) modulates the importance of the spatial (resp. range) proximity. The value S_v is high when the strongest error caused by neighboring vertices w.r.t. the quadric of v is low i.e., when the quadric fits well the neighborhood of v. We illustrate in Fig. 6 an example of this score.

Mode extraction. With our per-vertex representativity score in hand, we now seek for modes by adapting the Quick Shift algorithm. In particular, to obtain a clustering which reflects the similarity measure we designed through our filtered quadrics, we define the proximity measure p in the bilateral space of the quadric field:

$$p(v, v_i) := G_{\sigma_d}(\|v - v_i\|) G_{\sigma_q}\left(\sqrt{Q_{v_i}^{bl}(v)}\right).$$

Given this proximity measure, we can now link each vertex v to the neighbor v_i that (i) satisfies $S_{v_i} > S_v$ and (ii) maximizes $p(v, v_i)$. This *linking* process forms a tree structure on the surface (see Fig. 7 for a visualization of this structure), where only the vertices with the higher values S are not linked to other ones. We label each branch $\{v_i, v_j\}$ with the proximity $p(v_i, v_j)$ that generated it. To extract a partition from the so-defined modes, we cut all tree branches with proximity label smaller than a threshold θ, which leads to forest of trees, where each root is a mode (see Alg. 1). Each vertex of V can finally be associated to the cluster of its mode by traversing its chains of links until finding its root (see Alg. 2).

5. GPU Implementation

Each stage of our approach has been designed to maximize fine-grain parallelism and scalability. In this section, we detail the key

input mesh partition trees

Figure 7: **Links** established by the Quick Shift procedure. Each tree corresponds to a region. The nodes luminances (right) indicate their number of descendants (lighter node have more descendants). $\sigma_s = 2.0$, $\sigma_r = 1.0$, $\sigma_d = 5.0$, $\sigma_q = 1.0$ and $\theta = 0.75$. 142 clusters.

Algorithm 1: Parallel Quick Shift tree construction.

```
forall vertex v ∈ V in parallel do
    pmax ← 0
    next[v] ← v
    // tree creation
    forall neighbors v_i of v do
        if p(v, v_i) > pmax and S_{v_i} > S_v then
            pmax ← p(v, v_i)
            next[v] ← v_i
        end
    end
    // branch cutting
    if p(v, next[v]) < θ then
        next[v] ← v
    end
end
```

Algorithm 2: Parallel per-vertex mode search.

```
forall vertex v ∈ V in parallel do
    current ← v
    parent ← next[v]
    while current ≠ parent do
        current ← parent
        parent ← next[current]
    end
    mode[v] ← current
end
```

components of our GPU implementation. We implemented our approach in C++ using the Nvidia CUDA SDK for GPU execution, together with the Thrust library for basic GPU algorithms. Reported experiments were conducted on an Intel Xeon E5-1620 CPU at 3.9Ghz, and a NVidia Geforce GTX 980 Ti GPU.

In a nutshell, our filtering algorithm consists of 3 GPU passes, each of them taking the form of a CUDA kernel:

1. base quadric computation (Q_v), with one thread per triangle;
2. neighborhood quadric computation (Q'_v), with one thread per vertex;
3. quadric diffusion and vertex position update, with one thread per vertex.

The clustering algorithm starts with the same filtering steps, without updating the vertex positions. Additionally, 3 passes are performed to generate the partition:

1. representativity score computation, with one thread per vertex;
2. linking, with one thread per vertex;
3. mode finding, with one thread per vertex.

Error quadrics are particularly convenient to manipulate on the GPU, as they locally summarize the geometry in a compact way and can be combined easily by summing them up. We store each quadric as an array of 10 floating point values (4x4 symmetric matrix). In the first step of our algorithm, the base vertex quadrics Q_v associated with each vertex v are computed in a single kernel.

Model	Method	Parameters	RMSE	MFNE
Fandisk (12.9K tri.)	[JD03]	($\sigma_s=2.0$, $\sigma_r=1.5$)	0.002206	0.148220
	[SRML07]	($T=0.5$, $n_1=30$, $n_2=20$)	0.001466	0.046387
	[ZFAT11]	($\sigma_s=0.35$, $\lambda=0.0001$, $v=20$)	0.001022	0.037008
	[HS13]	(default)	0.005094	0.195904
	[ZDZ*15]	($r=2x$, $\sigma_r=0.35$, $k_{iter}=20$, $v_{iter}=15$)	0.000818	0.039276
	Ours	($\sigma_s=2.0$, $\sigma_r=1.0$)	0.001536	0.120709
Bunny (69.6K tri.)	[JD03]	($\sigma_s=2.0$, $\sigma_r=2.0$)	0.000869	0.103577
	[SRML07]	($T=0.5$, $n_1=5$, $n_2=20$)	0.000654	0.102727
	[ZFAT11]	($\sigma_s=0.3$, $\lambda=0.001$, $v=15$)	0.000743	0.117505
	[HS13]	(default)	0.000771	0.131686
	[ZDZ*15]	($r=2x$, $\sigma_r=0.55$, $k_{iter}=5$, $v_{iter}=15$)	0.000458	0.091487
	Ours	($\sigma_s=2.0$, $\sigma_r=0.7$)	0.000725	0.099304
Caesar (772K tri.)	[JD03]	($\sigma_s=3.5$, $\sigma_r=3.0$)	0.000314	0.121606
	[SRML07]	($T=0.6$, $n_1=12$, $n_2=12$)	0.000165	0.117080
	[ZFAT11]	($\sigma_s=0.45$, $\lambda=0.0001$, $v=10$)	0.000161	0.116400
	[HS13]	(default)	0.000205	0.121308
	[ZDZ*15]	($r=2x$, $\sigma_r=0.35$, $k_{iter}=20$, $v_{iter}=20$)	0.000170	0.119206
	Ours	($\sigma_s=4.0$, $\sigma_r=1.0$)	0.000185	0.117429
Gargoyle (1.72M tri.)	[JD03]	($\sigma_s=4.0$, $\sigma_r=3.0$)	0.000417	0.242417
	[SRML07]	($T=0.4$, $n_1=15$, $n_2=15$)	0.000188	0.227175
	[ZFAT11]	($\sigma_s=0.35$, $\lambda=0.001$, $v=10$)	0.000159	0.195790
	[HS13]	(default)	0.000208	0.149341
	[ZDZ*15]	($r=2x$, $\sigma_r=0.35$, $k_{iter}=15$, $v_{iter}=15$)	0.000150	0.135382
	Ours	($\sigma_s=4.0$, $\sigma_r=1.0$)	0.000226	0.172294

Table 1: Parameters, RMSE and Mean Face Normal Error (or MFNE, reported in radians) measured for models from Fig. 8.

For every triangle in parallel, the (planar) quadric of the triangle, weighted by its area, is accumulated on its incident vertices. We achieve this summation in a single pass using *atomic* operations, which are well supported on modern GPU architectures.

To obtain the quadric Q'_v accounting for the local neighboring geometry, the parallel computation is not as straightforward, as we need to collect quadrics for all vertices in the neighborhood. We address this problem first by considering a purely spatial neighborhood over a topological one. This makes possible to use an efficient fixed-radius nearest neighbor search algorithm, based on a uniform grid partition, as described by Hoetzlein [Hoe14]. Additionally, this allows to handle non-manifold meshes. In this method, Hoetzlein improves upon the work of Green [Gre10] by observing that, with recent GPU architectures, a parallel *counting sort* based on atomic operations is more efficient than a *radix sort* to affect points to grid cells, and allows for efficient access to them. We use this approach several times throughout our algorithm, to compute the initial quadric field, the bilateral filtering, the representativity score and the linking step of our Quick Shift clustering (Alg. 1). The tree structure resulting from this process can then be traversed in parallel, using a single kernel on every vertex (Alg. 2).

The reported performance and scalability of our method rely on this algorithmic layout only, and does not require any low-level optimization, which makes our approach easy to reproduce. Further speed-up may however be achieved with additional architecture-dependent engineering.

6. Results

Filtering experiments. We compare our filtering results with several state-of-the-art methods [JD03, SRML07, ZFAT11, HS13, ZDZ*15] in Fig. 8, and present quantitative comparisons in Tab. 1. We used the CPU implementation provided by Zhang et al. [ZDZ*15] to generate the results for [SRML07, ZFAT11, HS13,

Figure 8: Comparisons for models with synthetic Gaussian noise of standard deviation σ_{noise}, and a real noisy scanned model. σ_{noise} given in terms of mean edge length.

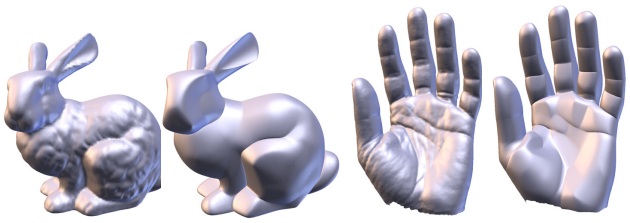


Figure 9: **Stylization by filtering**. The main structures are emphasized while the rest of the shape is completely smoothed.

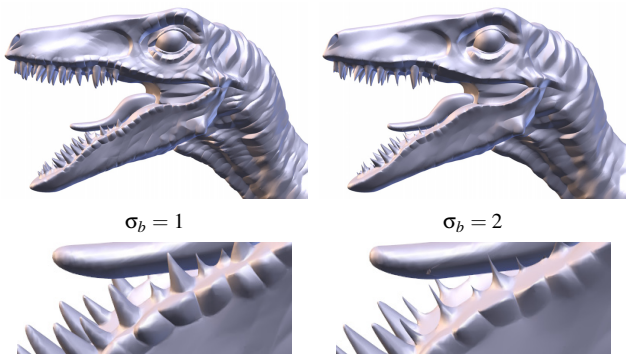


Figure 10: **Influence of scale threshold** σ_b. ($\sigma_s = 4$, $\sigma_r = 0.7$)

ZDZ*15], and we implemented a GPU accelerated version of the bilateral filter [JD03]. For each method, we kept the best results obtained after fine tuning the parameters.

In terms of performance, the main competitor of our quadric filter (or QF) is a GPU implementation of the original bilateral mesh filter [JD03] (or BF), as it shares similar properties (local, non iterative, solver-free, parallel scalable, real-time on medium size models). Fig. 8 – in particular the Gargoyle model – illustrates the benefit brought by our quadric-based approach. On this manufactured example, we introduced a significant amount of artificial noise so that we could measure the RMSE (Tab. 1) for both filters. While both methods succeed at removing most of the noise, our approach better recovers important salient structures. The smooth geometry we obtain shows the higher ability of the diffused quadric field to distinguish noise from structure, compared to the normal field used by the classical bilateral filter. The approach by Sun et al. [SRML07] is also quite fast, as it is simple and requires only a few iterations. A GPU implementation would most likely give timings close to our GPU bilateral filter. However, we observe in Fig. 8, on the Gargoyle and the Iron closeups, that our method produces visually smoother surfaces, with sharper features.

The last three methods [ZFAT11,HS13,ZDZ*15] are much more computationally expensive. They offer different trade-offs between quality and time, with [ZDZ*15] giving overall the best denoising results in terms of RMSE as shown in Tab. 1. We measured these RMSE values against the original noiseless models in order to evaluate numerically how close the denoised results are to the ground truth. Our method tends to enhance and sharpen features, which is particularly beneficial for CAD-like models, such as the fandisk. However, the RMSE is blind to this perceptual notion and does not show significant improvement over the recent fast algorithms considered in these experiments. Therefore, we also report the *mean face normal error* [ZDZ*15] (or MFNE), which is more sensitive to feature preservation. From a visual assessment, one can easily see the combination of large smooth areas delimited by sharper features obtained with our method. Indeed, from a pure sharpness reproduction perspective, our approach appears as superior to all the other ones, including the (slower) high quality alternatives.

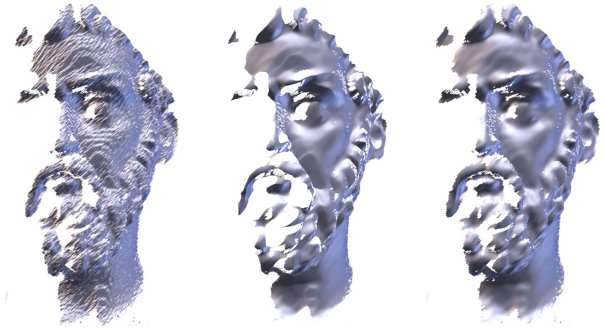
Our filter is not limited to pure denoising scenarios, as one of its main characteristics is its ability to strongly enhance features. We show in Fig. 9 that pushing the spatial support σ_s to high values creates *stylized* shapes, where only the main structure remains, with large smooth regions delimited by prominent lines. This behavior is also instrumental with non-organic shapes, such as the Fandisk and the Iron models processed in Fig. 8, where our filter recovers smooth regions while enhancing sharp edges. As such, our filter appears as a useful quick preprocessing for shape analysis algorithms which need to extract high level structures in 3D objects, such as shape approximation for instance.

Fig. 10 illustrates the influence of the parameter σ_b on a larger mesh. This parameter determines the size of the smallest feature that can be preserved by the filter. Our experiments show that setting $\sigma_b = \sigma_s/2$ usually gives good results but in some cases, such as to preserve the teeth for the raptor model, a lower value can be preferred. In any case, the performance of our filter allows the user to easily adjust this parameter and interactively explore the space of possible filtered shapes, as the entire process takes less than a second for this 2M triangles model. Ultimately, in the absence of a formal noise model derived from a particular 3D capture modality, any mesh filter requires adjusting its parameters depending on the intended effect and target application. Consequently, the speed of our filter significantly enhances the try-and-test workflow operated by the user.

Our approach builds entirely upon spatial queries and does not depend on the manifoldness of the input nor on the presence of holes, which makes it particularly adapted to the preprocessing of scans, as shown in Fig.11.

Fig. 12 illustrates how our filter behaves in the presence of various amounts of noise, at different scales and using alternative vertex placement strategies. We can observe how the unconstrained minimizer retains sharp structures up to high filtering levels.

As shown in Fig. 13, the quadric diffusion step is key to retain features while smoothing large areas at the same time: considering local quadrics without this diffusion i.e., similarly to the approach by Vieira et al. [VNMC10] (middle left, in the figure), cannot achieve both effects. In fact, the methods of Jones et al. [JD03] and Viera et al. [VNMC10] address the problem with orthogonal approaches: while the former designs a custom, bilateral-guided support to account for features, the later retains the traditional low-pass principle, but uses a better geometric optimization, beyond averaging. We remark that the range function used in standard bilateral filtering [JD03] is a quadric in terms of the vertex positions.



(a) input scan (b) unconstrained opt. (c) constrained opt.

Figure 11: Geometric filtering of a partial scan ($\sigma_s = 3.0$, $\sigma_r = 1.0$), using unconstrained (b) and normal-constrained (c) optimizations.

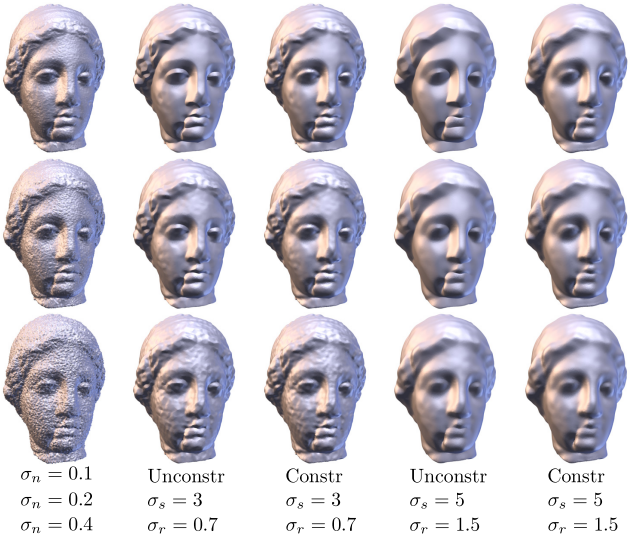


Figure 12: Filtered results for different levels of noise. Parameters given in terms of mean edge length.

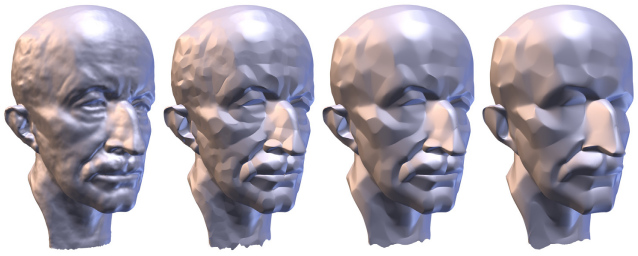


Figure 13: **Diffusion impact**. The quadric diffusion (middle right) better preserves the features around the eyes, mouth and nose, while smoothing more the rest of the shape. Without diffusion (middle left and right), both effects cannot be achieved at the same time.

Model	**#Tri.**	σ_s	σ_r	**QF** (ours)	**BF**
Fandisk (fig. 8)	20K	3	0.4	15	13
Bunny (fig. 9)	70K	5	2.0	68	51
Igea	200K	3	0.4	81	56
Max Planck	400K	6	0.8	332	266
Grog (fig. 3)	1M	4	0.4	459	347
Gargoyle (fig. 1)	1.7M	5	1.5	1106	855
Raptor (fig. 10)	2M	4	0.7	849	649
Dragon (fig. 14)	7M	5.5	1.0	6330	4756

Table 2: **Filtering timings** in milliseconds, without CPU-GPU memory transfer. Our method (QF) is compared with our GPU implementation of [JD03] (BF).

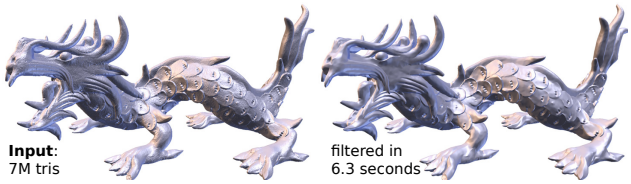


Figure 14: Our scalable filter processes meshes with several millions of polygons in a few seconds.

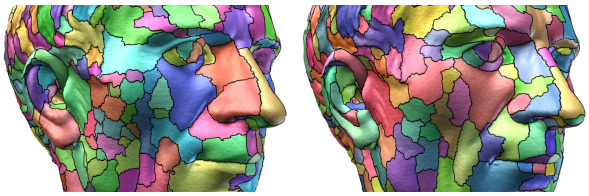
While Jones et al. [JD03] suggest to pre-filter the normal field, we propose to aggregate and filter the quadric range functions themselves instead. Consequently, while BF works fine in regions that are almost planar, we capture more complex curved regions (e.g., ears of the bunny, Fig. 8). So, overall, our filter retains the best of both concepts, revealing large curved regions and preserving sharp features at multiple scales, something not achievable with former high-speed approaches.

Finally, we report performances for all models illustrating this paper in Tab. 2. We see that our method achieves real-time performances up to 200k input polygons, without any preprocessing, and remains close to interactive for two million polygons and a fairly large spatial support. The only model for which the filtering time was much larger than 1s is the 7M triangle Dragon model, which is displayed in Fig. 14, and took up to 6.3s to denoise with our approach. Compared to our filter, the classical bilateral filter turns out to be 23% faster on average, for a significant loss in feature preservation and large scale behavior. Overall, thanks to the high performance GPU implementation one can easily achieve using on approach, our filter appears to be fast enough for several time-sensitive applications, such as interactive filtering design, on-the-fly preprocessing for the many geometric algorithms which work better on filtered shapes and shape repository batch processing for machine learning, as it can process several thousands of objects (e.g., scans) per hour on a single computer. The recent introduction of a new mesh-based graphics pipeline [Kub18] based on the concept of *mesh shaders* acting on *meshlets* makes also our clustering method an interesting candidate to generate efficiently such meshlets from a raw polygon mesh and feed mesh shaders.

Clustering results. Fig. 15 reports results of our quadric-guided clustering at different resolutions on a 1M triangle input. This clus-

$\theta = 0.5$, 2035 clusters $\theta = 0.75$, 1120 clusters $\theta = 0.99$, 449 clusters

Figure 15: Different partitions of a model, with θ varying. The other parameters are fixed : $\sigma_s = 2.0$, $\sigma_r = 1.0$, $\sigma_d = 6.0$, $\sigma_q = 1.0$

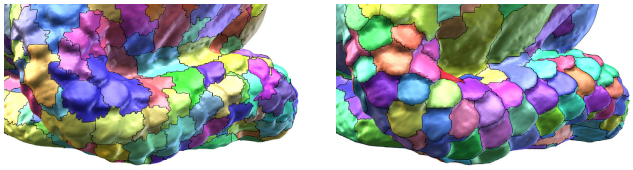


$\sigma_q = 3.5$ $\sigma_q = 1.0$

Figure 16: Influence of the quadric weight parameter σ_q with 600 clusters in both partitions ($\sigma_s = 2.0$, $\sigma_r = 1.0$, $\sigma_d = 7.0$).

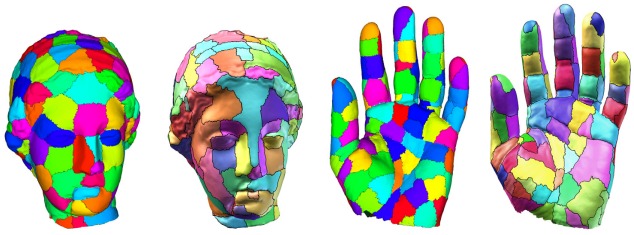
tering if performed in 1.7 sec. to 2.9 sec., depending on the number of output clusters, a smaller cluster count requiring to gather larger regions. We can observe that salient features of the input carry many of the cluster boundaries, which is a desirable behavior in super-pixel-like applications.

Fig. 16 illustrates the influence of the quadric weight parameter σ_q. This parameter controls the importance of the quadric error evaluation in the score and the proximity values. When this value is too high, the frontiers of the regions do not adhere properly to features, for example on the eye or the mouth. Once again, the speed of our clustering method makes possible for users to investigate interactively the parameters influence, and select the result that best serves their needs.



Quick Shift (euclid dist): 1877ms Our method: 3928ms

Figure 17: Comparison with a Quick Shift on the euclidean distance only (same parameters as for figure 1, 1119 clusters in both cases).



Superfacets: 248s Ours: 419ms Superfacets: 87s Ours: 174ms

Figure 18: Comparison of our method with the superfacets of Simari et al. [SPDF14]. Our method gives comparable results, in interactive time. The parameters used for our method are the same as is figure 6 and 7. For the superfacets, we selected the same number of regions, and $\alpha = 200$ (as recommended by the authors [SPDF14]).

Model	#Tri.	σ_s	σ_r	σ_d	σ_q	θ	#Clusters	Time
Bunny (fig. 20)	70K	3.0	1.0	5.0	0.8	0.9	89	161
Hand (fig. 7)	100K	2.0	1.0	5.0	1.0	0.75	142	174
Igea (fig. 6)	200K	2.0	1.0	6.0	1.0	0.87	156	419
Caesar (fig. 16)	750K	2.0	1.0	7.0	1.0	0.75	600	1691
Grog (fig. 15)	1M	2.0	1.0	6.0	1.0	0.5	2035	1676
		2.0	1.0	6.0	1.0	0.75	1120	1883
		2.0	1.0	6.0	1.0	0.99	449	2864
Gargoyle (fig. 1)	1.7M	3.0	1.0	6.0	1.0	0.95	1119	3928
Raptor (fig. 19)	2M	3.0	1.0	4.0	1.0	0.95	1389	2287
Dragon (fig. 19)	7M	4.0	1.0	3.0	1.0	0.999	2701	10557

Table 3: **Clustering timings** in milliseconds. σ_s and σ_r are the filtering parameters. σ_d, σ_q and θ are the Quick Shift parameters. σ_s, σ_r, σ_d and σ_q are given in terms of mean edge length.

We further evaluate the influence of our quadric field on the quick shift clustering by comparing it to a quick shift accounting only for the euclidean distance between vertices in Fig. 17. We can observe that our approach provides a significant quality improvement, with clusters fitting the geometric structures of the surface naturally, for about twice the amount of time. This lets us think that most applications, which can afford a basic quick shift clustering can therefore afford ours, for a much better feature sensitivity.

Fig. 18 compares our approach to the superfacets algorithm of Simari et al. [SPDF14], which shares the objective of reproducing the super-pixel concept on surfaces.

We use the reference CPU implementation provided by the authors. The two techniques give regions with comparable adherence to features. As our method is a Quick Shift variant, we trade off some compactness for performance. The superfacets produce more compact and regularly shaped regions, while our GPU approach is 2 to 3 orders of magnitude faster, hence suitable for interactive scenarios.

Tab. 3 reports clustering performance. We can observe that the clustering can be achieved at interactive framerate up to 200k input triangles and is scalable, with about 10 sec. to cluster a 7M triangle mesh into about 2700 clusters, which we showcase in Fig. 19.

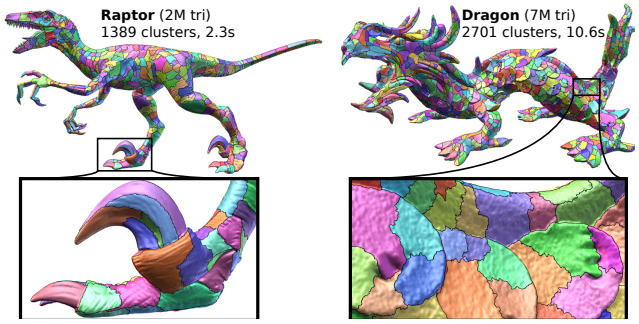


Figure 19: Our method is scalable and capable of clustering meshes with several millions of polygons in a few seconds. **Raptor :** $\sigma_s = 3.0$, $\sigma_r = 1.0$, $\sigma_d = 4.0$, $\sigma_q = 1.0$, $\theta = 0.95$. **Dragon :** $\sigma_s = 4.0$, $\sigma_r = 1.0$, $\sigma_d = 3.0$, $\sigma_q = 1.0$, $\theta = 0.999$.

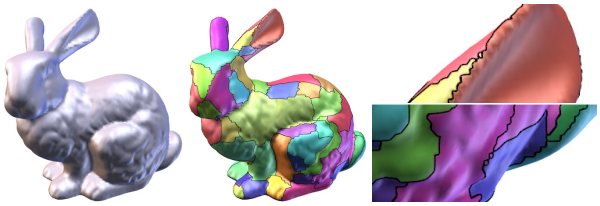


Figure 20: The region connectivity is not controlled, vertices can sometimes be disconnected from the region containing their mode. A post-processing step may be necessary if this is mandatory. $\sigma_s = 3.0$, $\sigma_r = 1.0$, $\sigma_d = 5.0$, $\sigma_q = 0.8$, $\theta = 0.9$. 89 clusters.

Limitations and future work. Our method has three main limitations that motivate future research. First, our aim at high-speed processing prevents us from accounting for geodesic approximation when gathering neighborhoods. While exact geodesic may not be desired – to still cope with non-manifold surfaces having holes – a parallel scalable neighborhood query mechanism that accounts for on-surface proximity would be an interesting improvement of our approach. Regarding filtering, the most advanced high quality filters account for non-local self-similarity and collaboratively filter surface samples independently from their actual 3D location. While being orders of magnitude slower than our approach, we think that our quadric field may be instrumental to better characterize long range relationships between distant surface structures. Finally, regarding clustering, our approach sometimes requires several try-and-test parameter adjustments to achieve the desired partition. Having an automatic adjustment mechanism that dynamically adjusts the cluster based on a-posteriori cluster shape analysis would be useful for several applications, in particular in compression and recognition. The topology of each cluster is also not guaranteed to be a single component (see Fig. 20), which may require an additional post-processing for applications requiring so.

7. Conclusion

We have introduced the anisotropic diffusion of error quadrics as a new mechanism to enrich surface meshes with a discriminative per-vertex attribute that has a fixed size, is easy to compute, can model large regions and be used as the basis to guide a new feature-preserving geometric filter or seek geometry-aware modes in a new fast clustering method. Our approach is well adapted to high-speed processing on GPU and is able to capture large salient structures, both in filtering and clustering, while providing a strong regularization effect anywhere else. So far, such a behavior was achievable only with offline methods, up to several orders of magnitude slower. We believe that the speed of our method can make it a serious candidate for a number of geometry processing primitives that require prefiltering their input or estimating large scale geometric structures. Its simplicity should also ease its integration into high-speed processing pipelines based, for instance, on mesh shaders.

Acknowledgments We thank Wangyu Zhang and Bailin Deng for providing implementations of various mesh denoisers. We also thank the reviewers for their helpful insights. This work has been partially funded by the French National Research Agency under grant ANR 16-LCV2-0009-01 ALLEGORI and by BPI France, under grant PAPAYA.

References

[AGDL09] ADAMS A., GELFAND N., DOLSON J., LEVOY M.: Gaussian kd-trees for fast high-dimensional filtering. In *Proc. SIGGRAPH* (2009), pp. 21:1–21:12. 3

[ASS*12] ACHANTA R., SHAJI A., SMITH K., LUCCHI A., FUA P., SÜSSTRUNK S.: Slic superpixels compared to state-of-the-art superpixel methods. *IEEE PAMI 34*, 11 (2012), 2274–2282. 4

[BCM05] BUADES A., COLL B., MOREL J.-M.: A non-local algorithm for image denoising. In *Proc. CVPR* (2005), vol. 2, pp. 60 – 65. 3

[CDR00] CLARENZ U., DIEWALD U., RUMPF M.: Anisotropic geometric diffusion in surface processing. In *Proc. Visualization* (2000), pp. 397–405. 3

[CM02] COMANICIU D., MEER P.: Mean shift: A robust approach toward feature space analysis. *IEEE PAMI 24*, 5 (2002), 603–619. 4

[CPD07] CHEN J., PARIS S., DURAND F.: Real-time edge-aware image processing with the bilateral grid. In *Proc. SIGGRAPH* (2007). 3

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *ACM Trans. Graph.* (2004), vol. 23, ACM, pp. 905–914. 4

[DMSB99] DESBRUN M., MEYER M., SCHRÖDER P., BARR A. H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. SIGGRAPH* (1999), pp. 317–324. 2

[DMSB00] DESBRUN M., MEYER M., SCHRÖDER P., BARR A. H.: Anisotropic feature-preserving denoising of height fields and bivariate data. In *Proc. Graphics interface* (2000), vol. 11. 3

[DR05] DECORO C., RUSINKIEWICZ S.: Pose-independent simplification of articulated meshes. In *Proc. I3D* (2005), pp. 17–24. 2

[FDCO03] FLEISHMAN S., DRORI I., COHEN-OR D.: Bilateral mesh denoising. In *Proc. SIGGRAPH* (2003), pp. 950–953. 3

[FS10] FULKERSON B., SOATTO S.: Really quick shift: image segmentation on a gpu. In *Proc. 11th European conference on Trends and Topics in Computer Vision* (2010), pp. 350–358. 4

[FYP10] FAN H., YU Y., PENG Q.: Robust feature-preserving mesh denoising based on consistent subneighborhoods. *IEEE TVCG 16*, 2 (2010), 312–324. 3, 5

[GAB12] GUILLEMOT T., ALMANSA A., BOUBEKEUR T.: Non local point set surfaces. In *Proc. 3DIMPVT* (2012), pp. 324–331. 3

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proc. SIGGRAPH* (1997), pp. 209–216. 2

[Gre10] GREEN S.: Particle simulation using cuda. *NVIDIA whitepaper 6* (2010), 121–128. 7

[GWH01] GARLAND M., WILLMOTT A., HECKBERT P. S.: Hierarchical face clustering on polygonal surfaces. In *Proc. I3D* (2001), pp. 49–58. 4

[Hoe14] HOETZLEIN R.: Fast fixed-radius nearest neighbors: interactive million-particle fluids. In *GPU Technology Conference* (2014), p. 18. 7

[Hop96] HOPPE H.: Progressive meshes. In *Proc. SIGGRAPH* (1996), pp. 99–108. 2

[HS13] HE L., SCHAEFER S.: Mesh denoising via l0 minimization. *ACM Trans. Graph. 32*, 4 (2013), 64:1–64:8. 3, 7, 8, 9

[HZR06] HE X., ZEMEL R., RAY D.: Learning and incorporating top-down cues in image segmentation. *Proc. ECCV* (2006), 338–351. 4

[JD03] JONES T. R., DURAND F.: Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graph. 22* (2003), 943–949. 3, 7, 8, 9, 10

[KCLU07] KOPF J., COHEN M. F., LISCHINSKI D., UYTTENDAELE M.: Joint bilateral upsampling. In *ACM Trans. Graph.* (2007), vol. 26, p. 96. 3

[KG05] KIRCHER S., GARLAND M.: Progressive multiresolution meshes for deforming surfaces. In *Proc. SCA* (2005), pp. 191–200. 2

[Kub18] KUBISCH C.: Turing mesh shaders. SIGGRAPH 2018 Exhibitor Talk, 2018. https://devblogs.nvidia.com/introduction-turing-mesh-shaders. 10

[LKK15] LEE J., KIM S., KIM S.-J.: Mesh segmentation based on curvatures using the gpu. *Multimedia Tools and Applications 74*, 10 (2015), 3401. 4

[LS09] LANDRENEAU E., SCHAEFER S.: Simplification of articulated meshes. In *Computer Graphics Forum* (2009), vol. 28, pp. 347–353. 2

[MPW*09] MOORE A. P., PRINCE S. J., WARRELL J., MOHAMMED U., JONES G.: Scene shape priors for superpixel segmentation. In *Proc. ICCV* (2009), pp. 771–778. 4

[NF10] NOCIAR M., FERKO A.: Feature-preserving mesh denoising via attenuated bilateral normal filtering and quadrics. In *Proc. SCCG* (2010), pp. 149–156. 3

[OBS02] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: Mesh smoothing by adaptive and anisotropic gaussian filter applied to mesh normals. In *Proc. VMV* (2002), pp. 203–2010. 3

[RM03] REN X., MALIK J.: Learning a classification model for segmentation. In *Proc. ICCV* (2003), pp. 10–17. 4

[SCBW14] SOLOMON J., CRANE K., BUTSCHER A., WOJTAN C.: A general framework for bilateral and mean shift filtering. *arXiv preprint arXiv:1405.4734* (2014). 3

[SKK07] SHEIKH Y. A., KHAN E. A., KANADE T.: Mode-seeking by medoidshifts. In *Proc. ICCV* (2007), pp. 1–8. 4

[SPDF14] SIMARI P., PICCIAU G., DE FLORIANI L.: Fast and scalable mesh superfacets. In *Computer Graphics Forum* (2014), vol. 33, pp. 181–190. 2, 4, 6, 11

[SRML07] SUN X., ROSIN P., MARTIN R., LANGBEIN F.: Fast and effective feature-preserving mesh denoising. *IEEE TVCG 13*, 5 (2007), 925–938. 3, 7, 8, 9

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *Proc. SIGGRAPH* (1995), pp. 351–358. 2

[TGB13] THIERY J.-M., GUY E., BOUBEKEUR T.: Sphere-meshes: Shape approximation using spherical quadric error metrics. *ACM Trans. Graph. 32*, 6 (2013), Art. No. 178. 2

[TGBE16] THIERY J.-M., GUY E., BOUBEKEUR T., EISEMANN E.: Animated mesh approximation with sphere-meshes. *ACM Trans. Graph. 35*, 3 (2016), 30:1–30:13. 2

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *Proc. ICCV* (1998), pp. 839–. 3

[VB11] VIALANEIX G., BOUBEKEUR T.: SBL Mesh Filter: A Fast Separable Approximation of Bilateral Mesh Filtering. In *Vision, Modeling, and Visualization (2011)* (2011), Eisert P., Hornegger J., Polthier K., (Eds.), pp. 97–103. 3

[VNMC10] VIEIRA A. W., NETO A. A., MACHARET D. G., CAMPOS M. F.: Mesh denoising using quadric error metric. In *Proc. SIBGRAPI* (2010), pp. 247–254. 3, 4, 9

[VS08] VEDALDI A., SOATTO S.: Quick shift and kernel methods for mode seeking. *Proc. ECCV* (2008), 705–718. 4, 6

[WFL*15] WANG P.-S., FU X.-M., LIU Y., TONG X., LIU S.-L., GUO B.: Rolling guidance normal filter for geometric processing. *ACM Trans. Graph. 34*, 6 (2015), 173:1–173:9. 3

[WLT16] WANG P.-S., LIU Y., TONG X.: Mesh denoising via cascaded normal regression. *ACM Trans. Graph. 35*, 6 (2016), 232. 3

[YBS06] YOSHIZAWA S., BELYAEV A., SEIDEL H.-P.: Smoothing by example: Mesh denoising by averaging with similarity-based weights. In *Proc. SMI* (2006), pp. 9–. 3

[YLL*05] YAMAUCHI H., LEE S., LEE Y., OHTAKE Y., BELYAEV A., SEIDEL H.-P.: Feature sensitive mesh segmentation with mean shift. In *Proc. SMI* (2005), pp. 236–243. 4

[YOB02] YAGOU H., OHTAKE Y., BELYAEV A.: Mesh smoothing via mean and median filtering applied to face normals. In *Proc. GMP* (2002), pp. 124–. 3

[YOB03] YAGOU H., OHTAKE Y., BELYAEV A.: Mesh denoising via iterative alpha-trimming and nonlinear diffusion of normals with automatic thresholding. In *Proc. CGI* (2003), pp. 28–33. 3

[YRP17] YADAV S. K., REITEBUCH U., POLTHIER K.: Mesh denoising based on normal voting tensor and binary optimization. *IEEE TVCG 24*, 8 (2017), 2366–2379. 3

[ZDZ*15] ZHANG W., DENG B., ZHANG J., BOUAZIZ S., LIU L.: Guided mesh normal filtering. In *Computer Graphics Forum* (2015), vol. 34, pp. 23–34. 3, 7, 8, 9

[ZFAT11] ZHENG Y., FU H., AU O. K.-C., TAI C.-L.: Bilateral normal filtering for mesh denoising. *IEEE TVCG 17*, 10 (2011), 1521–1530. 3, 7, 8, 9

[ZSL*13] ZHANG L., SONG M., LIU Z., LIU X., BU J., CHEN C.: Probabilistic graphlet cut: Exploiting spatial structure cue for weakly supervised image segmentation. In *Proc. CVPR* (2013), pp. 1908–1915. 4

[ZSXJ14] ZHANG Q., SHEN X., XU L., JIA J.: *Rolling Guidance Filter*. 2014, pp. 815–830. 3

[ZZW10] ZHANG S., ZHAO J., WANG B.: A local feature based simplification method for animated mesh sequence. In *Proc. ICCET* (2010), vol. 1, pp. V1–681. 2