Gaussian Curvature Filter on 3D Mesh

Wenming Tang[1], Yuanhao Gong[1], Kanglin Liu[2], Jun Liu[3], Wei Pan[4], Bozhi Liu[5], and Guoping Qiu[*]

Abstract—Minimizing Gaussian curvature of meshes is fundamentally important for obtaining smooth and developable surfaces. However, there is a lack of computationally efficient and robust Gaussian curvature optimization method. In this paper, we present a simple yet effective method that can efficiently reduce Gaussian curvature for 3D meshes. We first present the mathematical foundation of our method, which states that for any point on a developable surface there must be another point that lives on its tangent plane. Then, based on this theoretical insight, we introduce a simple and robust implicit Gaussian curvature optimization method named Gaussian Curvature Filter (GCF). GCF implicitly minimizes Gaussian curvature without the need to explicitly calculate the Gaussian curvature itself. GCF is highly efficient and is 20 times faster than the classical standard Gaussian curvature flow method. We present extensive experiments to demonstrate that GCF significantly outperforms state-of-the-art methods in minimizing Gaussian curvature, geometric feature preserving smoothing and mesh noise removal. This method can be used in a large range of applications that involve Gaussian curvature.

Index Terms—Gaussian curvature, mesh denoising, feature preserving, probability distributions.

1 Introduction

There are many ways to represent 3D models. Among these representations, triangular meshes are perhaps the most popular. Triangular meshes usually contain two parts. The first part is a set of vertices, representing 3D spatial locations of the surface. The other part is a set of triangular faces that indicate the connectivity between vertices. With the topological information of adjacent vertices, triangular mesh can represent the geometric details of a surface.

Automatic 3D mesh grneration has made great progress in the past few years. The way of generating triangular meshes can be roughly divided into three categories: interactive design from CAD software, 3D scanning, and end-to-end generation. In 3D scanning, a scanner can automatically obtain the 3D coordinates of the surface. For example, by modeling the environment around the scanned object, selecting the best path for automatic scanning to produce a high-quality 3D mesh [1]. In end-to-end methods, the 2D images are used to train a neural network, which generates the corresponding 3D mesh [2].

Unfortunately, the 3D mesh obtained through these technologies is often noisy, incomplete and/or contains outliers. The noise usually comes from the measurement device. Due to the limited view angle, the obtained mesh might be incomplete (missing partial data). As a result, the obtained 3D mesh can not be directly used in practice. For this reason, denoising and smoothing methods for 3D meshes become indispensable.

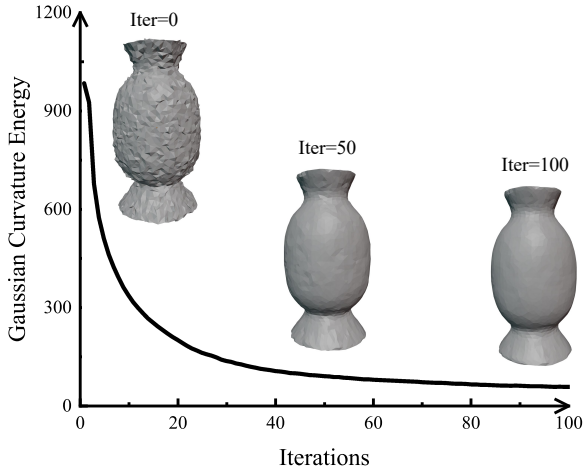


Fig. 1. Gaussian curvature filter on the noisy Vase mesh.

In the literature, various methods have been developed. These methods can be categorized into three types: optimization [3], [4], [5], training [6], [7], [8], and filtering [9], [10], [11], [12], [13], [14], [15]. These methods have their advantages and disadvantages. Most of the optimization methods for mesh smoothing need manually set some parameters, which often need to be optimized iteratively to satisfy the parameter requirements. Such methods are generally time consuming and sometimes do not converge for certain given parameter values. On the other hand, the training methods do not have this issue. Based on the given noise and ground truth model data, training methods can achieve satisfying results. However, the disadvantage of the training-based approach

- [1] Wenming Tang and Yuanhao Gong equally contributed to this work.
- Wenming Tang, Yuanhao Gong, Kanglin Liu, Jun Liu and Bozhi Liu are with the College of Information Engineering, Shenzhen University, Guangdong Key Laboratory of Intelligent Information Processing, Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen, China. E-mail: tangwenming@szu.edu.cn, gong@szu.edu.cn, max_liu@szu.edu.cn, ljunbit@126.com, Liu@szu.edu.cn.
- Wei Pan is with the School of Mechanical & Automotive Engineering, South China University of Technology, Department of Research and Development, OPT Machine Vision Tech Co., Ltd, Jinsheng Road, Chang'an, Dongguan 523860, Guangdong, China. E-mail: vpan@foxmail.com.
- Guoping Qiu (Corresponding author) is with the College of Information Engineering, Shenzhen University, Guangdong Key Laboratory of Intelligent Information Processing, Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen, China, and also with the School of Computer Science, University of Nottingham, Nottingham NG8 1BB, U.K. E-mail: guoping.qiu@nottingham.ac.uk.

is that it is difficult to find sufficient models to train the network parameters, to have good generalization capabilities for different meshes and different noise levels. Most filter methods, based on vertex and face normal, use global or local statistics, and may also rely on several manually set parameters.

Curvature is an important geometric feature of surfaces. Its value reflects how the surface is curved. It is often used as an important tool for surface analysis and processing. The literature has reported that using curvature features on 3D mesh surfaces can achieve good results [16], [17]. Michael Eigensatz et al. proposed a 3D geometry processing framework to achieve 3D mesh filtering and editing by utilizing the curvature distribution of the surface [18]. Gaussian curvature is a specific type of curvature. It is an intrinsic measurement of surfaces. It has been applied to images and 3D meshes [19], [20], [21], [22], [23].

1.1 Motivation and Contribution

A 3D mesh that contains noise has higher Gaussian curvature than its corresponding noise-free model. Therefore, reducing the curvature energy can smooth or denoise the meshes [20], [24]. Based on this observation, we can formulate the problem of noise removal on 3D mesh as that of reducing the Gaussian curvature.

However, minimizing Gaussian curvature is traditionally carried out by Gaussian curvature flow [25]. This method requires to explicitly compute Gaussian curvature. Explicitly computing Gaussian curvature, which implicitly assumes that the surface is second order differentiable. This condition can not be easily satisfied in real applications. Another problem with Gaussian curvature flow is that the time step has to be small to ensure numerical stability. As a result, such geometric flow is time consuming. These two issues hamper the application of Gaussian curvature on meshes.

To tackle these two issues, we propose a simple, easy to implement, and robust filter that can efficiently minimize Gaussian curvature. Our method can effectively remove noise and preserve geometric features as illustrated in Fig. 1. Different from most existing methods that have many free parameters, our method has only one. The contributions of this paper are as follows:

- We propose a simple and robust implicit Gaussian curvature optimization method, which we call Gaussian Curvature Filter. Our filter neither relies on Gaussian curvature calculation nor requires second-order differentiation of surfaces. This property is fundamentally different from conventional geometric flow, which requires the second-order differentiability of the surface.
- We developed a Gaussian curvature optimization algorithm using a 1-ring neighborhood. On this basis, we further introduce geometric constraints on the neighborhood vertices for each Gaussian curvature optimization step to preserve the model's original geometric features and simultaneously optimize the Gaussian curvature.
- Our algorithm has only one parameter - the number of iterations. And it does not depend on any a priori assumption. Our method is very simple, highly efficient, and is 20 times faster than conventional Gaussian curvature flow method. Our method is also more robust than existing methods and outperforms state of the art qualitatively and quantitatively. We will make our code and data open source after the paper is published.

2 RELATED WORK

Before presenting our method that minimizes Gaussian curvature energy and preserves the geometric features, we mainly discuss the related work from three aspects: the Gaussian curvature in image processing, Gaussian curvature in mesh processing, and finally, mesh denoising capability and feature preservation property.

2.1 Gaussian curvature in image processing

Researchers have made great progress in image denoising using the geometric properties of Gaussian curvature in past decades. Lee et al. design a Gaussian curvature-driven diffusion equation for image noise removal [21]. This method can maintain the boundary and some details better than the mean curvature. Jidesh et al. proposed Gaussian curvature to guide image denoising of fourth-order partial differential equations (PDE) [26]. It works for image denoising and maintains curved edges, slopes, and corners.

However, geometric flow algorithms require the surface to be differentiable. But discrete digital images usually can not satisfy this condition. Furthermore, the calculation formula of Gaussian curvature is generally complicated and has some numerical issues.

To overcome these issues, researchers found simple filters to optimize Gaussian curvature. Gong et al. proposed a locally weighted Gaussian curvature as a regularized variational model and designed a closed-form solution [27]. It has achieved excellent results in image denoising, smoothing, texture decomposition and image sharpening. They further proposed an optimization method for regularizers based on Gaussian curvature, mean curvature and total variation [20]. These pixel local filters can be used to quickly reduce the energy of the entire model, thus significantly reduce computational complexity because there is no need to explicitly calculate Gaussian curvature itself.

2.2 Gaussian curvature in mesh processing

In 3D geometry, the explicit Gaussian curvature calculation method requires the surface to be second-order differentiable. However, discrete surfaces usually do not satisfy this requirement. Researchers have found some approximation methods to calculate Gaussian curvature for discrete meshes [28], [29]. The Gaussian area is used to approximate the Gaussian curvature of the mesh surface [23]. The Gaussian curvature distribution of the surface is obtained by a lookup table. Zhao et al. applied Gaussian curvature geometry flow to mesh fairing [22]. They designed a diffusion equation whose evolution direction relies on the normal and the step size is a manually defined function of Gaussian curvature. The corner and edge features of the mesh are preserved during fairing. However, the Gaussian curvature of each vertex is explicitly calculated, and the computation complexity is too high. Moreover, the convergence of the algorithm is too slow.

In Gaussian curvature optimization methods, the Gaussian curvature flow is the most classical one. The Gaussian curvature flow method relies on high-precision Gaussian curvature calculations. It also requires the time step size to be small for ensuring numerical stability. If the step size is set too large, the algorithm may be unstable. If the step size is too small, the convergence speed is slow. How to set a reasonable step size is an open problem.

2.3 Mesh denoising and feature preservation

There are many types of 3D mesh denoising and feature preservation methods. Here, we mainly discuss the optimization-based and filter-based state-of-the-art methods.

In optimization-based methods, this type of methods achieve global optimization with given ground truth and noise distribution prior constraints. He et al. propose a L_0 minimization based method that achieves denoising by maximizing the plane area of the model [3]. In a model with rich planar features, this method can preserve some sharp geometric features during the denoising. Wang et al. implement denoising and feature preservation in two steps [4]. First, the global Laplace optimization algorithm is used to denoise, and then an L1-analysis compressed sensing optimization is used to recover sharp features.

Filter-based methods are mostly implemented by moving the vertex position. In [11], [12], [13], researchers perform the denoising and feature preservation by moving the vertex position of the model. The vertex is moved along the normal direction. And the moving step size is an empirical parameter. Lu et al. constructs geometric edges by extracting geometric features of the input model, and iteratively optimizes vertex positions for denoising and feature preservation by guiding the geometric edges [13]. In [13], [14], [15], [30], iterative optimization of the face normal is used as a guide for denoising and feature preservation. Li et al. present a non-local low-rank normal filtering method [15]. Denoising and feature preservation of synthetic and real scan models are achieved by guided normal patch covariance and low-rank matrix approximation.

Most of existing denoising and feature preservation algorithms have the following problems: 1) Excessive dependence on the a priori assumptions, thus resulting in many parameters to be manually set; 2) It is difficult to find the optimal parameters; 3) Based on the method of normal estimation, the original feature is easily damaged while denoising texture-rich models.

3 Gaussian Curvature Filter on Mesh

In this section, we show a simple iterative filter that can efficiently reduce Gaussian curvature for meshes. Meanwhile, our method can preserve geometric features of the input mesh during the optimization process. We first show a mathematical theory behind this filter, which guarantees to reduce Gaussian curvature. Then, we add more geometric constraints in this filter to preserve geometric features of the input mesh.

3.1 Variational Energy

In many applications, reducing Gaussian curvature is usually imposed by following variational model:

$$\arg\min_{\{v_i'\}} \sum_{i=1}^{N} [\frac{1}{2}(v_i - v_i')^2 + \lambda |K(v_i')|] \,, \tag{1}$$

where $\{v_i\}$ are the input vertices N is the number of vertices, v_i' is the desired output, $K(v_i')$ is the Gaussian curvature at v_i' and $\lambda > 0$ is a scalar parameter that usually is related to noise level. The first quadratic term measures the similarity between the input and the output. The second term measures the Gaussian curvature energy of the output mesh. The main challenge in this model is how to efficiently minimize the Gaussian curvature.

The Gaussian curvature energy (GCE) is defined as

$$\mathcal{E}_{\mathrm{GC}}(v_i') = \sum_{i=1}^{N} |K(v_i')| \,. \tag{2}$$

This energy measures the developability of the mesh $\{v_i'\}$. Different from the Eq. 1, this energy does not consider the similarity between the output $\{v_i'\}$ and input $\{v_i\}$. Therefore, only minimizing Gaussian curvature energy does not preserve the geometric features of the input mesh during the optimization. We will discuss how to minimize Gaussian curvature and preserve geometric features in our method.

When $\mathcal{E}_{\mathrm{GC}} = 0$, it is clear that $K = 0$ everywhere on the surface. Such surface is called a developable surface, which can be mapped to a plane without any distortion. That is why it is called "developable". Reducing the Gaussian curvature on the surface is trying to make the surface developable. Developable surfaces can be easily manufactured and produced in industry. This is one reason that minimizing Gaussian curvature is an important topic.

3.2 Mathematical Foundation

For any developable surface $\mathcal{S}$ (Gaussian curvature is zero everywhere on the surface), we denote $\mathcal{TS}$ as its tangent space. We have following theorem:

Theorem 1. $\forall \vec{x} \in \mathcal{S}, \forall \epsilon > 0, \exists \vec{x_0} \in \mathcal{S}, 0 < |\vec{x} - \vec{x_0}| < \epsilon$, s.t. $\vec{x_0} \in \mathcal{TS}(\vec{x})$.

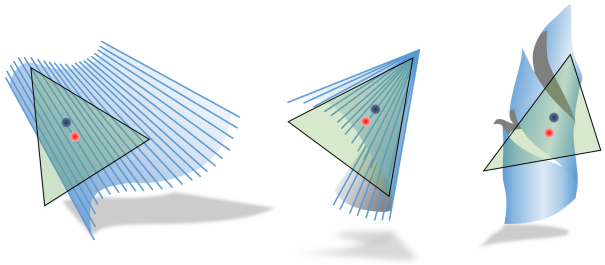
Proof. Let $\vec{x} = \vec{r}(u, v) \in \mathcal{S}$, where (u, v) is the parametric coordinate. Since $\mathcal{S}$ is developable, $\vec{r}(u, v)$ can be represented as $\vec{r}(u, v) = \vec{r}_A(u) + v\vec{r}_B(u)$ [31], where $\vec{r}_A(u)$ is the directrix and $\vec{r}_B(u)$ is a unit vector. Let $\vec{x}_0 = \vec{r}(u, v_0) \in \mathcal{S}$, where $v_0 = v + \epsilon$ and $\epsilon \neq 0$, then $\vec{x}_0 = \vec{r}_A(u) + (v + \epsilon)\vec{r}_B(u)$. For two arbitrary scalars α_1 and α_2, the tangent plane at $\vec{x}$ is

$$\mathcal{TS}(\vec{x}) = \vec{r} + \alpha_1 \frac{\mathrm{d}\vec{r}}{\mathrm{d}u} + \alpha_2 \frac{\mathrm{d}\vec{r}}{\mathrm{d}v} = \vec{r} + \alpha_1 \frac{\mathrm{d}\vec{r}}{\mathrm{d}u} + \alpha_2 \vec{r}_B(u) \,. \tag{3}$$

Because of Eq. 3, $\vec{x}_0$ is on the plane that passes $\vec{x}$ and is spanned by the two vectors $\frac{\mathrm{d}\vec{r}}{\mathrm{d}u}$ and $\vec{r}_B$. Therefore, $\vec{r}_0 \in \mathcal{TS}(\vec{x})$. □

This theorem indicates that for any point $\vec{x}$ on a developable surface there must be another neighbor point $\vec{x_0}$ that lives on its tangent plane. This conclusion is the theoretical foundation for our method.

This theorem can also be verified on developable surfaces. In mathematics, it is already known that there are only



(a) Cylindrical surface (b) Conical surface (c) Tangent surface

Fig. 2. Theorem 1 on three types of developable surfaces.

three types of developable surface: cylinder, cone and tangent developable. As shown in Fig. 2, for any point $\vec{x}$ (red point) on such surface, there is another point $\vec{x_0}$ (blue point) that lives on its tangent plane.

This theorem can also be explained from another point of view. In differential geometry, Gaussian curvature of a vertex on a surface is the product of the principal curvature κ_1 and κ_2 at the vertex. That is $K = \kappa_1\kappa_2$. In literature [24], it has been proved that minimizing a principal curvature in the Gaussian curvature of a vertex is to minimize the Gaussian curvature of the vertex for the 2D discrete images. More specifically, we have following relationship:

$$\kappa_1\kappa_2 = 0 \Longleftrightarrow \min\{|\kappa_1|, |\kappa_2|\} = 0. \tag{4}$$

This result is stronger than Theorem 1 because it tells where $\vec{x_0}$ should be. Instead of adopting this result, we use Theorem 1 because finding the principal curvature is challenging in practice.

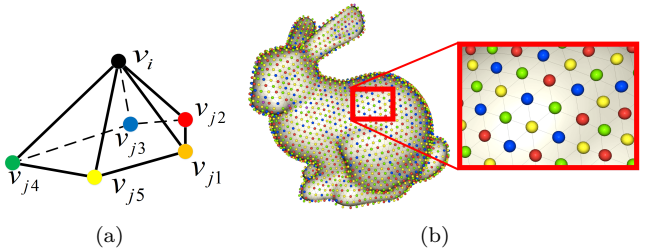
In this paper, we adopt Theorem 1 and apply it on mesh processing. According to Theorem 1, we can reduce the Gauss curvature of the vertex by moving its position such that one of its neighbors falls on its tangent plane. If the Gaussian curvature at the vertex is zero, then the moving distance is zero because one of its neighbors already lives on its tangent plane. Otherwise, the Gaussian curvature is high before the movement. After the movement, the processed vertex is closer to a developable surface. Therefore, the Gaussian curvature is reduced.

3.3 Discrete Neighborhood on Meshes

Based on the Theorem 1, there is always a neighbor point $\vec{x_0}$ that lives on the tangent plane of $\vec{x}$ for any point $\vec{x}$ on a developable surface. However, on the discrete mesh, this point $\vec{x_0}$ is not necessarily a vertex in real applications. To overcome this issue, we take all the 1-ring topology neighborhood vertices as possible candidates and finally adopt only one as an approximation to $\vec{x_0}$. Although such approximation introduces some numerical error, it simplifies the way to find $\vec{x_0}$ on triangular meshes. Our numerical experiments confirm that such approximation works well on triangular meshes in practical applications.

3.4 Our Method

Our method can be roughly divided into two stages. The first part is to classify all the vertices of a mesh according to their neighborhood relationship, so as to ensure that a certain vertex is moved in the local area and its neighborhood is fixed. Fixing the neighborhood ensures the convergence of the algorithm. It is also convenient to design a parallel algorithm. The second part is the vertex update algorithm. According to the Theorem 1, the absolute value of the minimum distance from the neighborhood point to the vertex tangent plane is calculated. Such minimum distance ensures the similarity between the input and filtered result. Finally, the vertex update is performed according to the normal direction and the minimum absolute distance.



(a) (b)

Fig. 3. The 1-ring neighborhood and the domain decomposition result on the Stanford bunny mesh. (a) is the basic structure of the 1-ring topology neighborhood. (b) is the result of the domain decomposition algorithm on the Stanford bunny.

Algorithm 1: DOMAIN DECOMPOSITION

```
Input: Vertexs V={v_1, v_2, ..., v_n}, Neighbor list
       P={P_1, P_2, ..., P_n}
Initialization each vertex color C_i = 0, i = 1, ··· , n;
Initialization k = 1
for i = 1 → n do
    //find the used color set in the neighbor list
    S= ∅
    for j = 1 → P[i].size() do
        if color(P[i][j]) > 0 then
            S.add(C(P[i][j]));
        end
    end
    //loop through the color list to find a label that is
     not used. Give current vertex this color.
    for m = 1 → k do
        if m ∉S then
            C_i=m;
        end
    end
    //if no free color available, add a new color label
    if C_i = 0 then
        C_i = k + 1
        //extend the color list
        k = k + 1
    end
end
Output: vertex domain set {D_1, D_2, ..., D_k}
where all vertices in D_i have the same color label.
```

3.4.1 Domain Decomposition

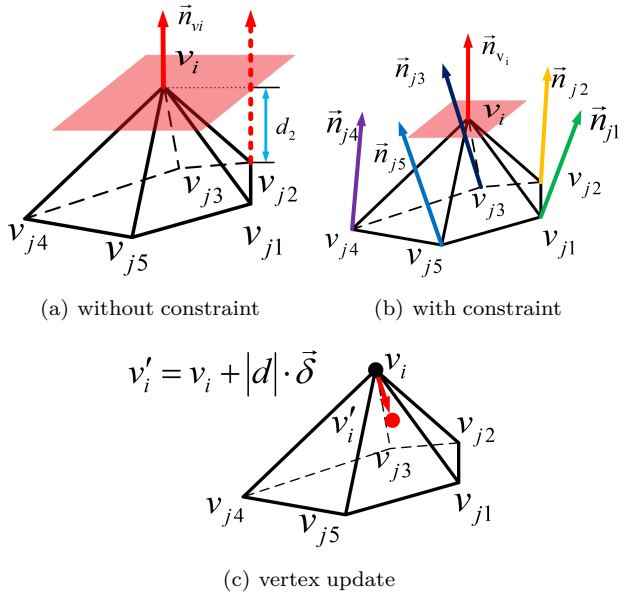
A 1-ring neighborhood of a triangular mesh is usually composed of the similar structure as Fig. 3 (a). The local shape structure consists of a vertex v_i and its neighborhood vertex

set P_i={$v_{j1}, ..., v_{j5}$}. Vertex and neighborhood vertices are connected by edges.

We optimize the entire triangular mesh by optimizing each vertex with 1-ring neighborhood. The Gaussian curvature of the v_i vertex is inextricably linked to its neighborhood vertex set P_i={$v_{j1}, ..., v_{j5}$}. If we are moving the vertex v_i, then the neighborhood vertices set $\{v_{j1}, ..., v_{j5}\}$ must not be moved at the same time. It's easy to understand that if the neighborhood vertices are moved at the same time, it will cause instability and the algorithm will not converge. So we need to separate the vertices in the mesh as shown in Fig. 3 (a). The vertex is separated from the neighborhood vertex set. As shown in Fig. 3 (b), we distinguish them by different colors.

Implementation of this algorithm is described in Algorithm 1, where $V = \{v_1, v_2, ..., v_n\}$ is the set of all vertices of a mesh, $P = \{P_1, P_2, ..., P_n\}$ is the set of neighborhood points of each vertex, and $D = \{D_1, D_2, ..., D_n\}$ is the color label of each vertex after domain decomposition (Algorithm 1).

The advantages of domain decomposition for mesh vertices are as follows: First, it can ensure that the vertex moves while the neighboring vertices do not move. This is an important prerequisite for ensuring convergence. Second, all vertices are divided into several independent sets. Therefore, each set can move independently. This makes it convenient to design parallel algorithms. This also is an important reason why our algorithm is faster than similar algorithms (almost 20 times, see table 1). The result of the domain decomposition of an actual mesh by Algorithm 1 is shown in Fig. 3 (b). We can see that each vertex color is different from the color of its neighborhood vertices, and all the vertices of the bunny are independently divided into several sets.



(a) without constraint (b) with constraint

$$v_i' = v_i + |d| \cdot \vec{\delta}$$

(c) vertex update

Fig. 4. Two projection strategies and the vertex update. (a) and (b) are projection strategy without and with feature preservation, respectively. (c) shows the vertex movement direction and amplitude.

3.4.2 Vertex Moving Direction

The differential coordinate of the i-th vertex $v_i = (x_i, y_i, z_i) \in V = \{v_1, v_2, ..., v_n\}$ is the difference between the absolute coordinates of v_i and the center of mass of its immediate neighbors $P_i = \{v_{j1}, v_{j2}, ..., v_{jm}\}$ in the mesh [32], i.e.

$$\vec{\delta}_{v_i} = (\delta_{x_i}, \delta_{y_i}, \delta_{z_i}) = v_i - \frac{1}{m} \sum_{v_j \in P(v_i)} v_j \,. \tag{5}$$

In differential geometry, the direction of the differential coordinate vector $\vec{\delta}_{v_i}$ can be approximated to the normal direction of the local area [33]. Following these works, we use the $\vec{\delta}$ unit vector Eq. 6 (reverse normal direction) as the moving direction.

$$\vec{\delta} = \frac{-\vec{\delta}_{v_i}}{\|\vec{\delta}_{v_i}\|} \,. \tag{6}$$

3.4.3 Vertex Moving Distance

After the domain decomposition, we update each independent vertex set separately. During each iteration, we have to find the moving direction of the vertex and also the corresponding moving distance.

We propose two different strategies for computing the moving distance and compare them in the experiment sections. They have different properties in minimizing Gaussian curvature and geometric feature preservation. And they can be adopted separately according to the specific task requirement.

- No Geometric Constraints: As can be seen from the previous theoretical part, our algorithm optimizes the overall Gaussian curvature. As shown in Fig. 4 (a), we optimize the Gaussian curvature of v_i by moving the vertex v_i. So we need to calculate the magnitude of the movement of the vertex v_i. In Fig. 4 (a), the projection strategy is to calculate the projection distance set d={d_1,d_2...d_5}. Each projection distance is from the neighbor vertex v_j to the tangent plane of the vertex v_i. We choose the smallest absolute value in this as the moving amplitude of vertex v_i.
- With Geometric Constraints: As shown in Fig. 4 (b), we compute the normal $\vec{n}_{vi}$ of the vertex v_i and then the normal of each neighborhood vertices. It should be noted that the neighborhood vertex normal is the cross-product unit vector of the two edges of the neighborhood vertices. More specifically, the $\vec{n}_{jk}$ unit vector in Fig. 4 (b) is given by

$$\vec{n}_{jk} = \frac{\overrightarrow{v_{jk}v_{j(k-1)}} \times \overrightarrow{v_{jk}v_{j(k+1)}}}{\|\overrightarrow{v_{jk}v_{j(k-1)}} \times \overrightarrow{v_{jk}v_{j(k+1)}}\|}. \tag{7}$$

We calculate the projection distance of the unit normal vector set {$\vec{n}_{vi}$,$\vec{n}_{j1}$...$\vec{n}_{j5}$} from the vertex of the 1-ring neighborhood structure. Then, the each edge {$\overrightarrow{v_iv_{j1}}$,$\overrightarrow{v_iv_{j2}}$...$\overrightarrow{v_iv_{j5}}$} has a projection to each normal {$\vec{n}_{vi}$,$\vec{n}_{j1}$...$\vec{n}_{j5}$}. As a result, we have all possible projection distances {d_1,d_2...d_n} (in this example $n = 5 \times 6 = 30$). We still choose the smallest absolute value in this set $|d|$ as the moving amplitude of vertex v_i.

In summary, the minimal moving distance is computed by

$$d = \begin{cases} \min\limits_k \{| < \vec{n}_{vi},\ \overrightarrow{v_iv_{jk}} > |\}\,, \text{without geometric constraint} \\ \min\limits_k \{\big| < \{\vec{N}\},\ \overrightarrow{v_iv_{jk}} > \big|\}, \{\vec{N}\} = \{\vec{n}_{vi}, \vec{n}_{j1} ... \vec{n}_{jm}\} \end{cases} \tag{8}$$

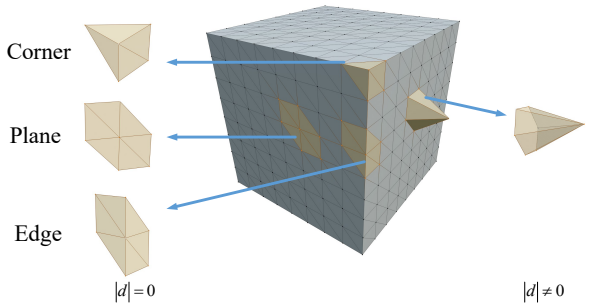


Fig. 5. Geometrically constrained projection distance.

where $<,\ >$ is the standard inner product, $k = 1, ..., m;\ i = 1, ...n$.

Adding the neighborhood geometry constraint as shown in Figure 4 (b) ensures that geometric features are preserved when optimizing the Gaussian curvature. This property is important for the vertices at the corner, edge, and plane geometry, as shown in Fig. 5. Since the vertices are contained in the above geometrical features, the minimum absolute value projection distance obtained by the projection strategy of Fig. 4 (b) is $|d| = 0$.

$$\exists\ \vec{n} \in \{\vec{N}\},\ \vec{n} \perp \overrightarrow{v_i v}_{jk}\ \Rightarrow\ < \{\vec{N}\},\ \overrightarrow{v_i v}_{jk} > \ = 0. \quad (9)$$

Therefore, the vertex moving distance 0 and the spatial position is preserved. This kind of constraint makes our algorithm have strong denoising and feature preservation capability for different noise levels. Not surprisingly, the performance of feature preservation in the noise-free model is still robust, as shown in Fig. 6.

As shown in Fig. 4, there are two strategies for obtaining $|d|$. Visual and quantitative results of the two projection strategies are shown in Fig. 6. We can see that although the projection strategy without geometric constraints can also reduce the Gaussian curvature energy of the model, some details of the model are unfortunately lost. So reducing the Gaussian curvature energy in this way can achieve the denoising of the model, but it can not preserve features.

3.4.4 Vertex Update Algorithm

Through the above computation, we obtain the minimum projection distance of the vertex v_i, see Fig. 4 (c). According to the mathematical theory, iteratively updating each vertex of the entire mesh will reduce the Gaussian curvature. algorithm 2: The vertex update then is given by

$$v' = v + |d| \cdot \vec{\delta}. \quad (10)$$

4 Experiments

In this section, we perform several experiments to show two properties of our method: minimizing the Gaussian curvature and denoising with feature preservation. In the aspect of Gaussian curvature optimization, the literature [22] is the closest to our method. In the aspect of geometric feature preserving noise removal, there are many methods, including optimization based methods [34] and [3], and filter based methods [35], [9], [10], [12], and [15]. We compare our approach with these methods in the two aspects respectively.

Algorithm 2: GAUSSIAN CURVATURE FILTER ON MESH

Input: Vertex set V=$\{v_1, v_2, ..., v_n\}$, Vertex normal set N=$\{\vec{n_1}, \vec{n_2}, ..., \vec{n_n}\}$, Neighbor List P=$\{P_1, P_2, ..., P_n\}$, Domains D=$\{D_1, D_2, ..., D_k\}$, IterationNumber

```
for i = 0 → IterationNumber − 1 do
    for j = 0 → k − 1 do
        //each vertex in the same domain
        for t = 0 → D_k[j].size() − 1 do
            index = D_k[j][t];
            current = V[index];
            if current on boundary then
                v'[index] = v[index];
            else
                Comupte ProjN by Eq. 7;
                put N[index] into ProjN;
                //find the min projection distance;
                MIN = +∞;
                for p = 0 → P[index].size() − 1 do
                    for r = 0 → ProjN.size() − 1 do
                        ProjectDistance =
                         abs((P[index][p] − current) ∗
                         ProjN[r]);
                        if ProjectDistance < MIN
                         then
                            MIN = ProjectDistance;
                        end
                    end
                end
                //compute δ⃗ by Eq. 6 and update;
                v'[index] = v[index] + δ⃗ ∗ MIN;
            end
        end
    end
end
```

Output: Vertexs V'=$\{v'_1, v'_2, ..., v'_n\}$

4.1 Minimize Gaussian Curvature

To show the property of minimizing Gaussian curvature, we compare our method with two approaches. The first one is the classical Gaussian curvature flow method [22]. We compare them by processing two commonly used but representative meshes Max Planck head and Vase. These two meshes are not developable. Therefore, minimizing the Gaussian curvature would change its geometry such that the mesh becomes more developable. Moreover, we also show the performance of both methods on these meshes when adding some random noise.

We further compare our method with a recent approach from SIGGRAPH2018 by Stein et. al [36]. Their method has to manually or automatically defined the edges that can not be moved during the optimization. We compare both methods on noise-free and noisy meshes respectively. We will discuss the results in later sections.

4.1.1 Parameter settings

In [22], the algorithm has five parameters to be manually adjusted, k, ρ, β, ϵ, α. For the specific meaning of each parameter, see Equation 2 in [22]. According to the author's

suggestion, we set $\beta = 2$, $\epsilon = 0.001$, and $\alpha = 0.0005$. The most important parameter of this method is the step size ρ. However, the algorithm is very sensitive to this parameter ρ. If this parameter is set too large, the algorithm will easily diverge. Otherwise, the convergence speed is very slow. After numeral adjustments, we determined that the best step size of the Gaussian curvature flow algorithm on Max Planck is $\rho = 1$. For the Vase, we set the step size $\rho = 0.01$. It is important to note that this parameter may have to be set different values for different models, or different noise levels of the same model. The author also mentioned the importance of an implicit step size algorithm.

In contrast, our algorithm only has one parameter, i.e., the iteration number. It should be noted that the algorithm generally converges after 40-50 iterations. In practical applications, the number of iterations relies on 1) how far the input mesh to a developable surface; 2) noise level on the mesh; 3) how close the output mesh to the original input.

4.1.2 Result analysis

In Fig. 6, we use two meshes, Max Planck head and Vase. These two meshes are not developable. Therefore, minimizing their Gaussian curvature would change their geometry. We perform the standard Gaussian curvature flow method, our method with and without feature preservationon these meshes, respectively. And the results are shown in the first row of Fig. 6. We report the Gaussian curvature energy during the iteration, along with the mean angle error.

On these two meshes, results from the standard Gaussian curvature flow method do not have obvious difference from the input. In contrast, our method leads to some visual differences. This fact indicates that our method has better performance in terms of optimizing Gaussian curvature.

Furthermore, we add random noise to these meshes and compare the performance of these algorithms on noisy meshes. The results are shown in the second row of Fig. 6. We set the noise level $\sigma_n = 0.3e_l$ which is proportional to the average edge length e_l of the input model mesh.

In Fig. 6, it can be seen from the color of the denoised meshes that the denoising results of Gaussian curvature filtering are significantly better than the Gaussian curvature flow. Gaussian curvature filter with feature preservation can remove the noise and also keep some details during the smoothing.

For further quantitative analysis, we show how Gaussian curvature energy and mean square angular error (MSAE, the smaller this value, the closer it is to the ground truth) changes with optimization iteration in Fig. 6. We can see that our method reduces Gaussian curvature energy faster than Gaussian curvature flow.

Without feature preservation, our algorithm has converged after 10 iterations. Fig. 4 (b) MSAE curve diverges after a rapid decline, and the Gaussian curvature flow MSAE curve decreases slowly, which is also comparable to its Gaussian curvature energy curve.

With feature preservation, our algorithm has converged after 40 iterations. In Fig. 6, the MSAE curve with the constraint of Fig. 4 (b) is consistent with the Gaussian curvature energy curve.

Combined with the two sets of curves in Fig. 6, we can easily get the following conclusions: First, through our algorithm, both Fig. 4 (a) and Fig. 4 (b) projection strategies are better than Gaussian curvature flow in Gaussian curvature energy optimization. Second, with the geometric constraint projection strategy of Fig. 4 (b), our algorithm can both denoise and preserve features, and the effect is much better than Gaussian curvature flow.

In addition, it is worth noting that our algorithms are highly parallel due to our domain decomposition algorithm. Our algorithm is nearly 20 times faster than the Gaussian curvature flow algorithm, see Table 1.

TABLE 1
Running time comparison between the standard Gaussian curvature flow method and our method. Our method is about 20 times faster.

Models	Methods	Parameters	Time(s)
Max Planck	[22]	$(100, 1.0, 2, 0.001, 0.0005)$	1016.41
	Ours	(100)	**56.39**
Vase	[22]	$(100, 0.01, 2, 0.001, 0.0005)$	744.75
	Ours	(100)	**40.34**

We further compare our method with the recently developed method [36]. The results are shown in Fig. 7. In the noise free cases, both methods achieve similar results. But our method reaches lower Gaussian curvature energy. In the noisy cases, our method leads to better results in the terms of similarity with the ground truth and also the Gaussian curvature energy. Our method does not generate singularities as shown in Fig. 7 (d). The quantitative results are summarized in Table 2. Our method is about 50 times faster than the method in [36].

TABLE 2
Quantitative comparison of [36] on the bunny ($|V|$: 3301, $|F|$: 6598) and mask ($|V|$: 12657, $|F|$: 24979) mesh. Our method is 50 times faster. The running time is measured in seconds.

Models	Methods	MSAE	GCE	Time(s)
Bunny	[36]	**5.34**	98.89	1613.10
noise-free	Ours	11.24	**82.44**	**35.70**
Mask	[36]	**1.97**	1076.50	5770.60
noise-free	Ours	2.69	**1075.44**	**132.40**
Bunny	[36]	12.59	323.21	1571.10
$\sigma_n = 0.3e_l$	Ours	**11.86**	**83.48**	**37.90**
Mask	[36]	5.56	1226.85	5636
$\sigma_n = 0.3e_l$	Ours	**3.75**	**1124.08**	**134.50**

4.2 Denoising with Feature Preservation

To evaluate how well our new method performs in feature preservation and noise removal, we choose seven representative state-of-art methods for comparison.

4.2.1 Parameter settings

In this set of experiment design, our algorithm iteration parameter is set to 40, because the Gaussian curvature energy curves in our previous experiments show that the algorithm will converge at around 40 iterations. Both the filter based methods and the optimization based methods use their default parameters. The detailed parameters are listed in Table 3. The meanings of the specific parameters can be found in the respective papers. In this section, GCE stands for Gaussian curvature energy and KLD is Kullback-Leibler divergence.

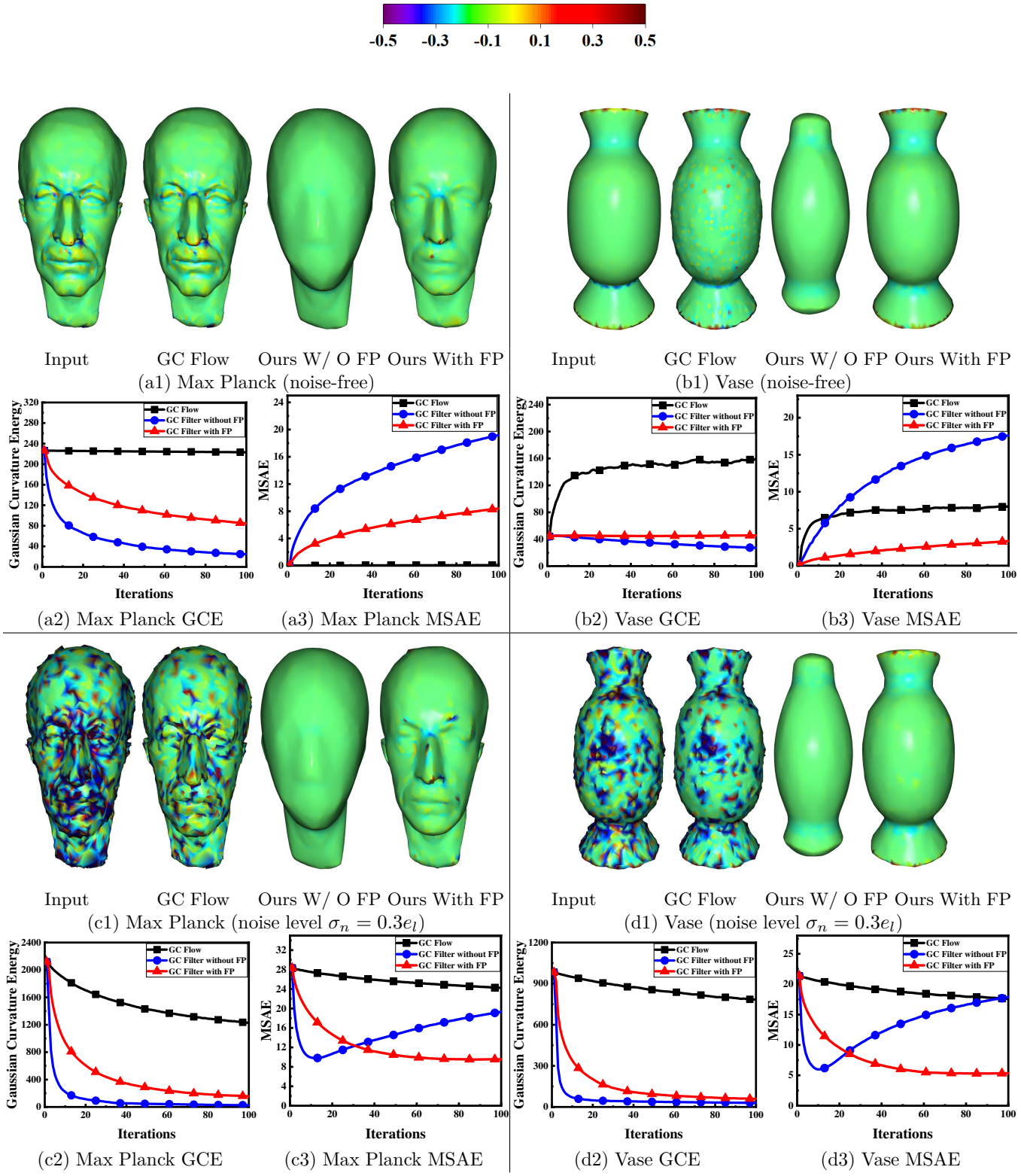


Fig. 6. Minimizing Gaussian curvature on noise-free meshes (top row) and noisy meshes (bottom row). In each panel, from left to right: the original, Gaussian curvature flow method, our method without feature preserving, our method with feature preserving. GC is an abbreviation for Gaussian curvature, and FP is an abbreviation for feature preservation.

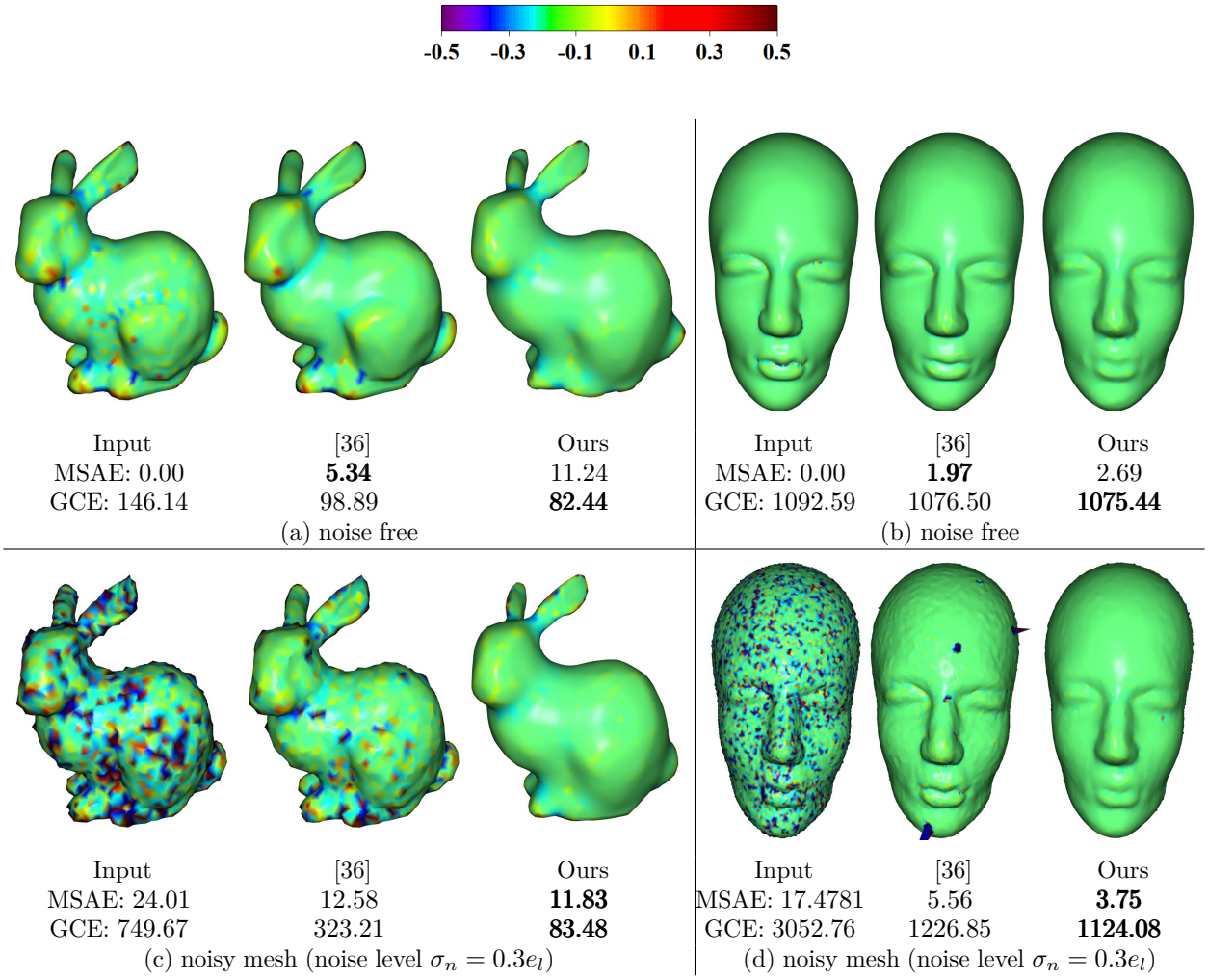
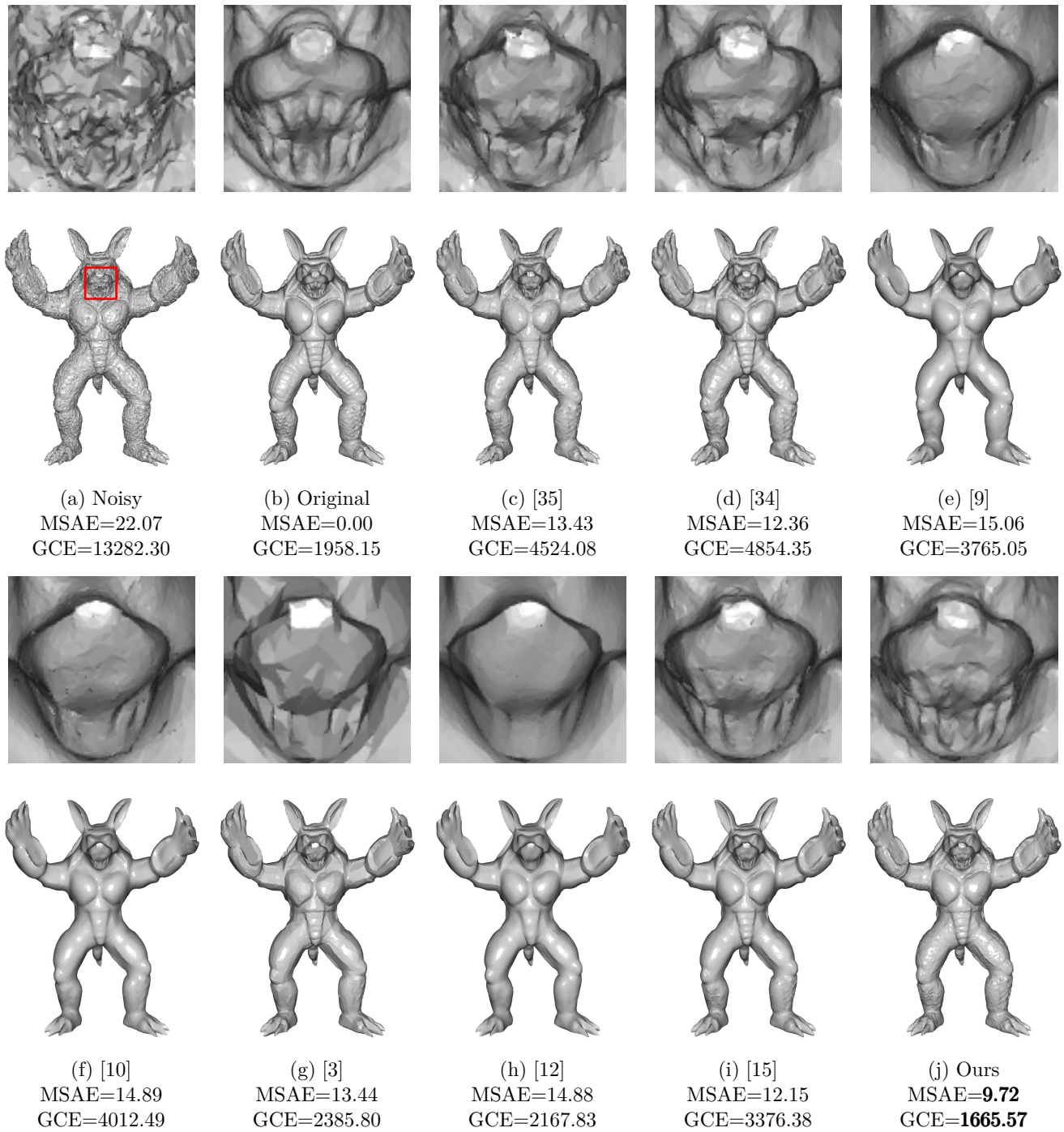


Fig. 7. Comparison between [36] and our algorithm. Top: noise free case. Bottom: noisy case. Both methods lead to similar results in the noise-free case. But our method is better in the noisy case. Notice the singularity in the middle image of (d). Both methods are performed 100 iterations.

4.2.2 Result analysis

In the experiments, we have selected four representative models with rich features: armadillo, bunny, Max Planck, and vaselion. In the armadillo model of Fig. 8, this represents a type of mesh with more vertices and faces, more features, and a complete shape. Figure $8(a)$ is an input model with Gaussian random noise. We can see that its Gaussian curvature energy value is very high, nearly 10 times that of the ground truth model. Most of the comparison methods are based on vertex normals or faces normals for denoising and feature preservation. The normal estimation method needs to be constrained by many artificial parameters, and has to rely on strong assumptions. Setting the normal weight parameter too large, will cause the estimated local plane to be too large thus losing the original features. Conversely, the denoising ability will be weakened. We reduce the Gaussian curvature energy of the overall model by minimizing the Gaussian curvature of each vertex according to the Gaussian curvature of the model itself. So in the end our model's Gaussian curvature energy value is closest to the ground truth. In the comparison method, our MSAE value is the smallest, which also shows that our algorithm preserves the features the best in the denoising process (see Table 3 in later section). As shown in Fig. 8, our results are the best in terms of both overall shape and local detail.

Figures 9 demonstrates that our method also works in models with few vertices and faces but rich features. In Fig. 9, we can see that for the method of [12], handling low-resolution mesh models can result in local regions using larger neighborhood filtering normals and calculating normals on larger patches, which can result in over-smoothing. It can be clearly seen from the bunny's head. In Fig. 9, we can see that our model is the closest to ground truth. For example, due to the rich feature of the nose portion, the Gaussian curvature energy is large. The algorithm does not simply reduce the Gaussian curvature of all vertices to 0 to achieve denoising, but denoises based on the geometric characteristics of the model itself. Fig. 11 represents a model with very rich feature texture and a large number of vertices and faces. Our algorithm is equally applicable to such models, and achieves the best results compared to several state of the art methods. To verify that our algorithm works at different noise levels, we designed experiments to process models with different noise levels. Figure 10 and Fig. 11 demonstrate the robustness of



(a) Noisy
MSAE=22.07
GCE=13282.30

(b) Original
MSAE=0.00
GCE=1958.15

(c) [35]
MSAE=13.43
GCE=4524.08

(d) [34]
MSAE=12.36
GCE=4854.35

(e) [9]
MSAE=15.06
GCE=3765.05

(f) [10]
MSAE=14.89
GCE=4012.49

(g) [3]
MSAE=13.44
GCE=2385.80

(h) [12]
MSAE=14.88
GCE=2167.83

(i) [15]
MSAE=12.15
GCE=3376.38

(j) Ours
MSAE=**9.72**
GCE=**1665.57**

Fig. 8. Comparison between our algorithm and the selected state-of-the-art algorithms on the armadillo ($\sigma_n = 0.3e_l$).

the algorithm at different noise levels.

The above comparisons mainly demonstrate visually that our method is superior to other state of the art methods. The quantitative comparison results are shown in Table 3. We compare the algorithms in terms of setting parameters, MSAE, GCE, and KLD. In Fig. 12, we plot the Gaussian curvature probability distributions of the models which can help better explain why our scheme achieves such a good effect. A wider Gaussian curvature probability distribution indicates a higher level of noise. As can be seen from the armadillo in Fig. 12, for the noisy model, the Gaussian curvature energy is the largest, so the curve is the widest. For the ground truth model, the Gaussian curvature energy is concentrated near 0, and the distribution curve is relatively narrow. Our method's distribution curve is closest to the ground truth distribution amongst all compared methods. The KLD values of our method in Table 3 is the smallest, indicating that the Gaussian curvature probability distribution of our method is the closest to the ground truth distribution. The MSAE values of our method are the smallest, indicating that the output of

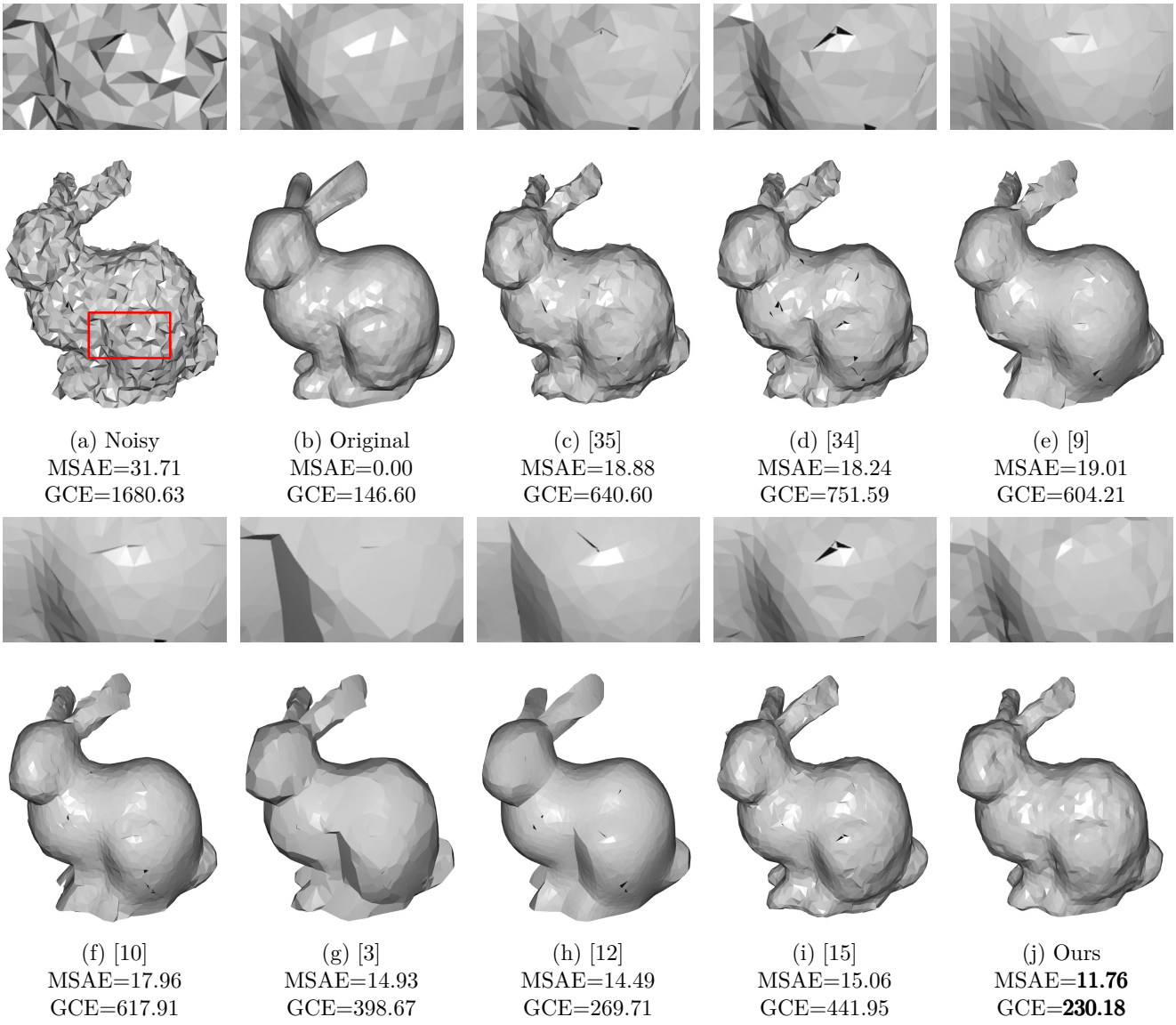


Fig. 9. Comparison between our algorithm and the selected state-of-the-art algorithms on the bunny ($\sigma_n = 0.5e_l$).

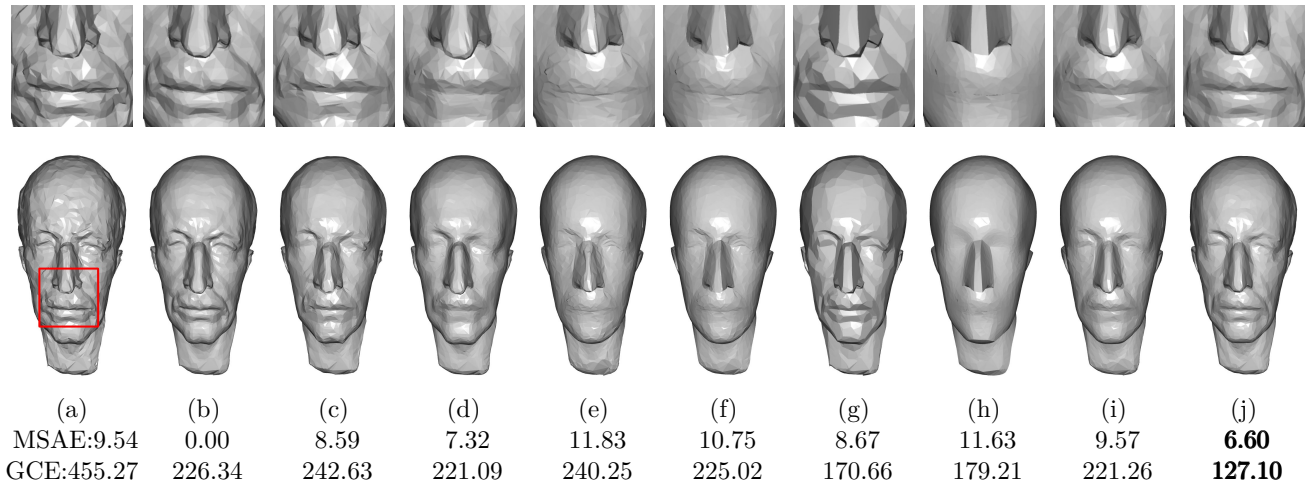
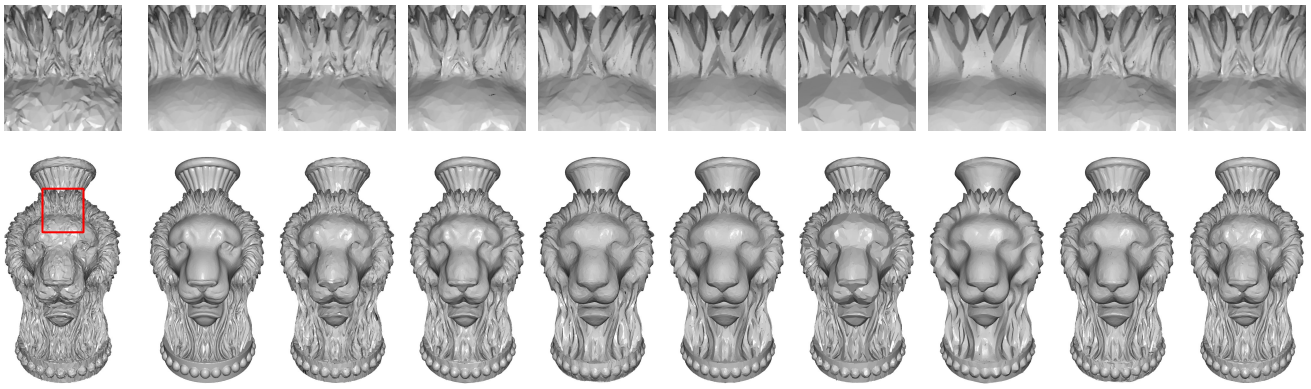


Fig. 10. Comparison between our algorithm and the selected state-of-the-art algorithms on the Max Planck ($\sigma_n = 0.1e_l$).(a) Noisy; (b) Original; (c) [35]; (d) [34]; (e) [9];(f) [10]; (g) [3]; (h) [12]; (i) [15]; (j) Ours;



(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)
MSAE:25.11	0.00	21.13	NaN	24.92	21.78	16.84	25.66	21.55	**12.72**
GCE:12933.10	2757.32	7435.05	8127.51	7017.39	6449.90	5356.49	4403.20	6184.21	**2741.82**

Fig. 11. Comparison between our algorithm and the selected state-of-the-art algorithms on the vaselion ($\sigma_n = 0.2e_l$). (a) Noisy; (b) Original; (c) [35]; (d) [34]; (e) [9];(f) [10]; (g) [3]; (h) [12]; (i) [15]; (j) Ours;

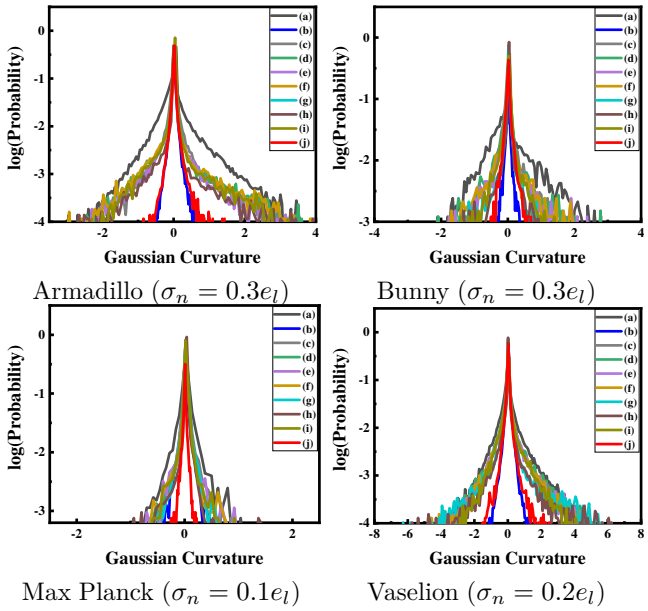


Fig. 12. Gaussian curvature distribution map corresponding to Figures 8- 11. The left axis is Gaussian curvature probability in Log scale. And the bottom axis is the Gaussian curvature value of the vertices on the model. The curves of the eight different colors are: (a) is the Noisy input, (b) is the Ground truth . (c) is the method [35], (d) is the method [34], (e) is the method [9], (f) is the method [10], (g) is the method [3], (h) is the method [12], (i) is the method [15], (j) is Ours. Our results are close to the ground truth. The quantative K-L devergence is summarized in the right column of Table 3.

our method can preserve the model's geometric feature the best. It is also interesting to observe that the GCE values of our method's outputs are also the lowest. Similar observation can be made for the other models in Fig. 12.

4.3 Applied to real scan models

We have seen that our method achieves state of the art effect on synthesized CAD meshes. We have applied the algorithm to process real 3D models. We use two real scanned meshes which contain unknown noise in the experiments and results are shown in Figures 13 and 14. Our method can remove the noise and simultaneously keep the geometric features.

5 Conclusion

In this paper, we propose an iterative filter that optimizes the Gaussian curvature of a triangular mesh. Our method does not need to explicitly calculate the Gaussian curvature. Our method does not require the mesh to be second-order differentiable. Our method is simple but effective and efficient, as confirmed by the numerical experiments.

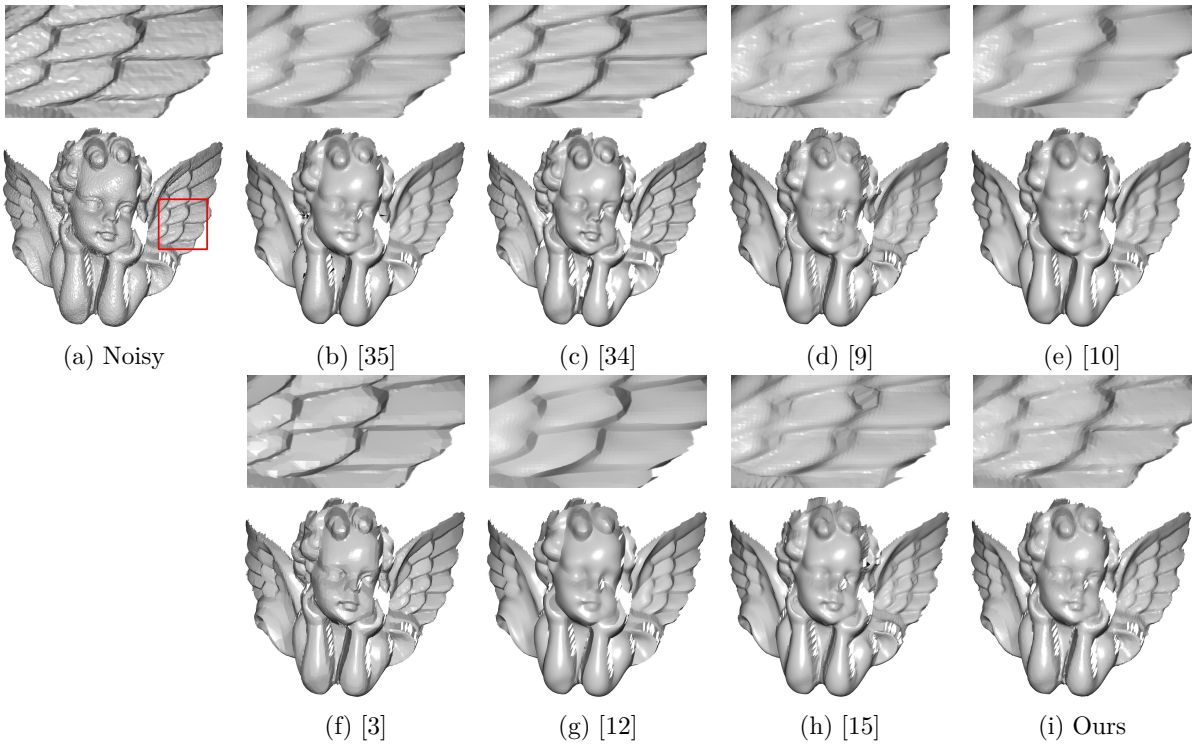
In addition, thanks to the addition of the geometric constraints, our algorithm has strong feature preservation while achieving fast denoising. Our algorithm is adaptive to the original geometric features of the model. Our algorithm has only one parameter in the form of the number of iterations, which makes our algorithm easier to use. From the results of multiple sets of experiments, whether it is visual or quantitative analysis, we have verified that our method exceeded the state-of-art. At the same time, our algorithm can also be applied to the triangular mesh of real scanning, and also achieves good denoising and feature preservation effects.

With this Gaussian curvature filter, minimizing Gaussian curvature on triangular meshes becomes much easier. This is important for many industries, such as ship manufacture, car shape design, etc. And we believe that our method can benefit both academical research and practical industries.

In the future, we will study the parallel Gaussian curvature filtering algorithm, which will further accelerate the optimization of Gaussian curvature. With such high performance, we can study the meshes with super large number of vertices and faces on GPU processing in real applications, such as modeling, manufacture and real-time mesh editing.

References

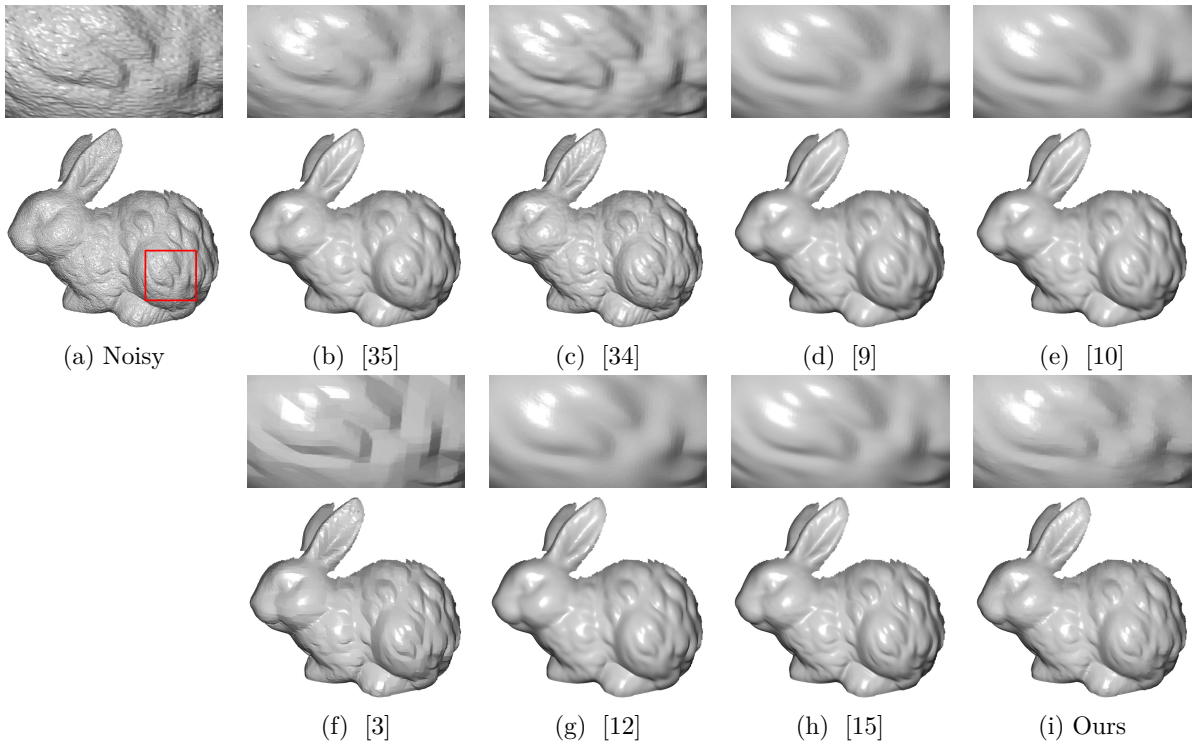
[1] S. Kriegel, C. Rink, T. Bodenmüller, A. Narr, M. Suppa, and G. Hirzinger, "Next-best-scan planning for autonomous 3d modeling," in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2012, pp. 2850–2856.



(a) Noisy (b) [35] (c) [34] (d) [9] (e) [10]

(f) [3] (g) [12] (h) [15] (i) Ours

Fig. 13. Comparison between our algorithm and the selected state of the art algorithms on the model of the real scan (angel).



(a) Noisy (b) [35] (c) [34] (d) [9] (e) [10]

(f) [3] (g) [12] (h) [15] (i) Ours

Fig. 14. Comparison between our algorithm and the selected state of the art algorithms on the model of the real scan (rabbit).

TABLE 3
Quantitative comparison of noise removal and feature preservation performances with other state of the art methods.

Models	Methods	MSAE	GCE	KLD
Armadillo $\sigma_n = 0.3e_l$ (Figure 8)	Noisy	22.07	13282.30	0.79
	Ground truth	0.00	1958.15	0.00
	[35]	13.43	4524.08	0.23
	[34]	12.36	4854.35	0.13
	[9]	15.06	3765.05	0.35
	[10]	14.89	4012.49	0.45
	[3]	13.44	2385.80	0.52
	[12]	14.88	2167.83	0.75
	[15]	12.15	3376.38	0.26
	Ours	**9.72**	**1665.57**	**0.08**
Bunny $\sigma_n = 0.3e_l$ (Figure 9)	Noisy	31.71	1680.63	1.70
	Ground truth	0.00	146.14	0.00
	[35]	18.88	640.60	0.35
	[34]	18.24	751.59	0.25
	[9]	19.01	604.21	0.19
	[10]	17.96	617.91	0.18
	[3]	14.93	398.67	0.36
	[12]	14.49	269.71	0.33
	[15]	15.06	441.95	0.12
	Ours	**11.76**	**230.18**	**0.05**
Max Planck $\sigma_n = 0.1e_l$ (Figure 10)	Noisy	9.54	455.27	0.11
	Ground truth	0.00	226.34	0.00
	[35]	8.59	242.63	0.08
	[34]	7.32	221.09	0.07
	[9]	11.83	240.25	0.12
	[10]	10.75	225.02	0.20
	[3]	8.67	170.66	0.18
	[12]	11.63	179.21	0.46
	[15]	9.57	221.26	0.14
	Ours	**6.60**	**127.10**	**0.07**
Vaselion $\sigma_n = 0.2e_l$ (Figure 11)	Noisy	25.11	12933.10	0.47
	Ground truth	0.00	2757.32	0.00
	[35]	21.13	7435.05	0.14
	[34]	NaN	8127.51	0.12
	[9]	24.92	7017.39	0.15
	[10]	21.78	6449.90	0.18
	[3]	16.84	5536.49	0.26
	[12]	25.66	4403.20	0.41
	[15]	21.55	6184.21	0.10
	Ours	**12.72**	**2741.82**	**0.02**
Parameter setting: [35] (10); [34] $(1, 1)$; [9] $(0.5, 20, 10)$; [10] $(1, 3.5 \times 10^{-1}, 20, 1.0 \times 10^{-2}, 10)$; [3] (Armadillo) $(1.4\times10^{-2}, 1.0\times10^{-3}, 1.0\times10^{3}, 0.5, 2.9\times10^{-3}, 3.0\times10^{-6})$, (Bunny) $(1.4 \times 10^{-2}, 1.0 \times 10^{-3}, 1.0 \times 10^{3}, 0.5, 5.2 \times 10^{-3}, 2.9 \times 10^{-4})$, (Max Planck) $(1.4 \times 10^{-2}, 1.0 \times 10^{-3}, 1.0 \times 10^{3}, 0.5, 3.8 \times 10^{-3}, 0.4)$, (Vaselion) $(1.4 \times 10^{-2}, 1.0\times10^{-3}, 1.0\times10^{3}, 0.5, 1.1\times10^{-3}, 2.0\times10^{-6})$; [12] $(2, 1, 0.35, 20, 1, 1.0 \times 10^{-2}, 10)$; [15] $(3.9 \times 10^{-1}, 20, 10)$; Ours (40);				

[2] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2mesh: Generating 3d mesh models from single rgb images," in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 52–67.

[3] L. He and S. Schaefer, "Mesh denoising via l 0 minimization," ACM Transactions on Graphics (TOG), vol. 32, no. 4, p. 64, 2013.

[4] R. Wang, Z. Yang, L. Liu, J. Deng, and F. Chen, "Decoupling noise and features via weighted l1-analysis compressed sensing," ACM Transactions on Graphics (TOG), vol. 33, no. 2, p. 18, 2014.

[5] M. Wei, J. Yu, W.-M. Pang, J. Wang, J. Qin, L. Liu, and P.-A. Heng, "Bi-normal filtering for mesh denoising," IEEE transactions on visualization and computer graphics, vol. 21, no. 1, pp. 43–55, 2014.

[6] P.-S. Wang, Y. Liu, and X. Tong, "Mesh denoising via cascaded normal regression." ACM Trans. Graph., vol. 35, no. 6, pp. 232–1, 2016.

[7] G. Arvanitis, A. S. Lalos, K. Moustakas, and N. Fakotakis, "Feature preserving mesh denoising based on graph spectral processing," IEEE transactions on visualization and computer graphics, vol. 25, no. 3, pp. 1513–1527, 2018.

[8] M. Wei, J. Wang, X. Guo, H. Wu, H. Xie, F. L. Wang, and J. Qin, "Learning-based 3d surface optimization from medical image reconstruction," Optics and Lasers in Engineering, vol. 103, pp. 110–118, 2018.

[9] X. Sun, P. Rosin, R. Martin, and F. Langbein, "Fast and effective feature-preserving mesh denoising," IEEE transactions on visualization and computer graphics, vol. 13, no. 5, pp. 925–938, 2007.

[10] Y. Zheng, H. Fu, O. K.-C. Au, and C.-L. Tai, "Bilateral normal filtering for mesh denoising," IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 10, pp. 1521–1530, 2011.

[11] L. Zhu, M. Wei, J. Yu, W. Wang, J. Qin, and P.-A. Heng, "Coarse-to-fine normal filtering for feature-preserving mesh denoising based on isotropic subneighborhoods," in Computer Graphics Forum. Wiley Online Library, 2013, pp. 371–380.

[12] W. Zhang, B. Deng, J. Zhang, S. Bouaziz, and L. Liu, "Guided mesh normal filtering," in Computer Graphics Forum. Wiley Online Library, 2015, pp. 23–34.

[13] X. Lu, Z. Deng, and W. Chen, "A robust scheme for feature-preserving mesh denoising," IEEE transactions on visualization and computer graphics, vol. 22, no. 3, pp. 1181–1194, 2015.

[14] S. K. Yadav, U. Reitebuch, and K. Polthier, "Mesh denoising based on normal voting tensor and binary optimization," IEEE transactions on visualization and computer graphics, vol. 24, no. 8, pp. 2366–2379, 2017.

[15] X. Li, L. Zhu, C.-W. Fu, and P.-A. Heng, "Non-local low-rank normal filtering for mesh denoising," in Computer Graphics Forum. Wiley Online Library, 2018, pp. 155–166.

[16] L. Dong, Y. Fang, W. Lin, and H. S. Seah, "Perceptual quality assessment for 3d triangle mesh based on curvature," IEEE Transactions on Multimedia, vol. 17, no. 12, pp. 2174–2184, 2015.

[17] Y. Lin, M. Yu, G. Jiang, Y. Song, and H. Shao, "A novel 3d mesh quality assessment method based on curvature analysis," in Optoelectronic Imaging and Multimedia Technology V. International Society for Optics and Photonics, 2018, p. 108170E.

[18] M. Eigensatz, R. W. Sumner, and M. Pauly, "A comparison of gaussian and mean curvatures estimation methods on triangular meshes," in Computer Graphics Forum. Wiley Online Library, 2008, pp. 241–250.

[19] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in Proceedings of the 26th annual conference on Computer graphics and interactive techniques. Citeseer, 1999, pp. 317–324.

[20] Y. Gong and I. F. Sbalzarini, "Curvature filters efficiently reduce certain variational energies," IEEE Transactions on Image Processing, vol. 26, no. 4, pp. 1786–1798, 2017.

[21] S.-H. Lee and J. K. Seo, "Noise removal with gauss curvature-driven diffusion," IEEE Transactions on Image Processing, vol. 14, no. 7, pp. 904–909, 2005.

[22] H. Zhao and G. Xu, "Triangular surface mesh fairing via gaussian curvature flow," Journal of Computational and Applied Mathematics, vol. 195, no. 1-2, pp. 300–311, 2006.

[23] H. Yamauchi, S. Gumhold, R. Zayer, and H.-P. Seidel, "Mesh segmentation driven by gaussian curvature," The Visual Computer, vol. 21, no. 8-10, pp. 659–668, 2005.

[24] Y. Gong, "Spectrally regularized surfaces," Ph.D. dissertation, ETH Zurich, 2015. [Online]. Available: http://e-collection.library.ethz.ch/eserv/eth:47737/eth-47737-02.pdf

[25] W. J. Firey, "Shapes of worn stones," Mathematika, vol. 21, no. 01, pp. 1–11, 1974.

[26] P. Jidesh and S. George, "Fourth-order gauss curvature driven diffusion for image denoising," International Journal of Computer and Electrical Engineering, vol. 4, no. 3, p. 350, 2012.

[27] Y. Gong and I. F. Sbalzarini, "Local weighted gaussian curvature for image processing," in 2013 IEEE International Conference on Image Processing. IEEE, 2013, pp. 534–538.

[28] J. Peng, Q. Li, C.-C. J. Kuo, and M. Zhou, "Estimating gaussian curvatures from 3d meshes," in Human Vision and Electronic

Imaging VIII. International Society for Optics and Photonics, 2003, pp. 270–281.
[29] T. Surazhsky, E. Magid, O. Soldea, G. Elber, and E. Rivlin, "A comparison of gaussian and mean curvatures estimation methods on triangular meshes," in 2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422). IEEE, 2003, pp. 1021–1026.
[30] X. Lu, X. Liu, Z. Deng, and W. Chen, "An efficient approach for feature-preserving mesh denoising," Optics and Lasers in Engineering, vol. 90, pp. 186–195, 2017.
[31] H. Pottmann and J. Wallner, Computational line geometry. Springer Science & Business Media, 2009.
[32] O. Sorkine, "Differential representations for mesh processing," in Computer Graphics Forum, vol. 25, no. 4. Wiley Online Library, 2006, pp. 789–807.
[33] G. Taubin, "A signal processing approach to fair surface design," in Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. ACM, 1995, pp. 351–358.
[34] T. R. Jones, F. Durand, and M. Desbrun, "Non-iterative, feature-preserving mesh smoothing," in ACM Transactions on Graphics (TOG). ACM, 2003, pp. 943–947.
[35] S. Fleishman, I. Drori, and D. Cohen-Or, "Bilateral mesh denoising," in ACM transactions on graphics (TOG). ACM, 2003, pp. 950–953.
[36] O. Stein, E. Grinspun, and K. Crane, "Developability of triangle meshes," ACM Transactions on Graphics (TOG), vol. 37, no. 4, p. 77, 2018.