Fast and Effective $L_0$ Gradient Minimization by Region Fusion

Rang M. H. Nguyen  
Michael S. Brown  
School of Computing, National University of Singapore  
goinguyenho@comp.nus.edu.sg | brown@comp.nus.edu.sg

Abstract

$L_0$ gradient minimization can be applied to an input signal to control the number of non-zero gradients. This is useful in reducing small gradients generally associated with signal noise, while preserving important signal features. In computer vision, $L_0$ gradient minimization has found applications in image denoising, 3D mesh denoising, and image enhancement. Minimizing the $L_0$ norm, however, is an NP-hard problem because of its non-convex property. As a result, existing methods rely on approximation strategies to perform the minimization. In this paper, we present a new method to perform $L_0$ gradient minimization that is fast and effective. Our method uses a descent approach based on region fusion that converges faster than other methods while providing a better approximation of the optimal $L_0$ norm. In addition, our method can be applied to both 2D images and 3D mesh topologies. The effectiveness of our approach is demonstrated on a number of examples.

1. Introduction and Related Work

This paper focuses on $L_0$ gradient minimization applied to images and 3D meshes. $L_0$ gradient minimization is used to control the global number of non-zero gradients between neighbors in a graph. When done properly, $L_0$ gradient minimization effectively creates a series of piecewise constant functions whose transitions correspond to important changes in the original function. This can be considered as a feature-preserving filter that has many applications in computer vision, from image and mesh denoising, to image enhancement and segmentation.

There are a number of feature-preserving filters in the literature. These can be broadly categorized as local or global filters. The trade-off among these various methods lies in their overall effectiveness for the task at hand and time complexity. Well-known local filters include the anisotropic filter [1, 16], the bilateral filter [2, 6, 15, 21, 26, 27], the guided filter [10, 11, 28], and the geodesic filter [9, 4]. These filters were originally designed for use on images, but have been extended to other domains, including 3D meshes [8, 29]. Local filters are popular given their simplicity and effectiveness in smoothing noise while preserving edges, however, they do require the tuning of the filter’s local support size. In addition, as discussed in [7], it can often be difficult to use local filters to achieve progressive coarsening.

$L_0$ gradient minimization falls into the category of global filters that impose a constraint on the entire image. Most global filters are realized in an optimization framework. One of the best known global methods is the work by Rudin et al. [17] and subsequent variations (e.g. [5, 23, 25]) that introduced the total variation (TV) minimization which essentially imposes an $L_1$ gradient minimization. Another effective global filter is the weighted-least-square filter [7, 13] that solves an $L_2$ objective function defined over the entire image, using image affinity based on local image gradients.
Recently, the $L_0$ gradient minimization was used by Xu et al. [24] for the task of image smoothing. As previously mentioned, this approach aims to limit the number of gradient transitions in the output image. However, in terms of computational complexity, minimizing the $L_0$ norm is NP-hard [14]. As such, approximation strategies must be used. Xu et al. [24] split their optimization function into two subproblems by introducing auxiliary variables and solved them by an $L_0$-$L_2$ iteration framework. A parameter $\kappa$ was used to balance the influence of the $L_0$ norm on the final filtered signal. A smaller $\kappa$ weights the $L_0$ regularization more, but at the cost of a longer running time. Figure 1 shows an example, where $F$ represents the final optimization energy and $T$ is the time in seconds. Cheng et al. [3] extended this work to propose a better approximation algorithm based on a fused-coordinate descent framework (as shown in Figure 1-d). However, their method is also slow to converge, often needing $700-1000$ iterations. Storath et al. [19] proposed an optimization method based on dynamic programming and alternating direction method of multipliers. Their approach obtains a good approximation of the $L_0$, but requires a significantly large running time (as shown in Figure 1-e). Shen et al. [18] defined a different optimization function that changed the $L_2$ norm term in the optimization function into an $L_1$ norm. As with local filters, the $L_0$ gradient minimization has been extended beyond images to work with 3D meshes (e.g. He et al. [12], Cheng et al. [3]).

**Contribution** We present a method for $L_0$ gradient minimization that is fast and is able to approximate the $L_0$ norm effectively. Our method uses a descent strategy based on region fusing. We show that at each minimization step, the objective function will decrease monotonically. This allows our method to converge quickly. Moreover, our method can be applied to arbitrary graphs such as image and mesh topologies. We detail our algorithm and demonstrate its effectiveness on a number of experiments showing that it provides a good approximation of the $L_0$ norm and is significantly faster than prior methods.

The remainder of paper is organized as follows. Section 2 overviews the $L_0$ gradient minimization framework and its application to different graph topologies. Section 3 describes our descent and fusion approach for approximating $L_0$ gradient minimization. Section 4 evaluates the performance of our method against existing $L_0$ minimization approaches. Section 5 shows the effectiveness of our approach on a number of applications. The work is concluded in Section 6.

### 2. $L_0$ Gradient Minimization

Here we describe $L_0$ gradient minimization for a general input signal. Let $I$ be the original signal and $S$ represent the filtered output. The gradient of the output $S$ is denoted by $\nabla S$. The objective function of the $L_0$ gradient minimization is formulated as follows:

$$ F = \min_S ||S - I||^2 + \lambda ||\nabla S||_0, \tag{1} $$

where $|| \cdot ||$ denotes the $L_2$ norm, $|| \cdot ||_0$ denotes $L_0$ norm, and $\lambda$ is the parameter to control the level of sparseness in the final signal $S$. A larger $\lambda$ produces a coarser result with less gradient.

Equation 1 can be rewritten as follows:

$$ F = \min_S \sum_{i=1}^{M} \left[ ||S_i - I_i||^2 + \lambda/2 \sum_{j \in N_i} ||S_i - S_j||_0 \right], \tag{2} $$

where $M$ is the length of the signal and $N_i$ is the neighboring set of the $i^{th}$ element. Here, $\lambda$ is divided by 2 since the neighboring relationship between $S_i$ and $S_j$ is counted twice. The neighboring set $N_i$ is defined for each case (e.g. 1D signal, 2D images, or 3D mesh models) as follows:

$$ N_i = \begin{cases} 
\{i - 1, i + 1\} & \text{1D} \\
\{\text{four-connected pixels}\} & \text{2D} \\
\{\text{all neighbor faces of the } i^{th} \text{ face}\} & \text{3D}
\end{cases} \tag{3} $$

### 3. Our Region Fusion Minimization

Our goal is to minimize the objective function introduced in Equation 2. Our method uses a fusion technique that examines neighboring regions in the signal that have nearly similar values and decides whether to fuse them to have the same value. By combining two regions, we create a larger single region, but also remove the gradients between the regions. The work by Cheng et al. [3] used a somewhat similar method termed the fused-coordinate descent. Their work, however, used a more complicated optimization mechanism that separated the fusion step from the coordinate descent step in a way that did not guarantee the objective function to monotonically decrease during the fusion step (see accompanying supplemental material for more details). Their method also required a longer running time. In contrast, our approach combines the fusion and descent into a single step that guarantees a decrease of the objective function and allows our method to converge quickly. Our optimization approach is detailed in the following.

#### 3.1. Optimization

As discussed in Section 1, the inclusion of the $L_0$ norm in the objective function makes it NP-hard. We approximate this objective function by considering each pair of neighboring elements in the graph one at a time, instead of the entire signal at once. Our descent optimization works as follows. We first assign the output signal $S$ to be the same as the input signal $I$. Our algorithm then loops through all the signal elements. At each step, we consider two neighboring elements $i$ and $j$. The amount these two elements contribute
to the objective function $F$ in Equation 2 can be expressed as follows:

$$f = \min_{S_i, S_j} ||S_i - I_i||^2 + ||S_j - I_j||^2 + \lambda ||S_i - S_j||. \quad (4)$$

Our goal here is to find the best $S_i$ and $S_j$ that minimize the sub-function $f$. To do this, we divide the problem into two cases: $S_i \neq S_j$, and $S_i = S_j$ to eliminate the $L_0$ term $||S_i - S_j||_0$ in Equation 4.

- **Case $S_i \neq S_j$:** The $L_0$ term $||S_i - S_j||_0$ is equal to 1 and the function $f$ in Equation 4 becomes:

$$f = \min_{S_i, S_j} ||S_i - I_i||^2 + ||S_j - I_j||^2 + \lambda. \quad (5)$$

In this case, we have a trivial solution as follows:

$$\begin{cases} S_i = I_i, & S_j = I_j \\ f = \lambda \end{cases}. \quad (6)$$

- **Case $S_i = S_j$:** The $L_0$ term $||S_i - S_j||_0$ is equal to 0 and the function $f$ in Equation 4 becomes:

$$f = \min_{S_i} ||S_i - I_i||^2 + ||S_i - I_j||^2. \quad (7)$$

Equation 7 is a quadratic equation that requires only one variable $S_i$ to be solved. By using the first derivative, its solution can be easily obtained as follows: $S_i = (I_i + I_j)/2$. Therefore, the solution for this case is:

$$\begin{cases} S_i = S_j = (I_i + I_j)/2 \\ f = (I_i - I_j)^2/2 \end{cases}. \quad (8)$$

Combining these two cases together, we have the solution for Equation 4 as follows:

$$\{S_i, S_j\} = \begin{cases} \{A, A\} & \text{if } ||I_i - I_j||^2/2 \leq \lambda \\ \{I_i, I_j\} & \text{otherwise} \end{cases}, \quad (9)$$

where $A = (I_i + I_j)/2$.

We call Equation 9 the fusion criterion. Note that Equation 9 still holds true in the case of $I_i = I_j$. According to this fusion criterion, we will decide whether to fuse these two elements into one group or not.

Our overall approach is described in Algorithm 1. Elements in the signal (e.g. pixels in an image or faces on a mesh) are denoted as $I_i$. Group (connected regions) of elements with the same values will be denoted as $G_i$. The number of elements in each group is denote as $w_i$. The number elements that connect group $i$ and $j$ is denote as $c_{i,j}$. To initialize the algorithm, each group $G_i$ contains exactly one element $i$. Therefore, the number of elements of each group $w_i$ is equal to 1. We use $Y_i$ to store the mean value of all elements in group $G_i$ which is initialized to the original signal $I_i$. All neighboring groups $N_i$ are initialized using Equation 3. We also define the number of initial connections between two neighboring groups as follows:

$$c_{i,j} = \begin{cases} 1 & \text{if } j \in N_i \\ 0 & \text{otherwise} \end{cases}. \quad (10)$$

The matrix that represents the connection numbers $c$ is very large, but sparse, since each element has only a few neighbors. Therefore, we can use a sparse matrix representation storing only non-zero values to save memory. Note that the connection number is equivalent to the number of gradients between two neighboring groups. Figure 2-(a) shows an example of group neighbors and the connection numbers for a 2D image. All pixels that belong to the same group have the same numerical value, e.g. $G_1$ has 10 elements in Figure 2-(a). Groups $G_1$ and $G_2$ have five elements connections together (shown along the green line with double arrows), therefore, their connection numbers are $c_{1,2} = c_{2,1} = 5$. Note that these connections are counted twice, once for $c_{i,j}$ and $c_{j,i}$, and account for why $\lambda$ is divided by two in Equation 2.

Our algorithm loops through all groups of a current filtered signal. For each group $G_i$, we consider its neighbors $G_j$. Like prior methods, we use an auxiliary parameter $\beta$ ($0 \leq \beta \leq \lambda$) that increases for each iteration. Details to this parameter are provided in Section 3.2. Factoring in the auxiliary parameter, Equation 4 becomes as follows:

$$\min_{S_i, S_j} ||S_i - Y_i||^2 + \lambda ||S_j - Y_j||^2 + \beta c_{i,j} ||S_i - S_j||. \quad (11)$$

Recall that $Y_i$ and $Y_j$ represent the mean signal values for the groups $G_i$ and $G_j$ containing $w_i$ and $w_j$ elements respectively. The above equation can be solved in the exact same manner as described for Equation 4 as follows:
Algorithm 1: Region Fusion Minimization for $L_0$

**Input:** signal $I$ of length $M$, the level of sparseness $\lambda$

1. $G_i \leftarrow \{i\}$, $Y_i \leftarrow I_i$, $w_i \leftarrow 1$
2. Initialize $N_i$ as Equation 3
3. Initialize $c_{i,j}$ as Equation 10
4. $\beta \leftarrow 0$, $\text{iter} \leftarrow 0$, $P \leftarrow M$
5. **repeat**
   6. $i \leftarrow 1$
   7. **while** $i \leq P$ **do**
      8. **for all** $j \in N_i$ **do**
         9. **if** $w_i w_j |Y_i - Y_j|^2 \leq \beta c_{i,j}(w_i + w_j)$ **then**
            10. $G_i \leftarrow G_i \cup G_j$
            11. $Y_i \leftarrow (w_i Y_i + w_j Y_j)/(w_i + w_j)$
            12. $w_i \leftarrow w_i + w_j$
            13. Remove $j$ in $N_i$ and delete $c_{i,j}$
            14. **for all** $k \in N_j \setminus \{i\}$ **do**
               15. **if** $k \in N_i$ **then**
                  16. $c_{i,k} \leftarrow c_{i,k} + c_{j,k}$
                  17. $c_{k,i} \leftarrow c_{i,k} + c_{j,k}$
               18. **else**
                  19. $N_i \leftarrow N_i \cup \{k\}$
                  20. $N_k \leftarrow N_k \cup \{i\}$
                  21. $c_{i,k} \leftarrow c_{k,j}$
                  22. $c_{k,i} \leftarrow c_{j,k}$
               23. **end if**
               24. Remove $j$ in $N_k$ and delete $c_{k,j}$
            25. **end for**
            26. Delete $G_j$, $N_j$, $w_j$
            27. $P \leftarrow P - 1$, $i \leftarrow i + 1$
         19. **end if**
      20. **end for**
   21. **end while**
   22. $\text{iter} \leftarrow \text{iter} + 1$
   23. $\beta \leftarrow g(\text{iter}, K, \lambda)$ $\triangleright$ Defined in Equation 13
24. **until** $\beta > \lambda$
25. **for** $i = 1 \rightarrow P$ **do**
      26. **for all** $j \in G_i$ **do**
         27. $S_j \leftarrow Y_i$
      28. **end for**
      29. **end for**
   30. **end for**

**Output:** filtered signal $S$ of length $M$

$${S_i, S_j} = \begin{cases} 
  \{B, B\} & \text{if } w_i w_j |Y_i - Y_j|^2 \leq \beta c_{i,j}(w_i + w_j) \\
  \{Y_i, Y_j\} & \text{otherwise}
\end{cases} \quad (12)$$

where $B = (w_i Y_i + w_j Y_j)/(w_i + w_j)$ is the weighted average of the two groups $G_i$ and $G_j$.

The criterion in Equation 12 is used to decide whether to fuse the group $G_j$ into the group $G_i$ or not. Note that changing the values of groups $G_i$ and $G_j$ can affect the magnitudes of gradients with their other neighbors but it does not create any new non-zero gradients. Therefore, this will not affect the other terms in Equation 2. In addition, $\beta \leq \lambda$, if $\beta$ satisfies Equation 12, then $\lambda$ does too. Therefore, the total objective function $F$ in Equation 2 will decrease by $\lambda c_{i,j} - w_i w_j |Y_i - Y_j|^2/(w_i + w_j)$ if we preform the fusion step, or will remain unchanged otherwise. As a result, our fusion-based method acts as a descent strategy to either lower the objective function or remain the same.

If these groups are fused, all elements in $G_j$ are joined into $G_i$, then the mean value $Y_i$ and the element number $w_i$ are updated. Next, all the neighbors of $G_j$ are inserted into $N_i$ and the corresponding connection numbers are updated. After each fusion step, all the information of the fused group $G_j$ is deleted and the number of remaining groups is reduced by one. Figure 2-(b) shows an example of how the connection numbers are updated during the fusion step. Here, group $G_2$ is fused into $G_1$. The connection numbers $c_{1,3}, c_{3,1}$ (between $G_1$ and $G_3$) and $c_{1,4}, c_{4,1}$ (between $G_1$ and $G_4$) are updated. All connection numbers related to $G_2$ are deleted.

Our algorithm is repeated until the auxiliary parameter $\beta$ reaches the regular parameter $\lambda$. At that time, the remaining groups contain all elements in the signal. Finally, the output value for each element $S_j$ will be assigned by the mean value of the group that it belongs to.

Figure 3 shows the value of the objective function, $F$, in Equation 2 with each iteration. For this example, the input signal is the image shown in Figure 1. The objective function monotonically decreases with each iteration, converging after approximately 50 iterations.

### 3.2. Auxiliary Parameter $\beta$

Like the other prior works [3, 24], we use an auxiliary parameter $\beta$ which gradually increases from 0 to $\lambda$ at each iteration. This parameter is used to make pairs of neighboring groups that have small differences in their mean values fuse together. We experimented with three different strategies to increase $\beta$: linear, non-linear, and multiplicative, de-

![Figure 3](image_url)
shows some examples of content-based color quantization. Figure 4 shows the results with different values of the maximum iteration number (e.g., 10, 20, 30, 40, 50) on three strategies for increasing β, i.e., linear, nonlinear, and multiplicative. The left shows the values for objective function, F, while the corresponding running-time are shown in the right.

\[ g(\text{iter}, K, \lambda) = \begin{cases} (\text{iter}/K)^\lambda & \text{linear} \\ (\text{iter}/K)^\gamma & \text{nonlinear} \\ \alpha(\text{iter} - K)^\lambda & \text{multiplicative} \end{cases} \] (13)

Figure 4 shows the results with different values for the maximum iteration number K (e.g., 10, 20, 30, 40, 50). The left plot shows the values for objective function while the corresponding run-time is shown in the right plot. Here, we use γ = 2.2 for the nonlinear increase function and α = 1.5 for multiplication strategy. As shown in Figure 4, the multiplication strategy is the worst in terms of the objective function value and running time. The linear strategy is faster than the nonlinear approach, but has a slightly larger value for the objective function. Based on this, we choose the nonlinear strategy for our \( L_0 \) gradient minimization, since it gives us the best approximation to the \( L_0 \) norm.

4. Experiments

In this section, we compare our method with the three other \( L_0 \) gradient minimizations proposed by Xu et al. [24], Cheng et al. [3], and Storath et al. [19]. The comparison examines the final objective function value F defined in Equation 2 and running-time T. All the experiments are run on a dual core 3.10 GHz PC with 16.0 GB RAM. Our method is implemented in C++. For processing an image sized approximately 600 × 400, our approach takes roughly 1 second. The other methods [3, 19, 24] are implemented in C++, Java, and Matlab by their authors. Xu et al.’s method [24] has a parameter \( \kappa \) to control the amount of \( L_0 \) minimization. A smaller \( \kappa \) gives results with better \( L_0 \) minimization, but requires more iterations. In our experiments, we report Xu et al.’s results for both \( \kappa = 2 \) and \( \kappa = 1.05 \).

Figure 5 shows the results of each method. As can be seen, the objective function values, F, obtained from Xu et al.’s method are notably large. This is because of \( L_2 \) step in their approximation that smooths the signal. As a result, there are still small gradients left in the output signal (shown in the plot in second row). Our results are very close to Storath et al. [19] that are the best in terms of minimizing the objective function, however, our running-time is significantly faster.

5. Applications

In this section, we show our approach applied to several applications involving images and 3D meshes. In particular, we demonstrate: image denoising, content-based color quantization, clip art compression artifact removal, and 3D mesh denoising. Additional results for each application are also included in the supplemental material.

5.1. Image Denoising

The \( L_0 \) gradient minimization can approximate the input signal by a series of piecewise constant functions. Therefore, it can be used to denoise the 2D images that have sparse colors with sharp edges.

In this experiment, we compare our method against two methods: the total variation method proposed by Dahl et al. [5], and \( L_0 \) gradient proposed by Storath et al. [19]. Figure 6 shows two examples of image denoising. Storath et al. [19] obtain the best results in terms of signal-to-noise ratio (SNR). Our quantitative results are very close to Storath et al.’s results, but again, our running-time is significantly faster.

5.2. Content-Based Color Quantization

Content-based color quantization is used to reduce the number of colors in an image. This is useful for tasks such as image segmentation or image retrieval since it reduces the color complexity of an imaged scene.

Figure 7 shows an example of content-based color quantization. Figure 7-(a) shows the original input image that over 100,000 different colors (number of colors denote as \( P \)). Figure 7-(b)-(e) show our results with different values for the level of sparseness \( \lambda \). Our method reduces the number of colors in image but still keep the overall image structure. Figure 8 shows some examples of content-based color quantization. As can be seen, our approach can obtain results with the least number of remaining colors in the fastest time.

5.3. Clip Art Compression Artifact Removal

Using JPEG compression on clip art images often create artifacts as shown in Figure 9-(a). Most of clip art images have sparse colors with sharp edges and artifacts are most noticeable near the sharp edges. Prior works [3, 22, 24] showed that using local filters can reduce these artifacts, but they also tend to blur the edges. The \( L_0 \) gradient minimization is well-suited to remove these artifacts.

In this experiment, we compare our method with Wang et al. [22], Xu et al. [24], and Cheng et al. [3]. The work by
Figure 5. This figure shows the details of the experiment that was already shown in Fig. 1 in Sec. 1. One more result for Xu et al.’s method with $\kappa = 2$ is added in here. (a) Input images. (b)-(c) Results of Xu et al. [24] with different $\kappa = 2$ and $\kappa = 1.05$ ($\lambda = 0.05$). (d) Results of Cheng et al. [3] ($\lambda = 0.2$). (e) Results of Storath et al. [19] ($\lambda = 0.2$). (f) Our results ($\lambda = 0.2$). The plot in the second row shows a 1D scanline from the green channel. We also report the running time (in seconds) and the objective function value defined in Equation 2.

Figure 6. This figure shows two examples of image denosing. (a) Ground truth images. (b) Noisy input images. (c) Results of total variation method proposed by Dahl et al. [5]. (d) Results of Storath et al. [19]. (e) Our results. We also report the running time (in seconds) and the signal-to-noise-ratio SNR (in dB).

Figure 7. This figure shows an example of content-based color quantization. (a) The original input image with 132752 different colors. (b)-(e) Our results with different values for $\lambda$. The numbers reported in parenthesis are the remaining colors $P$. 

213
Wang et al. [22] is explicitly designed for artifact reduction for JPEG compressed clip art. Figure 9 shows two examples of artifact reduction. Our method obtains comparable results to prior methods but with the fastest running time.

5.4. Mesh Denoising

Our approach can also be applied to 3D mesh topologies. We modified Algorithm 1 to use the two-step framework proposed by Sun et al. [20]. In step one, we use our $L_0$ gradient minimization to filter the noisy face normals. In step two, the vertex coordinates are reconstructed from the filtered face normals using the iterative updating vertex framework (see [20] for more details).

The nature of the $L_0$ gradient minimization will approximate the input signal by a series of piecewise constant functions. Therefore, it can be used to denoise the 3D meshes that have sharp transitions between their faces. In this experiment, we test our approach on several sharp-edge meshes and compare our results with the results of three methods: Sun et al. [20], He et al. [12], and Cheng et al. [3]. Sun et al.’s method is a local filtering method based on the weighted averaging of neighboring face normals. He et al.’s method is an extension of Xu et al.’s method [24] to work on 3D meshes. Figure 10 shows two examples of denoising on 3D mesh models. Here, the noisy 3D models are synthesized from the ground truth ones. The results show that our approach produces the best results and with the fastest running time.

6. Concluding Remarks

We have described a fusion-based descent method for $L_0$ gradient minimization. The nature of $L_0$ gradient minimization will approximate the input signal by a series of piecewise constant functions. Therefore, applying $L_0$ minimization to smooth signals will create artifacts that may be undesirable. For example, Figure 11 shows the example for the task of image smoothing. The $L_0$-$L_2$ iterative strategy proposed by Xu et al. [24] is arguably more appropriate than our approach for this task. Similarly, for the mesh models with smooth surfaces, our method creates a sharp mesh instead of a smooth one. He et al.’s method [12] based on Xu et al.’s strategy is more suitable when smooth regions are present (see Figure 12). However, for tasks for which $L_0$ gradient minimization is needed, the proposed method in this paper offers a fast and effective approach that can be applied to both 2D images and 3D mesh topologies.
Figure 9. This figure shows two examples of JPEG compression artifact removal on clip art images. The first and third rows show the complete images, while the second and fourth rows shows the corresponding close-ups in these images. (a) Input JPEG compression artifact images. (b) The ground truth images without compression. (c) Results of Wang et al. [22]. (d) Results of Xu et al. [24] (λ = 0.025, κ = 1.5 for the first image, and λ = 0.01, κ = 1.5 for the second image). (e) Results of Cheng et al. [3] (λ = 0.26 for the first image, and λ = 0.08 for the second image). (f) Our results (λ = 0.26 for the first image, and λ = 0.08 for the second image).

Figure 10. This figure shows two examples of denoising 3D mesh models. The first model contains 24578 vertices and 49152 faces, while the second one contains 10242 vertices and 20480 faces. (a) Noisy 3D mesh models. (b) The ground truth meshes. (c) Results of Sun et al. [20] (τ = 0.55 for the first model, and τ = 0.4 for the second model). (d) Results of Cheng et al. [3] (λ = 0.5 for the first model, and λ = 0.04 for the second model). (e) Our results (λ = 0.5 for the first model, and λ = 0.04 for the second model). We also report the running-time T (in seconds) for each method.
References


