A Projection Operator for Representing Sharp Features using Visibility

Hiroaki Kawata[1] and Takashi Kanai[2]

[1]The University of Tokyo, hiroaki@graco.c.u-tokyo.ac.jp
[2]The University of Tokyo, kanai@graco.c.u-tokyo.ac.jp

ABSTRACT

We propose a projection operator for point sets without normal vector information. Based on the visibility of points, it is possible to represent sharp features when points are projected on the surface without normal vector information. Our method is based on the Moving Least-Square Surface projection operator. Visibility computation is utilized to avoid average computation of points near sharp features when selecting points, in which visibility information is computed before starting the projection for selecting point clouds of one side. We also demonstrate the effectiveness of our visibility approach through several experiments.

Keywords: point-based, projection, sharp feature
DOI: 10.3722/cadaps.2013.33-44

1 INTRODUCTION

The projection operator is a geometrical process for moving a point onto a surface defined by its neighbor points. There are a number of applications such as noise reduction, point up-sampling, deformation, etc. In this study, we focus on a projection operator as a pre-process of surface reconstruction from point clouds [4].

The MLS (Moving Least-Square) surface projection operator [2] is a simple method to project a point to a smooth surface defined from point clouds (called point set surface) based on MLS [1]. In this projection method, a noisy point cloud also tends to be smooth. However, sharp features such as corners or creases cannot be preserved due to its original nature. Such features may also frequently appear due to the measurement of mechanical parts using 3D range scanners.

As other types of projection operators, LOP [6] and its variant WLOP [9] are proposed. Those approaches are both robust for noises including outliers. In addition, normal vectors are not also required. Another characteristic is that projected points are uniformly distributed. Unfortunately, their approaches do not also preserve sharp features.

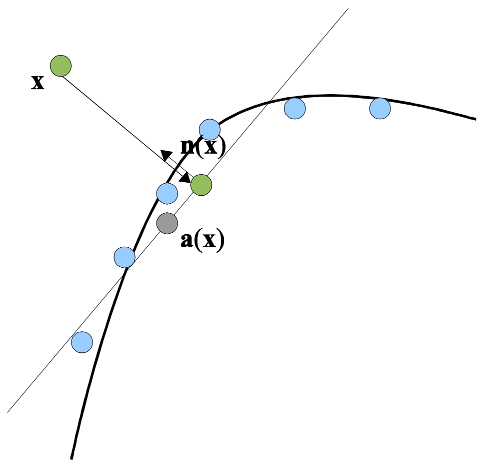


Fig. 1: Projection of point x to surface defined by point set P.

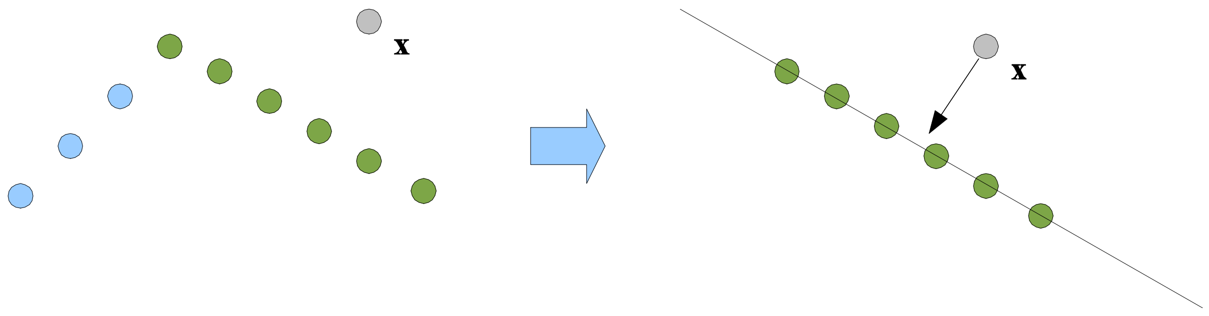
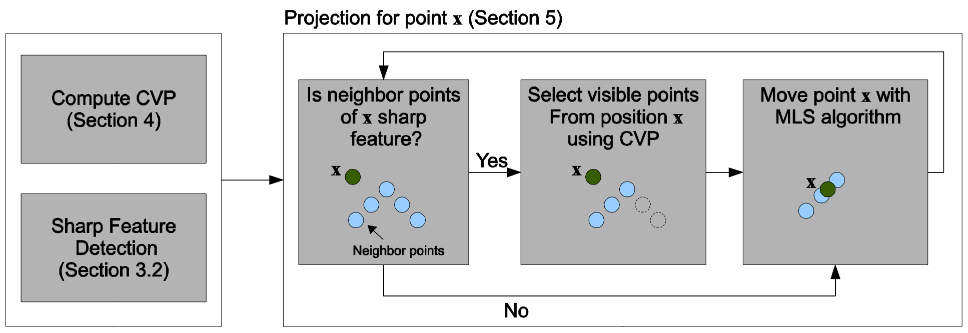


Fig. 2: Selection of points for point x (green points) and projection of point x to selected points.



Projection for point x (Section 5)
Compute CVP (Section 4)
Sharp Feature Detection (Section 3.2)
Is neighbor points of x sharp feature?
x
Neighbor points
Yes
No
Select visible points From position x using CVP
x
Move point x with MLS algorithm
x

Fig. 3: Proposed algorithm. CVP is computed and sharp features are detected before projection, and point x is projected using CVP and sharp feature information.

Several approaches have been considered for preserving sharp features based on MLS [3],[8],[10]. With [3], Least Median of Squares (LMS) is used to fit several planes to points near the region of sharp features. With [8], the weight function of MLS is optimized to preserve the relevant sharp features. However, these methods require normal vector information as an input. In general, computing normal vectors is a difficult task, especially from noisy point clouds obtained using range scanners. In this paper, we focus on projection to a point cloud which does not have normal vectors.

We also propose a novel projection operator based on MLS which preserves sharp features. To preserve sharp features, points which consist of each face of sharp features have to be separately selected before projecting a point. In our approach, a visibility test is utilized to handle such separation. Visibility test is based on the HPR operator [7] in which visibility is computed for points from the view position. In our algorithm including visibility tests, normal vectors are not necessarily required as an input and are computed on-the-fly. This is an advantage of our method, especially when it is applied to a point cloud taken from physical objects.

This paper is organized as follows. Section 2 reviews the MLS-based projection operator, Sections 3, 4, and 5 discuss the method, Section 6 discusses the results, and Section 7 summarizes our conclusions.

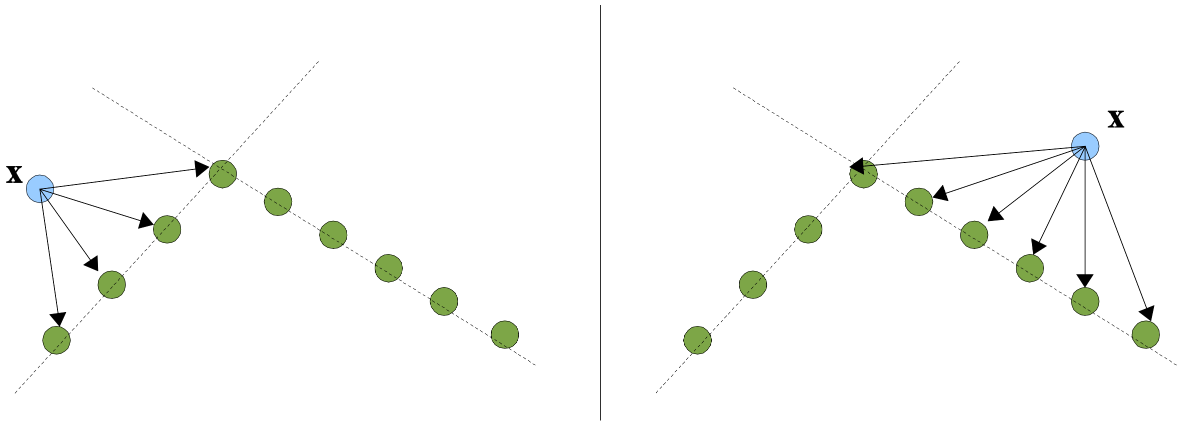


Fig. 4: Dependence of selected points on position of x.

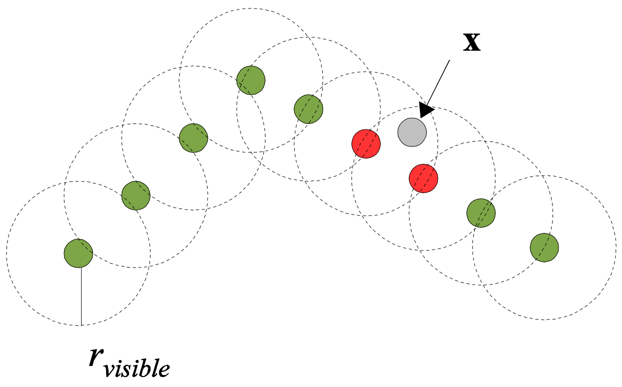


Fig. 5: Selection of only red points in selection of points using HDR operator.

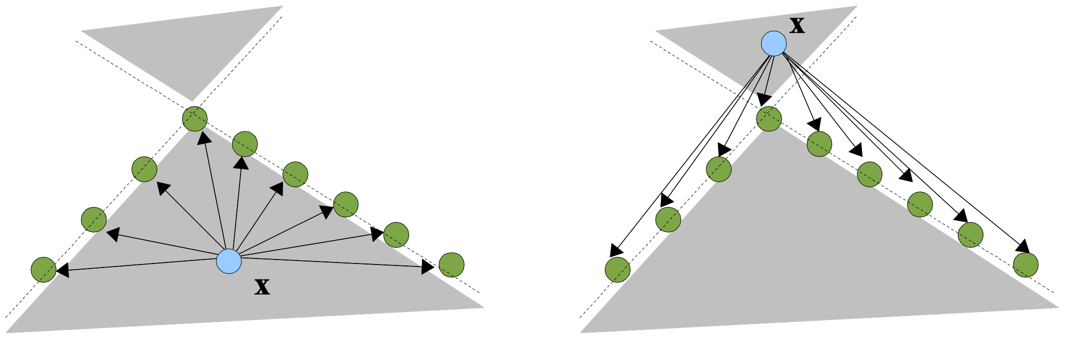


Fig. 6: Selection of points composed of more than two planes.

2 MOVING LEAST-SQUARE SURFACE PROJECTION OPERATOR

The Moving Least Squares (MLS) projection operator [2] is a convenient tool for cleaning up input point clouds without changing the number of points, which means that there are no adding/deleting operations. In this operator, a point $\mathbf{x} \in \mathrm{R}^3$ is projected onto a surface which is defined by input point $\mathbf{p}_i = (\mathrm{x}_i, \mathrm{y}_i, \mathrm{z}_i) \in P$. Let $f(\mathbf{x})$ be the distance function which computes the distance from a point $\mathbf{x}$ to a plane defined by a weighted average position $\mathbf{a}(\mathbf{x})$ of neighbor points and a weighted average $\mathbf{n}(\mathbf{x})$ of normal vectors:

$$f(\mathbf{x}) = \mathbf{n}(\mathbf{x})(\mathbf{a}(\mathbf{x}) - \mathbf{x}) = 0,$$
$$\mathbf{a}(\mathbf{x}) = \frac{\sum_{i=1}^{N} \theta(|\mathbf{p}_i - \mathbf{x}|)\mathbf{p}_i}{\sum_{i=1}^{N} \theta(|\mathbf{p}_i - \mathbf{x}|)},$$
$$\mathbf{n}(\mathbf{x}) = \frac{\sum_{i=1}^{N} \theta(|\mathbf{p}_i - \mathbf{x}|)\mathbf{n}_i}{\sum_{i=1}^{N} \theta(|\mathbf{p}_\mathbf{i} - \mathbf{x}|)},$$
$$\theta(d) = e^{-\frac{d^2}{h^2}},$$

where $\theta(d)$ denotes a weight function. In our assumption each point does not have a normal vector $\mathbf{n}_\mathrm{i}$ attached with a point $\mathbf{p}_i$ as an input. Alternatively, $\mathbf{n}_\mathrm{i}$ can be computed by Principal Component Analysis (PCA) with respect to neighbor points. In the projection operator, the orientation of normal vector is not so important since the function f can be evaluated as a signed value associated with the orientation of $\mathbf{n}(\mathbf{x})$. The algorithm of the projection operator is shown as follows:

1. Compute $\mathbf{a}(\mathbf{x})$ and $\mathbf{n}(\mathbf{x})$.
2. Compute $f(\mathbf{x}) = \mathbf{n}(\mathbf{x}) \cdot (\mathbf{a}(\mathbf{x}) - \mathbf{x})$.
3. Move a point to $\mathbf{x} + \mathbf{n}(\mathbf{x})f(\mathbf{x})$, and set the position to $\mathbf{x}$.
4. Algorithm terminates if $|f| < e$, otherwise, back to 2.

When computing $\mathbf{a}(\mathbf{x})$ and $\mathbf{n}(\mathbf{x})$, the weight function is set to 0, if d is greater than h. This is why they can be computed by the weighted average of only neighbor points at $\mathbf{x}$. Here we represent neighbor points of $\mathbf{x}$ as P_n. We efficiently search neighbor points by using k-D tree data structure. Fig. 1 shows the projection operator.

3 PROPOSED PROJECTION METHOD

In this section, we describe the projection operator for preserving sharp features based on MLS. One problem of the original MLS projection operator is that it always computes a weighted average $\mathbf{a}(\mathbf{x})$ and $\mathbf{n}(\mathbf{x})$ for all points around a sharp feature. In nature, points around sharp features configure two or more planes which cross each other. Therefore, only points which approximate a plane have to be selected and a point has to be projected on a plane defined by such selected points to preserve the sharp features as shown in Fig. 2.

In our projection algorithm, the visibility test is used for selecting points for sharp features. Using only points selected as visible points for fitting to a plane, these points can be projected onto such a plane composed of sharp features. The following subsection describes the details.

3.1 Overview of Our Algorithm

Our approach is based on the MLS projection operator. With the original MLS, the algorithm computes $\mathbf{a}(\mathbf{x}), \mathbf{n}(\mathbf{x})$ for all neighbor points P_n even around sharp features. Instead, our algorithm uses a part of neighbor points $P_{visible} \subseteq P_n$. Points in $P_{visible}$ are selected by the visibility test described later. If $P_{visible}$ is identical to P_n, the result of projection by our algorithm is the same as that of the original MLS. Fig. 3 describes our algorithm.

In the pre-processing stage, we judge whether a point in P is on sharp feature or not and store such a *"sharp feature" flag* (Section 3.2). We also compute *Cached View Positions* (CVPs) which satisfy the conditions of the visibility test (Section 4).

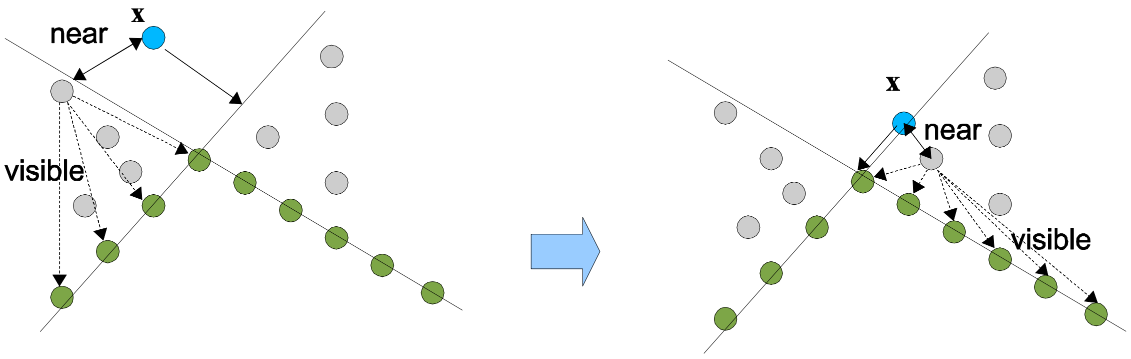


Fig. 7: Selection of visible points using CVPs (gray-colored points). First, a view position close to x is found, and then visible points $P_{visible}$ are computed from such a view position.

For the algorithm, we first find neighbor points P_n for a point $\mathbf{x}$. If at least one "sharp feature" flag is found in P_n, then we proceed with the process of preserving sharp features, otherwise we apply the original MLS projection operator by using P_n. In the process of sharp feature preservation, we find a cached view position close to $\mathbf{x}$, and thenfind visible points $P_{visible}$ from such a position by the visibility test and apply the MLS projection operator using $P_{visible}$ in place of P_n.

Fig. 8: (a) Input cube-shaped point set. (b) Cached view positions for point set.

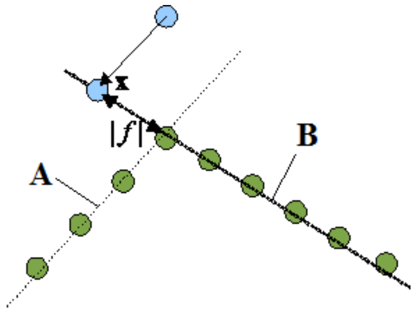


Fig. 9: Projection of point x to line B (Error value |f| for A is still large).

3.2 Sharp Feature Detection

In our projection approach, we find visible points if only the point to be projected is similar to a sharp feature shape. However, point$P$ does not provide information on whether it is on a plane composed of a sharp feature or not. For this reason, such information has to be computed and stored in advance.

To judge whether a point $\mathbf{p}_i \in P_n$ is similar to a sharp feature or not, we adopt a simple approach. A polynomial surface $z = f(x, y)$ is fitted to neighbor points around $\mathbf{p}_i$on its local coordinate system and the coefficients of a fitted surface are investigated:

$$z = f(x, y) = Ax^2 + By^2 + Cxy + Dx + Ey + F.$$

In this formula, coefficients of quadric terms A and B represent how a surface is curved. We judge that a point $\mathbf{p}_i \in P_n$ is similar to a sharp feature if a surface is considerably curved, namely, if $|A|$ or $|B|$ is larger than a threshold t_s. Such judgment information is then stored at each point as a binary “sharp feature” flag. Note that such a binary flag is not always accurate: We use this flag only for preserving sharp features. The number of neighbor points in P_n is sufficiently large enough to check the regions of sharp features.

3.3 Computing Visibility of Point Set

This subsection describes a method for selecting points for preserving sharp features by using visibility test. Using visibility test for selecting points related to sharp features comes from a simple idea; “It is likely that a point visible from a certain view position belongs to a point set which composes the plane of sharp feature” as shown [6].

Next, we explain the algorithm called HPR (Hidden Point Removal) operator to compute visibility from points proposed by Katz et al. [7]. HPR operator is a method for judging whether a point is visible or not by using the transformation for spherical flipping and the construction of convex hull. Mehra et al. also extended the HPR operator for noisy point clouds [12].

Point $\mathbf{p}_i$ is first transformed using the following formula,

$$\mathbf{p}_i^{\mathbf{x}} \leftarrow \mathbf{p}_i^{\mathbf{x}} + 2(R - |\mathbf{p}_i^{\mathbf{x}}|)\frac{\mathbf{p}_i^{\mathbf{x}}}{|\mathbf{p}_i^{\mathbf{x}}|},$$

where $\mathbf{p}_i^{\mathbf{x}} = \mathbf{p}_i - \mathbf{x}$, $\mathbf{x}$ denotes a view position and R denotes a maximum length of $|\mathbf{p}_i^{\mathbf{x}}|$. A convex hull of $\mathbf{p}_i^{\mathbf{x}}$ is then computed and visible points $P_{visible}$ composed of a circular arc in such a convex hull are selected.

In the following discussion, we use a *visible rate* $v_{visible} = N(P_{visible})/N(P_n)$ for evaluating how extent points are selected from P_n, where $N(P)$ denotes the number of points in P.

For noisy point clouds, we use a simplified algorithm of [12]. The algorithm also selects points in the range of $r_{visible}$ for each selected point by HPR operator, where $r_{visible}$ is a radius of point for computing visibility. But, the simplified algorithm cannot be used for noise levels greater than $r_{visible}$. Because, the high level noise points are not located in range of $r_{visible}$ from points selected using HPR operator.

However, there are two problems to be resolved when we apply the HPR operator to the selection of points in our algorithm. One issue is related to the visibility test from a view position much closer to a point set. The other issue is related to the visible rate $v_{visible}$ at a sharp feature point which is greater than a threshold $t_{visible}$. Details are described in the following subsections.

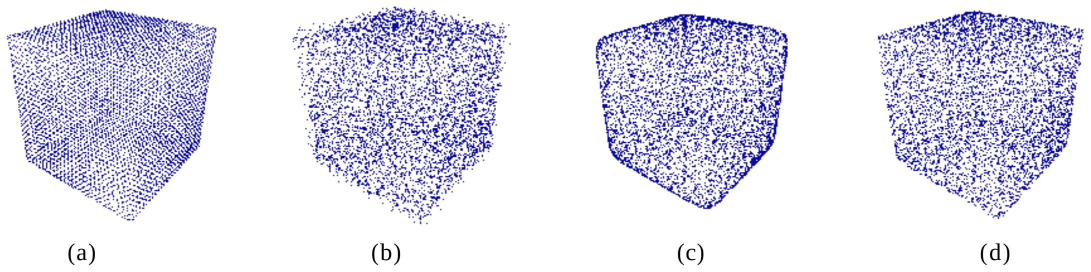


Fig. 10: Results for a “cube” point set.

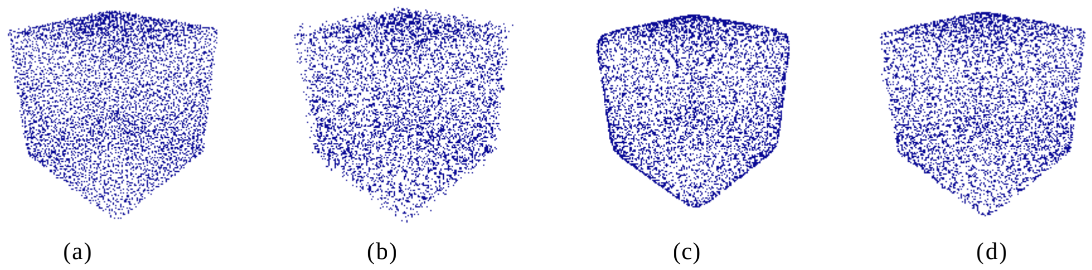


Fig. 11: Results for a “noise cube” point set.

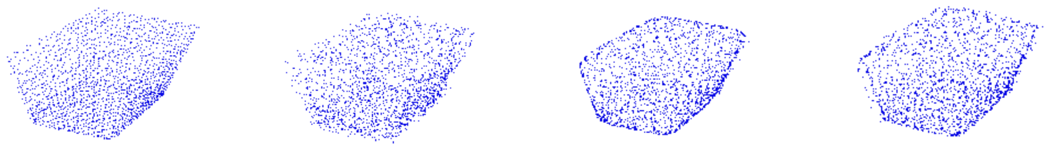

(a) (b) (c) (d)

Fig. 12: Results for “sharp cube” point set.

3.3.1 *Issue of Visibility Close to Neighbor Points*

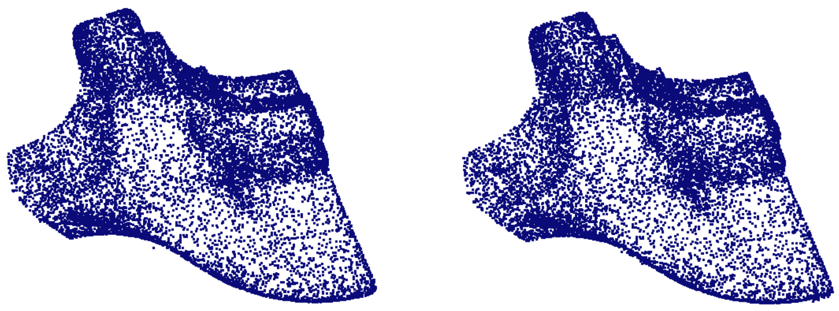
In the visibility test based on Katz et al.’s method, correct judgment cannot be made when a view position $\mathbf{x}$ is considerably close to neighbor points P_n (e.g. a view position is inside the sphere of a point in P_n whose radius is $r_{visible}$). Fig. 5 shows this situation. In Fig. 5, only red points are selected when a view position $\mathbf{x}$ is considerably close to P_n. This problem comes from the algorithm of the HPR operator.

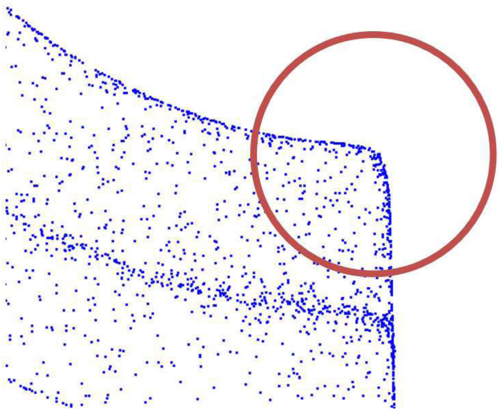
To address this issue, the algorithm is slightly modified. If a view position $\mathbf{x}$ is in the range of radius r_{visible} of $\mathbf{p}_i \in P_n$, we select all points of P_n. However, this indicates that the point selection failed.

3.3.2 *Issue of Selecting Visible Points using Visibility around Sharp Features*

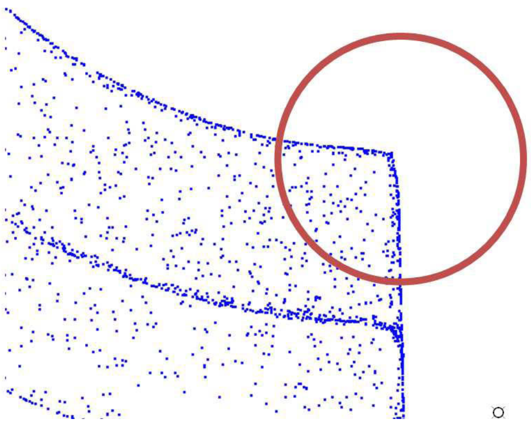
The other issue concerns the value of a visible rate $v_{visible}$. In some cases, a visible rate $v_{visible}$ can be larger than the threshold $t_{visible}$, even when there is at least one “sharp feature” flag in P_n. This implies that several groups of points composed of more than two planes are selected. Fig. 6 shows this situation. This problem occurs depending on where the view position is located, e.g., a view position is on a gray-colored region as shown in Fig. 6.

To resolve this problem, we attempted to move a view position to where only a group of points of a plane can be selected. Details are discussed in the next section.

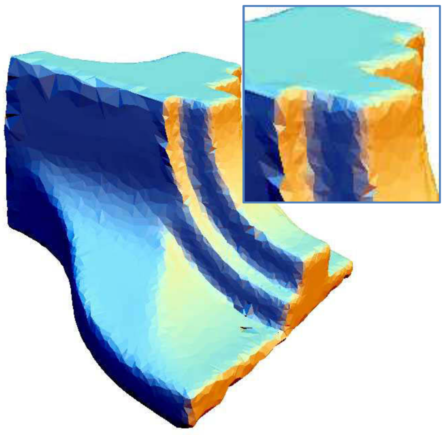

(a) (b)



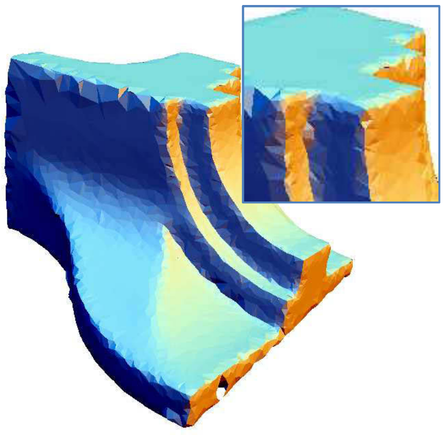
(c)



(d)



(e)



(f)

Fig. 13: Results for "fandisk" point set. (a) and (c): results by MLS projection operator, (b) and (d): results by our projection operator. (e) and (f): surface reconstruction results by [4]. (e): from points by MLS projection operator (f): from points by proposed by projection operator.

4 SELECTION OF VISIBLE POINTS FOR PROJECTION

In this section, we describe a method for selecting visible points if P_n has a "sharp feature" flag. To preserve sharp features by the projection operator, not only must a "sharp feature" flag be included in P_n but also the visible rate must be smaller than $t_{visible}$.

Our idea here is to compute view positions in advance to satisfy the above conditions. We call such view positions *Cached View Positions* (CVPs). It is guaranteed that all CVPs satisfy the conditions of visible positions and visible points can be computed. In Fig. 7, gray-colored points indicate such CVPs. CVPs are computed in the pre-process as follows: We first sample points randomly as view positions around a point $\mathbf{p}_i \in P$. For each sampled view position, we apply a visibility test and leave this point as a CVP if the visible rate $v_{visible}$ is smaller than a threshold $t_{visible}$. Otherwise, such a view position is removed. Fig. 8 shows CVPs for a cube-shaped point set.

Before projecting a point $\mathbf{x}$, we find a visible position close to $\mathbf{x}$ from CVPs as shown in Fig. 7. From a view position, we apply the HPR operator to select visible points $P_{visible}$. Using these visible points as neighbor points, a projection operator is applied for $\mathbf{x}$.

Even in the situation in which a point $\mathbf{x}$ is considerably close to a point set P as described in Section 3.3.1, the same approach can be applied. If the distance between a point $\mathbf{x}$ and the nearest point in P is smaller than $r_{visible}$, we find a view position from CVPs. The selected view position satisfies $v_{visible} < t_{visible}$. Also, in the case described in Section 3.3.2, the same approach can be applied to search visible points by finding a view position close to $\mathbf{x}$ from CVPs.

To search CVPs efficiently, k-D tree data structure can be used because CVPs are stored in the data structure only once and are re-used during the whole process. First, a view position close to x is found, and then visible points $P_{visible}$ are computed from such a view position.

5 PROJECTION OPERATOR WITH VISIBLE POINTS

Applying the projection operator described in Section 2 directly by selecting visible points $P_{visible}$ still has a problem. Fig. 9 shows such an issue in 2D case. In Fig. 9, input points compose a sharp feature point (edge in 3D case) and two lines A and B (planes in 3D case). A point $\mathbf{x}$ is projected to line B, and an error $|f|$ is close to 0. The algorithm is then terminated, however the final position is still far from line A. In this case, we need to project a point to where it is close to both lines A and B.

To resolve this issue, we prepare a stack e^f to store error values $|f|$. The algorithm terminates if all of stored error values are smaller than a threshold e.

A modified projection operator is described as follows:

1. Set i to 0.
2. Find a view position $\mathbf{vp}$ close to $\mathbf{X}$ from CVPs.
3. Select visible points $P_{visible}$ at $\mathbf{vp}$.
4. Compute $\mathbf{a}(\mathbf{x})$ and $\mathbf{n}(\mathbf{x})$ with $P_{visible}$.
5. Compute $f = f(\mathbf{x}) = \mathbf{n}(\mathbf{x}) \cdot (\mathbf{a}(\mathbf{x}) - \mathbf{x})$.
6. Compute position $\mathbf{x} + \mathbf{n}(\mathbf{x}) \cdot f$ and update the position to $\mathbf{x}$.
7. Repeat 2, 3, 4 and 5.
8. If $i > N_e$, remove the first value from e^f.
9. Add $|f|$ to e^f.
10. If max $(e^f) < e$, then the algorithm is terminated.
11. $i = i + 1$ and back to 2.

This modified algorithm is still based on MLS projection operator. The differences are (1) $P_{visible}$ is used for computing $\mathbf{a}(\mathbf{x})$ and $\mathbf{n}(\mathbf{x})$, and (2) is used to control error values. With our algorithm, we set N_e to ten for all our experiments. When we find a view position from the CVPs, we choose it randomly from several candidates close to $\mathbf{x}$; this avoids selecting the same view position from different positions of $\mathbf{x}$.

6 RESULTS

We implemented our algorithm using C# and C++. We evaluated our algorithm using Core 2 Quad 3GHz and 4.0 GB memory computer. We also use "Qhull" [15] for computing convex hulls for HPR operator algorithm. In all our experiments, we sample points randomly from a point set P and use them for points to be projected.

Name	Number of Points	h	t_s	t_{visible}	Visibility (msec)	Sharp Feature (msec)	Projection (msec)	MLS Time (msec)
cube	7,959	3.0	0.001	0.78	157,534	3,552	166,937	4,891
noise cube	7,959	3.0	0.05	0.68	109,770	3,522	162,595	4,994
sharp cube	2,653	3.0	0.001	0.85	47,948	842	48,142	1,035
fandisk	108,546	0.1	0.5	0.92	1,372,017	127,883	808,918	11,636

Tab. 1: Parameters and computation time of our projection operator and MLS.

Fig. 11 shows the results of projecting points to a “cube” point set (the diagonal of its bounding box is 51.9) and Fig. 11 shows the results for a “noise cube” point set (the diagonal of its bounding box is 52.9) In both Fig. 10 and 11, (a) indicates the input point set and (b) indicates randomly sampled points for projection. The results of the MLS projection operator are shown in (c), and (d) demonstrates the result of our visibility test based the projection operator. It can be seen from Fig. 10 that (d) represents sharp features well compared to (c). In Fig. 11(d), several points failed to project to sharp features. Fig. 12 shows the results for a “sharp cube” point set (the diagonal of its bounding box is 55.6). The results of Fig. 11 indicate that sharp features are well represented by our algorithm.

Fig. 13 shows the results of projecting points to a “fandisk” point set (the diagonal of its bounding box is 7.6). In Fig. 13, (a) and (c) show the results of projection by MLS projection operator, (b) and (d) show the results by our projection operator. In this figure, we project 1/8 of points P. (e) and (f) show the reconstruction results by using [4]. There are two types of reconstruction approaches; One compute implicit functions from point set [13],[14]. The other is to directly create triangles from point set [4]. Since our method assumes that an input point set does not have normal vectors, a reconstruction method which does not use normal vectors has to be used. From (e) and (f) we can see that the reconstruction result from points created by our projection operator also represents sharp features well compared to the results by MLS projection operator.

Tab. 1 shows parameters and computational time of our projection operator. Our approach needs considerable time for sharp feature detection (“Sharp Feature” in Tab. 1) and for computation for CVPs (“Visibility” in Tab. 1) as pre-processes. Also, our projection operator takes considerable time compared to the original MLS projection operator. In future works, we hope to speed up our algorithm.

7 CONCLUSION AND FUTURE WORK

In this paper, we proposed a MLS-based projection operator which can represent sharp features using visibility. This approach allows computation without the need to input normal vectors. Experiment results showed that the computation time of this method is slow and the representation of sharp features for noisy point sets needs more improvement.

We think it is possible to speed up our approach by using multi-core processors, because our projection operator can be processed independently for each point. The easiest way to improve our projection operator for noisy point sets is to replace the simple algorithm used with [12].

REFERENCES

[1] Alexa, M.; Behr, J.; Cohen-Or, D; Fleishman, S; Levin, D; Silva, C. T.: Point Set Surfaces, Proceedings of the conference on Visualization '01, IEEE Computer Society, 2001.

[2] Alexa, M.; Adamson, A.: On Normals and Projection Operators for Surfaces Defined by Point Sets. Symposium on Point-Based Graphics 2004.

[3] Fleishman, S.; Cohen-Or, D.; Silva C. T.: Robust Moving Least-Squares Fitting with Sharp Features, ACM Transaction on Graphics, 24(3), 2005, 544-552.

[4] Ohtake, Y.; Belyaev, A.; Seidel, H.-P.: An integrating approach to meshing scattered point data, Proceeding of the ACM symposium of Solid and physical modeling, SPM 05, 61-69, 2005.

[5] Guennebaud, G.; Gross, M.: Algebraic Point Set Surfaces, ACM Transactions on Graphics, 26(3), Article 23, 2007.

[6] Lipman, Y.; Cohen-Or, D.; Levin, D.; Tal-Ezer, H.: Parameterization-free Projection for Geometry Reconstruction, ACM Transaction on Graphics, 26(3), Article 22, 2007.

[7] Katz, S.; Tal, A.; Basri, R.: Direct Visibility of Point Sets, ACM Transaction on Graphics, 26(3), Article 24, 2007.

[8] Oztireli, A. C.; Guennebaud, G.; Gross, M.: Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression, Proceedings of Eurographics, Computer Graphics Forum, 28(2), 2009, 493-501.

[9] Huang, H.; Li, D.; Zhang, H.; Ascher, U.; Cohen-Or, D.: Consolidation of unorganized point clouds for surface reconstruction, ACM Transaction on Graphics, 28(5), 2009, 176:1-176:7.

[10] Avron, H.; Sharf, A.; Greif, C.; Cohen-Or, D:l_1-Sparse Reconstruction of Sharp Point Set Surfaces, ACM Transaction on Graphics, 29(5), Article 135, 2010, 135:1-135:12.

[11] Lipman, Y.; Cohen-Or, D.; Levin, D.: Data-Dependent MLS for Faithful Surface Approximation, Proceedings of the fifth Eurographics symposium on Geometry processing, Eurographics Association, 2007, 59-67.

[12] Mehra, R.; Tripathi, P.; Sheffer, A.; Mitra, N. J.: Visibility of noisy point cloud data, Computer & Graphics, 34(3), 2010, 219-230.

[13] Ohtake, Y.; Belyaev, A.; Alexa, M; Turk, G.; Seidel, H.-P. : Multi-level partition of unity implicits, ACM Transaction on Graphics, 22(3), 2003, 463-470.

[14] Kazhdan, M.; Bolitho, M.; Hoppe, H.: Poisson Surface Reconstruction, Proceedings of the fourth Eurographics symposium on Geometry processing, SGP '06, 2006, 61-70.

[15] Qhull: http://www.qhull.org