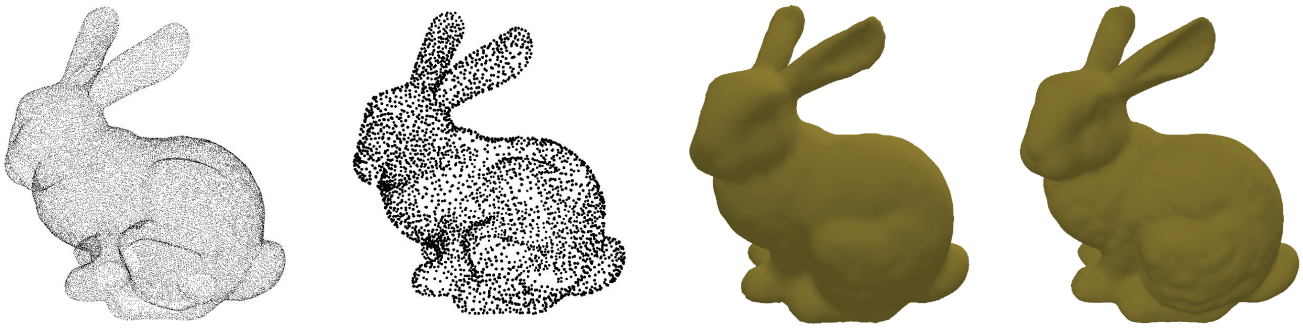
Moving Least Squares Multiresolution Surface Approximation

BORIS MEDEROS LUIZ VELHO LUIZ HENRIQUE DE FIGUEIREDO

IMPA–Instituto de Matemática Pura e Aplicada
Estrada Dona Castorina 110, 22461-320 Rio de Janeiro, RJ, Brazil
boris@impa.br lvelho@impa.br lhf@impa.br

Abstract. We describe a new method for surface reconstruction based on unorganized point clouds without normals. We also present a new algorithm for refining the inital triangulation. The output of the method is a refined triangular mesh with points on the moving least squares surface of the original point cloud.

Keywords: multi-resolution; surface reconstruction; k-nearest neighbors; moving least squares.



point cloud reduction triangulation refinement

1 Introduction

We consider the problem of surface reconstruction and refinement from scattered data points without normals. Several algorithms are known for this important problem [4–13], including a number of recent algorithms with theoretical guarantees [4–7]. Those algorithms use a 3D Delaunay triangulation of the original point cloud to compute a triangular surface mesh. Computing the Delaunay triangulation can be slow and susceptible to numerical errors.

Gopi, Krishnan, and Silva [12] proposed an algorithm based on Differential Geometry that projects the neighborhood of each sample point on a tangent plane, computes the 2D Delaunay triangulation of this projected neighborhood, and lifts it to 3D.

Hoppe et al. [10] estimate a tangent plane at each sample point using its k-nearest neighbors and use the distance to the plane of the nearest sample as an estimative of the signed distance function to the surface. The zero set of this distance function is extracted using the marching cubes algorithm.

Other algorithms [8, 9, 13] use a greedy approach. The ball pivoting algorithm [9] rolls a ball over the sample points; it requires the normal to the surface at each sample point, but it can create artifacts. Boissonnat [8] starts by finding an initial edge pq in the triangulated reconstruction; he then computes a tangent plane around the edge, projects the k-nearest neighbors of both vertices onto this plane, and determines the point r that maximizes the angle $\angle\, \bar{p}\, \bar{r}\, \bar{q}$ (where the bar represents projected points on the tangent plane). This point r determines a surface triangle prq. The process is repeated for each border edge, resulting in a triangulated surface. Boyer [13] proposes an incremental algorithm over the 3D Delaunay triangulation of the samples, and relies on regular interpolants.

Our algorithm uses a different approach: it computes a set of representative points near of the original point cloud and triangulates these representative points using a new incremental algorithm based on k-nearest neighbors. This is somewhat similar to what is done in other greedy algorithms [8, 9, 13]: For each border edge, the algorithm uses angle criteria to select a point to make a triangle with the edge. The initial triangulation is refined using a new method based on moving least square operators [1–3]. Our algorithm does not need Delaunay triangulations and it can handle surfaces with borders.

Section 2 describes in detail the four steps of our method: clustering, reduction, triangulation, and refinement. These steps are illustrated above. Section 3 discusses several examples of the method in action.

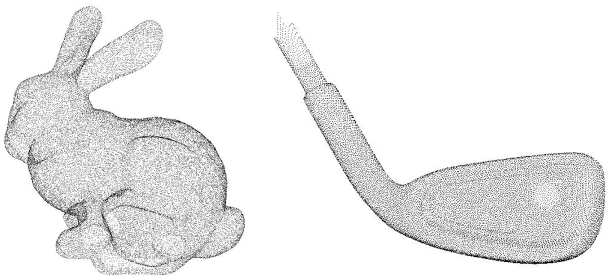


Figure 1: Point clouds

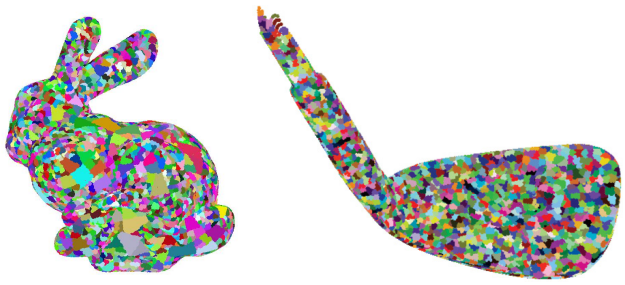


Figure 2: Clusters

2 Our Method

Starting from a point cloud $\mathcal{Q} \subseteq \mathbf{R}^3$ that samples an unknown C^1 surface $\mathcal{S}$, our method produces a mesh with points near the moving least-squares surface of $\mathcal{Q}$ [1–3]. The method has four steps:

1. *Clustering*: Split the point cloud into a set of clusters.
2. *Reduction*: Compute a representative point of each cluster on the moving least squares surface of its k-nearest neighbors.
3. *Triangulation*: Build a triangulated surface over the set of representative points.
4. *Refinement*: Refine the initial mesh into a finer mesh that is adapted to the geometry of the unknown surface $\mathcal{S}$.

We shall now describe each of these steps in detail.

2.1 Clustering

The goal of this first step is to partition the original set of points $\mathcal{Q}$ into a finite set of clusters, such that the curvature of the original surface $\mathcal{S}$ varies a little within each cluster. Since $\mathcal{S}$ is not known, its curvature must be estimated from the sample points $\mathcal{Q}$.

We use a hierarchical clustering method based on a BSP tree [3]. Each node in this tree contains a subset $\mathcal{P} = \{p_1, \ldots, p_n\}$ of the original point cloud $\mathcal{Q}$. We use the covariance matrix C of $\mathcal{P}$ to decide whether or not to subdivide the node:

$$C = \begin{bmatrix} p_1 - \bar{p} \\ p_2 - \bar{p} \\ \vdots \\ p_n - \bar{p} \end{bmatrix}^T \begin{bmatrix} p_1 - \bar{p} \\ p_2 - \bar{p} \\ \vdots \\ p_n - \bar{p} \end{bmatrix}, \quad \bar{p} = \frac{1}{n} \sum_{i=1}^{n} p_i.$$

Since C is a symmetric positive semi-definite 3×3 matrix, its three eigenvalues are real and we can order them: $\lambda_1 \leq \lambda_2 \leq \lambda_3$. These eigenvalues measure the variation of the points in $\mathcal{P}$ along the directions of their corresponding eigenvectors v_1, v_2, v_3.

The eigenvectors v_2 and v_3 define the directions of highest variation and define a regression plane Π for $\mathcal{P}$. The eigenvector v_1 is normal to Π and its eigenvalue λ_1 measures the variation of the points in $\mathcal{P}$ with respect to the plane Π. So, small values of λ_1 mean that all points in $\mathcal{P}$ are approximately on Π. Hence, the ratio

$$\sigma = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}$$

is a good measure of the flatness of the point set $\mathcal{P}$ and can be used as an estimate for the curvature of $\mathcal{S}$ around $\mathcal{P}$.

We use this flatness measure to define subdivision criteria for the BSP tree. A node is divided in two when both conditions below are satisfied:

1. The ratio σ is larger than a user-defined tolerance σ_{max};
2. The number n of points in $\mathcal{P}$ is larger than a user-defined value n_{min}.

The nodes are divided by classifying the points of $\mathcal{P}$ with respect to the regression plane Π. The points of $\mathcal{P}$ that are in the same side of Π will form a new node. The leaves of the BSP tree will be the clusters. Figure 2 shows the clusters computed for the point clouds shown in Figure 1.

2.2 Reduction

We find a representative point for each cluster using a new method based on the moving least squares theory [1–3]. The moving least square surface of a set of points R near the surface $\mathcal{S}$ is the set of fixed points of a projection operator $\Psi(R, \cdot)$ defined in [1–3]:

$$\text{MLS}(R) = \{y \in \mathbf{R}^3 : \Psi(R, y) = y\}.$$

Given a point point $r \in \mathbf{R}^3$, we approximate $\Psi(\mathcal{Q}, r)$ by $\Psi(K, r)$, where K is the set of k-nearest neighbors of r in $\mathcal{Q}$ (see [3] for details).

To determine a representative point for each cluster, we start with the centroid c of the set of points in the cluster. This point c is not necessarily near $\mathcal{S}$ and so we move

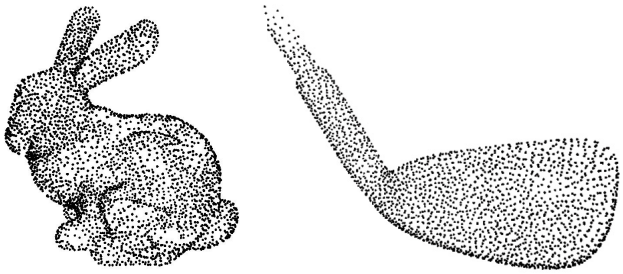


Figure 3: Representatives

it toward $\mathcal{S}$ and obtain the point $r = c + t_{min} n_c$, where n_c and t_{min} are explained below. The point r is then projected onto the moving least squares surface MLS($\mathcal{Q}$) of the initial point cloud and we obtain the point $r_{proj} = \Psi(Q, r)$, which is selected to represent the cluster. Figure 3 shows the representatives for the clusters shown in Figure 2.

The vector n_c used to define r is the eigenvector associated to the smallest eigenvalue of a weighted covariance matrix M over the set K of k-nearest neighbors of c in $\mathcal{Q}$. More precisely, M is the 3×3 matrix whose entries are

$$m_{ij} = \sum_{p \in K} (p_i - c_i)(p_j - c_j) \exp(\frac{-\|p - c\|^2}{h^2}),$$

where $p = (p_1, p_2, p_3)$, $c = (c_1, c_2, c_3)$, and h is a parameter reflecting the spacing between neighboring points in $\mathcal{Q}$ (see [1, 2] for details). We use h equal to three times the distance of c to its near neighbor in $\mathcal{Q}$.

Using the direction n_c as above, we compute

$$t_{min} = \arg\min_t \sum_{p \in K} \|p - c - t n_c\|^2.$$

We interpret this as finding the point $r = c + t_{min} n_c$ on the line through c in the direction of n_c that is closest to the original surface.

2.3 Triangulation

The next step is to build a triangulated surface over the set of representative points. We do this using an incremental algorithm that introduces new features in the basic framework of previous incremental algorithms [8, 9, 13].

The main idea behind the algorithm is to determine incrementally the restricted Delaunay triangles, because these triangles form a piecewise linear manifold homeomorphic to the original surface [4].

The algorithm computes a sequence of triangulated surfaces with border. At each step, it chooses a border edge, finds a new triangle associated to this edge, and updates the current surface. The algorithm maintains a half-edge data structure H that represents the current surface and a list L of half-edges that represents the current boundary. Here is a summary of the main steps:

Build a 3d-tree on the set of representatives.
Get an initial triangle and initialize H and L with it.
repeat
  Remove a half edge uv from L.
  $usv \leftarrow$ the triangle adjacent to uv in H.
  $P \leftarrow$ set of k-nearest neighbors of u.
  $q \leftarrow$ GET_POINT(usv, u, P)
  Insert uqv into H and its border edges into L.
until L is empty.

The initial surface contains a single triangle. This triangle is inserted into H and all its edges are inserted into L. To find the initial triangle, we select a point $p \in \mathcal{Q}$ and determine its nearest neighbor $q \in \mathcal{Q}$. These two points define the first border edge: pq. Among the k-nearest neighbors of p, we determine the point r that maximizes the angle $\angle\, p\, r\, q$. The initial triangle prq is in the Delaunay triangulation of the k-nearest neighbors of p (see [14], Lemma 3).

Once the initial surface has been found, we continue by removing edges from L until it is empty. For each edge uv, we select a point q among the k-nearest neighbors of u to create a new triangle uqv, which is added to H and L. (We use a 3d-tree built at the start of the algorithm to identify k-nearest neighbors.)

The number of edges of uqv inserted in L varies: it is zero when the triangle uqv fills a hole in H; it is one when uq or vq are consecutive edges of uv in the border of the current surface; otherwise, it is two. When q is already in H, we have to join two connected components of the border or to split one component in two new connected components. We determine the point q using the following algorithm:

GET_POINT (usv, u, P):
  $d_u \leftarrow$ distance from u to its nearest neighbor.
  $d_v \leftarrow$ distance from v to its nearest neighbor.
  $d_{min} \leftarrow \min\{d_u, d_v\}$.
  repeat
    $C_\theta \leftarrow$ set of points t such that:
    (i) the dihedral angle between the triangles usv and utv is in $[\pi - \theta, \pi + \theta]$
    (ii) Triangle utv does not violate the topology of H.
    (iii) $\dfrac{\max\{d(u,t), d(v,t)\}}{d_{min}} < \varepsilon_1$.
    if $C_\theta \neq \emptyset$ **then**
      Determine the set C'_θ of points $q \in C_\theta$ with maximum angle $\angle\, u\, q\, v$.
    else
      $\theta \leftarrow \theta + \varepsilon_2$.
  until $C'_\theta \neq \emptyset$ or $\theta > \frac{\pi}{2} + \varepsilon_2$

Here, θ, ε_1, and ε_2 are user-defined tolerances.

Condition (i) in this algorithm is based on the theorem below, which we prove in [14]:

Theorem 1 *Let F be a β, r-*sampling *of a surface $\mathcal{S}$ with $r < 1/4$. Then the angle between two adjacent restricted Delaunay triangles sharing an edge is at least $\pi - 2(\frac{2r}{1-4r} + \arcsin(\frac{\sqrt{3}r}{1-r}))$.*

This theorem show that the dihedral angle between two adjacent restricted Delaunay triangles converges to π as the sampling density increases (that is, as $r \to 0$). We start with an initial region $[\pi - \theta, \pi + \theta]$ with small θ to determine the set of point C_θ in this region that satisfy criteria (ii) and (iii). If C_θ is not empty, we find the point $q \in C_\theta$ with maximum angle $\angle\, u\, q\, v$; otherwise, we increase θ by ε_2 and repeat the process until we find a point q or θ gets too large.

Condition (ii) is used to eliminate the points t that are interior to H and the points on the boundary of H that would violate the topology if they were selected. When the point t is on the boundary of H, we first fit a plane Π to the set of points adjacent to t in H (the *star* of t) and project this start onto Π, resulting in a set of 2d triangles T. If these triangles do not form a triangulation, that is, is two edges intersect, then we have a topology violation.

Condition (iii) is used to determine border edges, and is based on the following theorem, which we prove in [14]:

Theorem 2 *Let F be a β, r-*sampling *of a surface $\mathcal{S}$ with $r < 1/3$. Let $T_1 = uvt$ and $T_2 = uvq$ be restricted Delaunay triangles sharing the edge uv. Then, the length of the longest edge of T_2 is at most $\frac{2\beta}{1-3r}d$, where $d = \min\{d_u, d_v\}$ and d_u (respectively, d_v) is the distance of u (respectively, v) to its nearest neighbor.*

This theorem shows that there is little variation between the length of the edges of two triangles adjacent to a interior edge. We observed that the length of edges of the adjacent triangles to a boundary edge are very different.

In practice, we have no way to estimate β and r, and so we use a predefined value ε_1 as the constant $\frac{2\beta}{1-3r}$ and a point t is eliminated if $\frac{\max\{d(u,t),d(v,t)\}}{d_{min}} > \varepsilon_1$. If C_θ is empty, the edge uv is a candidate border edge. If C_θ is empty for all θ tested, uv is a border edge.

If C_θ is not empty, we compute the subset C'_θ of points q with maximum angle $\angle\, u\, q\, v$. We justify this criterion as follows: In $\mathbf{R}^2$, given a Delaunay triangle uvs, its adjacent Delaunay triangle uvq is the one with maximum angle $\angle\, u\, q\, v$. Although this is a criterion for $\mathbf{R}^2$, we use it for surfaces because the surface normal varies little in the surface neighborhood of a point u containing the vertex of the restricted Delaunay triangulation (see [4–6]); in other words, we can consider this neighborhood as flat.

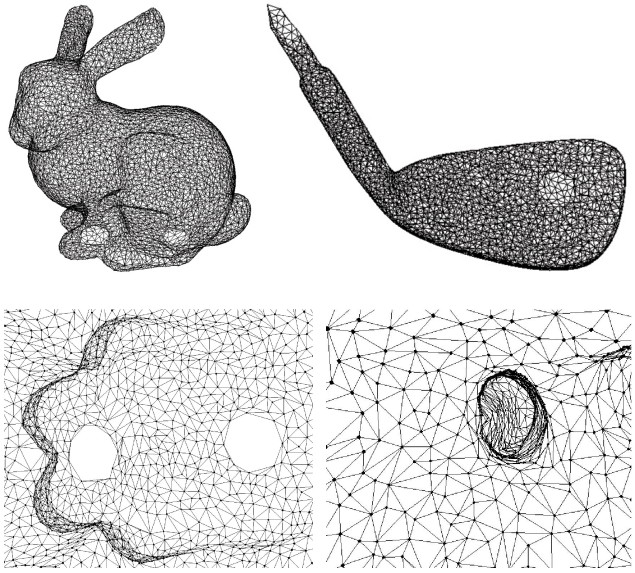


Figure 4: Triangulations

Figure 4 shows the triangulated surfaces over the set of representatives shown in Figure 3.

2.4 Refinement

The goal of this step is to refine the initial coarse triangulation into a finer triangulation adapted to the geometry of the unknown surface $\mathcal{S}$ from which the point cloud $\mathcal{Q}$ was sampled.

We propose the following method to refine an edge uv: We compute the middle point of the edge. If this point is too far from the original point cloud $\mathcal{Q}$, it is projected onto the moving least squared surface and a new edge is added to each adjacent triangle of the edge uv, dividing each triangle into two new triangles. We repeat this process for each edge in the original triangulation. The details are discussed below.

More precisely, for each edge uv we compute its midpoint $m = \frac{u+v}{2}$ and its normal

$$n_m = \frac{\frac{n_u+n_v}{2}}{\|\frac{n_u+n_v}{2}\|},$$

where n_u and n_v are the normals of u and v respectively, computed using the local triangulation (star) of each vertex.

Using the set K of k-nearest neighbors of m in the original points cloud, we minimize the following functional with respect to t :

$$\alpha = \min_t \sum_{p \in K} \|p - m - t \cdot n_m\|^2 = -\frac{1}{k} \sum_{p \in K} (p - m) \cdot n_m$$

Figure 5: Refinement

As in the reduction step, this is interpreted as finding the point on the line through m in the direction of n_m that is closest to the original surface $\mathcal{S}$.

We use α as a measure of the need of refinement. If α is smaller than some predefined value ε, we do not refine the edge; otherwise, we project the point $m + \alpha \cdot n_m$ onto the moving least squares surface, producing the new point $\bar{m} = \Psi(\mathcal{Q}, m + \alpha \cdot n_m)$ and replace the triangles auv and buv adjacent to uv with the new triangles $u\bar{m}a$, $v\bar{m}a$ and $u\bar{m}b$, $v\bar{m}b$, respectively. If the ratio between the length of the minimum edge and the maximum edge of the triangles auv or buv is too small, we do not divide the triangles.

This procedure is applied to all the old edges while the scalar t is larger than ε. In the resulting mesh, we flip the diagonals of each quadrilateral formed by the pairs of adjacent triangles if the length of one diagonal (the common edge) is larger than the length of the other diagonal (the segment joining the opposite vertex).

The final result is a triangulated surface whose vertices are on the moving least squares approximation surface of the original points cloud. Figure 5 shows two refinement of the initial triangulations shown in Figure 4.

3 Results

Our algorithm has been implemented using CGAL [15] for computing the half edge data structure and the operations in the refinement. The algorithm has been tested on several data sets available on the internet [16, 17] using two refinement steps. The results are shown in Figures 6–9 (and also in the previous figures). Table 1 shows the times (in seconds) taken in each step of the algorithm and the sizes (points and triangles) of the result models.

Model	Clustering	Triangulation	Ref 1	Ref 2
Bunny	3.68 s	22.48 s	2.33 s	7.9 s
35933 p	4457 p	8895 t	31623 t	107363 t
Club	1.36 s	5.3 s	1.11 s	3.86 s
16864 p	2103 p	4121 t	15037 t	51775 t
Cactus	0.21 s	1.58 s	0.2 s	0.68 s
3337 p	404 p	782 t	2776 t	9614 t
Cat	0.77 s	6.02 s	0.61 s	2.06 s
10000 p	1244 p	2453 t	29257 t	8535 t
Dragon	62.43 s	297.8 s	27.22 s	58.98 s
437645 p	55694 p	109889 t	361455 t	810367 t
Horse	4.67 s	30.83 s	3.17 s	10.76 s
48482 p	6164 p	12046 t	42860 t	144828 t

Table 1: Results on a 1.8Ghz Pentium 4 with 512Mb of RAM running Linux (s=seconds, p=points, t=triangles)

The pictures in this paper are available in full size and color at `http://www.impa.br/~boris/sib2003.html`.

4 Conclusion

We presented a new algorithm for reconstruction and refinement of surface, giving as output a refined triangular mesh with points in the moving least squares surface of the original points cloud.

The new method proposed to computing representatives points produces a simplified sample on the moving least squares surface. The new triangulation algorithm does not need to computed 3D Delaunay triangulations, which in the worst case requires quadratic time and is prone to numerical errors. The new refinement method is fast and gives a fine triangular mesh adapted to the geometry of the unknown surface.

Acknowledgments. The authors are partially supported by CNPq research grants. The authors are members of Visgraf, the computer graphics laboratory at IMPA, which is sponsored by CNPq, FAPERJ, FINEP, and IBM Brasil.

References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, C. Silva. Point set surfaces. *IEEE Visualization 2001*, pp. 21–28.

[2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, C. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics* 9(1):3–15, 2003.

[3] M. Pauly, M. Gross, L. Kobbelt. Efficient simplification of point-sampled surfaces. *IEEE Vizualization 2002*, pp. 163–170.

[4] N. Amenta, M. Bern, M. Kanvisellis. A new Voronoi based surface algorithm. *SIGGRAPH'98*, pp. 415–421.

[5] N. Amenta, S. Choi. One-pass Delaunay filtering for homeomorphic 3D surface reconstruction. *Technical Report TR99-08*, University of Texas at Austin, 1999.

[6] N. Amenta, S. Choi, R. Kolluri. The power crust. *6th ACM Symposium on Solid Modeling*, pp. 249–260, 2001.

[7] J. D. Boissonnat, F. Cazals. Smooth surface reconstruction via natural neighbor interpolation of distance function. *Computational Geometry* 22(1–3):185–203, 2002.

[8] J. D. Boissonnat. Geometric structures for three-dimensional shape reconstruction. *ACM Transactions on Graphics* 3(4):266–289, 1984.

[9] F. Bernardini, J. Mittelman, H. Rushmeier, C. Silva, G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics* 5(4):349–359, 1999.

[10] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH'92*. pp. 71–78.

[11] D. Freedman. Efficient simplicial reconstruction of manifolds from their samples. *IEEE Transaction on Pattern Analysis and Machine Intelligence* 24(10):1349–1357, 2002.

[12] M. Gopi, S. Krishnan, C. Silva. Surface reconstruction based on lower dimensional localized Delaunay triangulation. *Computer Graphics Forum* 19(3):C467–C478, 2000.

[13] E. Boyer, S. Petitjean. Curve and surface reconstruction from regular and non regular point sets. *Computational Geometry* 19(2–3):101–126, 2001.

[14] B. Mederos, L. Velho, L. H. Figueiredo. A Simple Algorithm for Multiresolution Surface Reconstruction. *Technical Report TR03-03*, Instituto de Matemática Pura e Aplicada, 2003. `http://www.visgraf.impa.br/RefBib/Data/PS_PDF/tr0303/`

[15] `www.cgal.org`

[16] `www.cc.gatech.edu/projects/large_models`

[17] `www.research.microsoft.com/hoppe`

Figure 6: Dragon

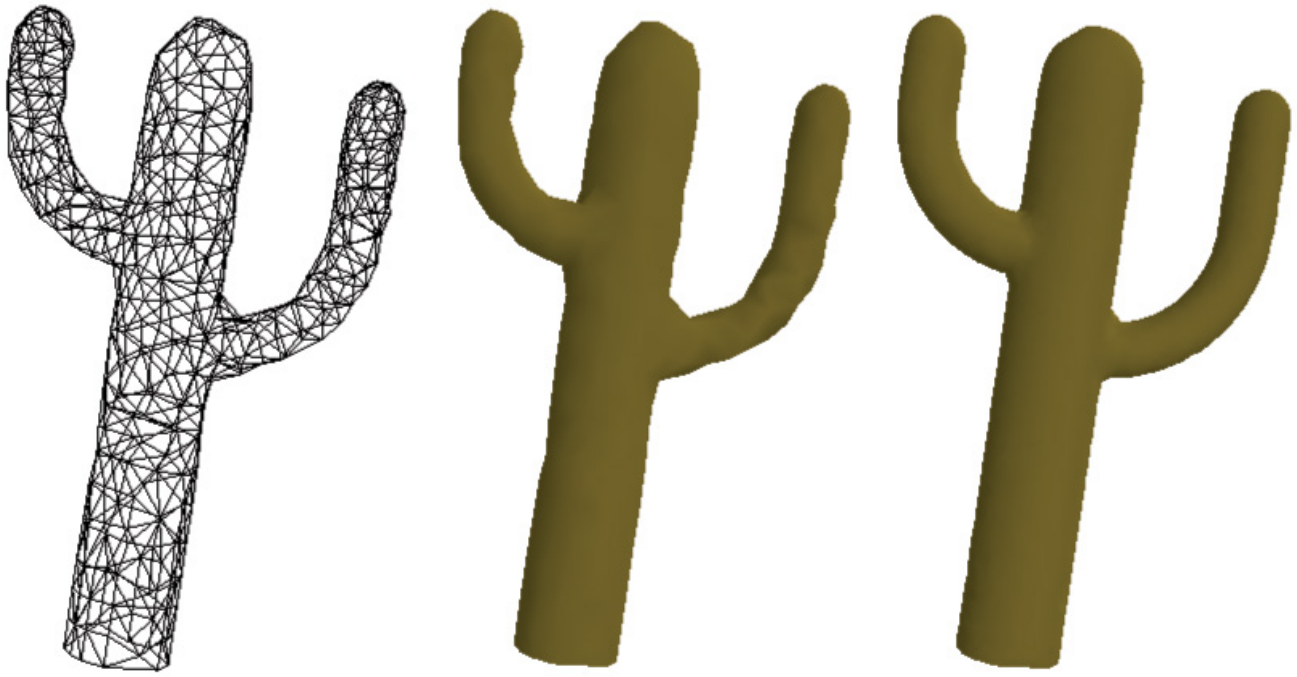


Figure 7: Cactus

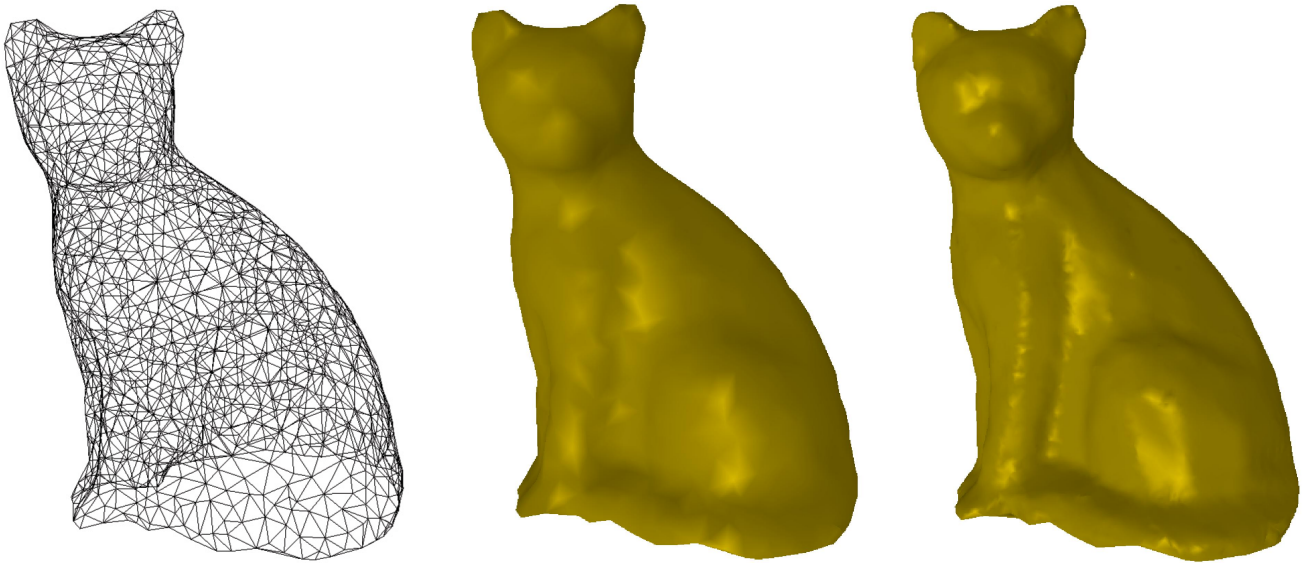


Figure 8: Cat

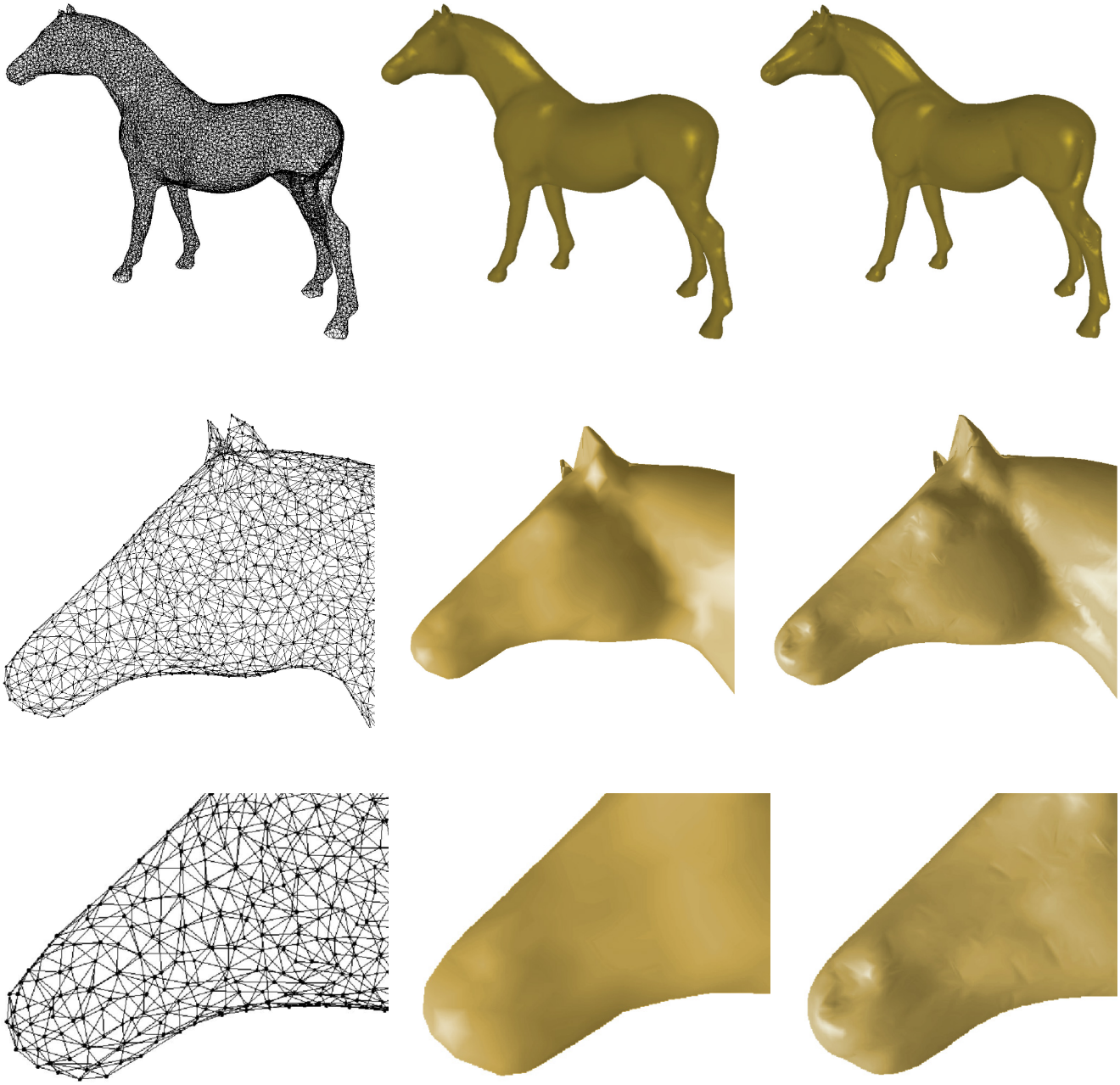


Figure 9: Horse