Visual Comput (2008) 24: 361–371
DOI 10.1007/s00371-007-0194-8

ORIGINAL ARTICLE

Jianning Wang
Manuel M. Oliveira
Haitao Zhang
Arie E. Kaufman

Reconstructing regular meshes from points

A parameterization-based approach

Published online: 1 February 2008
© Springer-Verlag 2008

J. Wang (✉) · H. Zhang · A.E. Kaufman
CVC, Computer Science,
Stony Brook University, USA
wangjianning@gmail.com

M.M. Oliveira
Instituto de Informática, UFRGS, Brazil

Abstract We propose an algorithm for reconstructing regular meshes from unorganized point clouds. At first, a nearly isometric point parameterization is computed using only the location of the points. A mesh, composed of nearly equilateral triangles, is later created using a regular sampling pattern. This approach produces meshes with high visual quality and suitable for use with applications such as finite element analysis, which tend to impose strong constraints on the regularity of the input mesh. Geometric properties, such as local connectivity and surface features, are identified directly from the points and are stored independent of the resulting mesh. This decoupling preserves most details and allows more flexibility for meshing. The resulting parameterization supports several direct applications, such as texturing and bump mapping. In addition, novel boundary identification and cut parameterization algorithms are proposed to overcome the difficulties caused by cuts, non-closed surfaces and possible self-overlapping parameter patches. We demonstrate the effectiveness of our approach by reconstructing regular meshes from real datasets, such as a human colon obtained from CT scan and objects digitized using laser scanners.

Keywords Surface reconstruction · Regular mesh · Point parameterization · Cut handling

1 Introduction

Given a set of points $\mathcal{P} = \{p_i = (x_i, y_i, z_i)\}$ sampled from an object's surface, the goal of surface reconstruction techniques [13, 19] is to compute a mesh $\mathcal{M}$ that approximates the underlying surface. This is essentially an ill-posed problem as it admits multiple solutions. Usually, the solution with minimal energy in some sense will be chosen as the optimal result. Thus, for instance, one may want to minimize the distance between the original points and the reconstructed surface. Often, however, it will be important to have well-shaped (e.g., nearly equilateral) triangles for best visual quality and for some subsequent use by other procedures, such as finite element analysis [5]. Triangles with nice aspect ratios are also important for animation and rendering (shading) [6].

While one would like to preserve the detailed information available in the original data as much as possible, surface reconstruction from points often leads to some loss. In this paper, we propose a new approach for reconstructing regular meshes from point clouds. Here the point connectivity is inferred using a neighborhood relation and detailed geometric features are preserved with the use of a new point parameterization. Given the set of input points, meshes with arbitrary resolutions can be extracted by sampling the patches using a regular grid.

The proposed approach consists of two steps as shown in Fig. 1. First, a parameterization is computed for the point cloud. Then, a mesh is extracted based on the resulting parameterization. The parameterization process can be summarized as follows: given a 3D point cloud representing a surface with arbitrary topology, it is segmented

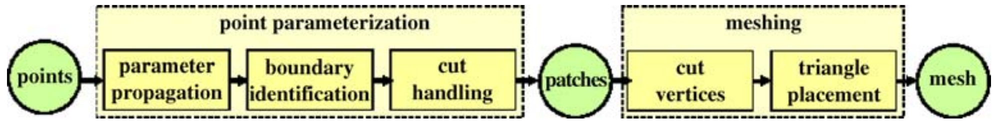

Fig. 1. Pipeline for creating regular meshes from point clouds. Given a point cloud, a point parameterization is used to create a set of 2D parameter patches, which are in turn used as input to the meshing process

into several non-overlapping patches, each topologically equivalent to a disk. The points in any patch are assigned a 2D parameterization via a propagation procedure. This results in a set of 2D *parameter patches*. Although patches do not overlap, a patch boundary may be adjacent to several other patches. During meshing, shared vertices are created along the patch boundaries, while the interior of the patches are tiled using a regular triangular pattern.

Surface remeshing techniques [1, 2, 21, 25] create regular meshes, but they take another mesh as input and, therefore, already have connectivity information. Basically, these techniques compute parameters for each original vertex, and new vertices are created by resampling [25]. In this paper, an alternative paradigm is employed: first we compute the point parameterization, and then resample the parameterization domain for meshing. This breaks the direct coupling between the original points and the resulting mesh, and has some advantages. For instance, it allows the preservation of details on the point side, and for more flexibility during meshing. Thus, meshes with arbitrary resolutions can be easily created. It is also a *direct* reconstruction approach without involving an intermediate mesh. The use of intermediate representations tends to cause loss of information. Our approach is similar to that of Tewari et al. [26]. Both algorithms try to parameterize the point cloud based on small neighborhoods. However, unlike the approach of Tewari et al. [26], our algorithm can handle non-closed surfaces and models with arbitrary genus number.

The proposed algorithm requires only the location of the points and, therefore, is general enough to handle all kinds of point datasets. Since a point parameterization is produced as a by-product of the reconstruction process, several operations can be directly applied to the resulting mesh, such as texture and bump mapping. As the digitization of real models using laser scanners is becoming increasingly popular, the automatic parameterization of reconstructed models provided by our approach can potentially save modelers some significant amount of work.

The main contributions of this paper include:

– A new approach for reconstructing regular meshes from point clouds that produces nearly equilateral triangles. It is based on a new paradigm that decouples parameterization from meshing, thus improving the flexibility of the meshing process, while the obtained parameterization can also be used for other applications (Sect. 3);
– A robust boundary-identification algorithm to find surface boundaries, including hole boundaries (Sect. 4.2);
– A novel approach for parameterizing cuts between parameter patches that allows points along cuts to have more than one associated parameter value (Sect. 4.3);
– A solution for creating triangles from the borders towards the center of the parameter patch, thus counteracting the effect that some patches might overlap (Sect. 5);
– A new approach to optimize the location of interior vertices so as to adjust the triangles around the patch boundary (Sect. 5.2).

2 Related work

Our approach falls into the category of representing geometrical information using images [17]. Geometry images [10] parameterize a 3D mesh to a rectangular domain based on geometric stretch. More theoretically sound results are obtained using spherical parameterization [9] and conformal mapping [15]. In comparison, we start with points without any connectivity information. In our approach, parameterized points form several patches, which may overlap and have non-rectangular shapes. The use of patches reduces the distortion and solves the nonuniform sampling problems inherent in the aforementioned approaches.

Our approach is also similar to Floater and Reimers [8] and Hormann and Reimers [14]. However, these methods force the parameterized surface to fit within a convex shape (e.g., a disk); thus, a large distortion will be introduced. Mencl and Muller [18] first create some skeleton edges from the point dataset. Then triangles are created to fill the surface under the constraint of the skeleton edges. In these algorithms, the original data points are used as vertices. As a result, no guarantees can be given to the aspect ratio of the resulting triangles. In contrast, vertices created by our approach are not necessarily located at the original data points. The approach of Sorkine et al. [25] takes advantage of the existing connectivity information for parameterization, while we have to infer the connectivity from point clouds.

Zwicker et al. [30] have proposed an interactive approach to locally compute a parameterization with minimal distortion. However, user interaction is needed to assign some feature points and this parameterization approach could not be applied to the entire surface of a manifold object. Moving least squares surface [3, 23] finds the location of the extremal surface from the point clouds. Its major advantage is to reduce noise. In [23], an adaptive direct meshing approach is employed. A guide field is used to control the triangle size according to curvature and other features. The major difference between this approach and ours is that the former can not output nearly equilateral triangles. In addition, it pre-computes the guide field while we compute the 2D parameter for each point via propagation.

3 Algorithm overview

The pipeline of the proposed algorithm is illustrated in Fig. 1. It includes two steps: point parameterization and meshing. For point parameterization, we compute several *3D patches* from the point cloud via propagation. The parameter values of the points inside the patches are computed, forming several 2D *parameter patches*. Each point p_i on a *3D patch* will then have an associated parameter $q_i = (u_i, v_i)$. Within any given 3D patch, there is a *nearly isometric mapping* to its *parameter patch*. That is, for any two points p_i and p_j in a local 3D neighborhood, their Euclidean distance nearly equals the distance computed in 2D using their corresponding parameters (i.e., $\|p_i - p_j\| \approx \|q_i - q_j\|$). Since each *3D patch* and its related *parameter patch* both refer to the same set of points, we often make no distinction between them and refer to both simply as *patches*. Whenever we need to refer to a particular kind of patch, we will use the specific terms.

A *patch* A boundary may represent the presence of non-sampled regions on the surface. Alternatively, it may represent a parameter discontinuity with respect to adjacent patches. The patch boundaries are termed *cuts* because they represent discontinuities in the parameter values.

In the stage of meshing (Fig. 1), we first add vertices along cuts and force them to be evenly distributed along the cuts. Then triangles are placed inside each patch using a regular pattern. In the algorithm, a Kd-tree is used to expedite the query of the k nearest neighborhood (*kNN*) of a point in 3D space.

4 Point parameterization

Isometric mappings only exist for developable surfaces [7]. In this paper, we instead find a *nearly isometric parameterization* for patches as large as possible. A propagation scheme is used to compute the parameter q for each point p. During the propagation, cuts may be introduced to separate different patches where a parameter discontinuity occurs. Since a cut is related to multiple patches, points along a cut might have multiple parameter values. This situation is illustrated in Fig. 3.

We parameterize a point cloud via propagation. During the process, a tag selected from the set {*unvisited, patch, gap, cut, boundary*} will be assigned to each point to indicate its current status. Initially, all points are set as *unvisited*. After propagation, points belonging to any patch become *patch* points. The remaining points are *gap* points, representing the boundaries of the *patches*. In the next step, we find among the *patch* points the exterior and interior (hole) boundaries of the surface and mark the related points as *boundary* points. Finally, we convert the *gap* and *boundary* points into several *cuts* and parameterize the cuts. Algorithm 1 presents the steps associated with the point parameterization stage of the pipeline shown in Fig. 1.

Algorithm 1. Point parameterization pipeline

```
// Point parameterization stage:
//
// (1) parameter propagation (Sect. 4.1)
//
for each point p in the point cloud do
    if p is not visited then
        CreateOnePatch(p)
    end
end
//
// (2) boundary identification (Sect. 4.2)
//
for each edge in the patch adjacency graph do
    if this is part of the boundary then
        Mark the edge points as boundary points
    end
end
//
// (3) cut handling (Sect. 4.3)
//
construct the gap point adjacency graph
find cut points by extracting pruned minimum
spanning trees from the graph
find the patch boundary curve from the boundary points
traverse the patch boundary curve to parameterize the cuts

function CreateOnePatch(point p)
begin
    push p to a stack
    while stack is not empty do
        pop the point at the top
        compute its parameters
        add adjacent unvisited points into the stack
    end
end
```

4.1 Parameter propagation

Our goal is to create a nearly equilateral triangular mesh from the point cloud. Since this problem is difficult to solve in 3D, we convert it to a simpler 2D problem. We do this by making the distance measured in a 2D

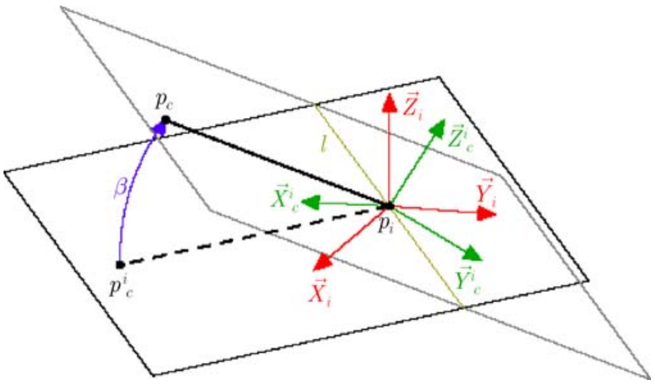

Fig. 2. Parameter propagation from a *patch* point p_i

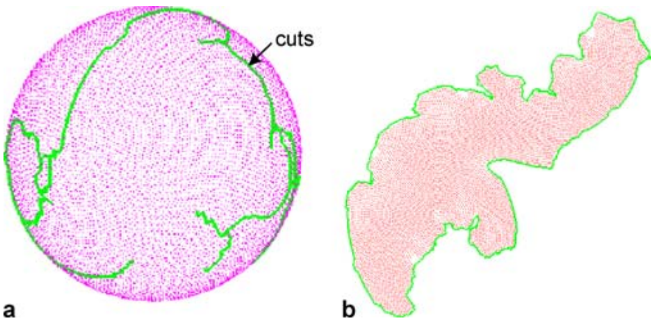

Fig. 3. **a** The cuts (*green curves*) separating patches, created from the point parameterization; **b** The 2D parameter patch is bounded by the cuts

parameter space (UV), for any two nearby points in 3D, nearly equal to their distance in 3D. The process of computing the point parameters is called parameter propagation.

We use our algorithm [29] to turn *unvisited* points into *patch* points, if applicable. Accordingly, each *patch* point p_i will be assigned a local coordinate frame $(\boldsymbol{X}_i, \boldsymbol{Y}_i, \boldsymbol{Z}_i)$ and a parameter $q_i = (u_i, v_i)$. Initially, all *unvisited* points are put into a priority queue, sorted by the number of *patch* neighbors in their *kNN*. While the queue is not empty, we pop the current top point p_c, which has the largest number of *patch* neighbors. If p_c has no *patch* neighbors in *kNN*(p_c), it is the first *patch* point in this patch. Its $\boldsymbol{Z}_c$ is computed as the eigenvector associated with the smallest eigenvalue of the covariance matrix defined in the points in the *kNN*(p_c). $\boldsymbol{X}_c$, $\boldsymbol{Y}_c$ and q_c are chosen arbitrarily ($q_c = (0, 0)$ in our experiments), given that $\boldsymbol{X}_c$, $\boldsymbol{Y}_c$ and $\boldsymbol{Z}_c$ form an orthonormal basis.

Case p_c already has some patch neighbors in *kNN*(p_c), $\boldsymbol{X}_c$, $\boldsymbol{Y}_c$, $\boldsymbol{Z}_c$ and q_c are computed from the *patch* neighbors of p_c. Let $N = \{p_i | p_i \in kNN(p_c)\}$ be the set of points in *kNN*(p_c). For each $p_i \in N$, we project p_c onto the $\boldsymbol{X}_i\boldsymbol{Y}_i$ plane as p^i_c (see Fig. 2). p^i_c is moved along the vector $\boldsymbol{p}_i\boldsymbol{p}^i_c$ such that $\|p_c - p_i\| = \|p^i_c - p_i\|$. The new local coordinate system $(\boldsymbol{X}^i_c, \boldsymbol{Y}^i_c, \boldsymbol{Z}^i_c)$ is computed for p_c by rotating $(\boldsymbol{X}_i, \boldsymbol{Y}_i, \boldsymbol{Z}_i)$ about l by β degrees, where l is a line passing through p_i and perpendicular to plane $\overline{p^i_c p_i p_c}$ (see Fig. 2). Parameters $q^i_c = (u^i_c, v^i_c)$ are computed from $q_i = (u_i, v_i)$ as $(u_i + \boldsymbol{p}_i\boldsymbol{p}^i_c \cdot \boldsymbol{X}^i_c, v_i + \boldsymbol{p}_i\boldsymbol{p}^i_c \cdot \boldsymbol{Y}^i_c)$. A weight is also computed for p_c from p_i as $w^i_c = 1/\|p_c - p_i\|$. We compute $\boldsymbol{X}_c$, $\boldsymbol{Y}_c$, $\boldsymbol{Z}_c$ and q_c as the weighted average of all $\boldsymbol{X}^i_c$, $\boldsymbol{Y}^i_c$, $\boldsymbol{Z}^i_c$ and q^i_c, respectively.

Two kinds of distortions are computed for point p_c: position distortion $D_{\text{pos}}(p_c)$ and orientation distortion $D_{\text{ori}}(p_c)$ using a variation of the estimates described in [29]. Note that the normalization of the weights w^i_c is embodied in the definition of D_{pos} and D_{ori}.

$$D_{\text{pos}}(p_c) = \max\left(\frac{w^i_c \|q^i_c - q_c\|}{\sum w^i_c}\right) \quad (1)$$

$$D_{\text{ori}}(p_c) = \max \frac{\left(w^i_c\left(2 - \boldsymbol{X}^i_c \cdot \boldsymbol{X}_c - \boldsymbol{Y}^i_c \cdot \boldsymbol{Y}_c\right)\right)}{\sum w^i_c} \quad (2)$$

If $\alpha D_{\text{pos}} + (1-\alpha)D_{\text{ori}} \geq E$, this point becomes a *gap* point, where α is the relative weight for the two errors and E is the maximum error. Otherwise, it becomes a *patch* point. We record the edges connecting p_c to all its *patch* neighbors in a *patch point adjacency graph* $\mathcal{G}_p$, which will be used later to parameterize the *cuts*. In all our experiments, we used $\alpha = 0.5$ and E is computed adaptively as the largest distance between a point and any neighbor in its *kNN*.

Once p_c becomes a *patch* point, the number of adjacent *patch* points for its adjacent *unvisited* points needs to be updated and the priority queue is adjusted accordingly. After parameter propagation, we obtain several *patches*. Note that a *parameter patch* might overlap with itself in parameter space. This might happen, for instance, if the boundary follows a curved path and overlaps itself in the 2D parameter space. Consider, for example, two *patch* points p_1 and p_2, which are far from each other in 3D, but whose 2D parameter values are very close. During propagation, the parameters of the *unknown* neighbors of p_1 are computed based only on the location of the point in 3D. Thus, the existence of a nearby point (i.e., p_2) in the 2D parameter space will not be taken into account. We use heap sorting to select the next point to be parameterized. Therefore, this is an $O(n \log n)$ algorithm, where n is the number of points.

4.2 Boundary identification

In this case the point model does not represent a closed two-manifold surface, we need an algorithm to identify the surface boundaries (exterior and interior, i.e., holes). In the literature, an angle criterion [11] is often used to check whether a point p_i belongs to the boundary. It consists of projecting the neighborhood N of p_i (including itself) onto its tangent plane (the plane fitted to N). By connecting p_i to its neighbor points on the tangent plane, one has a number of edges sharing p_i. The algorithm then finds the maximum angle between any two such adjacent edges. If such an angle is larger than a threshold, p_i is considered to be part of the boundary. Unfortunately, this approach is not robust in the case of varying sampling rates or high curvatures.

We propose a new criterion for identifying boundary edges that takes the computed local parameterization into account. Note that a boundary edge can be either a surface-boundary edge (in the case of a manifold with boundary) or a regular patch-boundary edge. Any surface-boundary edge also belongs to the edge of a given patch. The two kinds can be distinguished from one another because internal patch-boundary edges have *gap* points as some of their neighbors. Let $e = (v_0, v_1)$ be an edge in the *patch point adjacency graph* $\mathcal{G}_p$, connecting vertices v_0 and v_1. We say that e is part of the surface boundary if e is not shared by two triangles. Thus, let u_i be the 2D parameter value associated to vertex v_i and expressed in homogeneous coordintates. Thus, the line supporting e in parameter space can be represented as $u_0 \times u_1$, where $\times$ is the cross-product operator [12]. Thus, for any vertex v_j with associated parameter u_j, such a vertex is at one side of e if $(u_j \cdot (u_0 \times u_1)) > 0$ and is at the other side if $(u_j \cdot (u_0 \times u_1)) < 0$. The vertices of the adjacency graph represent *patch* points, while the edges represent neighboring relations. Thus, $e = (v_0, v_1)$ is considered a boundary edge if and only if for all vertices v_k in $\mathcal{G}_p$ and adjacent to both v_0 and v_1, the triple product $u_k \cdot (u_0 \times u_1)$ has the same sign for all v_k.

The algorithm for identifying boundary points has cost $O(m)$, where m is the number of edges in $\mathcal{G}_p$. Figure 4b shows the boundary points identified by the algorithm on a single range image of a mug model (Fig. 4a), whose point cloud was obtained using a structured light scanner [22]. Note the small holes on the model due to high reflectivity and some noise. This example illustrates the robustness of our algorithm to find boundaries.

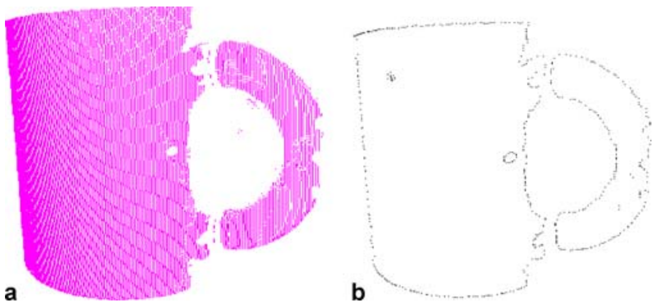

Fig. 4. **a** One range image of a mug; **b** The identified boundary points

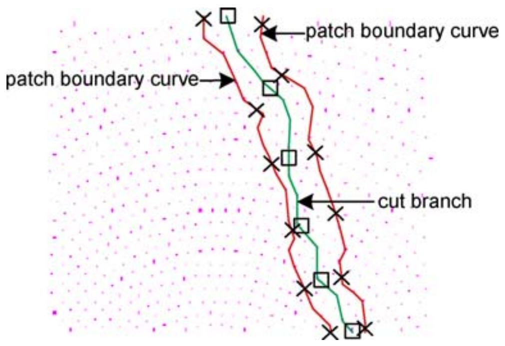

Fig. 5. We use *patch boundary curves* (shown in *red*) to help order and parameterize the points along a cut (*green curve*). Here, the squares represent points along the same *cut branch*, while the $\times$'s are their corresponding points along the patch boundary curves

4.3 Cut handling

The last step of the point parameterization stage (Fig. 1) is cut handling. A *cut* is a curve defined by points for which the accumulated error in the propagated parameter values is bigger than some threshold. Cuts separate different patches but can also define a limit between two regions on the same patch (Fig. 3).

Points along the cuts are special in the sense that they might have multiple parameters while points inside any patch have a single parameter value. *Cut points* are computed from *gap points*. For each gap point p_g we find its *kNN* using an approach similar to the one of Pauly et al. [20]. Edges connecting p_g and its *gap* neighbors are added to a *gap point adjacency graph* $\mathcal{G}_g$. Minimum spanning trees (MST) are then extracted from all connected subgraphs of $\mathcal{G}_g$[1]. From each spanning tree T_i, we remove all branches whose lengths are shorter than $3E$, where E is the average length of the *kNN* for all *patch* points. The value $3E$ was defined empirically based on our experiments. The pruned version of T_i becomes a *cut curve* C_i and its points are stored in a graph $\mathcal{G}_c$ (later, cut branches will be extracted from this graph and parameterized). Cut curves are shown as green curves in Fig. 3, which depicts the parameterization of a point cloud representing a sphere.

Starting with the set of boundary points, we also build MSTs and compute pruned versions of them. Each resulting curve is then closed by creating an edge between the first and last point, and is called a *patch boundary curve*. The set of all patch boundary curves tightly enclose the cuts. Patch boundary curves are shown in red in Fig. 5, while cuts are shown in green.

Conceptually, each cut point can be assigned an arbitrary number of different parameter values, which are obtained from patch boundary points. Note that, in this case, each parameter value is associated to a different

[1] A minimum spanning tree connects all vertices of a connected (sub)graph while minimizing the sum of the weights of the selected edges.

patch and is only used in the context of such a patch. Thus, let C be a cut and let p_c be a cut point with degree 1 (i.e., connected to only one other cut point). We proceed the cut parameterization by first finding, for each p_c, its nearest point in the patch boundary curves separated by C (Fig. 5). In case p_c separates two parts of the same patch, we find its two closest points in the patch boundary curve (one at each side of p_c).

The order of these nearest points along the patch boundary curve determines the order of *cut branches* (i.e., the minimum sets of connected edges in $\mathcal{G}_c$ enclosed by two cut points of degree 1). For each point on a *cut branch*, we find the nearest patch point along the corresponding portion of the patch boundary curve (Fig. 5). The approach described in Sect. 4.1 is used to parameterize the cut points.

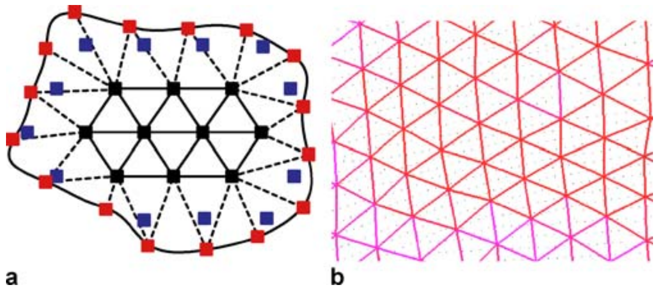

Fig. 6. **a** Vertex pattern used for resampling a patch. *Red squares* are samples along the boundary. *Black* and *blue squares* are samples inside the patch. The *blue* ones are too close to the *red squares* and are rejected (deleted). The *red* and *black* ones, together with the connection between them comprise a mass-spring system. **b** The resulting nearly regular mesh

5 Meshing from parameterization

Once the parameterizations of all patches are available, one can proceed to create a triangle mesh with arbitrary resolution. To create a complete mesh, the user needs to specify the side length L of the equilateral triangles. To preserve geometric details and avoid the occurrence of T-vertices in the resulting mesh, we first create vertices along the *cuts*, forming the boundary vertices of the parameter patches. Then, interior vertices are created using a regular sampling pattern. Algorithm 2 presents some pseudo code describing this process.

Algorithm 2. The meshing pipeline

```
Procedure Meshing()
    create shared vertices along the cuts
    place vertices inside patches
    created triangles via edge flipping
    optimize the location of interior vertices
```

5.1 Vertices along cuts

We create vertices along each *cut branch* by marching from one end to the other, adding vertices in-between. It is enforced that the distance in 3D between adjacent vertices is L. If the distance between one end and its nearest vertex becomes less than $\frac{L}{2}$, we delete that vertex. After that, this distance is in the range of $[\frac{L}{2}, \frac{3L}{2}]$. We relax the location of vertices, except those at the end points, along the *cut branch* and make the distances between every vertex and its previous/next neighbors equal. The 2D parameters and the 3D location of in-between cut vertices are obtained by linearly interpolating adjacent cut points.

5.2 Triangle placement

Interior vertices of a *parameter patch* are created by sampling it using a grid pattern composed of equilateral triangles, as shown in Fig. 6a. Interior vertices are located on the grid points and their parameters are determined implicitly. Conceptually, the meshing process can be understood as follows: let β be the set of points along the boundary of a patch P_i (matching its surrounding cut curves) and represented in 2D parameter space. Such points are shown in red in Fig. 6a. For each pair (β_i, β_j) of adjacent points along the boundary, we compute the coordinates of the interior vertex β_k that, together with β_i and β_j, would result in an equilateral triangle. However, in order to keep the mesh as regular as possible, we do not use β_k. Instead, we use the grid point closest to β_k as the third triangle vertex. This causes all interior vertices of the mesh to be on the grid pattern, and the triangles with vertices on the borders to be the only ones that might not be equilateral (Fig. 6a). An optimization procedure later tries to improve the aspect ratio of such triangles, which may slightly change the aspect ratio of some interior triangles. This triangle placement algorithm has the complexity of $O(m)$, where m is the number of triangles.

In practice, we start from one triangle on the boundary of the parameter patch which takes one interior vertex and two cut vertices, and propagate triangles around using an approach similar to the ball-pivoting strategy of Bernardini et al. [4]. Starting from one triangle, we keep flipping the triangle about its edges to create new triangles. The flipping stops when the current edge is part of the patch boundary or the flipped triangle has already been created. We show the triangles created after different numbers of propagation steps in Fig. 7.

For each interior vertex, we need to find its *nearest patch point* (NPP). For cut vertices, their NPP are their nearest points in the *patch boundary curve*. When we find a new triangle by flipping, we will need to find the NPP of the new vertex from the NPP of the vertices of the edge. Starting from the NPP of these two vertices of the edge, we repeatedly find the nearest neighbor point in the parameter space to the new vertex until the process stops. This process is illustrated in Fig. 8. Here the triangle $\overline{v_0 v_1 v_2}$ is flipped about edge (v_0, v_1) to create a new

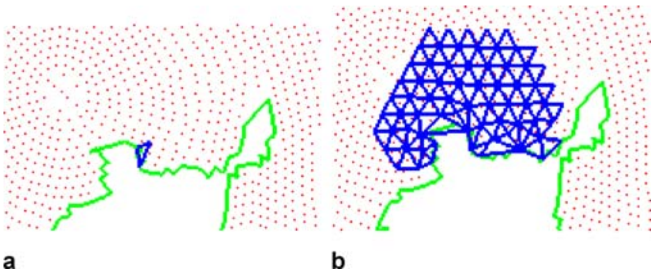

Fig. 7a,b. *Triangles* created inside a patch after **a** 1 step, and **b** 100 steps. *Red dots* are the original points and the *green line* represents a cut curve. The resulting triangles are shown in blue

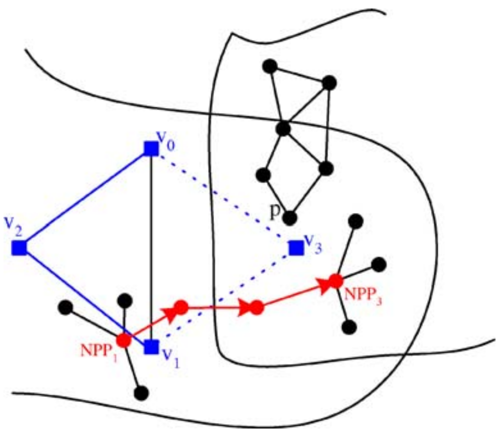

Fig. 8. The path (*red*) to find the NPP of an interior vertex v_3. Due to patch overlapping, NPP3 (the NPP of v_3) is found by following the path from NPP1. Note that although p is closest to v_3 in the 2D parameter space, NPP3 is closest to v_3 in 3D space

triangle $\overline{v_0 v_1 v_3}$. For the new interior vertex v_3, we need to compute its 2D parameter values as well as its 3D location. The 2D parameters are obtained with ease. To compute the 3D location, we first need to find the NPP of v_3, using the process described above. During this triangle-flipping process, we check the shortest distance between v_3 and any cut vertex. If this is $\leq \frac{L}{2}$, v_3 is too close to the boundary and is discarded.

Note that the triangles around the boundary are not equilateral, even in the 2D parameter space. To reduce this problem, we relax the obtained mesh using a mass-spring system. Each vertex becomes a mass point and any edge connecting vertices becomes a spring. The 2D parameters of the interior vertices are then relaxed to minimize the following energy function, while the 2D parameters of the vertices along the cut are fixed:

$$E = \sum_i \Big(\sum_{q_j \in kNN(q_i)} (q_i - q_j) \Big)^2. \tag{3}$$

The 3D location of an interior vertex is computed from the NPP and its *kNN* in $\mathcal{G}_p$. We use a moving-least-square fitting scheme [16, 27] to find the 3D location of this interior vertex. In this case, we compute the X, Y, Z coordinates separately. We try to minimize the following error function:

$$E(s) = \sum_{i=1}^{N} w_i(p_i)(s(p_i) - f_i)^2, \tag{4}$$

where

$$s(u, v) = a_0 + a_1 u + a_2 v + a_3 u^2 + a_4 v^2 + a_5 uv \tag{5}$$

is a low order polynomial surface and f_i is the height value related with each point. $w_i(p) = \frac{e^{-\alpha d_i^2(p)}}{d_i^2(p)}$ is the point-wise weight function and $d_i(p)$ is the distance from the query location p to point p_i. The parameter α controls the influence of vicinity features. Using the X, Y, Z coordinates of the points to define f, one obtains the various sets of coefficients a_0 to a_5, from which the the 3D location of the interior vertices can be resampled.

6 Results and discussion

We have used our algorithm to reconstruct meshes from a variety of point cloud datasets, including some real-world medical and 3D scanning data. All the experiments were carried on a Pentium 4 2.2 GHz PC with 768 M memory. The human colon dataset was obtained from CT scans of a real human body. The 3D points are computed using the marching cubes algorithm. We show the cuts generated by our algorithm for the parameterization of the virtual colon dataset in Fig. 11a. The reconstruction result is shown in Fig. 11b and Fig. 11c. Figure 12 presents a view of the inside of the colon. The resulting mesh is composed of triangles with good aspect ratios. Note that since the 3D mesh is seen in perspective, the projection of the triangles whose normals are almost perpendicular to the viewing direction appears distorted. The obtained mesh allows very nice colon visualizations during the exploration of the model, which contains 61 334 points. It took 30.23 s for parameter propagation and 50.04 s for meshing using $L = E$. The number of resulting triangles and vertices are 33 566 and 18 743, respectively.

The sphere model is used to show the quality of the resulting mesh. Figure 9 shows two meshes with different resolutions created from the original point cloud. Due to the constraint of shared vertices along cuts, one can expect some irregular arrangement of edges along the cuts. While within a patch, edges of triangles only have six possible directions, edges along a cut can have arbitrary directions. This tends to happen regardless of the resolution used to extract the resulting mesh. The sphere model has 10 270 points. It took 3.02 s for the propagation, and 4.87 s and

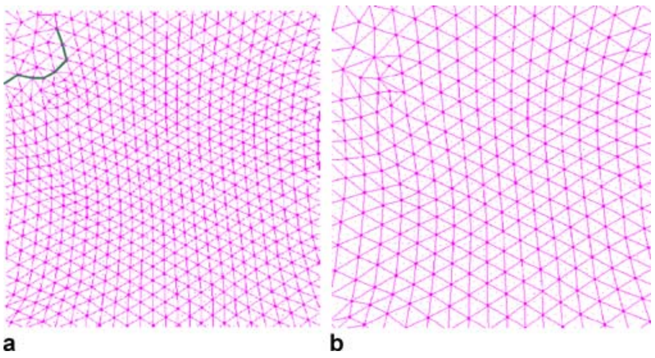
Fig. 9a,b. Reconstruction result of the sphere model: **a** $L = E$. **b** $L = 1.5E$. The *green curve* is a cut, where the triangles created around it have distorted shapes

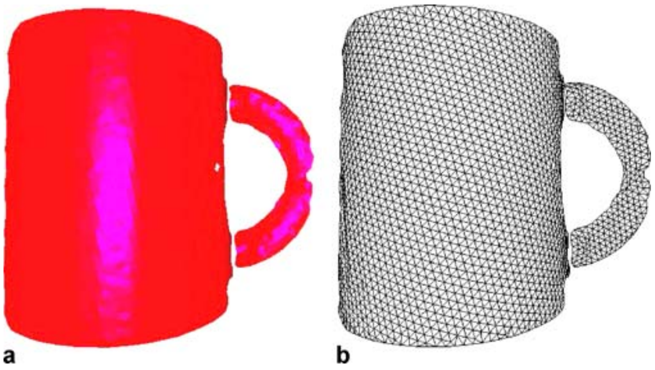
Fig. 10a,b. Reconstruction result of the mug model: **a** Shaded model. **b** Wireframe. Note the regular shapes of the resulting triangles

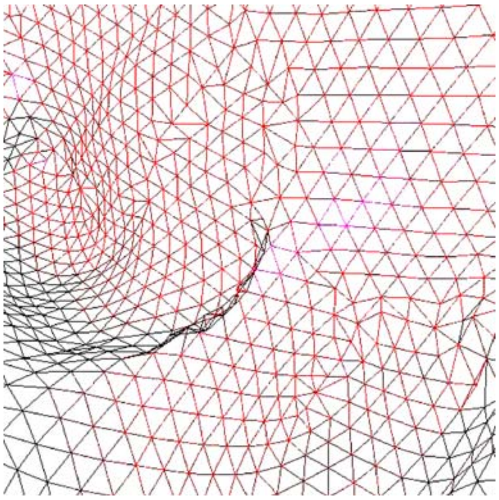
Fig. 12. An inside view of the reconstructed colon model seen in perspective, which cause the projection of some triangles to appear to have poor aspect ratio. In fact, all triangles are nearly equilateral

2.44 s to obtain the meshing results for these two models, respectively. For the model with $L = E$, we have 987 triangles and 710 vertices. For the model with $L = 1.5E$, we obtain 455 triangles and 320 vertices.

The reconstruction of the mug model is shown in Fig. 10. Note that the boundaries of this model, including a small hole on the surface, are well preserved by our algorithm. One should note the regular shape of the resulting triangles (Fig. 10b). This model has 34 916 points. It took 6.92 s and 6.77 s for the propagation and meshing, respectively. The resulting model with $L = E$ contains 3504 triangles and 1872 vertices.

The resulting parameterization can be used to perform texture mapping (Fig. 13a) and bump mapping (Fig. 13b). While cuts (parameter discontinuities) may introduce seams when using textures containing regular patterns, this problem can be alleviated with the use of stochastic textures (Fig. 13a). Haitao et al. [29] have proposed an approach to synthesize seamless textures using this kind of parameterization.

In our approach, a global parameterization assures a nearly globally isometric mapping between the 3D space and the parameterization space, producing nearly equilateral triangles. Since it is a two-step algorithm, the quality of the parameterization is critical for the meshing result. The chosen parameterization approach is robust to varying sampling rate and a moderate amount of noise. Because the parameterization relies on local operations, it could handle complex topologies as long as the local neighborhood represents a topological disk. However, the

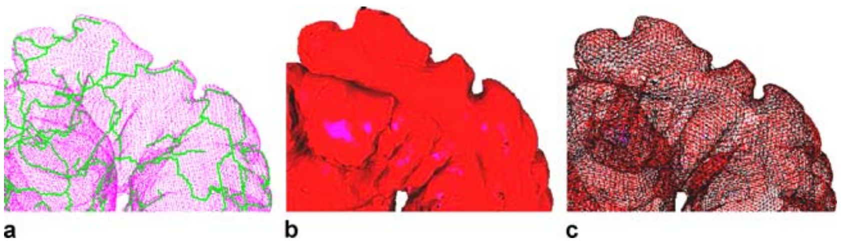
Fig. 11a–c. Reconstruction of a human colon model: **a** the point model and the cuts generated using point parameterization; **b** the shaded view of the reconstructed model; and **c** the wireframe view

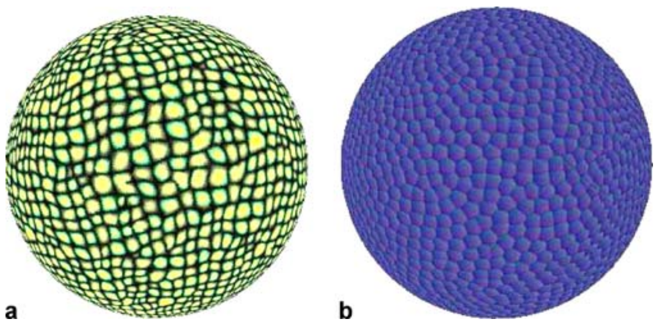

Fig. 13a,b. Applications of our point parameterization for **a** texture mapping and **b** bump mapping

error accumulates due to the progressive nature of the parameterization algorithm. The number of the parameter patches also increase with increased noise level. In this case, the mesh tends to become more irregular with longer cuts.

In this approach, we do not fill the holes. Instead, we preserve surface boundaries, including hole boundaries. One may apply the algorithm of Sharf et al. [24] to fill the holes directly on the point model before generating the mesh.

In addition, our point parameterization approach does not take shape features into account. In the case that sharp features exist, the parameters of a point may be distorted because the Euclidean distance may differ significantly from the Geodesic distance. In the meshing stage, the algorithm also does not pay attention to the existence of sharp features as well. If the feature is really sharp (an acute angle), the parameterization may fold over in the wrong direction. The reason for this problem is that the parameters of a point on one side of the sharp edge may be computed based on its k-neighborhood on the other side, causing a point to be projected on the wrong side of an edge in the 2D parameter space.

In the case of the presence of abruptly varying sample rates, we may also obtain a single parameterization patch. However, the parameter patch may consist of several small patches interconnected to each other. Although the parameterization within each small patch is reasonable, these small patches tend not to be well aligned globally, as shown in Fig. 15, which illustrates the ceiling of a real indoor scene. Due to the abrupt sampling rate change, we are not able to reconstruct it. The problem happens along the paths connecting these small patches. The points along the connection paths have been parameterized, based on one or two *patch* points for most of the time. The parameterization error accumulates and propagates after several steps. For a single point, its parameters are computed as an average of the contributions from its neighboring patch points, which significantly reduces distortions. With changing sampling rates, few neighboring patch points could have an much bigger contribution

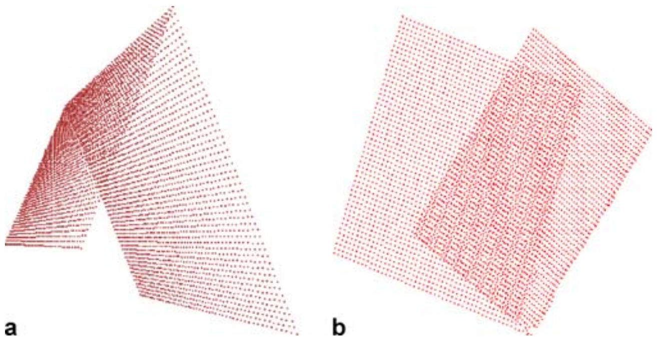

Fig. 14a,b. Problems of parameterization with respect to abrupt change of sampling rates: **a** a point model with a sharp edge; and **b** the parameterization patch of the model

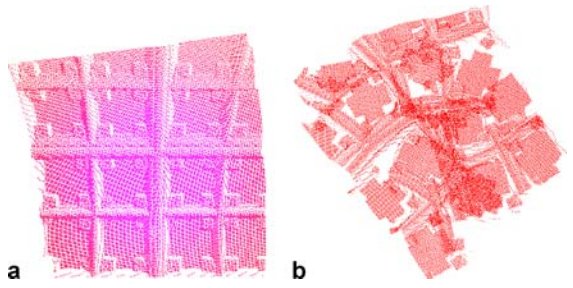

Fig. 15a,b. The indoor scene model has abruptly varying sample rates: **a** the point model; and **b** the parameterization patch

than others, thus introducing large errors. If this distortion propagates several times, it tends to become visually noticeable. In extreme situations, such distortions may even cause the resulting topology to be incorrect. In these situations, we will not have the correct regular mesh in the end.

The main limitation of our approach lies in that the parameterization procedure is not optimized within each parameter patch. Due to error propagation that happens during parameter propagation, points visited later in the same patch might accumulate considerable distortion, for which we do not have a measurement/control for the error bound. In addition, the discontinuity of the parameterization across cuts tends to exhibit seams when it is used for mapping highly regular texture patterns. Note that even in the presence of parameter discontinuities, the resulting meshes will be nice and regular. Thus, one could compute some global parameterization to the mesh as a post-processing using, for instance, the approaches described in [10, 15]. Alternatively, one could apply a transition rule at the vicinity of the cuts consisting of blending the multiple parameters (one from each adjacent patch that shares the vertex) associated to the cut vertices. The overall complexity of this algorithm is $O(n \log n)$, where n is the number of points in the input. This cost is dominated by the point parameterization stage.

7 Conclusion

This paper presented a new surface reconstruction algorithm for producing meshes with regular triangles from unorganized point clouds. The algorithm consists of generating a point parameterization before obtaining the resulting mesh. This approach preserves detailed geometric information and gives the meshing process great flexibility. The obtained meshes consist of nearly equilateral triangles, thus leading to better visualizations and making them suitable for specialized operations such as finite element analysis.

In the future, we would like to explicitly find certain kinds of geometric details before parameterization, such as surface boundary [28] and sharp features [20]. These features could then be used as additional constraints for propagation. By saving them as part of the cuts, vertices could be created along them, guaranteeing that they are more faithfully preserved. In addition, we intend to use a different re-sampling pattern to create meshes with the size of triangles adapted to some geometrical properties, such as curvature. Right now, all extracted triangles have approximately the same size.

Acknowledgement This work is partially supported by NSF grant CCR-0306438, CNPq Brazil (Processo No 4773442003-8) and Microsoft Brazil.

References

1. Alliez, P., Cohen-Steiner, D., Devillers, O., Levy, B., Desbrun, M.: Anisotropic polygonal remeshing. SIGGRAPH, pp. 485–493 (2003)
2. Alliez, P., Meyer, M., Levy, B., Desbrun, M.: Interactive geometry remeshing. SIGGRAPH, pp. 347–354 (2002)
3. Amenta, N., Kil, Y.J.: Defining point-set surfaces. SIGGRAPH, pp. 264–270 (2004)
4. Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., Taublin, G.: The ball-pivoting algorithm for surface reconstruction. IEEE Trans. on Visualization and Computer Graphics **5**(4), 349–359 (1999)
5. Botsch, M., Kobbelt, L.: A robust procedure to eliminate degenerate faces from triangle meshes. Vision Modeling and Visualization Conference, pp. 283–290 (2001)
6. Bunsen, O., Fleischmann, G.: Mesh optimization for animation purposes. SimVis, pp. 66–75 (1997)
7. Desbrun, M., Meyer, M., Alliez, P.: Intrinsic parameterizations of surface meshes. Eurographics, pp. 209–218 (2002)
8. Floater, M., Reimers, M.: Meshless parameterization and surface reconstruction. Comput. Aided Geom. Des. **18**, 77–92 (2001)
9. Gotsman, C., Gu, X., Sheffer, A.: Fundamentals of spherical parameterization for 3D meshes. SIGGRAPH, pp. 358 – 363 (2003)
10. Gu, X., Gortler, S., Hoppe, H.: Geometry images. SIGGRAPH, pp. 355–361 (2002)
11. Gumhold, S., Wang, X., McLeod, R.: Feature extraction from point clouds. 10th International Meshing Roundtable, pp. 293–305 (2001)
12. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press, Cambridge, ISBN: 0521623049 (2000)
13. Hoppe, H., DeRose, T., Duchamp, T., MacDonald, J., Stuetzle, W.: Surface reconstruction from unorganized points. SIGGRAPH, pp. 71–78 (1992)
14. Hormann, K., Reimers, M.: Triangulating point clouds with spherical topology. Curve and Surface Design, pp. 215–224 (2003)
15. Jin, M., Wang, Y., Yau, S., Gu, X.: Optimal global conformal surface parameterization. IEEE Visualization, pp. 267–274 (2004)
16. Lancaster, P., Salkauskas, K.: Curve and Surface Fitting, An Introduction. Academic Press (1986)
17. Losasso, F., Hoppe, H., Schaefer, S., Warren, J.: Smooth geometry images. Eurographics Symposium on Geometry Processing, pp. 138–145 (2003)
18. Mencl, R., Müller, H.: Graph-based surface reconstruction using structures in scattered point sets. Computer Graphics International, pp. 298–311 (1998)
19. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., Seidel, H.: Multi-level partition of unity implicits. SIGGRAPH, pp. 463–470 (2003)
20. Pauly, M., Keiser, R., Gross, M.: Multi-scale feature extraction on point-sampled models. Eurographics **22**(3), 281–289 (2003)
21. Praun, E., Hoppe, H.: Spherical parametrization and remeshing. SIGGRAPH, pp. 340–349 (2003)
22. Rusinkiewicz, S., Hall-Holt, O., Levoy, M.: Real-time 3D model acquisition. SIGGRAPH, pp. 438–446 (2002)
23. Scheidegger, C., Fleishman, S., Silva, C.: Triangulating point set surfaces with bounded error. Symposium on Geometry Processing, pp. 63–72 (2005)
24. Sharf, A., Alexa, M., Cohen-Or, D.: Context-based surface completion. SIGGRAPH, pp. 878–887 (2004)
25. Sorkine, O., Cohen-Or, D., Goldenthal, R., Lischinski, D.: Bounded-distortion piecewise mesh parametrization. IEEE Visualization, pp. 355–362 (2002)
26. Tewari, G., Gotsman, C., Gortler, S.: Meshing genus-1 point clouds using discrete one-forms. Comput. Graph. **30**(6), 917–926 (2006)
27. Wang, J., Oliveira, M.: Filling holes on locally smooth surfaces reconstructed from point clouds. Image Vis. Comput. **25**(1), 103–113 (2007)
28. Xia, C., Hsu, W., Lee, M., Ooi, B.: Border: Efficient computation of boundary points. IEEE Trans. Knowl. Data Eng. **18**(3), 289–303 (2006)
29. Zhang, H., Qiu, F., Kaufman, A.: Fast hybrid approach for texturing point models. Comput. Graph. Forum **23**(4), 715–725 (2004)
30. Zwicker, M., Pauly, M., Knoll, O., Gross, M.: Pointshop 3D: An interactive system for point-based surface editing. SIGGRAPH, pp. 322–329 (2002)

Jianning Wang is currently a Ph.D student of computer science at State University of New York at Stony Brook. His interests include computer graphics (especially surface reconstruction), its medical applications, geometrical modeling, and computer vision.

Manuel M. Oliveira is a faculty member at the Federal University of Rio Grande do Sul (UFRGS), in Brazil. He received his PhD in computer science from the University of North Carolina at Chapel Hill, in 2000. Before joining UFRGS in 2002, he was an Assistant Professor of Computer Science at the State University of New York at Stony Brook from 2000 to 2002. His research interests cover most aspects of computer graphics, but especially in the frontiers among graphics, vision, and image processing. This includes image-based and real-time rendering, representation and rendering of surface details, 3D photography, 3D scanning, and surface reconstruction from point clouds.

Haitao Zhang is also a Ph.D student of computer science at State University of New York at Stony Brook. He is interested in point model rendering and texture synthesis.

Arie Kaufman is a distinguished professor in the Computer Science Department at State University of New York at Stony Brook. His interests are computer graphics, visualization, user interfaces, virtual reality, multimedia, and computer architecture.