Visual Comput (2007) 23: 397–408
DOI 10.1007/s00371-007-0098-7

ORIGINAL ARTICLE

Haitao Zhang
Arie E. Kaufman

Point-and-edge model for edge-preserving splatting*

Published online: 31 March 2007

H. Zhang (✉) · A.E. Kaufman
Center for Visual Computing (CVC) and
Computer Science Department
Stony Brook University, Stony Brook, NY
11794-4400, USA
{haitao, ari}@cs.sunysb.edu

Abstract We introduce the point-and-edge model for edge-preserving modeling and rendering. Besides a set of surface points, the point-and-edge model also includes edge points representing the sharp edges in the model. The surface points and the sharp edges are relatively independent of each other. We present a feedback algorithm to simplify the point-and-edge model with bounded error based on an edge-preserving clustering method. An efficient constrained splatting method is used to preserve the sharp edges in the rendering, regardless of the surface point density.

Keywords Point-and-edge model · Edge-preserving simplification · Edge-preserving rendering · Constrained splatting

1 Introduction

A point-sampled model, which uses a set of surface points to represent the geometric shape has become an important model representation format. Algorithms for direct point rendering have been introduced in recent years [4–7, 18–20, 25, 26]. In order to reconstruct a hole-free visible surface from the irregular points without any coherence information, a splat is created on the image plane for each point. Usually, the splat is the projection of a disk defined on the tangent plane of the surface, which is centered at the point position with a pre-defined radius. The radii of the disks should be large enough to prevent holes, and should not be too large to create noticeable artifacts. The blending among the overlapping regions of nearby splats generates a smooth surface in the rendering. Artifacts may appear in the regions with high surface curvature, which can be minimized by increasing the point density. For a sharp edge, which is an important feature especially for man-made objects, a much higher point density is needed to prevent noticeable artifacts even when the geometry is very simple such as the intersection of two planes.

*This work has been partially supported by NSF grant CCR-0306438.

In order to preserve the sharp edges in direct point rendering with relatively low point sampling rate, we present a point-and-edge model. The point-and-edge model, an extension of the common point-sampled model, consists of a set of surface points, and 3D edge points for the sharp edges. Each sharp edge is represented by an edge point list. Unlike the splat clipping [26], in which at most two clip lines are defined on the splat plane (tangent plane of the model) for every point, in our point-and-edge model, the surface points and the edge points are relatively independent of each other. The sharp edges do not need to reside on the tangent planes of the surface points, and the number of edge segments is not limited by the number of surface points. Thus, sharp edges can be preserved in the modeling and rendering, regardless of the surface point density or the shape of the sharp edges. For model simplification, we use an edge-preserving clustering algorithm to reduce the number of surface points. In the simplification, a feedback method is used to limit the surface error within a given tolerance.

We present an edge-preserving splatting algorithm for the direct rendering of the point-and-edge model. Splatting is restricted by the edges, so that the sharp edges can be preserved in the rendering regardless of the surface point density and the viewing direction (Fig. 1). Our

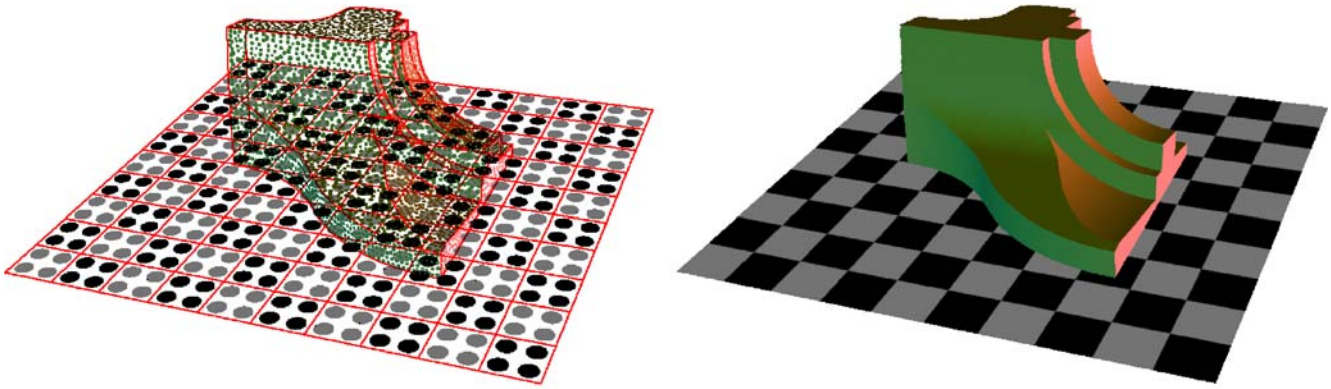

Fig. 1a,b. The Fandisk model above a checkboard model, with total of 5175 surface points. **a** Surface points (shown as dots with their original colors) and edges (shown as red lines) in the point-and-edge model. **b** Edge-preserving splatting of the point-and-edge model

GPU-based algorithm is an extension of the splat clipping proposed by Zwicker et al. [26]. Since our model supports unlimited clip line segments, the accuracy of the sharp edges is not compromised by the low density of the surface points.

The main contribution of this paper is the point-and-edge model for edge-preserving modeling and rendering, including:

- A point-and-edge model which preserves the exact sharp edge information regardless of the surface point density
- An error-bounded edge-preserving simplification method for the point-and-edge model
- A constrained splatting method for hardware-accelerated edge-preserving rendering

2 Related work

In order to use the splatting-based direct point rendering methods [4–7, 18–20, 25, 26], there are indirect point rendering systems [2, 9] which fit a piecewise quadratic function in a local region. In the rendering, these quadratic functions are resampled or converted into other rendering primitives such as meshes. The sharp edges would be preserved in the rendering, only if they are detected in the local fitting process. Ohtake et al. [15] detect the edges from the point cloud and use different quadratic functions for the different sides of the edges. Fleishman et al. [10] use a forward search method in the moving least squares (MLS) computation to preserve sharp edges. These indirect rendering methods are usually slower than the splatting-based methods.

Boolean operations on the point-sampled models could generate sharp edges. For the splats intersecting the sharp edges, Adams and Dutré [1] resample them into several smaller splats to keep the artifacts below a pre-defined threshold. Pauly et al. [17] clip the splats against the planes defined by the intersected splats during scan-conversion. Botsch et al. [6] also implement splat clipping [26] in their Phong splatting method. Wicke et al. [22] use a software-based renderer for the sharp edges created by CSG operations. They find the clipping partners in the rendering and use the two closest surfels for the inside/outside classification. While their methods support multiple clipping lines, artifacts may appear in some cases due to the simple classification method. Talton et al. [21] use silhouette clipping based on Voronoi rasterization for sparse point sets, which can clip point splats using the corresponding back-facing point splats on the other side of the edge. This method is unreliable, which fails when both sides of the edge are front-facing. For the clipping-based modeling and rendering, the number of clipping lines or planes is linear to the number of splats along the sharp edges, and the clipping lines are defined on the tangent planes of the splats. Thus, for the regions along curved sharp edges or near corners, the splat density should be high enough to preserve the original edge shape. In our point-and-edge model, the sharp edges are accurately preserved regardless of the surface point density.

Bala et al. [3] propose an edge-and-point renderer, which projects visible edges and shaded points onto the image plane, followed by a constrained interpolation to reconstruct the hole-free surface in the image space. The interpolation kernel is restricted by the visible edges for edge preservation. This method can be adapted to render our point-and-edge models. After projecting the edges onto the image plane, we splat the surface points and make sure that the splats do not go through nearby edges. Unlike Bala et al. [3], both the visible and invisible edges are projected onto the image plane. Depth comparison is used to constrain the visible splats only by the visible edges. If the depth difference between the splat and the edge is within a threshold, the splat cannot go through the edge. This cannot guarantee artifact-free rendering. For example, in

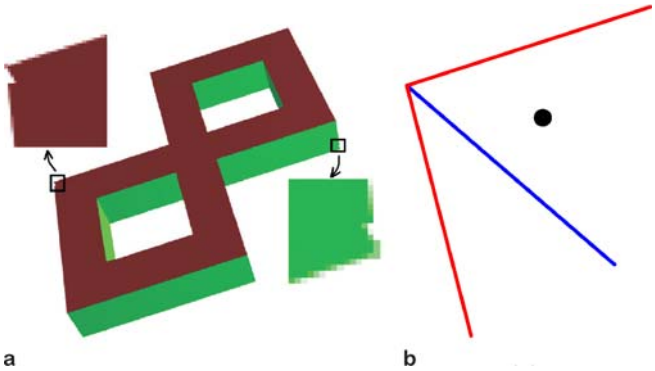

Fig. 2a,b. Rendering of a software-based constrained splatting method similar to [3]. **a** Artifacts appear inside the two black boxes, shown in the closeup views. **b** The blue line is an invisible edge meeting two visible edges (red lines) at a corner. The black dot is a visible surface point which is mistakenly constrained by the invisible edge

Fig. 2, the software-based constrained splatting creates artifacts near the two corners in the rendering. This is because the visible splats are mistakenly constrained by the invisible edges whose depth difference is within the depth threshold. In addition, since current GPUs do not support customized rasterization, this rendering method has to be implemented in software, resulting in at least one magnitude slower rendering speed compared to our GPU-based point rendering algorithm. The artifacts shown in Fig. 2 can be avoided by using our splatting-based renderer for the point-and-edge model, which is a GPU-based method for efficient and accurate edge-preserving rendering.

3 Point-and-edge model

A point-sampled model is represented by a set of surface points:

$$\mathrm{PM}_n = \{p_i | p_i = (d_i, n_i, c_i, r_i), i = 1 \text{ to } n\} \quad (1)$$

where n is the number of points, d_i, n_i and c_i are the position, normal and color attributes of point p_i, and r_i is the pre-defined splat radius for hole-free surface splatting.

Our point-and-edge model:

$$\mathrm{PAE} = \{\mathrm{PM}_n, \langle E_m, V_l \rangle\} \quad (2)$$

consists of two parts. PM_n is a point-sampled model containing all the surface points, and $\langle E_m, V_l \rangle$ consisting of m sharp edges $E_m = \{e_i, i = 1 \text{ to } m\}$ represented by l edge points $V_l = \{v_i, i = 1 \text{ to } l\}$. Each edge e_i is represented by an edge point list, with two neighboring edge points as a line segment. These connected line segments form a path or a loop. In the remainder of this paper, we will refer to these line segments as edge segments. Thus, e_i includes the number of edge points, the index of the first edge point, and whether it is a path or a loop for the ith edge. There are l edge points in the m edges, which are stored in a linear order according to E_m. In this way, the surface points and the edges are totally independent, and thus can be modeled separately.

Throughout this paper, we will use v for the edge points, and p for surface points. $\overrightarrow{p_i p_j}$ will be used to represent a vector from p_i to p_j, and $\overrightarrow{\overrightarrow{p_i p_j}}$ is the corresponding unit vector. Since our point-and-edge model is designed for splatting-based rendering, each surface point represents a surface splat on the surface of the model. From here on, we will use surface point and splat interchangeably.

In order to make use of the programmability of modern GPUs for fast and artifact-free rendering, we add three constraints to our point-and-edge model.

Constraint 1. If surface point p_i intersects with an edge segment $v_1 v_2$, $|n_i \cdot \overrightarrow{\overrightarrow{v_1 v_2}}| < T_n$. The term n_i is the normal of p_i, and T_n is a pre-defined threshold.

Constraint 2. If surface point p_i intersects with k edge segments and $k > 1$, these edge segments should form a linear connected path $\langle v_1, v_2, \ldots, v_{k+1} \rangle$.

Constraint 3. If surface point p_i intersects with k edge segments (a connected path $\langle v_1, v_2, \ldots, v_{k+1} \rangle$ according to *Constraint 2*) and $k > 2$, p_i should be on the same side of the k edge segments $\{\overrightarrow{v_i v_{i+1}}, i = 1 \text{ to } k\}$.

Figure 3 shows an edge segment $v_1 v_2$ intersecting with a splat p_i with radius r_i. The terms v_1^p and v_2^p are the projections of edge points v_1 and v_2 on the splat plane. The term $v_1 v_2$ *intersects with* splat p_i when and only when the distance between p_i and $v_1^p v_2^p$ is within r_i and *Constraint 1* is satisfied ($|n_i \cdot \overrightarrow{\overrightarrow{v_1 v_2}}| < T_n$). This guarantees that the projection of $v_1 v_2$ goes through the splat. If $\overrightarrow{v_1 p_i} \times \overrightarrow{v_1 v_2} \cdot n_i > 0$, p_i is on the *right side* of edge segment $v_1 v_2$;

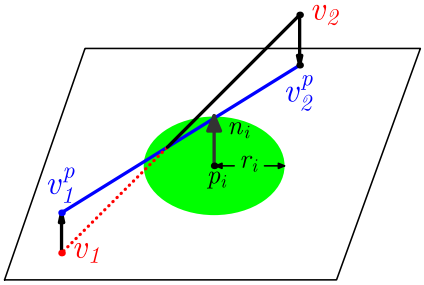

Fig. 3. *Constraint 1.* Line segment $v_1 v_2$ intersects with splat p_i when and only when $v_1^p v_2^p$ goes through splat p_i and $|n_i \cdot \overrightarrow{\overrightarrow{v_1 v_2}}| < T_n$, with n_i as the splat normal and $v_1^p v_2^p$ as the projection of $v_1 v_2$ on the splat plane

otherwise, p_i is on the *left side* of edge segment v_1v_2. In Fig. 3, p_i is on the right side of v_1v_2, or the left side of v_2v_1.

In the ideal case, an edge segment is on the plane defined by a corresponding intersecting splat, such as in the splat clipping [26], where the clipping lines are defined on the tangent plane of the points. When the curvature of the surface is not zero, or the edge itself is curved, this ideal case would not stand. In the worst case, the edge segment is perpendicular to the splat, resulting in bad modeling and difficulty in edge-preserving rendering. Thus, we introduce *Constraint 1* to restrict the deviation of the edge from the tangent plane of the model. In our experiments, we set $T_n = \cos(20°)$ so that the maximum deviation between the splat and the edge is 20°. *Constraint 2* and *Constraint 3* are introduced for the sake of efficient GPU-based rendering which will be discussed in Sect. 5. These two constraints are not necessary in the sense of modeling. Since the main purpose of our point-and-edge model is for fast edge-preserving rendering, we have added the two constraints.

In order to avoid the rendering artifacts shown in Fig. 2, for those surface points intersecting with edges, the correspondence information should be available. Because of *Constraint 1* and *Constraint 2*, for a point p_i intersecting with the edges, the correspondence can be represented by $\langle index, num, side \rangle$, where *index* is the first edge point index, *num* is the number of the intersected edge segments, and *side* is either *left* or *right*, for the side of the edges that p_i belongs to. If there are two edge segments ($num = 2$), *side* value is determined by the relationship between p_i and its nearest edge segment. If $num > 2$, because of *Constraint 3*, there is no conflict for the *side* value. Since all the edge points are stored in the order according to the edge they belong to, for the surface points near a corner or a looped edge, the corresponding edge points may not be stored continuously. If several edges are connected in a corner, each edge is extended by a portion of one of its neighboring edges to make our point-edge correspondence representation possible. Similarly, one end of a looped edge is extended by the other end of itself. This introduces some redundancy in the edge points, which is a tradeoff for the simplicity of the correspondence representation, which is essential for our GPU-based rendering.

Our point-and-edge model can accurately represent any sharp edge on a manifold surface, including corners. A surface point near a corner must reside between two neighboring edges that are connected at that corner. When the constraints are met, it intersects with either one of the edges or both of them. If both edges are intersected, according to *Constraint 3*, the intersected edge segments on the two edges are connected at the corner. One of the edges is extended by the other at this corner, and the corresponding edge points for this surface point are linearly stored in it.

The surface points are classified into four classes, depending on the number of the intersecting edge segments:

Simple point: points do not intersect with any edges
1-edge point: points intersecting with 1 edge segment
2-edge point: points intersecting with 2 edge segments
Complex point: points intersecting with more than 2 edge segments

Our point-and-edge model can be converted from other model formats, such as a triangular mesh or an implicit surface. Edges can also be extracted directly from the point-sampled model using some existing feature detection methods [10, 12–14, 17]. After acquiring the surface points and the sharp edges, we upsample the regions near the sharp edges to satisfy the three constraints if necessary and classify the surface points into the four classes. For each surface point, *Constraint 1* is used to find the intersecting edge segments. If *Constraint 2* or *Constraint 3* is not met, the corresponding region is progressively upsampled until the constraints are satisfied. If a point intersects with one or two connected edge segments, the constraints are guaranteed to be met. Thus, we can get a valid point-and-edge model from any existing model.

4 Model simplification

Mesh simplification methods [8, 11] can be adapted to simplify a point-sampled model. A point-sampled model can be simplified by clustering methods, iterative simplification, and particle simulation algorithms [16]. Wu and Kobbelt [23] select a subset of the original points as a hole-free approximation of the model within a prescribed error tolerance. Progressive splatting [24] uses two error metrics defined on the splats distance and normal difference in the clustering. These two error metrics are also used in our simplification method.

4.1 Edge-preserving simplification

We use an edge-preserving clustering method to simplify the point-and-edge model. Only the surface points are simplified and the edge points are kept in order to preserve the edges. Points on the different sides of an edge are prevented from being clustered together. Besides, the three constraints should not be violated in the simplification.

In the simplification, several nearby splats are clustered together to form a splat in the simplified model. We refer to the splats in the model to be simplified as *old* splats, and the splats in the simplified model as *new* splats.

Our clustering method uses a region growing algorithm. An unclustered old splat p_s is randomly selected as the seed for a cluster: $C = \{p_o\}$, with C as the set of old points in the cluster. Nearby old splats are added one by one based on some rules discussed below. For a cluster C, the center p_c and normal n_c of the corresponding

new splat are area-weighted average of the old splats:

$$p_c = \frac{\sum_{p_i \in C} \left(r_i^2 p_i\right)}{\sum_{p_i \in C} \left(r_i^2\right)} \tag{3}$$

$$n_c = normalize\left\{ \sum_{p_i \in C} \left(r_i^2 n_i\right) \right\} \tag{4}$$

where the squares of the old splats' radii are used as the weight. The radius of the new splat is set as the minimum length to ensure that the projections of all the old splats in C on the new splat plane are within the new splat.

Similar to [24], we define two error metrics based on the splat distance (L^2) and normal difference ($L^{2,1}$) between the old splats and the corresponding new splats:

$$E_d(C) = \sum_{p_i \in C} |p_i p_i'| \tag{5}$$

$$E_n(C) = \sum_{p_i \in C} (1 - n_i \cdot n_c) \tag{6}$$

where p_i' is the projection of old splat p_i on the corresponding new splat plane.

In the clustering, a nearby old splat p_i can be added into the cluster C if it meets the following three requirements:

- $R1$: p_i and p_c are on the same side of every nearby edge.
- $R2$: The resulting new splat $p_{\{C, p_i\}}$ and p_c are also on the same side of every nearby edge.
- $R3$: $E_d(\{C, p_i\}) < T_d^1$ and $E_n(\{C, p_i\}) < T_n^1$, where T_d^1 and T_n^1 are the given error tolerance.

A candidate set S_a is maintained, which is empty when a cluster begins. After adding an old splat p_i to the cluster, the unclustered splats in its k-neighborhood $N_k(p_i)$ satisfying $R1$ requirement are added to the candidate set. In our experiments, $k = 8$ is sufficient when there is no abrupt change in the surface point density. Thus, S_a contains the immediate neighbors of the current cluster C, which are not separated by any sharp edges.

Only those splats in S_a that meet $R2$ and $R3$ requirements are valid for adding into C. When there is more than one valid splat, an error metric is used to select the one with the minimum error for clustering:

$$E(C) = \alpha E_d(C) + \beta E_n(C) \tag{7}$$

where α and β are user-specified weights for the two error metrics.

The invalid splats in S_a are not considered at this time, but remain in C. This is because they may be valid after adding other splats to C first. For example, in Fig. 4, when $C = \{p_1\}$, p_2 cannot be added next because the resulting new splat p_a would be on a different side of edge $v_1 v_2 v_3$, violating $R2$. It would become valid when adding p_3 first followed by p_2.

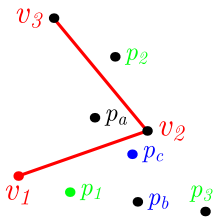

Fig. 4. $R2$ requirement. p_a, p_b and p_c are weighted averages of $\{p_1, p_2\}$, $\{p_1, p_3\}$ and $\{p_1, p_2, p_3\}$, respectively. p_a is on the wrong side of the edge $v_1 v_2 v_3$, while p_b and p_c are on the right side

C expands one old splat at a time in this way until no valid splats can be added. After generating a new splat from C, if there are still unclustered old splats, C and S_a are reset to empty to start the next cluster.

The simplification method can be used to construct a point-and-edge model hierarchy. In a model hierarchy, we often prefer that the surface point distribution in each level is relatively even. Thus, we introduce a parameter $s_{\max}$, the maximum number of old splats in any cluster. In the simplification, no old splat can be added into a cluster if the number of old splats inside the cluster reaches $s_{\max}$. If the points are evenly distributed in the original model, we will also get a simplified model with relatively even point distribution. Parameter $r_{\max}$ is the maximum radius for the new splats, which is used when there is a high variance in the point density of the original model. In the simplification, no old splat can be added into a cluster if the resulting new splat radius exceeds $r_{\max}$. The user can change these two parameters to adjust the coarseness of the resulting model. When there is no point distribution requirement in the simplification, we can set $s_{\max}$ to the number of old splats in the model, and $r_{\max}$ to the size of the model bounding box.

4.2 Feedback algorithm for bounded-error simplification

There are two types of errors in the splat-based simplification: (1) error between old splats and the corresponding new splats, and (2) error between neighboring new splats.

Type 1 error is bounded by the $R3$ requirement in the clustering. Type 2 error has not been considered so far, and large artifacts may appear in areas with high curvature but without sharp edges (Fig. 5). In order to take care of type 2 error, we extend the simplification method described above and introduce a feedback algorithm for bounded-error simplification.

Type 2 error can be defined as the maximum distance between the overlapping parts of two neighboring new splats which are not separated by any sharp edges. For two splats p_i and p_j with radii r_i and r_j, respectively, the error is:

$$E_2(p_i, p_j) = \max_p \left\{ |p, p'|, |p' p_j| \le r_j \right\} \tag{8}$$

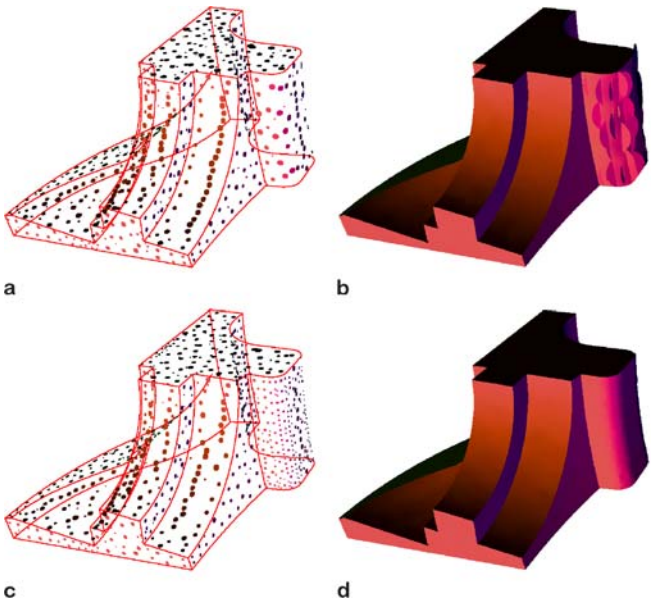

Fig. 5a–d. Artifacts caused by the discrepancy between neighboring splats. The Fandisk model is simplified using $s_{\max} = 16$. **a** Without error restriction resulting in 1154 points. **b** The corresponding rendering with artifacts caused by large type 2 error. **c** Simplified model (1529 points) from the feedback algorithm with error tolerance as 1.5 times the original model maximum type 2 error. **d** The corresponding rendering

where p is any point on p_i splat plane and within the radius ($|pp_i| \leq r_i$), and p' is the projection of p on p_j splat plane. This error has a large effect on the point splatting rendering method. In order to blend neighboring splats together, nearly all point splatting methods use a visibility splatting pass, in which the depth value is moved away from the viewer by an offset. To avoid artifacts from the discrepancy between two neighboring splats p_i and p_j, this offset should be above $E_2(p_i, p_j)$ and $E_2(p_j, p_i)$ (the two errors can have different values). If the error becomes too large, resulting in a very large offset, not only the rendering speed is affected, but also artifacts are introduced by blending invisible splats together. Furthermore, rendering artifacts from this error are view-dependent. Under some view directions, some parts of the splat are not blended with neighboring splats under any offset value.

We use a feedback algorithm to create a simplified model with bounded type 2 error. Below is the implementation of the feedback algorithm using a type 2 error tolerance T_d^2. Using this feedback algorithm, we can achieve adaptive simplification with bounded error:

Step 1. Put all old splats in the unclustered old splat set P_u. Set new splat set P_n to empty. The terms $s_{\max}$ and $r_{\max}$ are the user-defined parameters for maximum cluster size and splat radius in the simplification method.

Step 2. Cluster P_u using our edge-preserving simplification method. After the clustering, k clusters are generated: $C = \{C_1, C_2, \ldots, C_k\}$. P_u becomes empty.

Step 3. Add the generated k new splats into P_n. For any neighboring splat pair (p_i, p_j) in P_n, if $E_2(p_i, p_j) > T_d^2$ or $E_2(p_j, p_i) > T_d^2$, delete p_i and p_j from P_n, and move all corresponding old splats in C_i and C_j to P_u.

Step 4. Set $s_{\max} = s_{\max}/2$, $r_{\max} = r_{\max}/2$. Go to Step 2 if P_u is not empty.

Figure 6 shows the Fandisk model at four different point densities. Figure 6a is the original model with 13645 points. Figures 6b–d are the simplification results using our feedback algorithm with 3826, 2248, 1529 surface points, respectively. The error tolerance is set to be 1.5 times the maximum type 2 error in the original model, with the maximum cluster size $s_{\max}$ as 4, 8, 16, respectively.

5 Edge-preserving splatting

Bala et al. [3] use a software-based rendering method to reconstruct the continuous surface from visible shaded point samples and edges. While their method can be adapted to render our point-and-edge model, their software-based method is slow compared to some existing GPU-based splatting algorithms [4–7, 19, 26]. In order to take advantage of modern GPUs, we use a GPU-based

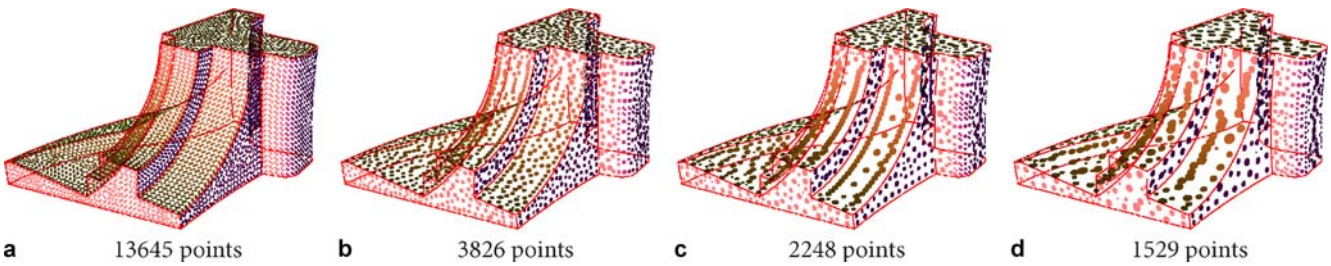

Fig. 6a–d. Point-and-edge model simplification using our feedback algorithm. **a** Original model. **b–d** Simplified model using $s_{max} = 4, 8, 16$, respectively

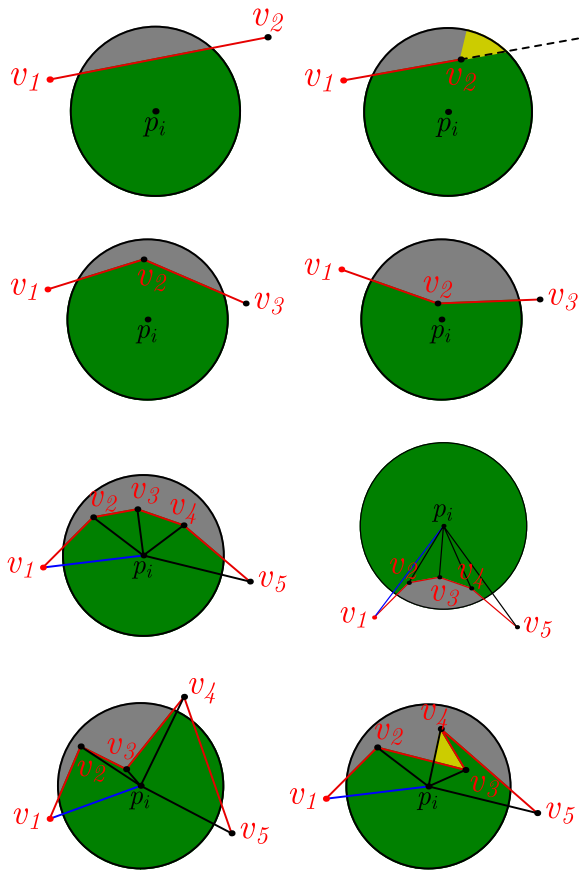

Fig. 7a–h. Edge-preserving splatting. **a**, **b** 1-edge points. **c** 2-edge point (convex). **d** 2-edge point (concave). **e–g** Complex points. **h** Invalid complex point which is prevented by *Constraint 3*

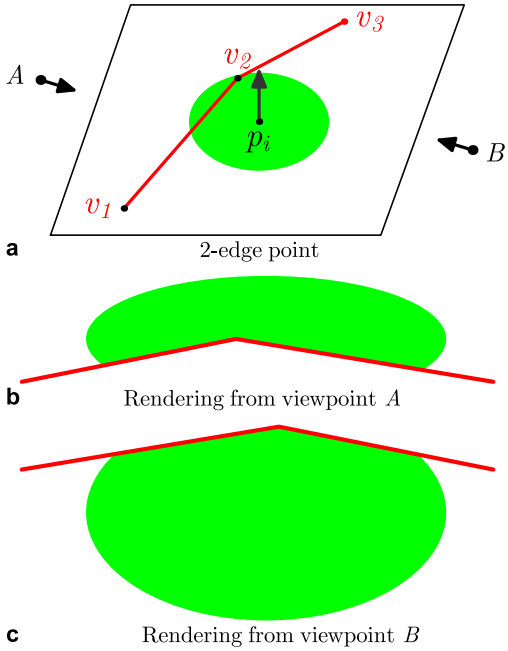

Fig. 8a–c. View-dependent convexness of 2-edge point. **a** 2-edge point p_i is intersecting with two edge segments v_1v_2 and v_2v_3. **b** Concave from viewpoint A. **c** Convex from viewpoint B

rendering algorithm, whose splat rasterization process is designed to preserve the sharp edges. Unfortunately, the GPU does not support customized rasterization in its pipeline. Thus, we adopt the splat clipping technique [26] and extend it for our edge-preserving splatting.

Most of existing GPU-based point splatting methods use a three-pass algorithm: visibility splatting pass, attribute splatting pass, and normalization/shading pass. Our edge-preserving splatting is also based on this three-pass algorithm. In the visibility splatting pass, the model is rendered with only depth buffer enabled, and the depth values are moved away from the viewer with a small offset. In the attribute splatting pass, with depth test enabled and update disabled, the model is rendered into the frame buffers. With floating-point precision blending which is available on modern GPUs, overlapping splats on the visible surface are blended together. If per-splat shading is used, shaded splat colors are computed and stored in a frame buffer. If a more complicated shading method is chosen, we use the Botsch et al. [4] rendering technique by splatting different needed attributes into multiple frame buffers, such as normal, surface position, etc. In the final pass, the accumulated attributes are normalized by the accumulated weights, and if necessary per-pixel shading is computed accordingly.

In our point-and-edge model, surface points are classified into four classes: simple points, 1-edge points, 2-edge points, and complex points. For the simple points, there is no edge constraint in the splat rasterization. Because our point-and-edge model is mainly used for man-made objects with sharp features, which usually contain many regions with low surface curvature, it does not require high point density in those regions, resulting in large splat size. With large splat size, it is very important to get accurate splat rasterization. We adopt the ray casting method [6] in order to get accurate perspective splats. In the fragment shader, the intersection between the ray through the pixel and the splat plane is computed, and the distance between the intersection and the splat center is used to determine whether this fragment should be discarded. If not, the blending weight is computed according to the distance. When the projected splat size is smaller than one pixel, the intersection may be outside the splat radius and there would be no contribution from this splat. In order to avoid this problem, in our algorithm, if the projected splat size s_p is smaller than one pixel, we omit the intersection computation and use s_p^2 as the weight for blending.

For the 1-edge points (Fig. 7a,b), each splat is constrained by one edge segment. The splat clipping technique [26] is extended by computing the image plane positions for the edge segment in the vertex shader and

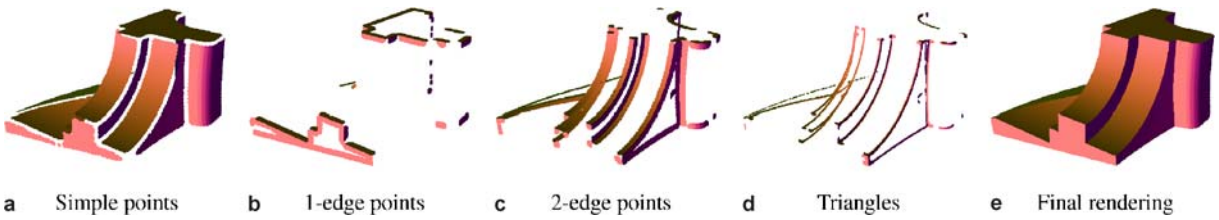

Fig. 9. GPU-based edge-preserving splatting for the Fandisk model

determining the clipping on the image plane in the fragment shader. For the 2-edge points (Fig. 7c,d), each splat is associated with two edge segments. Convexness of the edge segments should be determined in the vertex shader for correct clipping. If it is convex relative to the splat center (Fig. 7c), the fragment is discarded if it is clipped by either of the edge segments. If it is concave (Fig. 7d), the fragment is discarded if it is clipped by both of the edge segments. Since the edge segments may not reside on the splat plane, this convexness is view-dependent and should be determined on the image plane. For example, in Fig. 8, the edge segments can be either convex or concave from different viewpoints.

For the complex points associated with more than two edge segments, each splat is segmented into one 2-edge point and a connected group of triangles (triangle fan). In Fig. 7e, the complex point p_i is associated with four line segments $v_1v_2v_3v_4v_5$. It is segmented into a 2-edge point with $v_1p_iv_2$ as its clipping edge segments, and four connected triangles ($\Delta v_1p_iv_2$, $\Delta v_2p_iv_3$, $\Delta v_3p_iv_4$, $\Delta v_4p_iv_5$). Because of *Constraint 3*, the invalid case in Fig. 7h is impossible. These triangles may not totally reside inside the splats, such as $\Delta v_1p_iv_2$ in Fig. 7e, $\Delta v_3p_iv_4$ in Fig. 7g, etc. In the rendering of these triangle fans, the ray-splat plane intersection computation is also needed to discard the fragments outside the splat radius.

Different vertex/fragment shaders are designed for simple points, 1-edge points, 2-edge points, and the triangles for efficient rendering. Figure 9 shows the rendering of the simple points, 1-edge points, 2-edge points, triangles and the final image of the Fandisk model.

6 Results

We have implemented our edge-preserving simplification algorithm and the GPU-based constrained splatting method for the point-and-edge model. The tests have been conducted on a Xeon 2.40 GHz PC with a GeForce 6800 Ultra graphics card. The vertex/fragment shaders are written in the Cg language.

The running time for our simplification method is related to the number of surface points, the cluster size, type 2 error tolerance, and the geometry of the model. For the test models used in this paper, the simplification times range from 2 to 58 seconds. For example, the simplification times for the Fandisk model in Fig. 6 are 13, 19, and 29 seconds for Fig. 6b–d, respectively.

Figure 10 compares the rendering of the Fandisk model at different surface point densities using point-and-edge rendering (edge-preserving splatting) and point-sampled rendering (unconstrained splatting). Table 1 summarizes the corresponding rendering times and memory sizes. We compute the rendering performance by sending all the point primitives in the model to the rendering pipeline without any culling under an image size of 800×800. The rendering speed is affected by the number of surface points, the projected splat size on the image plane, and the percentage of surface points near the edges.

Figures 10g to 10k are the point-and-edge renderings of the Fandisk model at different resolutions, ranging from 53 794 to 1529 surface points. Sharp edges are preserved regardless of the surface point density. Figure 10b is the unconstrained splatting using the 13 645 surface points from the model of 10h. There are noticeable artifacts near the sharp edges. After upsampling the surface points near the edges, resulting in 39 910 surface points, these artifacts are eliminated and sharp edges are preserved (Fig. 10d). However, when the viewpoint moves much closer to the surface of the model, the edges become blurred as shown in Fig. 10e. Figure 10f is the same closeup rendering of the point-and-edge model with only 1529 points, whose sharp edges are preserved regardless of the viewpoint position.

Table 1. Rendering time and memory size for the Fandisk model at different resolutions with image size 800×800. PAE denotes our point-and-edge model, and PM for point-sampled model

Model type	Surface point number	Memory size (byte)	Rendering time (ms)
PM	215 149	6 884 768	16.8
PM	39 910	1 277 120	7.8
PM	13 645	436 256	6.2
PAE	215 149	7 208 456	17.5
PAE	53 794	1 900 772	9.3
PAE	13 645	531 764	8.1
PAE	3826	162 404	8.4
PAE	2248	103 652	9.8
PAE	1529	72 160	9.6

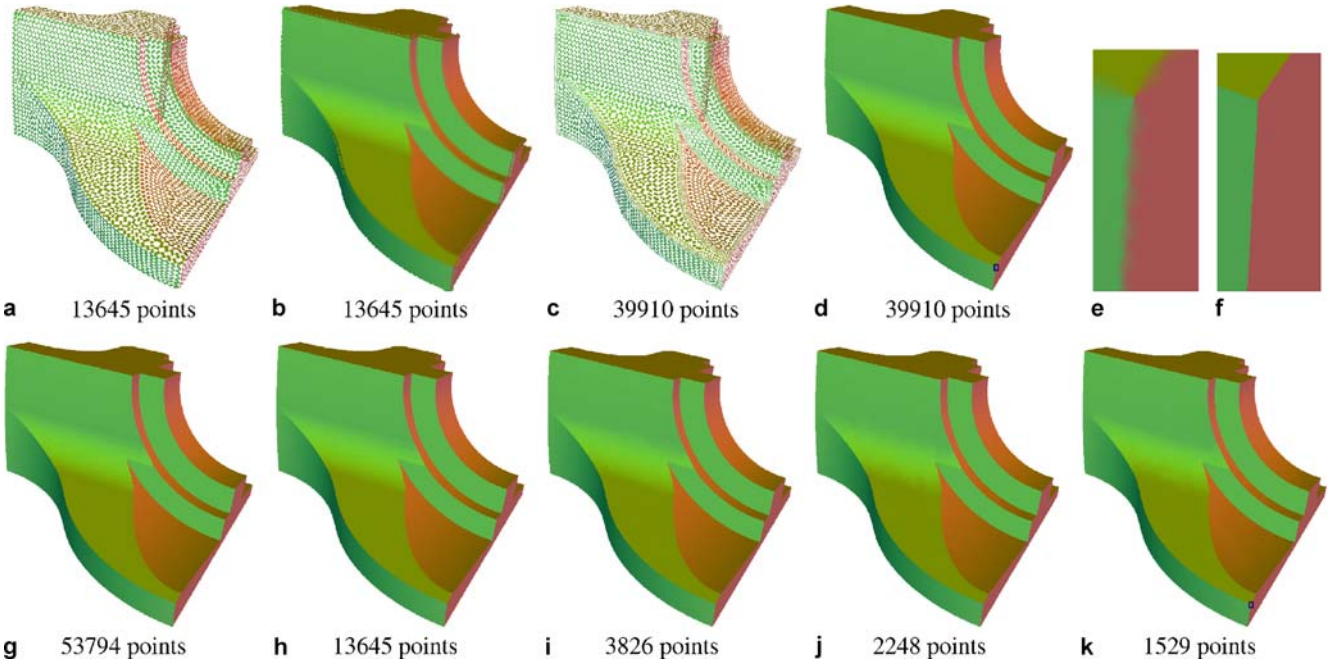

Fig. 10a–k. Rendering of the Fandisk model. Parts **a** and **c** are the point distribution; **b** and **d** are the corresponding unconstrained splatting. The model of **d** is generated by upsampling the points near the edges in the model of **b**. Parts **g**–**k** are the point-and-edge renderings of the model at different point densities. Parts **e** and **f** show two corresponding closeup renderings of a small region at a corner of the model from **d** and **k**, respectively, when the viewpoint is near the surface

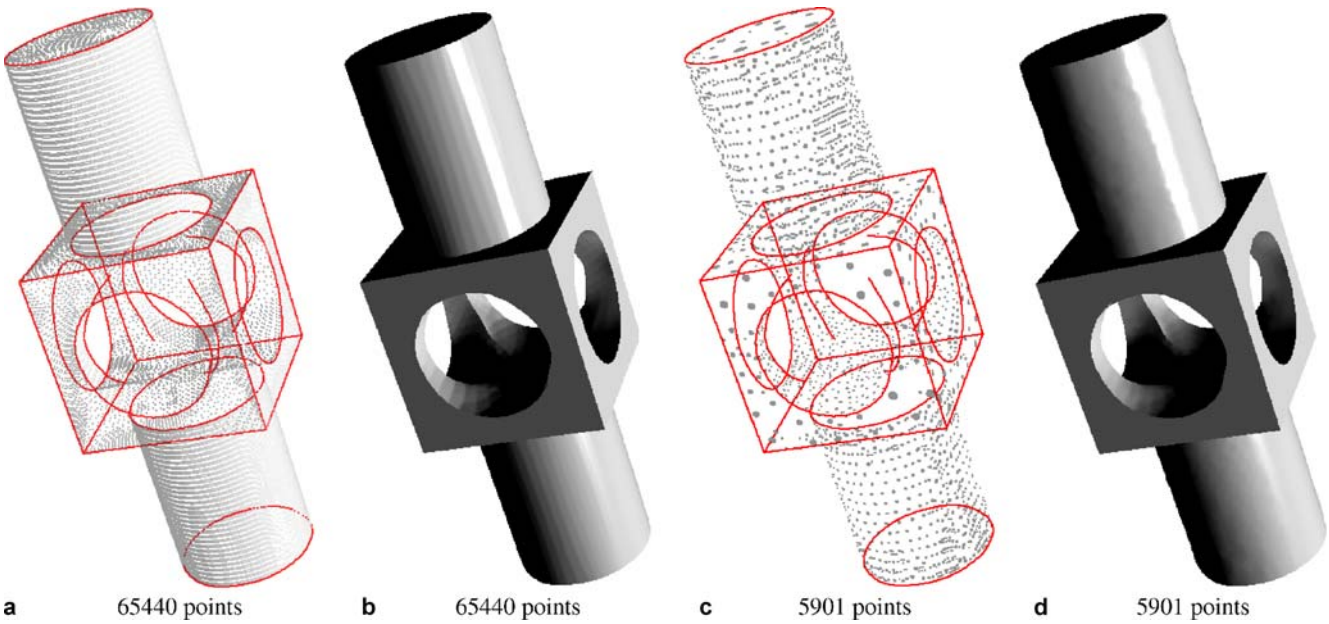

Fig. 11a–d. Machine Part model. Parts **a** and **c** show the point distribution (*grey dots*) and the sharp edges (*red lines*) with 65 440 and 5901 surface points, respectively. Parts **b** and **d** are the corresponding rendering

From Table 1, we can see that the point-and-edge rendering is slightly slower than the point-sampled rendering. This is due to the extra cost for computing the edge image space position and edge clipping, and switching between different shaders. For the model with 215 149 points, the rendering time for the point-and-edge model is 17.5 ms, or 12.3 M point/second, while for the point-sampled model, it is 16.8 ms, or 12.8 M point/second. When the point

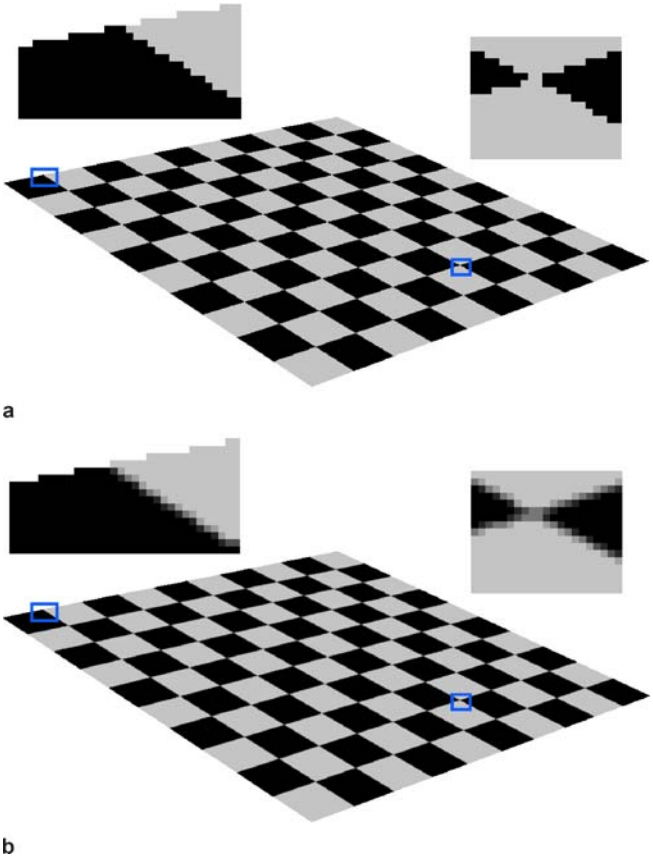

Fig. 12a,b. Rendering of a checkboard model. The closeup views of the regions inside the blue boxes are shown nearby. **a** Without edge anti-aliasing. **b** With edge anti-aliasing

density is high, the rendering time decreases with the surface point densities in the point-and-edge model. When the point density decreases under a certain value, the rendering time remains roughly the same or even increases a bit when the point density continues decreasing. This is because the projected splat size is large enough that the total number of the generated fragments is roughly the same regardless of the number of the splat primitives. The fragment shader becomes a bottleneck in the rendering pipeline. The reduction in the memory size is significant for the lower resolution model. For example, the model with 1529 points uses only 14% of the memory for the model with 13 645 points, without significant difference in the rendering time and quality.

Figure 11 shows the rendering of a Machine Part model. Figure 11a is the model with 65 440 surface points and 24 sharp edges, and Fig. 11b is the corresponding rendering. This model is simplified in 58 seconds into 5901 surface points (Fig. 11c,d) using our simplification method. The rendering times are 13.9 ms and 7.9 ms for Figs. 11b and 11d, respectively.

The sharp edges are preserved in the rendering of our point-and-edge models, but these edges may be aliased

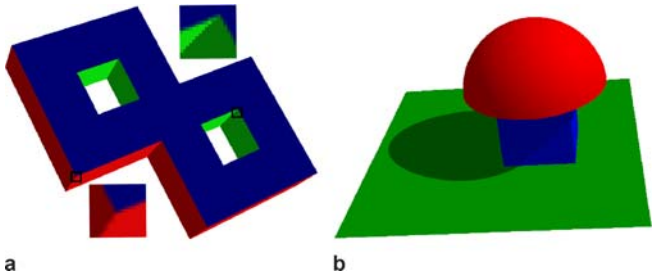

Fig. 13a,b. Point-and-edge rendering. **a** With edge anti-aliasing for the edges between two visible surfaces (the closeup views of the regions inside the black boxes are shown nearby). **b** With shadow mapping

in the final image such as in the checkboard shown in Fig. 12a. This is because we use the center of the pixel to determine the clipping in the fragment shaders. For those pixels that the edge goes through, the contribution comes from only one side of the edge. If the surface point density on the two sides of the edges are relatively the same, for the purpose of edge anti-aliasing, we can revise the fragment shaders and use the four corner of the pixel to determine the clipping. Assuming the clipping results from the four corners are s_1, s_2, s_3, s_4, with $s_j = 0$ for clipping and $s_j = 1$ otherwise, this pixel is on the edge if s_1, s_2, s_3, s_4 do not have the same value. We do not discard this fragment, but change the weight of this fragment by multiplying it by a coverage weight w_e:

$$w_e = (s_1 + s_2 + s_3 + s_4)/4 \tag{9}$$

From Fig. 12b, we can see that this only works for the edges between two visible surfaces. For an edge as a boundary or silhouette in the final image, only the points on one side of the edge make the contribution, resulting in no effect of this coverage weight after the normalization in the normalization/shading pass.

Figure 13a is another example using edge anti-aliasing. Figure 13b shows the rendering of a mushroom model with shadow mapping. Three frame buffers are used to store the accumulated surface position, normal, and color, and an additional pass is used to render the depth from the light source direction. In the shading pass, this information is used to get the final rendering with shadow.

7 Conclusions and future work

This paper introduces a point-and-edge model to represent models with sharp edges. An edge-preserving simplification method is presented in order to create the model hierarchy. We can generate a simplified model with bounded error using a feedback algorithm. An efficient constrained splatting algorithm is used for the direct rendering of the model, which utilizes the advanced features of modern

GPUs. This representation is intended to model objects with sharp edges so that relatively low point density is required while preserving the edges in the rendering. This is a hybrid model, which capitalizes on the advantages of both the unstructured point-sampled model and the structured mesh model.

Besides the geometric sharp edges, our point-and-edge model can also be used for texture discontinuity, such as the checkboard (Fig. 12). Instead of upsampling of surface points near the texture discontinuity, edges can be added to preserve the discontinuity.

There is room to improve our simplification method. Because the seed for clustering is randomly selected, the surface points on the simplified model are not optimally distributed. We will also explore other methods for edge anti-aliasing for the constrained splatting. One possible solution is to find and smooth the visible edges in the final image with the knowledge of the edge information.

References

1. Adams, B., Dutré, P.: Interactive boolean operations on surfel-bounded solids. In: SIGGRAPH '03, pp. 651–656 (2003)
2. Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C.T.: Point set surfaces. In: IEEE Visualization, pp. 21–28 (2001)
3. Bala, K., Walter, B., Greenberg, D.P.: Combining edges and points for interactive high-quality rendering. In: SIGGRAPH '03, pp. 631–640 (2003)
4. Botsch, M., Hornung, A., Zwicker, M., Kobbelt, L.: High-quality surface splatting on today's GPUs. In: Symposium on Point-Based Graphics, pp. 17–24 (2005)
5. Botsch, M., Kobbelt, L.: High-quality point-based rendering on modern GPUs. In: Pacific Graphics, pp. 335–442 (2003)
6. Botsch, M., Spernat, M., Kobbelt, L.: Phong splatting. In: Symposium on Point-Based Graphics, pp. 25–32 (2004)
7. Botsch, M., Wiratanaya, A., Kobbelt, L.: Efficient high quality rendering of point sampled geometry. In: Eurographics Workshop on Rendering, pp. 53–64 (2002)
8. Cohen-Steiner, D., Alliez, P., Desbrun, M.: Variational shape approximation. In: SIGGRAPH '04, pp. 905–914 (2004)
9. Fleishman, S., Cohen-Or, D., Alexa, M., Silva, C.T.: Progressive point set surfaces. ACM Trans. Graph. **22**(4), 997–1011 (2003)
10. Fleishman, S., Cohen-Or, D., Silva, C.T.: Robust moving least-squares fitting with sharp features. In: SIGGRAPH '05, pp. 544–552 (2005)
11. Garland, M., Willmott, A., Heckbert, P.: Hierarchical face clustering on polygonal surfaces. In: ACM Symposium on Interactive 3D Graphics, pp. 49–58 (2001)
12. Gumhold, S., Wang, X., Macleod, R.: Feature extraction from point clouds. In: 10th International Meshing Roundtable, pp. 293–305 (2001)
13. Hubeli, A., Gross, M.: Multiresolution feature extraction from unstructured meshes. In: IEEE Visualization, pp. 287–294 (2001)
14. Kobbelt, L.P., Botsch, M., Schwanecke, U., Seidel, H.: Feature sensitive surface extraction from volume data. In: SIGGRAPH '01, pp. 57–66 (2001)
15. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., Seidel, H.P.: Multi-level partition of unity implicits. In: SIGGRAPH '03, pp. 463–470 (2003)
16. Pauly, M., Gross, M.: Efficient simplification of point-sampled surfaces. In: IEEE Visualization, pp. 163–170 (2002)
17. Pauly, M., Keiser, R., Gross, M.: Multi-scale feature extraction on point-sampled surfaces. In: Eurographics, pp. 281–289 (2003)
18. Pfister, H., Zwicker, M., v. Baar, J., Gross, M.: Surfels: Surface elements as rendering primitives. In: SIGGRAPH 2000, pp. 335–342 (2000)
19. Ren, L., Pfister, H., Zwicker, M.: Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In: Eurographics, pp. 461–470 (2002)
20. Rusinkiewicz, S., Levoy, M.: QSplat: A multiresolution point rendering system for large meshes. In: SIGGRAPH 2000, pp. 343–352 (2000)
21. Talton, J.O., Carr, N.A., Hart, J.C.: Voronoi rasterization of sparse point sets. In: Symposium on Point-based Graphics, pp. 33–38 (2005)
22. Wicke, M., Teschner, M., Gross, M.H.: CSG tree rendering for point-sampled objects. In: Pacific Graphics, pp. 160–168 (2004)
23. Wu, J., Kobbelt, L.: Optimized sub-sampling of point sets for surface splatting. In: Eurographics, pp. 643–652 (2004)
24. Wu, J., Zhang, Z., Kobbelt, L.: Progressive splatting. In: Symposium on Point-based Graphics, pp. 25–32 (2005)
25. Zwicker, M., Pfister, H., van Baar, J., Gross, M.: Surface splatting. In: SIGGRAPH '01, pp. 371–378 (2001)
26. Zwicker, M., Rásánen, J., Botsch, M., Dachsbacher, C., Pauly, M.: Perspective accurate splatting. In: Graphics Interface, pp. 247–254 (2004)

HAITAO ZHANG was a PhD student in the computer science department at Stony Brook University (SUNY). His research interests are computer graphics and visualization, including 3D model acquisition, modeling and rendering. He received a BS (1997) in computer science and engineering from Zhejiang University, China, a MS (2000) in pattern recognition and intelligent systems from the Institute of Automation, Chinese Academy of Sciences, and a PhD (2006) from Stony Brook University.

ARIE E. KAUFMAN is a Distinguished Professor and Chair of the Computer Science Department and the Director of the Center for Visual Computing (CVC) at Stony Brook University (SUNY). He was the founding Editor-in-Chief of the *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 1995–1998. Kaufman has been the (co-)Chair for multiple *Graphics Hardware Workshops*, *Volume Graphics Workshops*, *Volume Visualization Symposium*, and the *Visualization Conferences*, and the co-founder and member of the steering committee of the *Visualization Conference* series. He has previously chaired and is currently a director of the IEEE Computer Society Technical Committee on Visualization and Computer Graphics. He is an IEEE Fellow and the recipient of IEEE Outstanding Contribution Award (1995), ACM Service Award (1998), IEEE Computer Society's Meritorious Service Award (1999), elected member of European Academy of Sciences (2002), State of New York Entrepreneur Award (2002), IEEE Harold Wheeler Award (2004), State of New York Innovative Research Award (2005), and IEEE Visualization Career Award (2005). Kaufman has conducted research for over 30 years in volume visualization; graphics architectures, algorithms, and languages; virtual reality; user interfaces; multimedia; and biomedical applications. He received a BS (1969) in Mathematics and Physics from Hebrew University of Jerusalem, Israel, an MS (1973) in Computer Science from Weizmann Institute of Science, Rehovot, Israel, and a PhD (1977) in Computer Science from Ben-Gurion University, Israel.