Available online at www.sciencedirect.com
ScienceDirect
Computers & Graphics 31 (2007) 175–189
COMPUTERS & GRAPHICS
www.elsevier.com/locate/cag

Deferred blending: Image composition for single-pass point rendering

Yanci Zhang, Renato Pajarola*

Visualization and MultiMedia Lab, Department of Informatics, University of Zürich, Switzerland

Abstract

In this paper, we propose novel GPU accelerated algorithms for interactive point-based rendering (PBR) and high-quality shading of transparent point surfaces. By introducing the concept of *deferred blending* we are able to formulate the smooth point interpolation problem as an image compositing post-processing task. Consequently, our new PBR algorithm does not suffer from an extra visibility-splatting pre-render pass, for conservative ε–z-buffer visibility culling, as this is eventually performed together with the smooth point interpolation during image compositing. Moreover, this new *deferred blending* concept enables hardware accelerated transparent PBR with combined effects of multi-layer transparency, refraction, specular reflection, and per-fragment shading. *Deferred blending* is based on a separation of the point data into not self-overlapping minimal independent groups, a multi-target rendering pass and an image compositing post-processing stage. We present different grouping algorithms for off-line and on-line processing. For basic opaque surface rendering and simple transparency effects, our novel algorithm only needs a single geometry rendering pass. For high-quality transparent image synthesis one extra rendering pass is sufficient. Besides transparency, per-fragment reflective and refractive multi-layer effects (e.g. environment mapping) are supported in our algorithm.
© 2006 Elsevier Ltd. All rights reserved.

Keywords: Point based rendering; Transparency; Alpha blending; Hardware acceleration; GPU processing

1. Introduction

Point-based rendering (PBR) has attracted growing interest in the last few years as points as geometric modeling and rendering primitives have shown to be an interesting alternative to triangle meshes [1–4]. Points are the basic geometry defining elements of 3D objects and surfaces. Moreover, most geometric modeling tasks can be performed directly on point sets as demonstrated in [5–7].

As the significance and adoption of point-based object representation increases [1–4], the illumination and shading feature set has to be fully developed. Basic PBR on the GPU has received much attention with many algorithms for real-time rendering of opaque surfaces. Also smooth shading of points has been addressed (see also Section 2). However, most GPU-based PBR algorithms [3,4,8] generally suffer from $2+1$ rendering passes; two passes over the geometry and one image processing pass. To achieve smooth interpolation and resolve correct visibility of overlapping point splats, a separate *visibility-splatting* rendering pass is needed. In a second *point-blending* rendering pass the smooth interpolation between visible overlapping points is performed. This separation into visibility splatting and blending is a fundamental problem of PBR.

Moreover, GPU-based interactive rendering of transparent point has been a daunting task. This is mainly due to the difficulty of integrating the following two different blending operations simultaneously on the GPU:

1. *Transparency-blending* is used to α-composite transparent surface layers in a back-to-front order to generate the effect of transparency. For this the z-buffer must be turned off to include all fragments from all transparent layers.
2. *PBR-blending* is used to interpolate between overlapping point splats within the same surface layer to achieve smooth rendering results. To interpolate between overlapping splats in one layer, the z-buffer is turned on to cull fragments farther than some ε in depth from the visible surface, and pass all others.

*Corresponding author. Tel.: +41 44 635 4370; fax: +41 44 6356809.
E-mail address: pajarola@ifi.unizh.ch (R. Pajarola).

0097-8493/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2006.11.012

Our solution to the above problems is the separation of the input point data into multiple not self-overlapping minimal independent groups, and formulating the PBR-blending as an image compositing post-processing task.

The contributions of this paper include the definition of the new *deferred blending* concept and GPU-based PBR framework, which implements the ε–z-buffer visibility culling in the image compositing stage, and algorithms for point grouping as required to perform *deferred blending*. This paper is an extended version of [9,10] and includes the following major additions:

- weighted-graph coloring based off-line point grouping for high-quality transparent shading,
- dynamic on-line point grouping algorithm for deformable objects,
- experimental results of different grouping algorithms,
- in-depth comparison to depth-peeling and [11].

The paper is organized as follows: Section 2 briefly reviews related work and in Section 3 we review the basic *deferred blending* concept. Section 4 describes off-line as well as on-line point grouping algorithms. The *deferred blending* rendering algorithms for opaque and transparent point objects are given in Section 5. Following the experimental results in Section 6, Section 7 concludes the paper.

2. Related work

Splatting-based techniques are the most widely adopted approach in PBR because of the good tradeoff between performance and rendering quality. Most of the PBR algorithms are derived in some way from a software algorithm for surface splatting [12,13].

Hardware accelerated point rendering techniques for high-quality shading include antialiasing filters [14–16], point-splat normal fields [17] and per-fragment smooth shading [16–18]. Also the combination of point and triangle primitives have been proposed [19–22] to improve rendering quality and performance.

A common feature of GPU-accelerated PBR algorithms [3,4,8] is a $2+1$-pass rendering concept: two passes over the geometry data for visibility and smooth blending between points, and one image pass for normalization (and shading) of smoothly blended point attributes. As splatting primitive for rasterization, either quads [14] and triangles [14,23] with blending textures or depth-corrected point-sprites [24] are typically employed.

However, the two rendering passes over the point geometry data in GPU-accelerated PBR algorithms are highly undesirable. The two passes are expensive iterations over the point geometry data not only due to the transform and lighting cost, but also in particular due to the complex vertex and fragment shaders required to rasterize depth-corrected elliptical splats in image-space [15,16,18].

With respect to PBR of transparency, only a software algorithm has been proposed to date [13]. It uses a software frame buffer with multiple depth layers per fragment. Unfortunately, this solution cannot be mapped onto GPUs as they do not support multiple depths per fragment nor the simultaneous read and write of the target buffer as required by this solution.

In principle, depth-peeling [25,26] can be applied to PBR of transparent surfaces [11]. Its idea is to render the k-nearest layers in k geometry passes to different target α-images and then α-blend these images together back-to-front. However, as it requires several iterations over the geometry—each itself a multi-pass PBR algorithm—it is impractical for interactive PBR.

3. Visibility splatting

In this section we review and establish the basic framework for *deferred blending* as it has been introduced in [9,10].

3.1. Smooth point interpolation

A point set $\mathcal{S}$ covers a 3D surface by a set of overlapping point splats $s_{0\ldots n-1}$. The projection of $\mathcal{S}$ in image-space must interpolate for each fragment f the contribution of multiple overlapping splats s_i as illustrated in Fig. 1. For smooth interpolation, the contribution of each splat s_i to the fragment f depends on the distance $|\mathbf{f}_i - \mathbf{p}_i|$ of its projection $\mathbf{f}_i$ onto the plane of splat s_i in object-space, where p_i is the center of splat s_i.

The fragment color $\mathbf{c}(f)$ is eventually computed from all overlapping splats s_i as the weighted sum of colors

$$\mathbf{c}(f) = \frac{\sum_i w_i(\mathbf{f}_i) \cdot \mathbf{c}_i}{\sum_i w_i(\mathbf{f}_i)}, \tag{1}$$

where w_i defines a smooth blending kernel centered on $\mathbf{p}_i$. For the remainder we will limit ourselves to circular disks, but elliptical splats can be handled analogously.

To avoid contribution from occluded splats s_j to Eq. (1), an ε–z-buffer visibility test [14,15,18,23,24] is used as illustrated in Fig. 2. It discards any fragments from occluded splats s_j farther back than ε from the nearest visible splat s_i.

Since GPUs do not offer such a weak visibility z-test, hardware accelerated implementations of Eq. (1) resort to a $2+1$-pass rendering algorithm. First, all point samples in

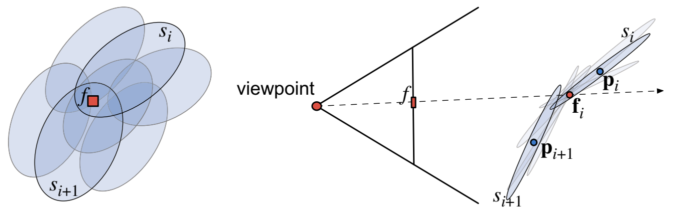


Fig. 1. Each fragment f receives contributions from multiple overlapping splats s_i which are smoothly interpolated.

$\mathcal{S}$ are rendered at ε-offset to initialize the depth-buffer of the point surface $\mathcal{S}$. Second, with lighting and α-blending enabled but z-buffer writing disabled, the terms $\sum_i w_i(\mathbf{f}_i) \cdot \mathbf{c}_i$ and $\sum_i w_i(\mathbf{f}_i)$ of Eq. (1) are accumulated into color $\mathbf{c}_{\text{rgb}}(f)$ and α $\mathbf{c}_\alpha(f)$ channels for each fragment f, respectively. The ε-offset in the first, together with disabled z-buffer writing in the second pass, achieves the desired ε–z-visibility. In an image normalization post-processing pass, the final fragment color $\mathbf{c}_{\text{rgb}}(f)/\mathbf{c}_\alpha(f)$ is generated as indicated by Eq. (1).

3.2. Deferred blending

To avoid multiple passes over the point geometry data we introduce a *deferred blending* concept that delays the ε–z-buffer visibility test as well as smooth point interpolation according to Eq. (1) to an image post-processing pass.

As illustrated in Fig. 3, we note that if a point set $\mathcal{S}$ is sufficiently split into multiple groups $\mathcal{S}_k$, with $\mathcal{S} = \bigcup_k \mathcal{S}_k$, overlapping splats in image-space can be avoided. Let us focus for now on the splats of $\mathcal{S}$ which are part of the nearest visible surface layer and ignore all others. Assuming non-overlapping point groups $\mathcal{S}_k$, the accumulation in Eq. (1) can be separated into summations over the individual groups as follows:

$$\mathbf{c}(f) = \frac{\sum_{s_i \in \mathcal{S}} w_i(\mathbf{f}_i) \cdot \mathbf{c}_i}{\sum_{s_i \in \mathcal{S}} w_i(\mathbf{f}_i)} = \frac{\sum_k \sum_{s_i \in \mathcal{S}_k} w_i(\mathbf{f}_i) \cdot \mathbf{c}_i}{\sum_k \sum_{s_i \in \mathcal{S}_k} w_i(\mathbf{f}_i)}. \tag{2}$$

Based on Eq. (2), for each group $\mathcal{S}_k$ we can form a partial image I_k with fragment colors $\mathbf{c}_{\text{rgb}}(f)_k = \sum_{s_i \in \mathcal{S}_k} w_i(\mathbf{f}_i) \cdot \mathbf{c}_i$ and fragment weights $\mathbf{c}_\alpha(f)_k = \sum_{s_i \in \mathcal{S}_k} w_i(\mathbf{f}_i)$. The final image can then be formed by an image compositing step over all partial images I_k,

$$\mathbf{c}_{\text{rgb}}(f) = \frac{\sum_k \mathbf{c}_{\text{rgb}}(f)_k}{\sum_k \mathbf{c}_\alpha(f)_k}. \tag{3}$$

Moreover, as there is no overlap in image-space between splats of one group $\mathcal{S}_k$, the fragment color and weight of I_k can in fact simply be set to

$$\mathbf{c}_{\text{rgb}}(f)_k = w_i(\mathbf{f}_i) \cdot \mathbf{c}_i \quad \text{and} \quad \mathbf{c}_\alpha(f)_k = w_i(\mathbf{f}_i), \tag{4}$$

for the only visible splat $s_i \in \mathcal{S}_k$ that covers the fragment f. Therefore, no α-blending nor ε–z-buffer visibility culling is required to generate the image I_k of a single point group $\mathcal{S}_k$.

If $\mathcal{S}_k$ only contains splats s_i of the nearest visible layer not overlapping in image-space, then Eq. (4) indicates that splats $s_i \in \mathcal{S}_k$ simply have to be rasterized into image I_k. A single rendering pass over $\mathcal{S}_k$ can write the per-fragment weighted color and weight itself into the RGB α-channels. For all groups this amounts to one full data traversal since $\mathcal{S} = \bigcup_k \mathcal{S}_k$. Post-process image composition and normalization of all I_k according to Eq. (3) yields the final smooth point interpolation.

In practice, however, a group $\mathcal{S}_k$ contains not only points from the nearest visible surface layer. On the other hand, if all splats $s_{i,j} \in \mathcal{S}_k$ have no overlap in object-space, that is $|\mathbf{p}_i - \mathbf{p}_j| \geqslant r_i + r_j$, where r_i is the radius of the splat s_i, and $|\mathbf{p}_i - \mathbf{p}_j|$ is the Euclidean distance between splats s_i and s_j, then simple z-buffer visibility determination guarantees that all visible fragments from splats s_i in the nearest surface layer of $\mathcal{S}_k$ are included in the image I_k as shown in Fig. 4. Additionally, fragments from splats $s_j \in \mathcal{S}_k$, but occluded by $\mathcal{S} \backslash \mathcal{S}_k$, may also occur in I_k. However, the complementary images $I_{l \neq k}$ will contain the data of the visible splat fragments to perform ε−z-buffer visibility culling as will be described below. In addition to the fragment color, the images I_k include per-fragment depth information $\mathbf{c}_d(f)_k$ as well.

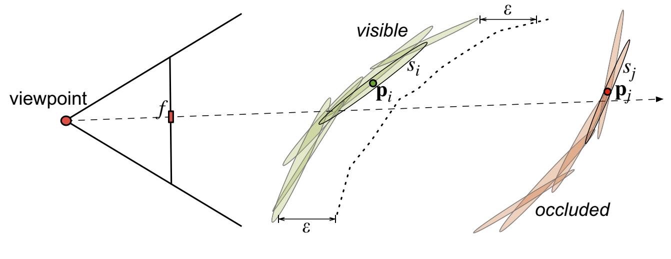


Fig. 2. Each fragment f receives contributions from multiple overlapping splats s_i which are smoothly interpolated.

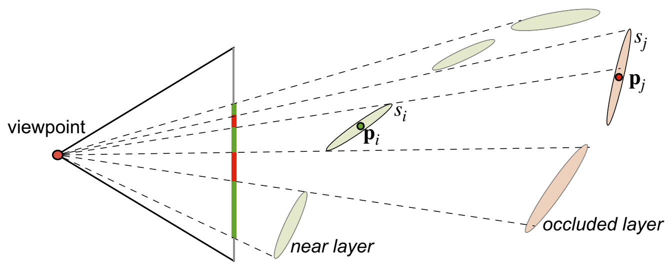


Fig. 4. For each point group $\mathcal{S}_k$, any fragments generated by splats s_i from the nearest visible surface layer will win the z-buffer visibility determination over any occluded splats s_j and will be kept in the image I_k.

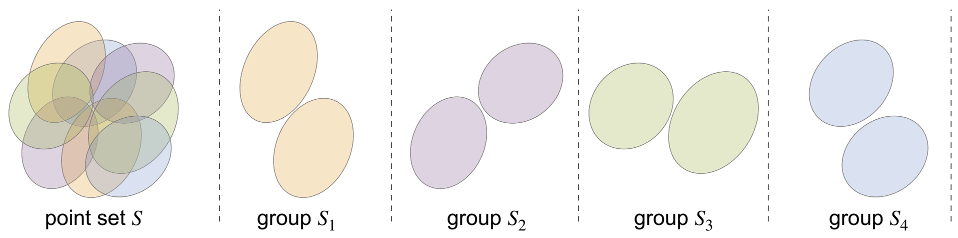


Fig. 3. Separation of the input point set $\mathcal{S}$ into non-overlapping sub-sets $\mathcal{S}_k$.

The color-and-depth images I_k of all groups $\mathscr{S}_k$ can then be combined, as suggested in Fig. 5, using the depth information to perform the $\varepsilon - z$-buffer visibility culling as outlined in the previous section. We can now outline the image compositing operation $\oplus$ over all K depth-images I_k to compute Eq. (3) under the $\varepsilon - z$-visibility constraint (given in Fig. 6).

The conservative ε–z-buffer visibility test is implemented in Fig. 6 by line 4 and the if statement on line 6. Due to the weighted color as from Eq. (4), lines 7 and 8 implement the summation, while line 11 performs the division of Eq. (3). Therefore, unlike in prior methods, ε–z-buffering, smooth point interpolation as well as color normalization are all formulated as an image compositing post-process.

Additional features such as deferred shading [13,16,18] or Voronoi rasterization [27] are integrated into the basic approach outlined here, see also Section 5.

3.3. Transparent points

As mentioned in the introduction, the main difficulty of rendering transparent points is the conflict of z-buffer usage. The introduced concept of *deferred blending* can be extended to solve this problem by separating the two blending operations into separate rendering passes. As illustrated in Fig. 7(a), transparency blending between

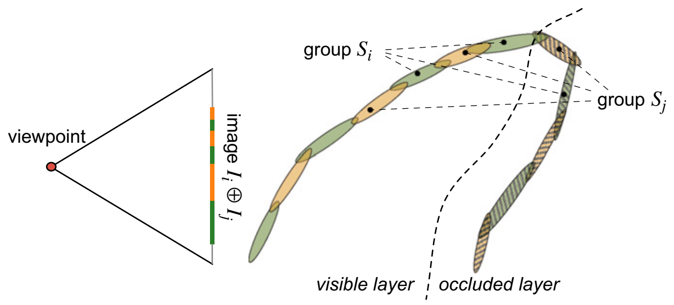


Fig. 5. Contributions from multiple depth-images I_k can be visibility culled and blended into the final result $I = \bigoplus_k I_k$, taking the z-depth and ε tolerance into account.

```
$I = \bigoplus_{k=0}^{K-1} I_k$:

1  foreach $f \in I$ do
2     $\mathbf{c}_{\mathrm{rgb}}(f) = \mathbf{0}$;
3     $\mathbf{c}_{\alpha}(f) = 0$;
4     $d = \min_k(\mathbf{c}_d(f)_k)$;
5     for $k = 0$ to $K-1$ do
6        if $\mathbf{c}_d(f)_k \le d + \varepsilon$ then
7           $\mathbf{c}_{\mathrm{rgb}}(f) = \mathbf{c}_{\mathrm{rgb}}(f) + \mathbf{c}_{\mathrm{rgb}}(f)_k$;
8           $\mathbf{c}_{\alpha}(f) = \mathbf{c}_{\alpha}(f) + \mathbf{c}_{\alpha}(f)_k$;
9        endif
10    endfor
11    $\mathbf{c}_{\mathrm{rgb}}(f) = \frac{\mathbf{c}_{\mathrm{rgb}}(f)}{\mathbf{c}_{\alpha}(f)}$;
12 endforeach
```

Fig. 6. Post-process image compositing performing smooth point interpolation as well as ε–z-visibility testing.

surface layers and smooth point interpolation within a surface layer cannot be told apart while performing back-to-front α-blending of fragments. Our solution is illustrated in Fig. 7(b) where the competing splats overlapping within a layer are separated into different groups A and B. Rendering group A into one target image I_A, using per-fragment material opaqueness α, yields the resulting fragment color $\alpha_2 \cdot a_2 + (1-\alpha_2)(\alpha_1 \cdot a_1 + (1-\alpha_1) \cdot$ *background*). The same proper back-to-front transparency α-blending is accomplished in image I_B for group B. Finally, smooth point interpolation is achieved by averaging the two results into the final image $I = \frac{1}{2} \cdot (I_A + I_B)$.

Note that smooth blending kernels cannot be supported that way as the interpolation weights interfere with the transparent α-blending. Hence, each fragment contributes equally to the final point interpolation. However, the visual artifacts introduced by this are largely suppressed due to the following two observations: (1) Artifacts are reduced dramatically by multiple transparent surface layers. (2) With current 8-bit color and α resolutions any errors below a value of $\frac{1}{256}$ have no effect. Moreover, the artifacts can be made virtually unnoticeable by considering the *nearest transparent layer* and render it separately in high quality using smooth point blending kernels.

Furthermore, we observe that the above concept works well if points within a group have no or only minimal overlap, but each group must cover the object's surface such that no holes exist within a transparent layer. These aspects are addressed by an extended grouping algorithm discussed in the following sections.

4. Grouping algorithm

The division of $\mathscr{S}$ into K groups $\mathscr{S}_{k=0\ldots K-1}$ as discussed above can be formulated as a *graph coloring* problem which is conducted in a pre-process prior to rendering. Depending on the definition of graph, two different grouping algorithms are introduced. The first algorithm is called *minimal graph coloring* which only considers whether two

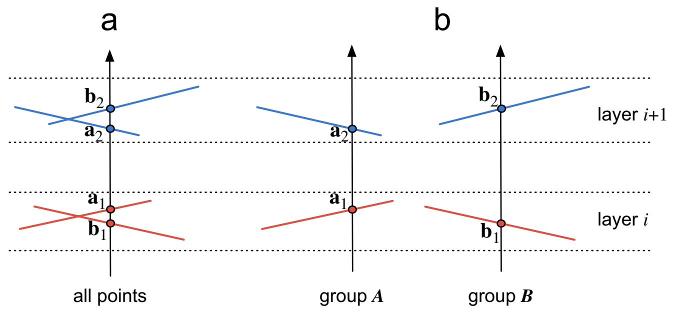


Fig. 7. (a) Traditional PBR cannot distinguish between point interpolation and transparency α-compositing during per-fragment blending; (b) dividing points into groups A and B: a_1, b_1 are transparency α-blended with a_2, b_2, respectively, and then PBR-interpolated in an image compositing post-process.

splats have overlaped or not, and the number of groups is unknown before grouping. The second algorithm is called *weighted graph coloring* which considers the amount of overlap between two splats, and the number of groups which is a user-defined parameter. Since these two grouping algorithms based on graph coloring are conducted in a pre-process stage, they cannot be applied for deformable point-based objects whose geometry may be changed dynamically. A third dynamic grouping algorithm, which offers better performance than the graph-coloring-based approach, is designed to handle grouping of dynamically changing points.

4.1. Minimal graph coloring

For *deferred blending* to work, it is sufficient that the point sets $\mathcal{S}_k$ must be independent groups in the sense that $\forall s_{i,j} \in \mathcal{S}_k$ it holds that $|\mathbf{p}_i - \mathbf{p}_j| \geqslant r_i + r_j$. Hence we can formulate a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V} = \{\mathbf{p}_i\}$ from all $s_i \in \mathcal{S}$ and edges

$$\mathcal{E} = \{e_{i,j} | |\mathbf{p}_i - \mathbf{p}_j| < r_i + r_j\}. \tag{5}$$

Other pairs of points need not define edges in $\mathcal{E}$ as they do not conflict in group assignment.

The required partitioning of $\mathcal{S}$ is thus defined as the solution to the *minimal graph coloring* (MGC) of $\mathcal{G}$ [28], and the number K of groups is $\mathcal{G}$'s *chromatic number* $\mathcal{X}(G)$. Since MGC is an NP-hard problem we apply an approximate solution as described below. Nevertheless, since $\mathcal{X}(G) \leqslant \Delta(\mathcal{G})$, the maximal degree of $\mathcal{G}$, we know an upper bound on K for any given point sample set $\mathcal{S}$.

We use the *largest first* (LF) graph coloring algorithm [29] to solve our point grouping problem. Given an ordered set of nodes $\mathcal{O} = [v_0, \ldots, v_{n-1}]$ $(v_i \in \mathcal{V})$ of the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ according to non-increasing degrees, assign color 0 to the first node v_0. If nodes $v_0, \ldots, v_i$ (with $i \geqslant 0$) have already received colors then v_{i+1} will be assigned the smallest color not yet assigned to any of its neighbors v_j (with $e_{i,j} \in \mathcal{E}$). Despite the fact that the LF algorithm is a simple algorithm to approximate the minimum graph coloring problem, it is very efficient and achieves almost the same results as the other more complex algorithms in the case of low edge-density.

Since each point group $\mathcal{S}_k$ is rendered to an individual target image I_k, which are later composited together, we prefer a small number K in practice. A smaller K means less memory overhead and fewer texture lookups during image compositing. Therefore, we apply the following modifications to the definition of edges $\mathcal{E}$ of graph $\mathcal{G}$ as given in Eq. (5) to reduce the number K of groups:

1. If two overlapping splats s_i and s_j are virtually co-planar, resulting in almost the same shading result, we exclude edge $e_{i,j}$ from $\mathcal{E}$. This allows to put s_i and s_j in the same group $\mathcal{S}_k$.
2. Ignore overlap condition in Eq. (5) if splat normals $\mathbf{n}_i$ and $\mathbf{n}_j$ point into opposite directions, that is if $\mathbf{n}_i \cdot \mathbf{n}_j < 0$.
3. Relax the overlap condition in Eq. (5) to $|\mathbf{p}_i - \mathbf{p}_j| < c \cdot (r_i + r_j)$, where $c \in [0, 1]$ is a user-defined parameter.

The side effect of the above modifications is that splats s_i and s_j in one group $\mathcal{S}_k$ may have a small overlap. However, for (1) as long as s_i and s_j are basically co-planar and have the same material color no rendering artifacts will result from this modification. Modification (2) allows points from different but close together surface layers to be in the same group which also causes no rendering artifacts. While (3) may introduce some rendering artifacts, these will be fairly small as the splats s_i and s_j will primarily overlap in the peripheral area of their disks which due to the smooth point blending kernels $w_{i,j}$ have less effect on the overall image generation. Furthermore, in the context of rendering opaque point surfaces, the artifacts caused by overlapping splats within the same group are further reduced by the Voronoi splat rasterization as described in Section 5.

4.2. Weighted graph coloring

As we mentioned above, a small K is preferred in practice which introduces some overlap between splats. In the MGC algorithm, the overlap is restricted by relaxing the overlap condition. Though it would be optimal if the overlap is minimized. Another drawback of MGC is that K is unknown before grouping and we need to write different shader programs for different K.

In order to address the above issues, we introduce another grouping algorithm called *weighted graph coloring* (WGC) which has the following two features: (1) the overlap between splats is minimized; (2) K is a user-defined parameter.

For the K-colors *WGC* problem, let us start with defining the weighted graph $G(\mathcal{S}, \mathcal{E})$ over the points $\mathcal{S} = \{\mathbf{p}_{1\ldots n}\}$ with edges $\mathcal{E} = \{e_{ij} | |\mathbf{p}_i - \mathbf{p}_j| < r_i + r_j\}$ and weights $\mathcal{W} = \{w_{ij} | \text{overlap between } \mathbf{p}_i \text{ and } \mathbf{p}_j\}$. The goal of K-colors WGC is to assign one of K colors to each node in G while minimizing the cost:

$$L(G, K) = \sum_{k=1}^{K} \sum_{\forall e_{ij} \in \mathcal{E}_k} w_{ij}, \tag{6}$$

where $G_k = (\mathcal{S}_k, \mathcal{E}_k)$ is a sub-graph of G of points $\mathcal{S}_k$ with color k and all edges $\mathcal{E}_k$ connecting points within $\mathcal{S}_k$.

The overlap weights $\mathcal{W}$ are given by

$$w_{ij} = (A_{ij} + A_{ji}) \cdot (1 - \mathbf{n}_i \cdot \mathbf{n}_j), \tag{7}$$

with normal vectors $\mathbf{n}_i$, and area A_{ij} of region $\mathcal{R}$ on splat $\mathbf{p}_i$ overlapped by $\mathbf{p}_j$ as shown in Fig. 8 given by

$$\mathcal{R} = \{\mathbf{x} | d_1 < \varepsilon \wedge d_2 < r\}, \tag{8}$$

where $d_1 = |(\mathbf{x} - \mathbf{p}_j) \cdot \mathbf{n}_j|$ and $d_2 = |(\mathbf{x} + d_1 \mathbf{n}_j) - p_j|$; ε is the constant used in the visibility-splatting pass.

We use a two-step greedy WGC strategy as follows:

1. *Initialization step*
   (a) Create a priority queue Q containing all *edges* $e_{ij} \in \mathscr{E}$ with priorities w_{ij}.
   (b) Process edges in Q in descending order. For edge e_{ij}, if vertex $\mathbf{p}_j$ has no color yet, call function *MinimizeOverlap* ($\mathbf{p}_j$) to assign one.
2. *Optimization step*
   (a) Create a priority queue Q_1 containing all *vertices* $\mathbf{p}_i \in \mathscr{S}$ in graph with the priority $\sum w_{ij}$, with points $\mathbf{p}_j$ having the same color as $\mathbf{p}_i$.
   (b) Process vertices in Q_1 in descending order. For vertex $\mathbf{p}_i$, call function *MinimizeOverlap(*$\mathbf{p}_i$*)* to assign new color to decrease cost function $L(G, K)$ of Eq. (6). If $\sum w_{ij}$ is below some user-defined threshold, remove $\mathbf{p}_i$ from Q_1, otherwise put it to Q_1 again with the new priority value.

Function *MinimizeOverlap(*$\mathbf{p}_i$*)* evaluates the best color k for $\mathbf{p}_i$, for which the sum $\sum A_{ij}$ of overlap between $\mathbf{p}_i$ and all $\mathbf{p}_j \in \mathscr{E}_k$, given by k, is smallest.

4.3. Dynamic grouping

One big drawback of graph-coloring-based grouping algorithms is that their performance is not fast enough to be applied on-line for dynamic point sets. In order to handle the rendering of dynamic points or deformable objects, a hash-based grouping algorithm is designed to achieve on-line performance and acceptable grouping results.

The basic idea of dynamic grouping is simple. A 2D hash table containing group indices is defined according to the number of groups defined by user. Then a 2D coordinate (x, y) calculated for each splat s_i based on its position in 3D space is used as an index into the hash table. The returned hash value is used as the group index for s_i.

In our hash-based grouping algorithm, we have to solve two basic problems: how to define the hash function for hash table lookup and how to handle the collisions. In the remainder of this section, we will show our strategy to solve the two problems.

For instance, suppose we want to divide all splats into 8 groups. A 2D hash table *hashTable*(8, 8) is shown in Fig. 9.

Our hash function is built in the following way: We use the average normal vector $\tilde{\mathbf{n}}$ of leaf node of the BSP-tree, which is used for efficient back-to-front traversal (Section 5.2). Given the largest dimension d of $\tilde{\mathbf{n}}$ we map s_i to the 2D coordinates

$$x_i = \left\lfloor \frac{s_i[(d+1)\%3]}{r_i} \right\rfloor, \quad y_i = \left\lfloor \frac{s_i[(d+2)\%3]}{r_i} \right\rfloor. \tag{9}$$

A hash function is then defined by $hash(s_i) = hashTable[x_i\%8][y_i\%8]$.

Note that the above algorithm does not handle collisions, which means that different splats may be hashed to the same group index. In fact, if two different splats s_i and s_j are mapped to the same hash entry, but $x_i \neq x_j$ or $y_i \neq y_j$, then the distance $|\mathbf{p}_i - \mathbf{p}_j|$ is large enough to guarantee that there is no overlap between them, hence it is safe to put them in the same group. Consequently, the only collision problem we have to deal with is if s_i and s_j satisfy $x_i = x_j$ and $y_i = y_j$. We adopt the following strategy to handle this collision:

1. Clear buffer Q.
2. Calculate (x_i, y_i) for each splat s_i. If (x_i, y_i) has not been tagged, s_i is put to group $hashTable[x_i\%8][y_i\%8]$ and (x_i, y_i) is tagged, otherwise s_i is pushed to Q.
3. For each splat s_i in Q, calculate its total overlap between its neighbors belonging to group k $(k = 0, 1, \ldots, 7)$ and assign it to group $k_{\min}$ with the minimal overlap value.

Even without dealing with the collision problem, the hash-based algorithm can generate fairly good grouping results for splats which are not very close to the eye point. Based on this fact, the performance of the hash-based grouping algorithm can be improved by only dealing with the collisions for the splats which are sufficiently close to the current eye point.

4.4. Extended grouping

The above grouping algorithms may not directly result in point groups suitable for transparent point rendering for the following two reasons, which will be addressed next:

1. *Too many fragments per pixel*: Despite overlap minimization, significant overlap may still exist within a

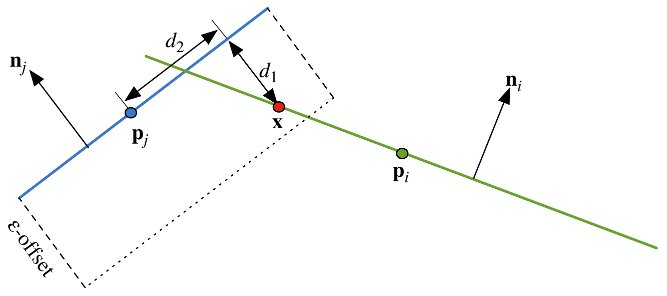


Fig. 8. Definition of overlap between two splats.

0	3	6	1	4	7	2	5
1	4	7	2	5	0	3	6
2	5	0	3	6	1	4	7
3	6	1	4	7	2	5	0
4	7	2	5	0	3	6	1
5	0	3	6	1	4	7	2
6	1	4	7	2	5	0	3
7	2	5	0	3	6	1	4

Fig. 9. Hash table for 8 groups.

single group $\mathcal{S}_k$. The overlapping splats will be transparency-blended back-to-front into image I_k which may result in excessive attenuation of other surface layers.
2. *Too few fragments per pixel*: The basic grouping algorithm does not guarantee that splats in a single group $\mathcal{S}_k$ cover the object's surface. This may result in holes within layers in some images I_k, and these missing fragments will introduce incorrect transparency-blending results.

4.4.1. Fragment culling

Optimally, in each transparent surface layer there is exactly one fragment that contributes to α-blending per pixel. We achieve this goal by reducing the precision of the per-fragment depth value. Let us assume that the z-test is on and set to pass fragments with smaller depth, and splats are rendered back-to-front. Now consider three fragments for the same pixel: f_1 with depth d_1 on a far surface layer, and f_2 and f_3 with depths d_2 and d_3, respectively, in the same near layer. Hence, $d_1 > d_2 \approx d_3$.

As f_1 is the first fragment in the pipeline it passes the z-test. Second is f_2 which also passes since $d_2 < d_1$, and colors are α-blended $\alpha\mathbf{c}_2 + (1-\alpha)\mathbf{c}_1$. Last f_3 enters the pipeline and should be rejected to avoid causing extra attenuation as it is in the same layer as f_2. This can be achieved by lowering depth precision to make $\tilde{d}_2 = \tilde{d}_3$, so that f_3 can be culled by z-test. Thus we can set the low precision fragment depth to

$$\tilde{d}_f = floor\left(\frac{\mathrm{d}_f - d_{\min}}{\mathrm{d}_{\max} - d_{\min}} \cdot n\right) \cdot n^{-1}, \tag{10}$$

where $d_{\min}$ and $d_{\max}$ are the nearest and farthest depths from the object to the eye, the fragment depth d_f is given from the hardware rasterization, and n is a constant that can be set to a value larger or equal to $(\mathrm{d}_{\max} - d_{\min})/\varepsilon$ based on the ε–z-buffer offset.

4.4.2. Surface coverage

The solution to covering the object surface is to change splats in each group $\mathcal{S}_k$ so as to cover more surface while keeping the overlap as small as possible. We propose two methods to do this: (1) adding splats and (2) enlarging splat radii.

(1) To better cover the object by group $\mathcal{S}_k$, points from other groups are duplicated and added to $\mathcal{S}_k$ as follows, where *Clipped(*$\mathbf{p}_i, r, k$*)* is the area of $\mathbf{p}_i$ overlapped by splats in $\mathcal{S}_k$:

1. Create a priority queue Q containing all splats $\mathcal{S}\backslash\mathcal{S}_k$, with priority p_i being $Clipped(\mathbf{p}_i, r, k)$.
2. Process splats in Q in descending order. For each $\mathbf{p}_i$, update its priority $p_i^{\mathrm{new}} = Clipped(\mathbf{p}_i, r, k)$ as $\mathcal{S}_k$ may have changed (with $p_i^{\mathrm{new}} \geqslant p_i$).
   (a) If p_i^{new} is too big, $\mathbf{p}_i$ is removed from Q and the next splat of Q is considered, otherwise proceed.
   (b) If p_i^{new} equals to the old p_i, $\mathbf{p}_i$ is added to $\mathcal{S}_k$, otherwise assign $p_i = p_i^{\mathrm{new}}$ and keep it in Q.

(2) Though a better surface coverage can be achieved by duplicating splats in multiple groups as above, the number of processed points and amount of overlap is also increased. Alternatively, we can cover more object surface by $\mathcal{S}_k$ by enlarging its splat radius.

The surface area covered by $\mathcal{S}_k$ can be calculated by

$$CoveredArea = n \cdot \pi r^2 - \sum_{\forall \mathbf{p}_i \in \mathcal{S}_k} Clipped(\mathbf{p}_i, r, k), \tag{11}$$

where $n = |\mathcal{S}_k|$ and r the (uniform) radius of splats.

Suppose the object's surface area is A, which can be calculated similarly to Eq. (11) for all points in $\mathcal{S}$. Enlarging the splat radii to $\tilde{r}$ should achieve

$$A \equiv n \cdot \pi \tilde{r}^2 - \sum_{\forall \mathbf{p}_i \in \mathcal{S}_k} Clipped(\mathbf{p}_i, \tilde{r}, k). \tag{12}$$

Notice that an enlarged radius $\tilde{r} > r$ also causes increased clipping $Clipped(\mathbf{p}_i, \tilde{r}, k) > Clipped(\mathbf{p}_i, r, k)$. Based on this observation, a simple iterative solution of Eq. (13) for $\tilde{r}_{s+1}$ is applied until the difference between $\tilde{r}_s$ and $\tilde{r}_{s+1}$ is small enough (with $\tilde{r}_0 = r$),

$$n \cdot \pi \tilde{r}_{s+1}^2 = A + \sum_{\forall \mathbf{p}_i \in \mathcal{S}_k} Clipped(\mathbf{p}_i, \tilde{r}_s, k). \tag{13}$$

Note that the above calculation is quite time consuming so that it is hard to be used for dynamic points. An approximation for the enlarged radius can be calculated in the following way based on the hash-based algorithm. According to the hash table, we can get a rough estimate about the distribution of splats in a single group. As shown in Fig. 10, the light blue disks show the ideal distribution of splats in group 3. Considering isosceles triangle $\triangle abc$ where $|ab| = |ac| = \sqrt{10} \cdot r$ and $|bc| = 2\sqrt{2} \cdot r$, the enlarged splat radius should be the radius of the circumcircle of $\triangle abc$, which is $\frac{\sqrt{50}}{4} \cdot r$. In practice, we can adopt a slightly bigger value than the ideal one to guarantee the surface coverage.

5. Rendering algorithm

5.1. Rendering opaque point surfaces

Based on the *deferred blending* concept and the grouping solution, we can now describe our basic rendering algorithm as illustrated in Fig. 11. The 1 + 1-pass rendering algorithm includes one pass over the point splat geometry $\mathcal{S} = \bigcup_k \mathcal{S}_k$ defined by the grouping process, and a second image compositing pass over the corresponding partial depth-images I_k.

As discussed in Section 4, if we want to improve rendering efficiency by reducing the number K of groups, we may suffer minor artifacts caused by small overlap between splats s_i and s_j belonging to the same group $\mathcal{S}_k$. In fact, the rendering algorithm in Fig. 11 guarantees that only one point splat will contribute its color and weight to the fragment f in the overlap region between splats s_i and s_j. This is because the z-visibility test is activated and hence

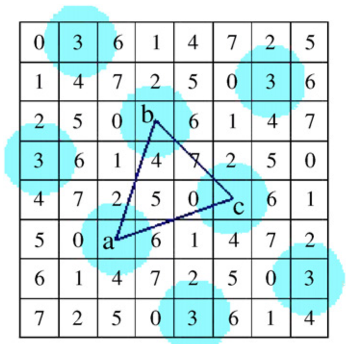


Fig. 10. Calculating the approximated enlarged splat radius using the hash-table.

only one fragment, the nearest with smallest depth, from either s_i or s_j will survive.

To avoid disturbing artifacts from extended flaps of overlapping splats resulting from the above simple z-visibility culling, Voronoi point rasterization can be used [27]. In the overlap between splats s_i and s_j, this technique assigns the color $\mathbf{c}_j$ and weight $w_j(\mathbf{f}_j)$ values of the splat s_j with $w_j(\mathbf{f}_j) \leqslant w_i(\mathbf{f}_i)$ to the fragment f. Thus in the overlap region, not the fragments with larger depth but with lower kernel weights will be culled.

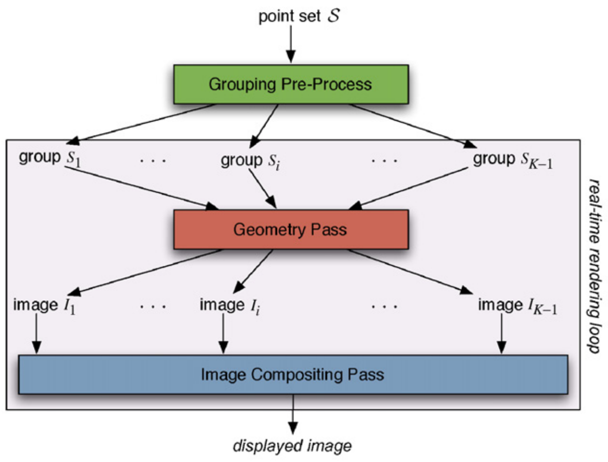
However, in contrast to [27] we do not introduce an extra rendering pass to implement Voronoi rasterization but realize it by outputting a Voronoi enhanced depth value in addition to the regular z-depth on line 9 of the *Geometry Pass* in Fig. 11. Given the current fragment's depth $d_f = \mathbf{c}_d(f)_k$ as z-distance of $\mathbf{f}_i$ to the eye point and the distance $d_i = |\mathbf{f}_i - \mathbf{p}_i|$ of the fragment-splat intersection $\mathbf{f}_i$ from the splat center, we define this modified z-depth value as

$$z = z_{\text{lowres}} + z_{\text{voronoi}} = \tilde{d}_f + \frac{\mathrm{d}_i}{r_i} \cdot n^{-1}, \tag{14}$$

where $\tilde{d}_f$ is defined in Eq. (10), r_i is the splats disk radius and n is an integer constant. The constant n is defined in Section 4.4.1.

The first term z_{low} is a low-precision depth which limits the depth values of all fragments to the range $[0, 1/n, 2/n, \ldots, 1]$. It is used to distinguish and separate fragments coming from different surface layers. The second part z_{voronoi} is a fragment-point distance ratio scaled to $[0, 1/n]$. Overlapping splats in the same surface layer should have the same z_{lowres} depth value and only distinguish in z_{voronoi}. Hence, in the nearest visible surface layer, fragments from s_i with the smallest z_{voronoi} value win the hardware z-visibility test against any fragments from other overlapping splats s_j. On the other hand, fragments of splats from different occluded surface layers will have a larger z_{low}, with the minimum difference of $1/n$ being larger than the maximum z_{voronoi}, and thus be culled.

In fact, the enhanced depth value of Eq. (14) is used for hardware z-buffering while the standard depth d_f is additionally stored for the fragment in the current target buffer I_k. This d_f is used in the compositing step for ε–z-visibility determination and blending.



Algorithm-1:

```
Geometry Pass:
1  turn on $z$-test and $z$-update;
2  for $k = 0$ to $K - 1$ do
3    clear $z$-depth and color of depth-image texture $I_k$;
4    render group $\mathcal{S}_k$ to depth-image texture $I_k$;
5    foreach $s_i \in \mathcal{S}_k$ do
6      transform, project and rasterize splat $s_i$;
7      foreach generated fragment $f \in I_k$ do
8        output color $\mathbf{c}_{\mathrm{rgb}}(f)_k$ and kernel weight $\mathbf{c}_\alpha(f)_k$
         according to Eq. 4;
9        output $z$-depth $\mathbf{c}_d(f)_k$;
10     endforeach
11   endforeach
12 endfor

Image Compositing Pass:
   As listed in Fig. 6
```

Fig. 11. Overview of 1 + 1-pass point rendering algorithm.

5.2. Rendering transparent point surfaces

5.2.1. Basic transparency

For efficient back-to-front ordering of the point data we use a BSP-tree data organization and traversal [30]. Based on this and the outlined extended grouping of splats, we can now define the following 1 + 1-pass PBR algorithm for transparent point objects:

Algorithm 1.

1. *Geometry pass* (*Transparency-blending*): Turn on z-test and α-blending. Render all splats s_i of each group $\mathscr{S}_k$ using modified radii $\tilde{r}_i$ into separate target images I_k. Perform back-to-front α-blending (using the material opacity for α_i and $1 - \alpha_i$). Adjust the fragment depth according to Eq. (10).

2. *Compositing pass* (*PBR-blending*): Combine (average) all K images I_k into final frame buffer.

Algorithm-1 implements a basic transparent point rendering solution. As such it suffers from the fact that each image I_k contributes equally to the final interpolation between point splats since no smooth interpolation blending kernels are supported. As demonstrated by our experiments, however, the artifacts introduced by this omission are hardly noticeable as shown in Figs. 18(a)) or 19(a)).

5.2.2. High-quality transparency

The point interpolation artifacts of Algorithm-1 can further be reduced by rendering the closest transparent surface layer separately and in higher quality (see Fig. 18(b)). This, however, will require a separate geometry pass for this first visible layer.

Therefore, we achieve high-quality transparency by rendering the nearest transparent layer in a separate pass to perform smooth point interpolation, and all other layers using the geometry pass of Algorithm-1. The two sets of images are then combined into a high-quality blended final result. In fact, this compositing pass performs three blending operations simultaneously: (i) smooth PBR interpolation of the nearest layer (including per-fragment color normalization); (ii) simple PBR interpolation of the other layers, and (iii) transparent α-blending of the nearest with the other layers.

Algorithm 2.

1. *Geometry pass for nearest layer*: Use the geometry pass of Algorithm-1 to render the point groups $\mathscr{S}_k$ to K target images I_k, including the depth information of the nearest fragments d_f and interpolation-kernel weight h_f.
2. *Geometry pass for other layers*: Use the geometry pass of Algorithm-1 to render the point groups $\mathscr{S}_k$ to K target images O_k, but culling all fragments from the nearest layer using the depth-mask Z from the first pass.
3. *Compositing pass*: Combine images I_k together where fragments f_k with depth $d_{f_k} - \min_k(d_{f_k}) > \varepsilon$ are occluded and discarded. All others, $\widehat{f}_k$, are composited together for a smoothly interpolated image C_{F} of the nearest visible layer with colors $\sum h_{\widehat{f}_k} \cdot \mathbf{c}_{\widehat{f}_k} / \sum h_{\widehat{f}_k}$. Then average the images O_k into C_O for the other layers. Finally high-quality transparency is achieved given the opacity α by $I = \alpha \cdot C_{\mathrm{F}} + (1-\alpha) \cdot C_O$.

Note that our transparency algorithms support varying material opacities, possibly different for each individual point splat, as the α_i values can be specified for each splat s_i and are processed on the fragment level.

5.2.3. Reflections and refractions

Besides basic transparency, refraction effects and specular reflections of the environment dramatically improve the rendering realism. Both effects are derived from the incident viewing vector and surface normal, and include a reflective and refractive environment map lookup which can all be added to the first geometry pass of Algorithm-2.

Note, however, that this way refraction and reflection can only be incorporated for the nearest visible layer. But visual realism can further be increased by adding multi-layer transparency effects such as multiple ray refraction and light absorption through semi-transparent material.

We can approximate visual multi-layer effects exploiting the GPU feature of associating different α-blending modes to the color and opacity (α-) channels, respectively. Setting the mode of the α-channel for both *SRC_ALPHA* and *DST_ALPHA* to 1.0 in the second geometry pass of Algorithm-2 causes accumulation of opacity over all layers $\alpha_{\mathrm{total}} = \sum_{\mathrm{layers}} \alpha_i$, that is in each image O_k separately for each group $\mathscr{S}_k$. Assuming a constant material opacity α we derive the number of layers from $l = \alpha_{\mathrm{total}}/\alpha$.

We extend our PBR algorithm using the layer number l to approximate the distance that light travels through semi-transparent material. Our approximation defines the light absorption ratio as

$$AbsorptionRatio = (1-\alpha)^l. \tag{15}$$

For multi-layer refraction effects, we simulate a transmitted total refraction angle θ_{T} by Eq. (16) which assumes equal refraction ratios at all layer interfaces. This is clearly a heuristic, but it provides good multiple layer transparency cues. Given the refraction ratio η and incidence angle θ_{I} we get

$$\sin_{\theta_{\mathrm{T}}} = \eta^l \cdot \sin_{\theta_{\mathrm{I}}}. \tag{16}$$

Although Eqs. (15) and (16) are not physically correct, they produce appealing visual multi-layer transparency effects (see also Section 6).

Additional lighting phenomena, also shown in Figs. 18(c) and (d), that can be simulated based on refractive and reflective environment mapping including *Fresnel effect* and *chromatic dispersion*.

5.2.4. Per-fragment shading

To achieve smooth illumination and shading effects, lighting, refraction and reflection are computed per fragment using a *deferred shading* approach as outlined in [13,16]. Deferred shading not only interpolates per-point colors, but in fact any attributes that are needed for shading. Thus per-point surface normal, and position if necessary, are interpolated for each fragment and rendered into separate attribute buffers as done for color. In the compositing pass, each set of attribute buffers (for the K groups) is handled the same way as color in Algorithm-2. Then Phong lighting, environment map reflection, (multi-layer) refraction and attenuation are calculated using the composited per-fragment attributes. If the number of textures exceeds the multi-texturing limit of a graphics card, the work can be split into multiple compositing passes.

While single-layer transparency effects could be achieved without deferred shading, the multi-layer effects introduced above depend on the number of layers l which is only available after all geometry has been processed. Hence, attenuation and refraction are done after geometry processing in the compositing pass. Additionally, deferred shading can support further effects such as bump-mapping.

6. Experimental results

We have implemented our point rendering algorithm in DirectX on a PC with a 2.8 GHz CPU and NVidia GeForce 7800GTX GPU.

6.1. Grouping algorithm

The first experiments are with respect to the MGC algorithm described in Section 4.1. As point-based surface models inherently depend on a significant overlap ratio between neighboring splats to generate a smoothly blended surface rendering, it comes as no surprise that a basic graph coloring solution with edges defined as in Eq. (5) may result in a fairly high number of colors K. In Table 1 we show the graph coloring results for different overlap relaxation parameters c used in the proposed extension (3). With decreasing c also the chromatic number $\mathcal{X}(G)$ drops rapidly.

We use overlap factor $Overlap(p_i) = \sum_k (2r - |p_i - p_k|)/r$ to measure the overlap for splat s_i with its overlapping neighbors s_k. We show the average overlap factor of the dragon model for the three grouping algorithms mentioned in this paper in Table 2 where $K = 8$. WGC algorithm provides the best grouping results because it exactly calculates the overlap between splats and aims at minimizing the overall overlap. On the other hand, it is the most time-consuming method and it takes about 4.6 s to finish the basic grouping. For the hash-based grouping algorithm, we have three different cases: (1) without and (2) with handling collisions in the third and fourth column, respectively, and (3) the mixed case in the fifth column. In practice, we prefer a fixed number of groups defined by the user so that a fixed shader program can be used. Both WGC and hash-based grouping can support this feature.

In Table 3 we show the performance of hash-based grouping algorithm in three different cases as Table 2 has. Notice that the data listed here is measured for the worst case, which means that we performed the dynamic grouping for all point splats and refreshed the whole index buffer for each rendered frame. In practice, however, regrouping all points for every frame is not necessary if only some parts of the deformable object are changed.

In Fig. 12, we compare the hash-based grouping algorithm to WGC algorithm. In the first row we show three grouping results from WGC and hash-based grouping algorithm ($K = 8$). It is clear that the WGC algorithm produces the best grouping results. There is almost no overlaps between splats. For the hash-based algorithm, some overlap appears especially in the case of ignoring the collisions problem. In the second row we show the corresponding rendering results, and we can see that the hash-based algorithm produces comparable image quality to the WGC algorithm if collisions are handled. Some artifacts appear in the high-curvature part if collisions are ignored in the hash-based algorithm. Note that the artifacts are visible only if the object is very close to the eye point.

6.2. Rendering opaque point surfaces

In Fig. 13 we show different rendering results for different overlap relaxation parameters c used by the MGC algorithm. We can see that in comparison to the standard PBR blending result, there are hardly any visible artifacts introduced even if the parameter c is set as low as 0.4, which has shown to be an acceptable value with respect to the group number $K = \mathcal{X}(G)$ from graph coloring and rendering image quality.

Our Voronoi rasterization implementation using the z-visibility test defined by the modified z-depth value in Eq. (14) is demonstrated in Fig. 14. It shows the effective removal of flaps between overlapping splats and the resulting faceted surface similar to [27]. This surface model is basically the depth-map, combined from all I_k, for the ε–z-buffer visibility test in conventional PBR.

Table 1
MGC point grouping results for different overlap relaxation parameters c

Model	Points	$K = \mathcal{X}(G)$/maxdegree/avgdegree			
	$\lvert\mathcal{S}\rvert$	$c = 1.0$	$c = 0.8$	$c = 0.6$	$c = 0.4$
Dhead	2000K	18/37/17.2	14/31/11.6	11/24/9.4	7/8/3.9
Dragon	1100K	14/34/8.8	12/29/6.3	8/15/3.0	5/7/0.8
Female	303K	19/49/18.9	15/32/13.2	10/18/6.9	8/9/2.3
Balljoint	137K	17/31/18.6	12/23/13.6	9/14/7.1	5/7/2.3

Table 2
Average overlap factor for our grouping algorithms

WGC	MGC	Dynamic grouping		
		(1)	(2)	(3)
0.08	0.11	1.24	0.19	1.03

Table 3
Performance for dynamic grouping algorithm

Model	Points $\lvert\mathcal{S}\rvert$	Dynamic grouping (s)		
		(1)	(2)	(3)
Dragon	1100K	0.19	0.58	0.33
Female	303K	0.06	0.19	0.12
Balljoint	137K	0.02	0.07	0.04

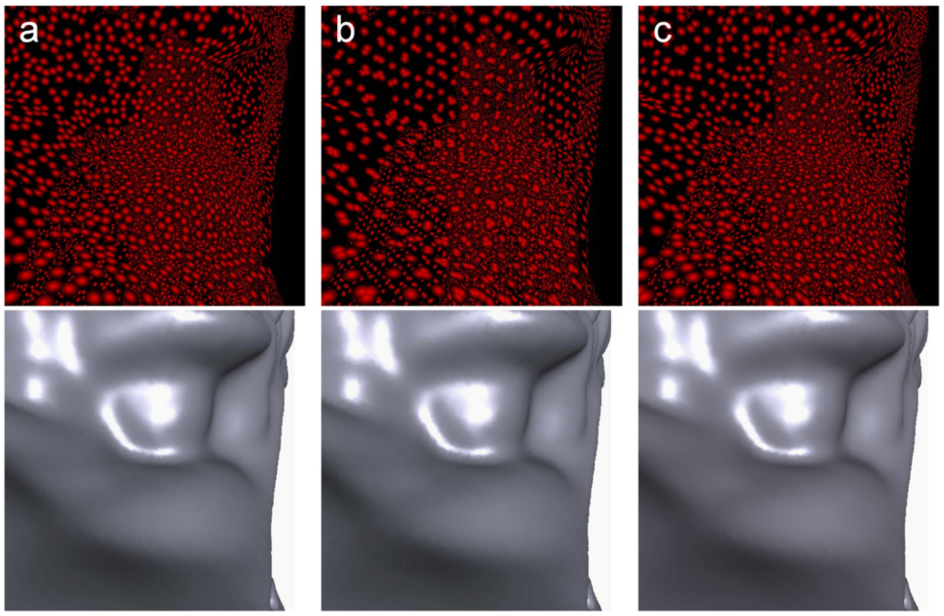


Fig. 12. Comparison of dynamic grouping algorithms: (a) WGC algorithm; (b) hash-based grouping algorithm without dealing with collisions; (c) hash-based grouping algorithm resolving collisions.

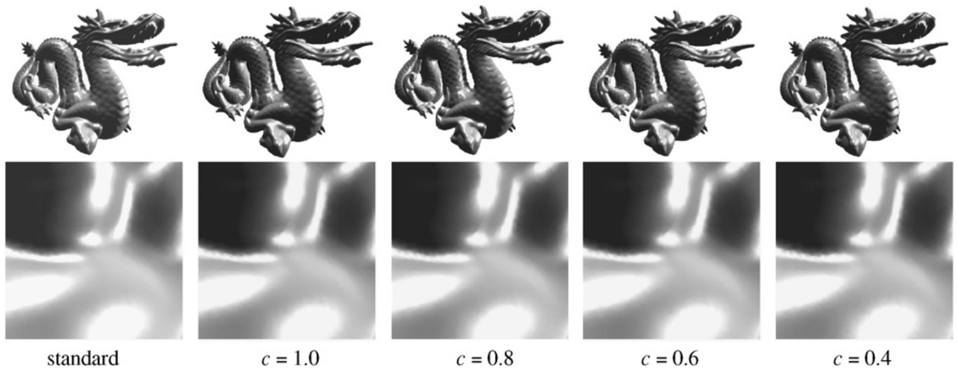


Fig. 13. Comparison of smooth point blending results for different overlap relaxation parameters c with respect to standard PBR blending.

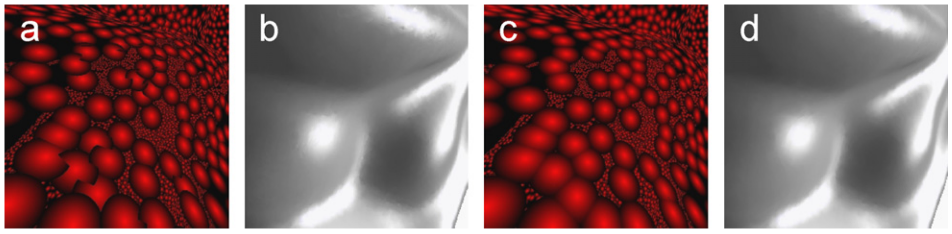


Fig. 14. Voronoi rasterization. In (a) and (b) we show the rasterization and shading examples without Voronoi rasterization enabled, hence fragments with smaller z-depth simply override any other. In (c) and (d), fragments with smaller Voronoi-depth as defined in Eq. (14) win the z-buffer visibility test.

Rendering performance is demonstrated in Table 4. We can see that for large point models, our algorithm can improve the rendering efficiency up to 50%, depending on the parameter c, and hence on the achieved grouping value K. For very small models where geometry processing is negligible, our $1+1$-pass algorithm may in fact be slower than a standard $2+1$-pass point rendering implementation. This can be expected for small enough models where the geometry rendering pass is less costly than image compositing. The *Image Compositing Pass* in Figs. 11 and 6 requires K texture lookups, and it accesses color, blending weight and fragment depth values from two color channels to avoid expensive pack and unpack operations. For $c=0.8$ in Table 4, Voronoi rasterization is disabled as the grouping of points is so effective that no significant point overlap is noticeable. Voronoi rasterization is only enabled for $c = 0.4$ which results in low grouping numbers K. Note also that for the models with around 1M points or less, the point geometry data can easily be cached in GPU memory which results in significantly better frame rates than for larger models which are kept in CPU main memory (i.e. the David head model).

Additional $1+1$-pass solid point surface rendering results are presented in Fig. 15, demonstrating smooth images at improved frame rates for large models.

6.3. Rendering transparent point surfaces

With respect to the graph coloring algorithm, the choice of K can make a difference. From experiments using different values for K, we have found that it is sufficient to set $K = 4$ to achieve a good separation of points into groups. Fig. 16 shows a good sampling of the surface for $K = 4$ compared to a larger value. Using a small K and to achieve good surface coverage for our transparent point rendering algorithms, it is feasible to use the group extension (1) proposed in Section 4.4.2. At the expense of points duplicated in multiple groups a good surface coverage can be achieved. For the dragon model, the sum of points in all groups increased the base data set by 45%. While this is not a negligible ratio, the results presented show that good display quality at good rendering performance can be achieved.

Table 4
Frame rate performance of our novel $1+1$-pass point rendering algorithm compared to a standard $2+1$-pass PBR implementation

Model	Points $\lvert\mathcal{S}\rvert$	FPS		
		$2+1$-pass	$c = 0.8$	$c = 0.4$
Dhead	2000K	0.96	1.2	1.4
Dragon	1100K	15.04	19.70	22.62
Female	303K	32.65	32.11	37.76
Balljoint	137K	65.68	52.96	70.37

If a larger K is required, the radius enlargement method (2) described in Section 4.4.2 is a better choice to achieve good surface coverage and to avoid a large point duplication ratio. At the expense of increased texture lookups and image compositing cost, method (2) can in fact avoid any point duplication at all.

In Fig. 17, our transparent PBR algorithms are compared to two depth-peeling methods, one is the standard depth-peeling method which generates the correct back-to-front α-blending result, and the other is the recently proposed method in [11] whose basic idea is only to use several front-most layers in rendering to improve the performance of depth-peeling. In contrast to depth-peeling, which conducts smooth point interpolation on each surface layer by a standard opaque rendering method, our algorithms perform the approximated point interpolation for all layers in Algorithm-1. We can observe that any so introduced visual artifacts are masked by the transparency attenuation and are hardly visible using Algorithm-1, and virtually no visual difference can be observed using Algorithm-2.

Furthermore, an error measurement $\varepsilon = \sum(|R(i,j) - R_d(i,j)| + |G(i,j) - G_d(i,j)| + |B(i,j) - B_d(i,j)|)/numPixel$ is employed to measure the error of the transparent rendering algorithm with respect to the standard depth-peeling, where $R(i,j), G(i,j), B(i,j)$ and $R_d(i,j), G_d(i,j),$

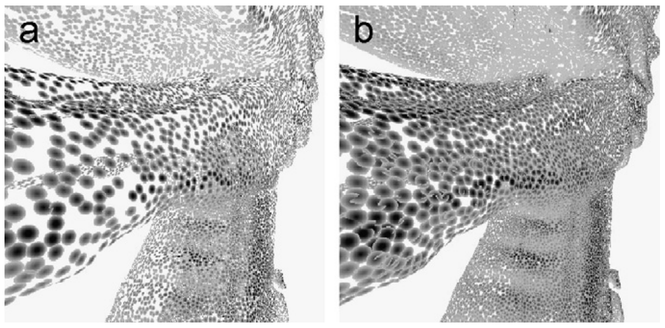


Fig. 16. Grouping results: (a) Splats have smaller overlaps but less surface coverage for $K = 8$; (b) splats have bigger overlaps but better surface cover for $K = 4$.

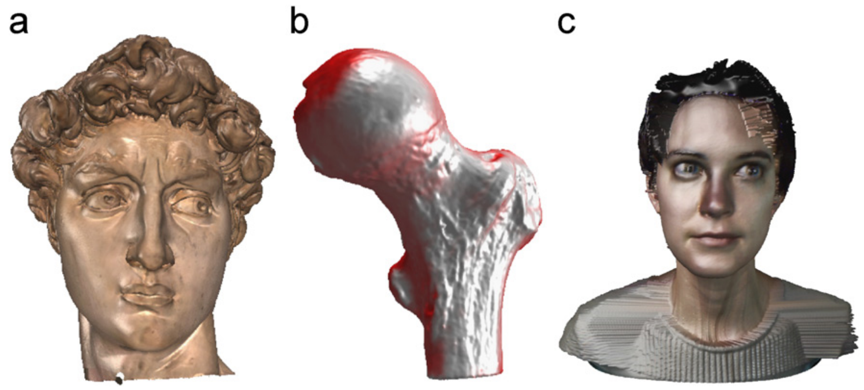


Fig. 15. Rendering results for various point models: (a) David head model rendered at 1.4 FPS; (b) balljoint model rendered at 70 FPS and (c) female model displayed at 37 FPS, using $c = 0.4$ and Voronoi rasterization.

$B_d(i,j)$ are color values of pixel (i,j) of the image generated by a transparent rendering and the standard depth-peeling algorithm, respectively, and *numPixel* is the total number of pixels in the image. The ε values of our $1+1$-pass Algorithm-1, $2+1$-pass Algorithm-2 and depth-peeling [11] method are 4.71, 3.42, 5.82, respectively.

The basic frame rate for the different transparent point rendering algorithms are: our $1+1$-pass transparent point rendering Algorithm-1 achieves 9 FPS, on the other hand our high-quality $2+1$-pass Algorithm-2 reaches 5 FPS. This compares very well to depth-peeling, which attains only less than 2 FPS for an upper limit of 8 layers. And for depth-peeling with 3 layers [11] the frame rate is about 4.5 FPS.

The above experimental data shows that our method produces more accurate blending results than [11]. And the performance of our methods, even with the $2+1$-pass high quality transparent algorithm, is also better than [11] because even if only using three front-most layers, their method requires to process the entire point data three times, while our high quality transparent algorithm only needs two passes for arbitrary number of transparent layers at the cost of some precomputing and BSP-tree traversal to provide back-to-front rendering order.

Several small bouncing opaque balls are added to the scene in Figs. 18(a), (b) and 19 to demonstrate that our algorithms generate the correct α-blending results when combining opaque and transparent objects. We demonstrate in Fig. 19 that our $1+1$-pass transparent PBR algorithm achieves high visual rendering quality for viewing configurations which do not exhibit extreme close-up views.

Figs. 18(c), (d) and 20 show rendering results of combining high-quality transparency and environment mapping. Note that both the Fresnel effect and chromatic dispersion are simulated in these images. In the close-up views of Fig. 20 we can also see the subtle differences between single- and multi-layer transparency effects such as the approximated multiple refractions and increased attenuation. All of these effects provide important visual clues about the existence of multiple transparent surface layers.

7. Summary

In this paper we present a new framework for GPU accelerated PBR algorithms based on the new concept of *deferred blending*. The basic idea is the division of the point splats into non-overlapping subsets such that smooth point interpolation can be deferred to a final image compositing pass. This concept allows us to perform only a single rendering pass over the point geometry data. Our new framework provides two solutions for the rendering of opaque and transparent point surfaces, respectively. With respect to the rendering of opaque surfaces, we only need one pass over the geometry data. The rendered images show that our algorithm can provide a very good rendering quality. The experimental data also shows that our algorithm is more efficient than a standard $2+1$-pass algorithm, in particular for the larger point data sets.

With respect to transparency, we present two novel GPU-based algorithms for high-quality rendering of transparent point surfaces. The major challenge of

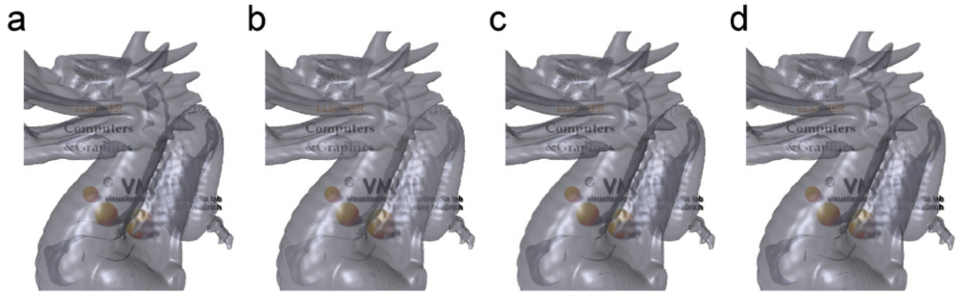


Fig. 17. Transparent image rendering quality for: (a) depth-peeling with 8 layers; (b) depth-peeling as proposed in [11], rendering only the front-most 3 layers ($\varepsilon = 5.82$); (c) $2+1$-pass Algorithm-2 ($\varepsilon = 3.42$); (d) $1+1$-pass Algorithm-1 ($\varepsilon = 4.71$).

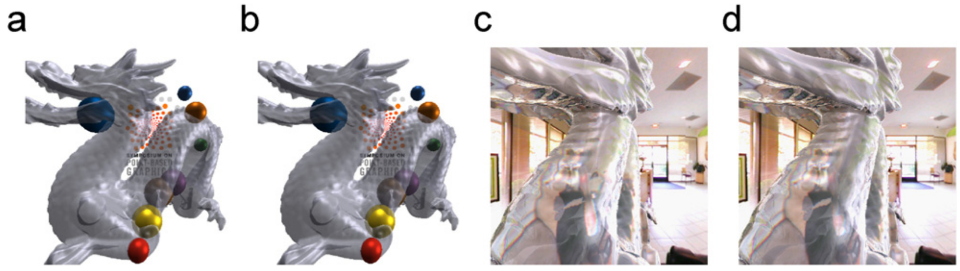


Fig. 18. Rendering transparent point objects on the GPU. Transparent and opaque objects with: (a) single-pass algorithm and (b) two-pass algorithm. Reflective and refractive environment mapping with: (c) single and (d) multi-layer effects.

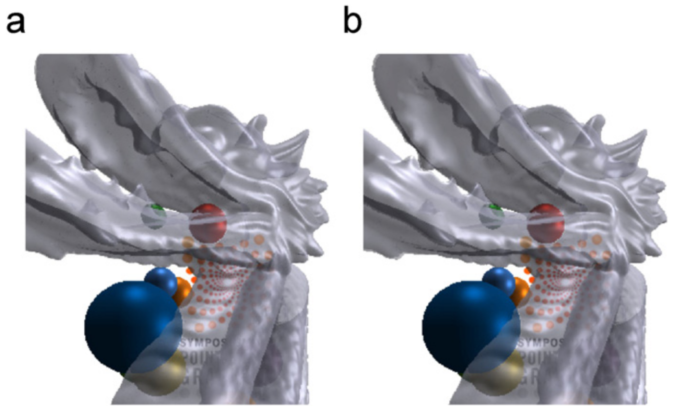


Fig. 19. Opaque and transparent objects, (a) 1 + 1-pass Algorithm-1 and (b) 2 + 1-pass Algorithm-2.

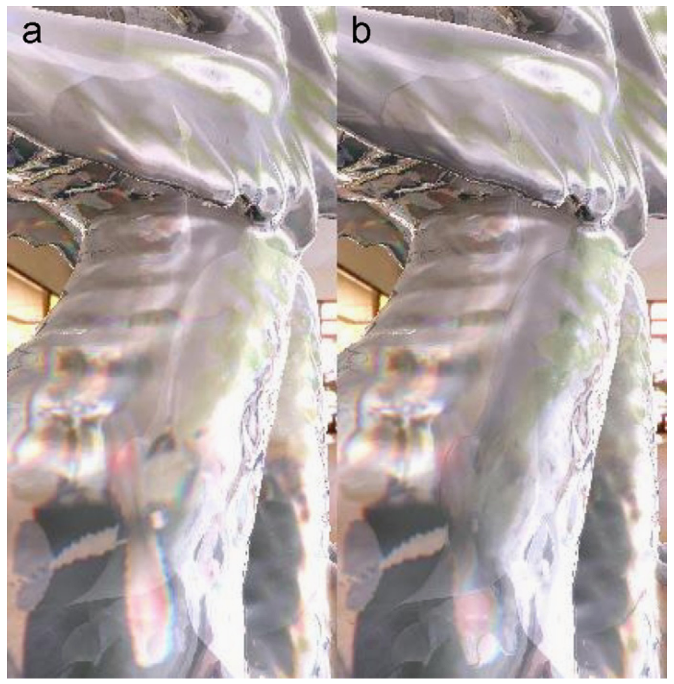


Fig. 20. (a) Single-layer, (b) multi-layer transparent refraction and specular reflection environment mapping effects.

handling the conflicting smooth point interpolation and transparent α-blending simultaneously is solved by separating them to different rendering passes, again exploiting the concept of *deferred blending*. We not only introduce the first GPU accelerated approaches to render multi-layer transparent point surfaces, but in fact provide a basic transparency α-blending of multiple transparent point layers in a single geometry processing pass over the point data. The more advanced 2 + 1-pass algorithm achieves very high-quality transparency blending and incorporates effective simulations of multi-layer refraction and reflection effects.

With respect to the involved point grouping problem, we have presented different practical solutions to perform the grouping as an off-line pre-process or even dynamically on-line. The off-line grouping methods are based on two different graph coloring algorithms and provide excellent results for the *deferred blending* rendering approach. For dynamic points or deformable surfaces we also presented an efficient hash-based grouping algorithm that can be used on-line for interactive rendering results.

Acknowledgments

We would like to thank the Stanford 3D Scanning Repository and Digital Michelangelo projects as well as Cyberware for providing the 3D geometry test data sets. This work was partially supported by the Swiss National Science Foundation Grant 200021-111746/1.

References

[1] Gross MH. Are points the better graphics primitives? Computer Graphics Forum 2001;20(3) Plenary Talk Eurographics 2001.
[2] Pfister H, Gross M. Point-based computer graphics. IEEE Computer Graphics and Applications 2004;24(4):22–3.
[3] Sainz M, Pajarola R. Point-based rendering techniques. Computers & Graphics 2004;28(6):869–79.
[4] Kobbelt L, Botsch M. A survey of point-based techniques in computer graphics. Computers & Graphics 2004;28(6):801–14.
[5] Zwicker M, Pauly M, Knoll O, Gross M. Pointshop 3D: an interactive system for point-based surface editing. In: Proceedings ACM SIGGRAPH. ACM Press; 2002. p. 322–9.
[6] Pauly M, Keiser R, Kobbelt L, Gross M. Shape modeling with point-sampled geometry. ACM transactions on graphics 2003;22(3): 641–50.
[7] Botsch M, Kobbelt L. Real-time shape editing using radial basis functions. Computer graphics forum 2005;24(3):611–21 Eurographics 2005 Proceedings.
[8] Sainz M, Pajarola R, Lario R. Points reloaded: point-based rendering revisited. In: Proceedings symposium on point-based graphics. Eurographics Association; 2004. p. 121–8.
[9] Zhang Y, Pajarola R. Single-pass point rendering and transparent shading. In: Proceedings symposium on point-based graphics. Eurographics Association; 2006. p. 37–48.
[10] Zhang Y, Pajarola R. GPU-accelerated transparent point-based rendering. In: ACM SIGGRAPH sketches & applications catalogue; 2006.
[11] Guennebaud G, Barthe L, Paulin M. A full-featured hardware-oriented splatting frame-work. In: Proceedings symposium on point-based graphics. Eurographics Association; 2006. p. 49–56.
[12] Pfister H, Zwicker M, van Baar J, Gross M. Surfels: surface elements as rendering primitives. In: Proceedings ACM SIGGRAPH. ACM SIGGRAPH; 2000. p. 335–42.
[13] Zwicker M, Pfister H, vanBaar J, Gross M. Surface splatting. In: Proceedings ACM SIGGRAPH. ACM SIGGRAPH; 2001. p. 371–8.
[14] Ren L, Pfister H, Zwicker M. Object space EWA surface splatting: a hardware accelerated approach to high quality point rendering. In: Proceedings EUROGRAPHICS 2002, p. 461–70, also in Computer Graphics Forum 21(3).
[15] Zwicker M, Ranen J, Botsch M, Dachsbacher C, Pauly M. Perspective accurate splatting. In: Proceedings of graphics interface; 2004. p. 247–54.
[16] Botsch M, Hornung A, Zwicker M, Kobbelt L. High-quality surface splatting on today's GPUs. In: Proceedings symposium on point-based graphics. Eurographics Association; 2005.
[17] Kalaiah A, Varshney A. Differential point rendering. In: Proceedings eurographics workshop on rendering techniques. Springer; 2001. p. 139–50.

[18] Botsch M, Spernat M, Kobbelt L. Phong splatting. In: Proceedings symposium on point-based graphics. Eurographics; 2004. p. 25–32.

[19] Chen B, Nguyen MX. POP: a hybrid point and polygon rendering system for large data. In: Proceedings IEEE Visualization; 2001. p. 45–52.

[20] Cohen JD, Aliaga DG, Zhang W. Hybrid simplification: combining multiresolution polygon and point rendering. In: Proceedings IEEE Visualization; 2001. p. 37–44.

[21] Coconu L, Hege H-C. Hardware-oriented point-based rendering of complex scenes. In: Proceedings Eurographics Workshop on Rendering; 2002. p. 43–52.

[22] Dey TK, Hudson J. PMR: point to mesh rendering, a feature-based approach. In: Proceedings IEEE visualization. Computer Society Press; 2002. p. 155–62.

[23] Pajarola R, Sainz M, Guidotti P. Confetti: Object-space point blending and splatting. IEEE Transactions on Visualization and Computer Graphics 2004;10(5):598–608.

[24] Botsch M, Kobbelt L. High-quality point-based rendering on modern GPUs. In: Proceedings Pacific graphics 2003. IEEE Computer Society Press; 2003. p. 335–43.

[25] Mammen A. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. IEEE Computer Graphics & Applications 1989;9(4):43–55.

[26] Everitt C. Interactive order-independent transparency. Technical Report, available at ⟨http://www.nvidia.com.⟩; 2002.

[27] Talton JO, Carr NA, Hart JC. Voronoi rasterization of sparse point sets. In: Proceedings symposium on point-based graphics. Eurographics Association; 2005. p. 33–7.

[28] Jensen TR, Toft B. Graph coloring problems. New York: Wiley-Interscience; 1994.

[29] Leighton FT. A graph coloring algorithm for large scheduling problems. Journal of Research of the National Bureau of Standards 1979;84:489–506.

[30] Samet H. The design and analysis of spatial data structures. Reading, Massachusetts: Addison Wesley; 1989.