Eurographics Symposium on Point-Based Graphics (2006)
M. Botsch, B. Chen (Editors)

Single-Pass Point Rendering and Transparent Shading

Yanci Zhang and Renato Pajarola

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

Abstract

Hardware accelerated point-based rendering (PBR) algorithms have suffered in the past from multiple rendering passes; possibly a performance limiting factor. Two passes over the point geometry have been necessary because a first visibility-splatting pass has been necessary for conservative ε-z-buffer visibility culling in the following point-interpolation rendering pass. This separation into visibility-splatting and point-blending, hence processing the point geometry twice, is a fundamental drawback of current GPU-based PBR algorithms. In this paper we introduce a new framework for GPU accelerated PBR algorithm whose basic idea is deferred blending. *In contrast to prior algorithms, we formulate the smooth point interpolation problem as an image compositing post-processing task. This is achieved by separating the input point data in a pre-process into not self-overlapping minimal independent groups of points. As an extension of this concept, we can for the first time render transparent point surfaces as well on the GPU. For simple transparency effects, our novel algorithm only needs a single geometry rendering pass. For high-quality transparent image synthesis an extra rendering pass is sufficient. Furthermore, per-fragment reflective and refractive multilayer effects are supported in our algorithm.*

Categories and Subject Descriptors (according to ACM CCS): I.3 [Computer Graphics]: I.3.3 [Picture/Image Generation]: Display algorithms I.3.5 [Computational Geometry and Object Modeling]: Surface representations I.3.7 [Three-Dimensional Graphics and Realism]: Color, shading, shadowing, and texture

Keywords: point based rendering, 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 [Gro01, PG04, SP04, KB04]. Points are the basic geometry defining elements of three-dimensional objects and surfaces. Moreover, most geometric modeling tasks can be performed directly on point sets as demonstrated in [ZPKG02, PKKG03, BK05].

While the significance and adoption of point-based geometric modeling and rendering steadily increases, full-featured point processing and shading algorithms must be developed. Real-time PBR algorithms to date can achieve high-quality rendering results and incorporate standard shading features. However, efficient GPU-based PBR algorithms [KB04, SP04, SPL04] generally suffer from 2+1 rendering passes; two passes over the geometry and one image processing pass. In particular, to achieve smooth interpolation and resolve correct visibility of overlapping point splats, a separate *visibility-splatting* rendering pass is employed to initialize the visibility-determining depth-buffer. In a second *point-blending* rendering pass the smooth interpolation between visible overlapping points, and smooth shading, is performed. This separation into visibility-splatting and blending, which requires processing the point geometry twice, is one remaining fundamental drawback of PBR.

Moreover, GPU-based interactive rendering of transparent point surfaces 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 smoothly interpolate between overlapping point splats within the same surface layer. To interpolate between overlapping splats in one layer, the *z*-buffer must be turned on to cull fragments farther

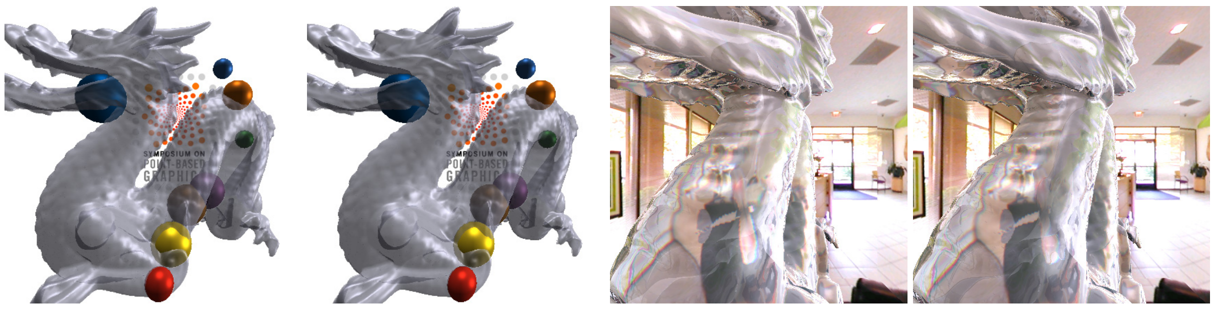


Figure 1: *Rendering transparent point objects on the GPU. Transparent and opaque objects with: a) single-pass algorithm and b) two-pass algorithm. Reflective and refractive environment mapping with: c) single and d) multi-layer effects.*

than some ε in depth from the visible surface, and pass all others.

In this paper we present a new framework for GPU-based PBR. Our framework is based on the new concept of *deferred blending* which delays the ε-*z*-buffer visibility test to an image post-processing pass so that only one pass over the geometry data is required. The main contributions are:

- The first GPU accelerated PBR algorithm that only requires one geometry processing pass.
- Two algorithms that implement rendering and shading of transparent point surfaces as shown in Fig. 1, a 1+1 (geometry + image compositing) pass rendering algorithm for simple transparency and a 2+1-pass algorithm for high-quality transparent shading.

2. Related Work

Splatting-based PBR as introduced in [PZvBG00, ZPvBG01] is the most widely adopted technique. It provides a good tradeoff between performance and rendering quality and is amenable for hardware acceleration. A wide range of GPU-accelerated point splatting algorithms such as [RPZ02, BK03, ZRB*04, BSK04, PSG04, BHZK05] have been proposed in the past and are surveyed in [SPL04, KB04, SP04].

Hardware accelerated point rendering techniques for high-quality shading include antialiasing filters [RPZ02, ZRB*04, BHZK05], point-splat normal fields [KV01] and per-fragment smooth shading [KV01, BSK04, BHZK05]. Also the combination of point and triangle primitives have been proposed [CN01, CAZ01, CH02, DH02] to improve rendering quality and performance.

A basic and common feature of virtually all GPU-accelerated PBR methods is the use of a separate visibility-splatting pre-rendering pass, see also [SP04, SPL04, KB04]. Smooth point interpolation and shading is then achieved in a second rendering pass which resolves visibility using the depth-buffer generated during visibility-splatting. The 2+1-pass rendering approach is completed by a color normalization – including optional per-fragment shading – image processing pass. The two rendering passes over the point geometry data are highly undesirable. The reduction of geometry processing to a single rendering pass is the goal of this work.

With respect to transparency, only a software algorithm has been proposed to date [ZPvBG01]. It uses a software frame buffer with multiple depth layers per fragment. Unfortunately, this solution cannot be mapped onto GPUs as they neither support multiple depths per fragment nor the simultaneous read and write of the target buffer as necessary by this solution.

In principle, depth-peeling [Eve02, Mam89] can be applied to PBR of transparent surfaces. 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

3.1. Smooth Point Interpolation

A point set $\mathcal{S}$ covers a 3D surface by a set of overlapping elliptical point splats $s_{0\dots n-1}$. The projection of $\mathcal{S}$ in image space must interpolate for each fragment f the contribution of multiple overlapping splats s_i. 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 the fragment's intersection $\mathbf{f}_i$ with the splat plane of point p_i in object-space.

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 which is centered on point $\mathbf{p}_i$ and parameterized by its radius r_i. For the remainder we will limit us to circular disks, but elliptical splats can be handled analogously.

Splats s_j from occluded surface layers must not contribute to the final color in Eq. 1. For this to work, an ε-z-buffer visibility test [RPZ02, BK03, BSK04, ZRB*04, PSG04] discards any fragments from hidden splats s_j farther back than some ε from the nearest contribution of a visible splat s_i.

Since GPUs do not offer such a *fuzzy* visibility z-test, hardware accelerated implementations of Eq. 1 resort to a 2+1-pass rendering algorithm. First, all point samples in $\mathcal{S}$ are rendered, without shading but applying an ε offset, such as to initialize a 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 of the first rendering pass together with the disabled z-buffer writing in the second achieves the desired ε-z-visibility. In a third image normalization post-processing pass, the final fragment color $\frac{\mathbf{c}_{\text{rgb}}(f)}{\mathbf{c}_\alpha(f)}$ is generated as indicated by Eq. 1.

The first two passes are expensive iterations over the point geometry data not only due to the transform & lighting cost, but also in particular due to the complex vertex and fragment shaders required to rasterize depth-corrected elliptical splats in image-space [BSK04, ZRB*04, BHZK05].

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 into an image post-processing pass.

We note, as illustrated in Fig. 2, that if a given 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 for a moment only consider splats of $\mathcal{S}$ which are part of the nearest visible surface layer and that all other occluded splats can be ignored. Assuming such non-overlapping point groups $\mathcal{S}_k$, the accumulation in Eq. 1 can be separated into summations over the individual groups as follows:

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

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 complete rendering result can then be formed by an image compositing step over all partial images I_k,

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

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

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

for the only splat $s_i \in \mathcal{S}_k$ that covers the fragment f. Hence each fragment of I_k gets the contribution from exactly one – the only visible – splat in $\mathcal{S}_k$. Therefore, no more α-blending and ε-z-buffer visibility culling is required to generate the image I_k of an individual point group.

If the group $\mathcal{S}_k$ only contains splats s_i of the nearest visible layer not overlapping in image-space, then Eq. 4 can easily be implemented as the splats $s_i \in \mathcal{S}_k$ only 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 RGBα-channels. For all groups this requires exactly one full traversal of the point data 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 $\mathcal{S}_k$ will not only contain 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 $|\mathbf{p}_i - \mathbf{p}_j| \geq r_i + r_j$, 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. 3. 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 corresponding images $I_{l \neq k}$ will contain the necessary data required to perform ε-z-buffer visibility culling as is described below. For this, the images I_k additionally include per-fragment depth information $\mathbf{c}_d(f)_k$.

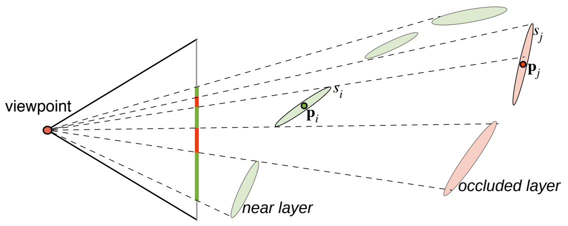


Figure 3: *For each point group $\mathcal{S}_k$, any fragments generated by splats s_i from the nearest visible surface layer will win the z-buffer visibility determination over any occluded splats s_j and will be kept in the image I_k.*

The depth-images I_k of all point groups $\mathcal{S}_k$ can then be combined, as suggested in Fig. 4, using the depth information to perform the ε-z-buffer visibility culling as outlined in the previous section. We can now outline the image compositing operation ⊕ over all K depth-images I_k to compute Eq. 3 under the ε-z-visibility constraint (given in Fig. 5).

The conservative ε-z-buffer visibility test is implemented in Fig. 5 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.

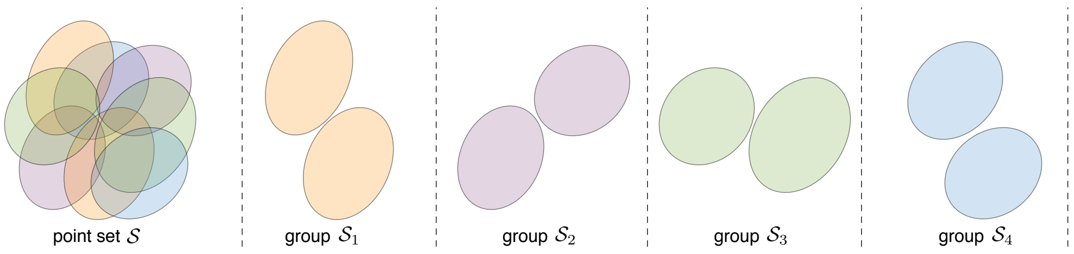


Figure 2: *Separation of the input point set $\mathcal{S}$ into non-overlapping sub-sets $\mathcal{S}_k$.*

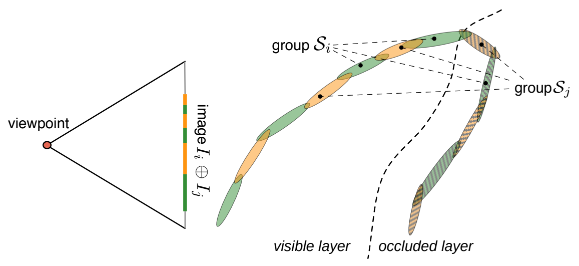


Figure 4: *Contributions from multiple depth-images I_k can be visibility culled and blended into the final result $I = \oplus_k I_k$, taking the z-depth and ε tolerance into account.*

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

```
1  foreach $f \in I$ do
2    $\mathbf{c}_{\text{rgb}}(f) = \mathbf{0}$;
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 $\mathbf{c}_d(f)_k \leq d + \varepsilon$ then
7        $\mathbf{c}_{\text{rgb}}(f) = \mathbf{c}_{\text{rgb}}(f) + \mathbf{c}_{\text{rgb}}(f)_k$;
8        $\mathbf{c}_\alpha(f) = \mathbf{c}_\alpha(f) + \mathbf{c}_\alpha(f)_k$;
9      endif
10   endfor
11   $\mathbf{c}_{\text{rgb}}(f) = \frac{\mathbf{c}_{\text{rgb}}(f)}{\mathbf{c}_\alpha(f)}$;
12 endforeach
```

Figure 5: *Post-process image compositing performing smooth point interpolation as well as ε-z-visibility testing.*

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 [ZPvBG01, BSK04, BHZK05] or Voronoi rasterization [TCH05] can also be 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 point surfaces 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. 6-a), transparency blending between 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 approach is illustrated in Fig. 6-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 $\alpha_2 \cdot a_2 + (1-\alpha_2)(\alpha_1 \cdot a_1 + (1-\alpha_1) \cdot \mathit{background})$. 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 $I = 1/2 \cdot (I_A + I_B)$.

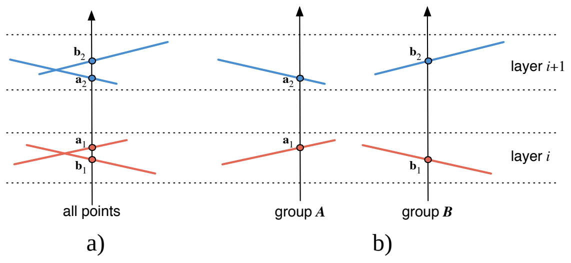


Figure 6: *a) Traditional PBR cannot distinguish between point interpolation and transparency α-compositing during per-fragment blending. b) Dividing points into groups A and B: a_1, b_1 are transparency α-blended with a_2, b_2 respectively, and then PBR-interpolated in an image compositing post-process.*

Note that point blending kernels cannot be supported in the above outlined approach 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 simplified PBR-blending 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 1/256 have no effect. Moreover, the artifacts can be made virtually unnoticeable by separately considering the nearest of the transparent layers. Thus we can render the *nearest layer* exclusively and separately in high quality using smooth point blending kernels.

Furthermore, we observe that the above concept works well if points within a group have minimal overlap, as no interpolation will be performed within a single group. Additionally, 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. Minimal Independent Grouping

The division of $\mathcal{S}$ into K groups $\mathcal{S}_{k=0...K-1}$ as discussed above can be formulated as a *graph coloring* problem which is conducted in a pre-process prior to rendering.

4.1. Basic Grouping

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 $|\mathbf{p}_i - \mathbf{p}_j| \geq r_i + r_j$. 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

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

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* of $\mathcal{G}$ [JT94], and the number K of groups is $\mathcal{G}$'s *chromatic number* $\mathcal{X}(G)$. Since minimal graph coloring is an NP-hard problem we apply an approximate solution as described below. Nevertheless, since $\mathcal{X}(G) \leq \Delta(\mathcal{G})$, the maximal degree of $\mathcal{G}$, we know an upper bound on K for a given point sample set $\mathcal{S}$.

We use the *Largest First* (LF) graph coloring algorithm [Lei79] to solve our point grouping problem. Given an ordered of nodes $\mathcal{O} = [v_0, \dots, v_{n-1}]$ ($v_i \in \mathcal{V}$) 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 $v_0, \dots, v_i$ (with $i \geq 0$) 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 approach the minimum graph coloring problem, it is very efficient and achieves almost the same results as 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 the image compositing post-process. Furthermore, current generation GPUs support only up to 16 texture samplers in the fragment shader, which would cause the image compositing process to take multiple passes for $K > 16$. 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 do not include edge $e_{i,j}$ in $\mathcal{E}$. 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, thus 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. Extended Grouping

The above basic grouping algorithm 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 single group $\mathcal{S}_k$. The overlapping splats will be transparency-blended back-to-front into image I_k which may results 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.

4.2.1. 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:

$$\tilde{d}_f = floor\left(\frac{d_f - d_{min}}{d_{max} - d_{min}} \cdot n\right) \cdot n^{-1} \qquad (6)$$

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 $\frac{d_{\max}-d_{\min}}{\varepsilon}$ based on the ε-z-buffer offset.

4.2.2. Surface Coverage

The solution to covering the object 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 $p_i^{\text{new}} =$ *Clipped*$(\mathbf{p}_i, r, k)$ as $\mathcal{S}_k$ may have changed. (with $p_i^{\text{new}} \geq p_i$)
   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 $p_i = p_i^{\text{new}}$ 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

$$CoveredArea = n \cdot \pi r^2 - \sum_{\forall \mathbf{p}_i \in \mathcal{S}_k} Clipped(\mathbf{p}_i, r, k) \qquad (7)$$

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. 7 for all points in $\mathcal{S}$. Enlarging the splat radii to $\tilde{r}$ should achieve:

$$A \equiv n \cdot \pi \tilde{r}^2 - \sum_{\forall \mathbf{p}_i \in \mathcal{S}_k} Clipped(\mathbf{p}_i, \tilde{r}, k) \qquad (8)$$

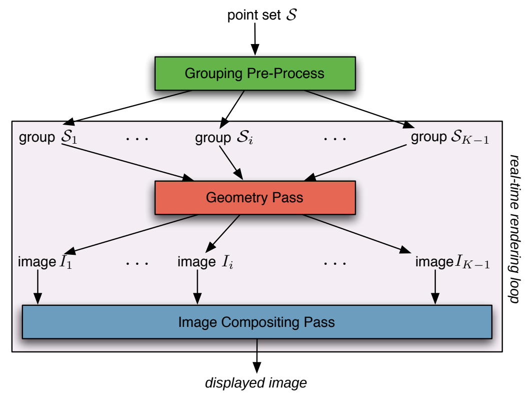
Notice that an enlarged radius $\tilde{r} > r$ also causes increased clipping $Clipped(\mathbf{p}_i, \tilde{r}, k) > Clipped(\mathbf{p}_i, r, k)$. Based on this observation, a simple iterative solution of Eq. 9 for $\tilde{r}_{s+1}$ is applied until the difference between $\tilde{r}_s$ and $\tilde{r}_{s+1}$ is small enough (with $\tilde{r}_0 = r$).

$$n \cdot \pi \tilde{r}_{s+1}^2 = A + \sum_{\forall \mathbf{p}_i \in \mathcal{S}_k} Clipped(\mathbf{p}_i, \tilde{r}_s, k) \qquad (9)$$

5. Rendering Algorithm

5.1. Rendering Opaque Point Surfaces

Based on the deferred blending concept and the grouping solution, we can describe our basic rendering algorithm as illustrated in Fig. 7. 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 pre-process and a second image compositing pass over the corresponding partial depth-images I_k.



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 $\mathbf{c}_{rgb}(f)_k$ 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. 5

Figure 7: *Overview of 1+1-pass point rendering algorithm.*

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 overlaps of splats s_i and s_j belonging to the same group $\mathcal{S}_k$. In fact, the rendering algorithm in Fig. 7 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 only one fragment, the nearest with smallest depth, from either s_i or s_j will survive.

To avoid disturbing artifacts due to flaps of overlapping splats resulting from the above simple z-visibility culling, Voronoi point rasterization can be used [TCH05]. In areas of 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 $w_j(\mathbf{f}_j) \leq w_i(\mathbf{f}_i)$ 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 [TCH05] we do not introduce an extra rendering pass to implement Voronoi rasterization but realize this by outputting an Voronoi enhanced depth value in addition to the regular z-depth on line 9 of the *Geometry Pass* in Fig. 7. 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 modifed z-depth value as

$$z = z_{lowres} + z_{voronoi} = \tilde{d}_f + \frac{d_i}{r_i} \cdot n^{-1}, \qquad (10)$$

where $\tilde{d}_f$ is defined in Eq. 6, r_i is the splats disk radius and n is an integer constant. The constant n is defined in Eq. 6.

The first term z_{lowres} is a low-precision depth which limits the depth values of all fragments to the range $[0, \frac{1}{n}, \frac{2}{n}, \ldots, 1]$. 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 $[0, \frac{1}{n}]$. 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_{lowres}, with the minimum difference of $\frac{1}{n}$ being larger than the maximum $z_{voronoi}$, and thus be culled.

In fact, the enhanced depth value of Eq. 10 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.

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 organization and traversal ([Sam89]). 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-2:

1. *Geometry Pass (Transparency-blending)*: Turn on z-test and α-blending. Render all splats $\mathbf{p}_i$ of each group $\mathcal{S}_k$ using modified radii $\tilde{r}_i$ into separate target images I_k. Perform back-to-front α-blending (using the material opacity for α and $1-\alpha$). Adjust the fragment depth according to Eq. 6.
2. *Compositing Pass (PBR-blending)*: Combine (average) all K images I_k into final frame buffer.

Algorithm-2 implements the basic transparent point rendering concept. 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. 1-a) or 13-a).

5.2.2. High-Quality Transparency

The point interpolation artifacts in Algorithm-2 can further be reduced by rendering the closest transparent surface layer separately and in higher quality (Figs. 1-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-2. The two sets of images are then combined into a high-quality blended final result. Observe that α-blending of *far layers* is conducted in the geometry pass while α-blending with the *nearest layer* is achieved in the image compositing pass. 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-3:

1. *Geometry Pass for Nearest Layer*: Use the geometry pass of Algorithm-1 to render the point groups $\mathcal{S}_k$ 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-2 to render the point groups $\mathcal{S}_k$ 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 F_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 $\frac{\sum h_{\widehat{f_k}} \cdot \mathbf{c}_{\widehat{f_k}}}{\sum h_{\widehat{f_k}}}$. Then average the images O_k into C_O for the other layers. Finally high-quality transparency is achieved given the opacity α by $I = \alpha \cdot C_F + (1-\alpha) \cdot C_O$.

Note that our transparency algorithms support varying material opacities, possibly different for each individual point splat, as the α values can be specified for each point sample 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-3.

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-3 causes accumulation of opacity over all layers $\alpha_{\text{total}} = \sum_{\text{layers}} \alpha_i$, that is in each image O_k separately for each group $\mathcal{S}_k$. Assuming a constant material opacity α we derive the number of layers from $l = \frac{\alpha_{\text{total}}}{\alpha}$.

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

$$AbsorptionRatio = (1-\alpha)^l \quad (11)$$

For multi-layer refraction effects, we simulate a transmitted total refraction angle θ_T by Eq. 12 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:

$$\sin_{\theta_T} = \eta^l \cdot \sin_{\theta_I} \quad (12)$$

Although Eqs. 11 and 12 are not physically correct, they produce appealing visual multi-layer transparency effects (see also Section 6).

Additional lighting phenomena, also shown in Figs. 1-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 [ZPvBG01, BHZK05]. 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-3. 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 bumb-mapping.

6. Experimental Results

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

6.1. Rendering Opaque Point Surfaces

The first experiments are with respect to the graph coloring based point grouping algorithm described in Section 4. As point-based surface models inherently depend on a significant overlap ratio between neighboring splats to generate a smoothly blended surface rendering, it comes at 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.

In Fig. 8 we show different rendering results for different overlap relaxation parameters c. We can see that, in comparison to a 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. 10 is demonstrated in Fig. 9. It shows the effective removal of flaps between overlapping splats and the resulting faceted surface similar to [TCH05]. This surface model is basically the depth-map, combined from all I_k, for the ε-z-buffer visibility test in conventional PBR.

Rendering performance is demonstrated in Table 2. We

Model	Points $|\mathcal{S}|$	$K = \mathcal{X}(G)$ / maxDegree / avgDegree			
		$c = 1.0$	$c = 0.8$	$c = 0.6$	$c = 0.4$
David Head	2,000K	18 / 37 / 17.2	14 / 31 / 11.6	11 / 24 / 9.4	7 / 8 / 3.9
Dragon	1,100K	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 1: *Graph coloring point grouping results for different overlap relaxation parameters c.*

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 an image compositing step. The *Image Compositing Pass* in Figs. 7 and 5 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 2 Voronoi rasterization is disabled as the grouping of points is so effective that no significant point overlaps are 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).

Model	Points $|\mathcal{S}|$	FPS		
		2+1-pass	c=0.8	c=0.4
D-Head	2,000K	0.96	1.2	1.4
Dragon	1,100K	15.04	19.70	22.62
Female	303K	32.65	32.11	37.76
Balljoint	137K	65.68	52.96	70.37

Table 2: *Frame rate performance of the novel 1+1-pass point rendering algorithm compared to a standard 2+1-pass PBR implementation.*

Additional 1+1-pass rendering results are presented in Fig. 10, demonstrating smooth images at improved frame rates for large models.

6.2. 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. 11 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 4.2.2. 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.

If a larger K is required, the radius enlargement method (2) described in Section 4.2.2 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.

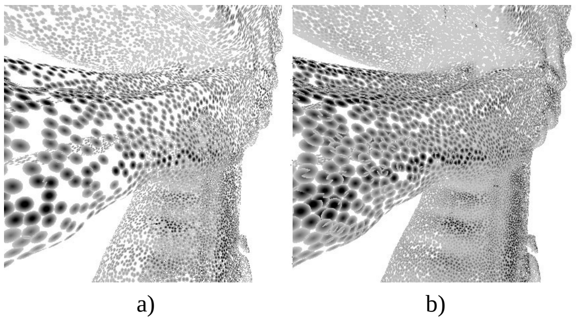


Figure 11: *Grouping results. a) Splats have smaller overlaps but less surface coverage for* $K = 8$. *b) Splats have bigger overlaps but better surface cover for* $K = 4$.

The basic frame rate for different transparent point rendering algorithms are: our 1+1-pass transparent point rendering Algorithm-2 achieves 9 FPS, on the other hand our high-quality 2+1-pass Algorithm-3 reaches 5 FPS. This compares very well to depth-peeling, which attains only less than 2 FPS for an upper limit of 8 layers. For comparison, a standard opaque point splatting algorithm reaches 14 FPS.

In Fig. 12, our transparent PBR algorithms are compared to depth-peeling which generates the correct back-to-front α-blending result. In contrast to depth-peeling, which conducts smooth point interpolation on each surface layer by a standard opaque rendering method, our algorithms perform the point interpolation for all layers in Algorithm-2, and except for the nearest visible layer in Algorithm-3. We can observe that any so introduced visual artifacts are masked by the transparency attenuation and are hardly visible using Algorithm-2, and virtually no visual difference can be observed using Algorithm-3.

Several small bouncing opaque balls are added to the

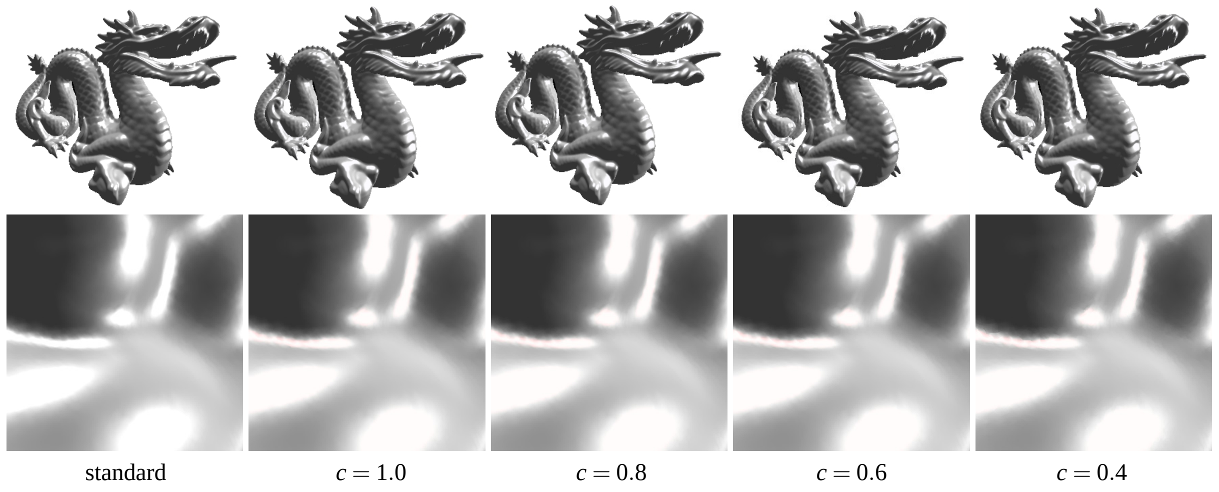


Figure 8: *Comparison of smooth point blending results for different overlap relaxation parameters c with respect to a standard PBR blending.*

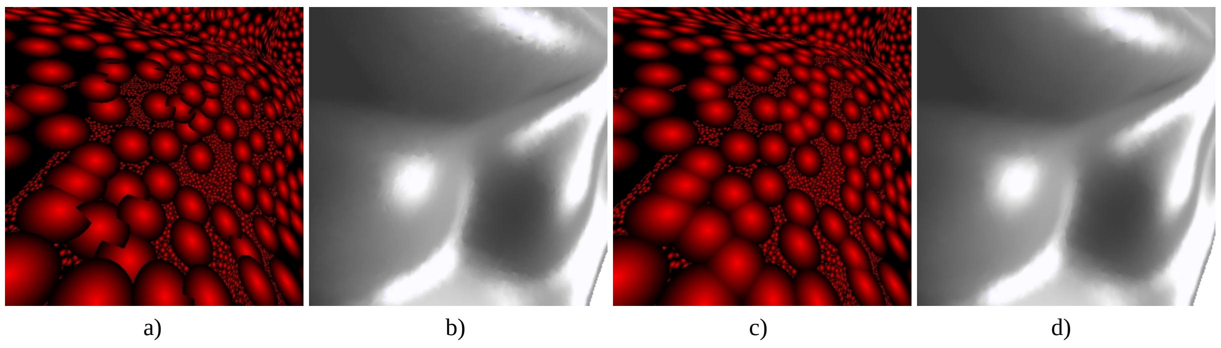


Figure 9: *Voronoi rasterization. In a) and b) we show the rasterization and shading examples without Voronoi rasterization enabled, hence fragments with smaller z-depth simply override any other. In c) and d), fragments with smaller Voronoi-depth as defined in Eq. 10 win the z-buffer visibility test.*

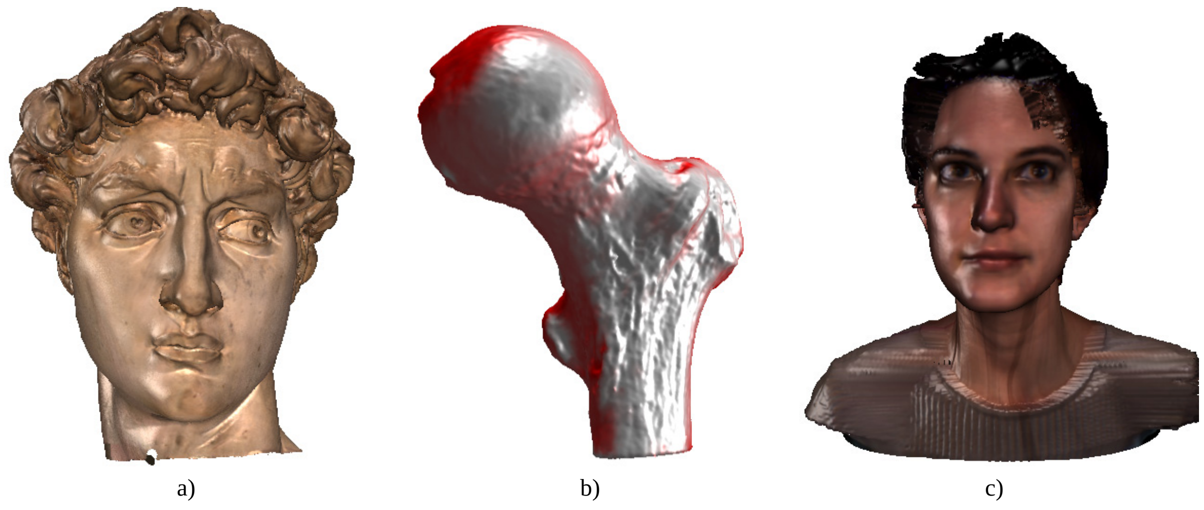


Figure 10: *Rendering results for various point models. a) David head model rendered at 1.4 FPS, b) Balljoint model rendered at 70 FPS and c) Female model displayed at 37 FPS, using* $c = 0.4$ *and Voronoi rasterization.*

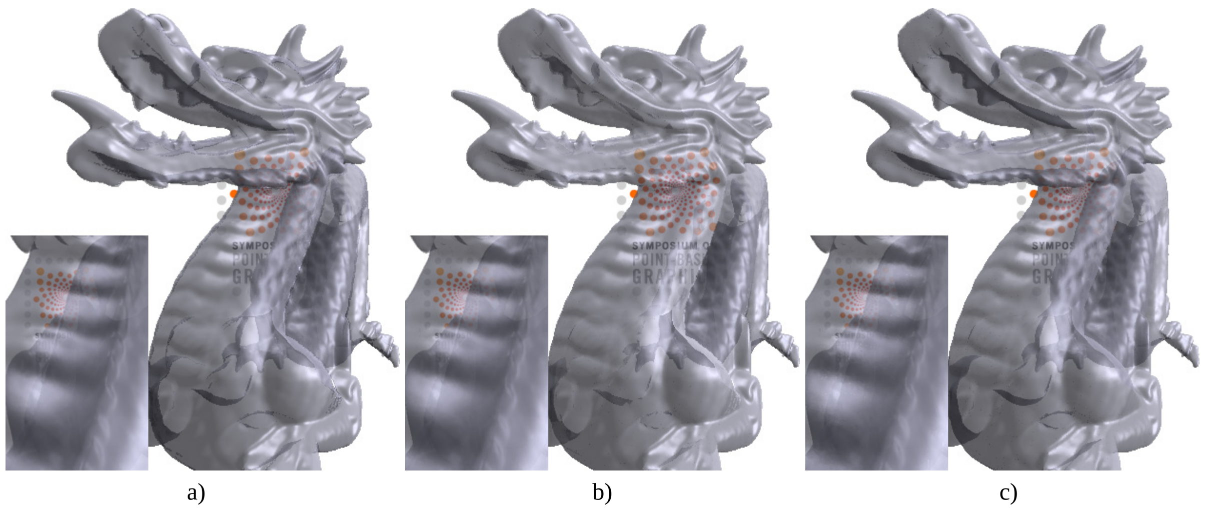


Figure 12: *Transparent image rendering quality for a) depth-peeling, b) Algorithm-3 and c) Algorithm-2.*

scene in Figs. 1-a), b) and Fig. 13 to verify that our algorithms generate the correct α-blending results when combining opaque and transparent objects. We demonstrate in Fig. 13 that our 1+1-pass transparent PBR algorithm achieves high visual rendering quality for viewing configurations which do not exhibit extreme close-up views.

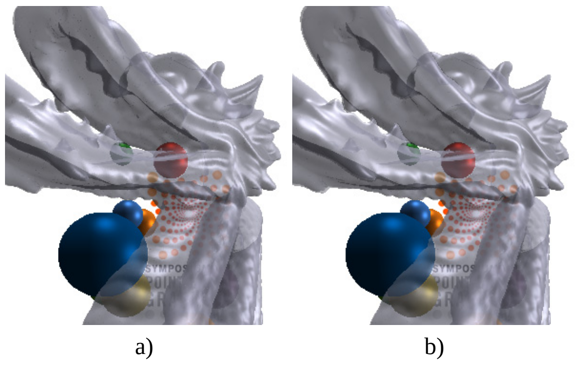


Figure 13: *Opaque and transparent objects, a) 1+1-pass Algorithm-2 and b) 2+1-pass Algorithm-3.*

Figs. 1-c), d) and Fig. 14 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. 14 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. Conclusion

This paper presents a new framework for GPU accelerated PBR algorithm based on the concept of deferred blending.

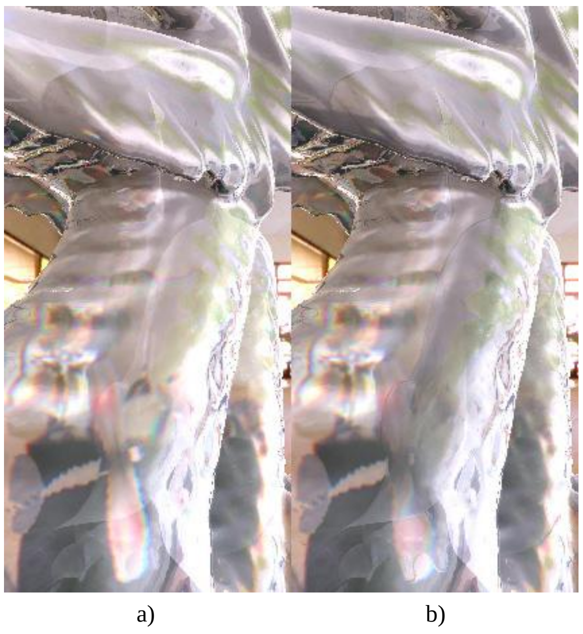


Figure 14: *a) Single-layer, b) multi-layer transparent refraction and specular reflection environment mapping effects.*

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 geometry data. The rendered images show that our algorithm can provide 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 the rendering of transparent surfaces, the major challenge of handling the conflicting point interpolation and transparent α-blending simultaneously is solved by separating them to different rendering passes. We have not only provided the first GPU accelerated approaches to render transparent point surfaces, but in fact presented a basic transparency α-blending of multiple transparent point layers in a single geometry processing pass over the point data. Our Algorithm-3 achieves very high-quality transparency blending and incorporates effective simulations of multi-layer refraction and reflection effects.

Acknowledgements

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

[BHZK05] BOTSCH M., HORNUNG A., ZWICKER M., KOBBELT L.: High-quality surface splatting on today's GPUs. In *Proceedings Symposium on Point-Based Graphics* (2005), Eurographics Association, pp. –.

[BK03] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern GPUs. In *Proceedings Pacific Graphics 2003* (2003), IEEE, Computer Society Press, pp. 335–343.

[BK05] BOTSCH M., KOBBELT L.: Real-time shape editing using radial basis functions. *Computer Graphics Forum 24*, 3 (2005), 611–621. Eurographics 2005 Proceedings.

[BSK04] BOTSCH M., SPERNAT M., KOBBELT L.: Phong splatting. In *Proceedings Symposium on Point-Based Graphics* (2004), Eurographics, pp. 25–32.

[CAZ01] COHEN J. D., ALIAGA D. G., ZHANG W.: Hybrid simplification: Combining multi-resolution polygon and point rendering. In *Proceedings IEEE Visualization* (2001), pp. 37–44.

[CH02] COCONU L., HEGE H.-C.: Hardware-oriented point-based rendering of complex scenes. In *Proceedings Eurographics Workshop on Rendering* (2002), pp. 43–52.

[CN01] CHEN B., NGUYEN M. X.: POP: A hybrid point and polygon rendering system for large data. In *Proceedings IEEE Visualization* (2001), pp. 45–52.

[DH02] DEY T. K., HUDSON J.: PMR: Point to mesh rendering, a feature-based approach. In *Proceedings IEEE Visualization* (2002), Computer Society Press, pp. 155–162.

[Eve02] EVERITT C.: Interactive order-independent transparency. Technical Report, 2002.

[Gro01] GROSS M. H.: Are points the better graphics primitives? Computer Graphics Forum 20(3), 2001. Plenary Talk Eurographics 2001.

[JT94] JENSEN T. R., TOFT B.: *Graph Coloring Problems*. Wiley-Interscience, 1994.

[KB04] KOBBELT L., BOTSCH M.: A survey of point-based techniques in computer graphics. *Computers & Graphics 28*, 6 (2004), 801–814.

[KV01] KALAIAH A., VARSHNEY A.: Differential point rendering. In *Proceedings Eurographics Workshop on Rendering Techniques* (2001), Springer-Verlag, pp. 139–150.

[Lei79] LEIGHTON F. T.: A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards 84* (1979), 489–506.

[Mam89] MAMMEN A.: Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Computer Graphics & Applications 9*, 4 (July 1989), 43–55.

[PG04] PFISTER H., GROSS M.: Point-based computer graphics. *IEEE Computer Graphics and Applications 24*, 4 (July-August 2004), 22–23.

[PKKG03] PAULY M., KEISER R., KOBBELT L., GROSS M.: Shape modeling with point-sampled geometry. *ACM Transactions on Graphics 22*, 3 (2003), 641–650.

[PSG04] PAJAROLA R., SAINZ M., GUIDOTTI P.: Confetti: Object-space point blending and splatting. *IEEE Transactions on Visualization and Computer Graphics 10*, 5 (September-October 2004), 598–608.

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proceedings ACM SIGGRAPH* (2000), ACM SIGGRAPH, pp. 335–342.

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Proceedings EUROGRAPHICS* (2002), pp. 461–470. also in Computer Graphics Forum 21(3).

[Sam89] SAMET H.: *The Design and Analysis of Spatial Data Structures*. Addison Wesley, Reading, Massachusetts, 1989.

[SP04] SAINZ M., PAJAROLA R.: Point-based rendering techniques. *Computers & Graphics 28*, 6 (2004), 869–879.

[SPL04] SAINZ M., PAJAROLA R., LARIO R.: Points reloaded: Point-based rendering revisited. In *Proceedings Symposium on Point-Based Graphics* (2004), Eurographics Association, pp. 121–128.

[TCH05] TALTON J. O., CARR N. A., HART J. C.: Voronoi rasterization of sparse point sets. In *Proceedings Symposium on Point-Based Graphics* (2005), Eurographics Association, pp. 33–37.

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3D: An interactive system for point-based surface editing. In *Proceedings ACM SIGGRAPH* (2002), ACM Press, pp. 322–329.

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings ACM SIGGRAPH* (2001), ACM SIGGRAPH, pp. 371–378.

[ZRB*04] ZWICKER M., RÄSÄNEN J., BOTSCH M., DACHSBACHER C., PAULY M.: Perspective accurate splatting. In *Proceedings of Graphics Interface* (2004), pp. 247–254.