Confetti: Object-Space Point Blending and Splatting

Renato Pajarola, *Member*, *IEEE Computer Society*,
Miguel Sainz, *Member*, *IEEE Computer Society*, and Patrick Guidotti

Abstract—In this paper, we present Confetti, a novel point-based rendering approach based on object-space point interpolation of densely sampled surfaces. We introduce the concept of a transformation-invariant covariance matrix of a set of points which can efficiently be used to determine splat sizes in a multiresolution point hierarchy. We also analyze continuous point interpolation in object-space and we define a new class of parameterized blending kernels as well as a normalization procedure to achieve smooth blending. Furthermore, we present a hardware accelerated rendering algorithm based on texture mapping and α-blending as well as programmable vertex and pixel-shaders.

Index Terms—Point-based rendering, multiresolution modeling, level-of-detail, hardware accelerated blending.

1 Introduction

Point-based surface representations have recently been established as viable graphics rendering primitives [1]. The advantage of point-based representations over triangle meshes is that explicit surface reconstruction (i.e., [2], [3], [4]) and storage of mesh connectivity is avoided. The major challenge of *point-based rendering* (PBR) methods is to achieve a continuous interpolation between discrete point samples that are irregularly distributed over a smooth surface. Furthermore, it must support correct visibility as well as efficient rendering. In this paper, we propose a novel point blending and rendering technique that is based on the direct interpolation between point samples in 3D and define the blending of surface points as a weighted interpolation in object-space. We analyze the smooth interpolation between points in object-space and define a new class of parameterized blending kernels. Also, we provide an efficient technique to calculate splat sizes in a multiresolution point hierarchy. Furthermore, our approach exploits hardware acceleration. An example rendering of our approach for the textured Female model is shown in Fig. 1. The contributions of this paper are threefold. First, we introduce the notion of a transformation-invariant homogeneous covariance of a set of points to efficiently compute hierarchical LOD splat sizes. Second, we analyze and define object-space blending functions that smoothly interpolate the texture color of irregularly distributed point samples. Third, we present an efficient hardware accelerated point rendering algorithm for visibility splatting and color blending.

- *R. Pajarola and M. Sainz are with the Computer Science Department, University of California at Irvine, Irvina, CA 92697. E-mail: pajarola@acm.org, msainz@ice.uci.edu.*
- *P. Guidotti is with the Department of Mathematics, University of California at Irvine, Irvine, CA 92697. E-mail: gpatrick@math.uci.edu.*

Manuscript received 25 Aug. 2003; revised 3 Dec. 2003; accepted 8 Dec. 2003. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0075-0803.

The remainder of the paper is organized as follows: Section 2 discusses related work and Section 3 the effective determination of elliptical splat sizes in a multiresolution point hierarchy. In Section 5, we introduce our concept of object-space point interpolation and a new class of parameterized blending kernels. Our point rendering algorithm is described in Section 6. Section 7 presents experimental results and Section 8 concludes the paper.

2 Related Work

Points as rendering primitives have been discussed in [5] but only later were rediscovered as a practical approach to render complex objects [6], [7]. *QSplat* [7] is a fast point rendering system that uses a memory efficient region-octree hierarchy for LOD selection and simple point rendering for fast display. Another memory efficient multiresolution point hierarchy is proposed in [8] along with a precomputed footprint table rendering approach. The *surfels* approach [9] generates a hierarchical orthographic sampling of an object in a preprocess and surfel colors are obtained by texture prefiltering. Its rendering algorithm is a combination of approximate visibility splatting, texture mipmapping, and image filtering to fill holes. In [10], the principle of EWA texture filtering [11] is applied to irregularly sampled texture information on 3D surfaces, and a two-pass hardware accelerated rendering method is given in [12]. Recent approaches such as [13] or [14] address high-speed rendering of point data by exploiting hardware acceleration and on-board graphics memory as geometry cache. More complex per-point surface attributes are used in [15], [16] to render points as normal mapped surface elements for high rendering quality at low sampling rate.

Independently of our work, hardware accelerated blending and rendering of points in object-space using elliptical disks with per-pixel blending and normalization has been proposed in [12] and [13]. We compare our rendering performance to prior fast PBR systems [7], [12], [13], [14] in Section 8.

Fig. 1. Female head model at $\tau = 1$ pixels screen tolerance (rendered 113,012 out of 302,948 points at 2,647,209 splats per second).

Further related techniques include the integration of polygons and points [17], [18], [19], [20], simplification [21], [22], interactive editing [23], and sampling of procedural objects [24]. Also, the randomized z-buffer method [25] is closely related to PBR.

3 Surface Representation

3.1 Point-Sampled Geometry

In this project, we consider blending and rendering techniques for surfaces represented as dense sets of point-samples organized in a space-partitioning multiresolution hierarchy. The surface elements may be irregularly distributed on the surface; however, here we assume that the input reasonably samples the surface (i.e., satisfies the Nyquist or other sufficient surface sampling criteria as in [26]). Like other work, our approach assumes that the point set reasonably samples the surface's color texture.

The full resolution input data set consists of point samples s with attributes for coordinates $\boldsymbol{p}$, normal orientation $\boldsymbol{n}$, and surface color $\boldsymbol{c}$. Furthermore, it is assumed that each sample also contains the information about its spatial extent in object-space. This *size* information specifies an elliptical disk e centered at $\boldsymbol{p}$ and perpendicular to $\boldsymbol{n}$. For correct visibility, these elliptical disks must cover the sampled object nicely without holes and, thus, overlap each other in object-space as shown in Fig. 2. As noted in [9], tangential planar disks may not completely cover a surface if it is strongly bent or under extreme perspective projections. However, this is not often noticeable in practical situations.

An elliptical disk e consists of major and minor axis directions $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ and their lengths. Together with the normal $\boldsymbol{n}$, the axis directions $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ define the local tangential coordinate system of that point.

In the remainder of this section, we focus on the efficient generation of a spatial-partitioning multiresolution hierarchy defined over the input point set and we assume that the point splat sizes of the input are already given. Initial splat ellipses can be derived from locally computed Voronoi cells

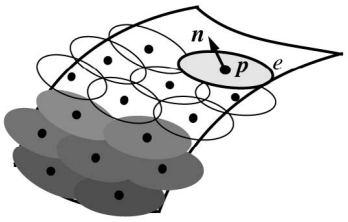


Fig. 2. Elliptical surface elements covering a smooth and curved 3D surface.

[19], [27] or from local neighborhood and covariance analysis [22], as described in Section 4.3.

3.2 Multiresolution Hierarchy

The multiresolution point representations considered in this paper are hierarchical space-partitioning data structures [28], [29]. Each cell or node c of such a hierarchy H, containing a set of k samples $S_c = \{s_1 \dots s_k\}$ has a representative sample s_c with average coordinates $\boldsymbol{p}_c = k^{-1} \sum_{i=1}^{k} \boldsymbol{p}_k$, as well as average normal $\boldsymbol{n}_c$ and color $\boldsymbol{c}_c$. Furthermore, for efficient view-frustum and back-face culling, each cell $c \in H$ also includes the sphere radius r_c and normal-cone [30] semi-angle θ_c bounding all samples in S_c. Several conforming space-partitioning multiresolution hierarchies have been proposed for PBR [7], [8], [22] which can be used with our techniques. In our work, we use a *point-octree* [29], [31] which partitions the space adaptively to the sample distribution (data-driven) rather than regularly in space (space-driven) like *region-octrees* [29], [31] which have been proposed more commonly.

Given n input points $s_1 \dots s_n$, a point-octree can efficiently be generated by a single depth-first traversal in $O(n \log n)$. In point-octrees, the 1-to-8 subdivision is determined by the average of the points in each node. Therefore, given the k points $S_c = \{s_1 \dots s_k\}$ of a node $c \in H$ and the average position $\boldsymbol{p}_c$, S_c is partitioned into sets S_1 to S_8 according to the eight octants defined by the split coordinate $\boldsymbol{p}_c$.

As shown in Fig. 3, the input data to each node c not only consists of the samples S_c but also of the average position $\boldsymbol{p}_c$ (split coordinate), normal $\boldsymbol{n}_c$, and color $\boldsymbol{c}_c$. Therefore, while dividing S_c into subsets S_i, their averages $\boldsymbol{p}_i$, $\boldsymbol{n}_i$, and $\boldsymbol{c}_i$ are computed. In one linear traversal of the points in S_c, the current node c does the following:

1. Divides S_c into the subsets S_i with $s \in S_i$ in octant i with respect to split coordinate $\boldsymbol{p}_c$,
2. Accumulates averages $\boldsymbol{p}_i$, $\boldsymbol{n}_i$, and $\boldsymbol{c}_i$ for each S_i,
3. Computes bounding sphere radius r_c of all points in S_c with respect to center $\boldsymbol{p}_c$, and

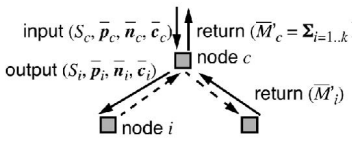


Fig. 3. Top-down and bottom-up point-octree generation data flow.

4. Computes bounding normal cone semi-angle θ_c of all normals in S_c with respect to center normal $\boldsymbol{n}_c$.

This process is repeated recursively passing the information $(S_i, \boldsymbol{p}_i, \boldsymbol{n}_i, \boldsymbol{c}_i)$ to the eight child nodes c_i.

As noted previously, each LOD node c stores its spatial extent in the form of an oriented elliptical disk e_c. To compute these elliptical disks efficiently, we use a novel technique based on a *generic homogeneous covariance matrix* introduced in [32]. This allows us to propagate covariance information in form of a homogeneous 4×4 matrix efficiently bottom-up during the generation of the multiresolution hierarchy, as illustrated in Fig. 3. In the following Section 4, we explain this new concept and explain how to efficiently derive the elliptical disk of a set of points therefrom.

4 Generic Homogeneous Covariance

4.1 Covariance Analysis

Since the smallest bounding ellipsoid of a set of points is hard to compute, covariance analysis—basically a singular value decomposition approach—is often used to determine an elliptical Gaussian distribution fitting a given point set [33].

The covariance matrix of a set of n points $\boldsymbol{p}_1 \dots \boldsymbol{p}_n \in \mathbf{R}^3$ and their average $\boldsymbol{p}_{\text{avg}}$ is defined by

$$M = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{p}_i - \boldsymbol{p}_{\text{avg}}) \cdot (\boldsymbol{p}_i - \boldsymbol{p}_{\text{avg}})^{\text{T}}. \tag{1}$$

The mean $\boldsymbol{p}_{\text{avg}}$ denotes the center of the ellipsoid and the unit-length eigenvectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, and $\boldsymbol{v}_3$ of M are the axis directions of the ellipsoid. The ratios of the axis lengths are given by the ratio of the eigenvalues λ_1, λ_2, and λ_3 of M. In the local coordinate system with center $\boldsymbol{p}_{\text{avg}}$, and x, y, and z-axis aligned with eigenvectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, and $\boldsymbol{v}_3$, the ellipsoid distribution is then defined by $x^2/\lambda_1^2 + y^2/\lambda_2^2 + z^2/\lambda_3^2 = f^2$. With the appropriate scaling factor f, the so-defined ellipsoid is a close approximation of the smallest bounding ellipsoid of the given point set, feasible to compute and widely used in practice.

The main problem with this formulation is the strong dependency of (1) on the mean $\boldsymbol{p}_{\text{avg}}$. Therefore, if the covariance matrices M^O and M^Q of two point sets $O = \{\boldsymbol{o}_1 \dots \boldsymbol{o}_n\}$ and $Q = \{\boldsymbol{q}_1 \dots \boldsymbol{q}_m\}$ as well as the covariance $M^{O \cup Q}$ of their union are to be calculated, (1) has to be computed individually for sets O and Q, as well as their union $O \cup Q$. Hence, the outer product $(\boldsymbol{p}_i - \boldsymbol{p}_{\text{avg}}) \cdot (\boldsymbol{p}_i - \boldsymbol{p}_{\text{avg}})^{\text{T}}$ is evaluated $|O| + |Q| + |O \cup Q| = 2 \cdot (n + m)$ times. In a multiresolution hierarchy of N points, this leads to a cost of $O(NlogN)$ outer products. Despite the fact that the cost of generating a multiresolution hierarchy is $O(NlogN)$, it is desired to avoid excessive calculations such as the expensive outer product of two vectors.

4.2 Homogeneous Covariance

In homogeneous space with $\boldsymbol{p}_i'^{\text{T}} = (\boldsymbol{p}_i^{\text{T}}, 1)$, we can rewrite the product $(\boldsymbol{p}_i - \boldsymbol{p}_{\text{avg}}) \cdot (\boldsymbol{p}_i - \boldsymbol{p}_{\text{avg}})^{\text{T}}$ to $(T \cdot \boldsymbol{p}_i') \cdot (T \cdot \boldsymbol{p}_i')^{\text{T}}$ with the transformation matrix T denoting the translation by $-\boldsymbol{p}_{\text{avg}}$. Thus, we can revise (1) to

$$\begin{aligned} M_{\mathcal{H}} &= \frac{1}{n} \sum_{i=1}^{n} (T \cdot \boldsymbol{p}_i') \cdot (T \cdot \boldsymbol{p}_i')^{\text{T}} \\ &= \frac{1}{n} \sum_{i=1}^{n} (T \cdot \boldsymbol{p}_i' \cdot \boldsymbol{p}_i'^{\text{T}} \cdot T^{\text{T}}) \\ &= \frac{1}{n}\, T \cdot \left(\sum_{i=1}^{n} \boldsymbol{p}_i' \cdot \boldsymbol{p}_i'^{\text{T}} \right) \cdot T^{\text{T}} \\ &= \frac{1}{n}\, T \cdot M_{\mathcal{GH}} \cdot T^{\text{T}}, \end{aligned} \tag{2}$$

with $M_{\mathcal{GH}} = \sum_{i=1}^{n} \boldsymbol{p}_i' \cdot \boldsymbol{p}_i'^{\text{T}}$ denoting the new *generic homogeneous covariance matrix* of points $p_1 \dots p_n$. This matrix $M_{\mathcal{GH}}$ expresses the covariance of a set of points with respect to the origin of the coordinate system. Hence, we can get the homogeneous covariance $M_{\mathcal{H}}$ with respect to any arbitrary center $\boldsymbol{o}$ given by the translation $T = (-o_x, -o_y, -o_z)$ as $M_{\mathcal{H}} = n^{-1}\, T \cdot M_{\mathcal{GH}} \cdot T^{\text{T}}$.

In fact, we can transform the covariance into any arbitrary coordinate system given by translation T and rotation R:

$$M_{\mathcal{H}} = n^{-1}\, R \cdot T \cdot M_{\mathcal{GH}} \cdot T^{\text{T}} \cdot R^{\text{T}}. \tag{3}$$

From the 4D homogeneous covariance matrix $M_{\mathcal{H}}$, we can get the 3D Cartesian M by dropping the fourth row and column of $M_{\mathcal{H}}$. This corresponds to an orthogonal projection along the homogeneous axis. Similarly, we can express the covariance in any lower-dimensional subspace and, thus, we get the covariance M_{xy} of points projected into the x, y-plane from the upper-left 2×2 submatrix of $M_{\mathcal{H}}$.

The introduced generic homogeneous covariance matrix $M_{\mathcal{GH}}$ allows efficient updates in the multiresolution hierarchy construction as indicated in Section 3. Since $M_{\mathcal{GH}}$ is invariant to transformations, the union $O \cup Q$ of two point sets O and Q can be handled efficiently: Given their covariance matrices $M^O_{\mathcal{GH}} = \sum \boldsymbol{o}_i' \cdot \boldsymbol{o}_i'^{\text{T}}$ and $M^Q_{\mathcal{GH}} = \sum \boldsymbol{q}_i' \cdot \boldsymbol{q}_i'^{\text{T}}$, the combined generic homogeneous covariance matrix $M^{O \cup Q}_{\mathcal{GH}}$ of $O \cup Q$ is given by

$$M^{O \cup Q}_{\mathcal{GH}} = M^O_{\mathcal{GH}} + M^Q_{\mathcal{GH}}. \tag{4}$$

Therefore, expensive outer products are only computed once on the leaf level of a multiresolution hierarchy, thus only $O(N)$ times. All nonleaf nodes compute their covariance matrix by componentwise addition from the child nodes by (4).

4.3 Splat-Size Determination

The elliptical disks e_c of nodes $c \in H$ must cover the surface at all levels in the multiresolution hierarchy. Based on the generic homogeneous covariance matrix, we show how to efficiently determine tangential elliptical disks for all nonleaf nodes. The basic principle is to project the points $\boldsymbol{p}_1 \dots \boldsymbol{p}_k$ of a node c onto the tangent plane $\kappa_c : \boldsymbol{n}_c \cdot (\boldsymbol{x} - \boldsymbol{p}_c) = 0$ defined by $\boldsymbol{p}_c$ and the normal $\boldsymbol{n}_c$, followed by calculating a bounding ellipse e_c in this tangent plane κ_c, as illustrated in Fig. 4.

Using the generic homogeneous covariance matrix $M_{\mathcal{GH}}$ of a node c, we perform the following steps:

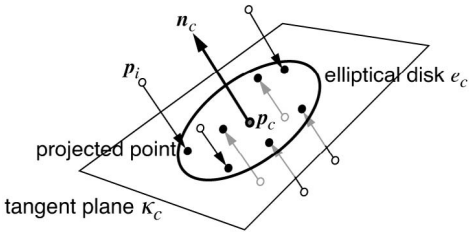


Fig. 4. Projection of points onto tangent plane κ_c at position p_c and with normal n_c.

1. Express the covariance in the 2D subspace of the tangent plane κ_c,
2. Compute the elliptical distribution of the projected points in κ_c, and
3. Adjust the ellipse equation to bound all projected points in κ_c.

Let us first outline how to get the ellipse axis and its axis-ratio in the tangent plane κ_c. For a node c and the covariance matrix $M_{\mathcal{GH}}$ of its points $p_{i=1\ldots k}$, we apply a coordinate system transformation RT according to (3) to the local tangent space. Thus, the translation matrix T has the last column being $(-p_x, -p_y, -p_z, 1)^{\mathrm{T}}$ from the node's position p_c and the rotation matrix R has the row vectors R_x, R_y, R_z defined by the normal n_c as: $R_x = R_y \times R_z$, $R_y = (0, -n_z, n_y, 0)$, and $R_z = (n_x, n_y, n_z, 0)$. The resulting transformed homogeneous matrix $M_{\mathcal{H}} = k^{-1}\, RT \cdot M_{\mathcal{GH}} \cdot T^{\mathrm{T}} R^{\mathrm{T}}$ then expresses the covariance in the local tangent-space coordinate system. Moreover, the covariance M_{κ_c} of the points $p_{i=1\ldots k}$ projected in the tangent plane κ_c is given by the upper-left 2×2 submatrix of $M_{\mathcal{H}}$.

Next, we get the axis-ratio of an ellipse fitting the points in κ_c from the eigenvalue decomposition of M_{κ_c}. We obtain the eigenvalues λ_1 and λ_2 from solving the quadratic equation $\lambda^2 + \mathrm{trace}(M_{\kappa_c}) \cdot \lambda + \det(M_{\kappa_c}) = 0$. Furthermore, we obtain the major and minor ellipse axis orientations in the tangent plane κ_c from the eigenvectors v_1 and v_2 by solving $M_{\kappa_c} \cdot v_i = \lambda_i \cdot v_i$. Note that v_1 and v_2 are given in $\mathbf{R^2}$, the tangent plane κ_c. However, with respect to the tangent-space coordinate system with the z-axis perpendicular to κ_c, we get vectors in $\mathbf{R^3}$ by $v_i'^{\mathrm{T}} = (v_i{}^{\mathrm{T}}, 0)$. Then, the world-coordinate system ellipse axis e_1 and e_2 are obtained by applying the inverse rotation R^{-1} and normalization to unit length, $e_i = R^{-1} \cdot v_i'/|v_i'|$. Now, we have defined an elliptical disk e_c in the world coordinate system with center p_c, axis directions e_1 and e_2 perpendicular to n_c, as well as major and minor axis lengths $a = \lambda_1$ and $b = \lambda_2$.

Finally, we have to adjust the axis lengths by a factor f since the so-defined ellipse does not yet bound all points p_i projected in the plane κ_c. We obtain the maximal scale factor f by evaluating the ellipse equation $f^2 = x^2/a'^2 + y^2/b'^2$ in the tangent plane κ_c spanned by e_1 and e_2 for all points p_i, with $x_i = (p_i - p_c) \cdot e_1$ and $y_i = (p_i - p_c) \cdot e_2$. However, since every sample s_i represents an elliptical disk e_i, we generate bounding ellipses that not only include the points p_i, but cover the entire disks e_i, approximated by oriented bounding boxes as illustrated in Fig. 5.

The outlined generation of elliptical disks can be used in a similar way to compute the initial elliptical disks of the input point samples, as indicated in Section 3.1. For each input point p, the generic homogeneous covariance $M_{\mathcal{GH}} = \sum_{i=1}^{k} p_i \cdot p_i{}^{\mathrm{T}}$ of its k-nearest neighbors $p_{i=1\ldots k}$ is computed. Then, the covariance is expressed in the local tangent space, the elliptical distribution is determined, and the bounding ellipse is calculated as described above.

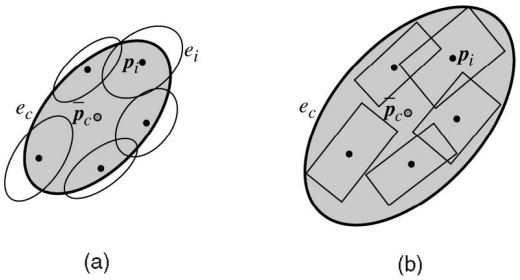


Fig. 5. (a) Ellipse e_c bounding only the points p_i and (b) conservatively bounding the disks e_i.

5 Point Blending

5.1 Continuous Interpolation

We interpret the smooth interpolation of surface parameters between points in object-space as a weighted blending of control point samples $s_1 \ldots s_n$. In fact, we need to compute the interpolated color $c_{\vec{p}}$ of a pixel $\vec{p}$ which is the perspective projection of a point p. Equation (5) interpolates color between visible point samples s_i whose elliptical disks e_i have a nonempty intersection with the projection $\vec{p}$ as illustrated in Fig. 6.

$$c_{\vec{p}} = \sum_{\forall i: \vec{p} \cap e_i \neq \emptyset} \Psi_i(u_i, v_i) \cdot c_i. \tag{5}$$

The blending function Ψ_i of a sample s_i must only be defined over its *local support*, the elliptical disk e_i, and zero outside. The local parameterization of Ψ_i with respect to a pixel $\vec{p}$ is given by the intersection $p_{e_i} = \vec{p} \cap e_i$ in the plane of e_i. The intersection parameters u_i, v_i define the linear combination $p_{e_i} = u_i \cdot e_1 + v_i \cdot e_2$ in the axis directions of e_i. Note that u_i and v_i are never explicitly computed since the blending functions Ψ_i are implemented as α-textures over e_i (see Section 6.3).

In order to achieve a continuous interpolation, the blending functions Ψ_i must satisfy the *positivity* $\Psi_i(\vec{p}) \geq 0$ and *partition-of-unity* $\sum_{\forall i: \vec{p} \cap e_i \neq \emptyset} \Psi_i(\vec{p}) = 1$ criteria for any

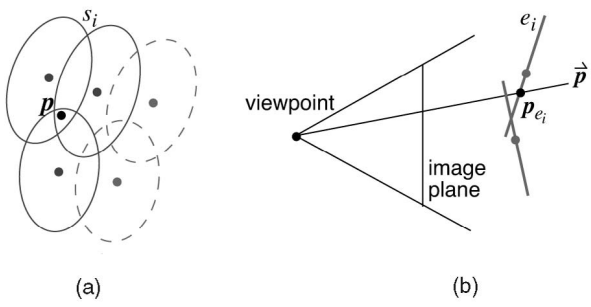


Fig. 6. (a) Surfels s_i overlapping p (dashed surfels do not contribute). (b) Side view of projection $\vec{p}$ intersecting the planar elliptical disks e_i.

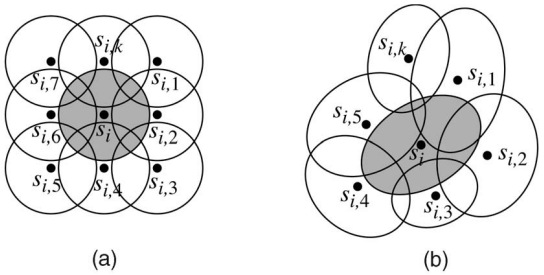


Fig. 7. (a) Regular grid of control points. (b) Irregular set of surface elements.

given pixel ray $\vec{p}$. We can define conforming blending functions Ψ_i as a normalization of rotation-symmetric blending kernels ψ_i. Given a sample s_i and its overlapping neighbors $s_{i,1} \ldots s_{i,k}$, as illustrated in Fig. 7, we define its conforming blending function as

$$\Psi_i(\vec{p}) = \frac{\psi_i(\vec{p})}{\psi_i(\vec{p}) + \sum_{j=1}^{k} \psi_{i,j}(\vec{p})}. \tag{6}$$

Thus, to achieve partition-of-unity, given a generic positive blending kernel ψ used for each point, its final contribution to a pixel is normalized by the sum of all other points' blending weights. Per-pixel visibility determination of points is detailed in Section 6.4.

For control points on a regular grid as shown in Fig. 7a, a solution to (6) is simple to achieve and a single global blending function Ψ can be used for each point. However, this is not the case for point samples irregularly distributed on the surface as illustrated in Fig. 7b. Hence, conforming object-space blending functions Ψ_i according to (6) must be computed for each individual point sample s_i, resulting in a computationally intractable solution. Note that the Ψ_i are not rotationally symmetric.

An effective solution to (6) is to separate (5) into separate summations for the numerator and the denominator as

$$\boldsymbol{c}_{\vec{p}} = \frac{\sum_{\forall i:\vec{p}\cap e_i\neq\emptyset} \psi_i(\vec{p}) \cdot \boldsymbol{c}_i}{\sum_{\forall i:\vec{p}\cap e_i\neq\emptyset} \psi_i(\vec{p})}. \tag{7}$$

Thus, we can perform the interpolation of (5) using a simple blending kernel ψ and get an *intermediate* blended color (the numerator of (7))

$$\boldsymbol{c}'_{\vec{p}} = \sum_{\forall i:\vec{p}\cap e_i\neq\emptyset} \psi_i(\vec{p}) \cdot \boldsymbol{c}_i. \tag{8}$$

This intermediate color $\boldsymbol{c}'_{\vec{p}}$ of a pixel $\vec{p}$ after blending all contributing points using ψ has a weight (or opacity) of $w_{\vec{p}} = \sum_{\forall i:\vec{p}\cap e_i\neq\emptyset} \psi_i(\vec{p})$, which is the denominator of (7) and may not equal to 1. Thus, the weights of (8) do not partition unity. However, since the weight $w_{\vec{p}}$ is the value of the denominator of (7), we obtain the correct color $\boldsymbol{c}_{\vec{p}}$ in a post-process per-pixel normalization by dividing $\boldsymbol{c}'_{\vec{p}}$ by its weight $w_{\vec{p}}$. Note that this normalization must be performed per pixel and its implementation is explained in Section 6.5. In case the representation of weights does not allow values larger than 1, a scaling of ψ (i.e., proportional to the maximum number of overlapping points) is sufficient in practice to prevent overflow of weights $w_{\vec{p}}$.

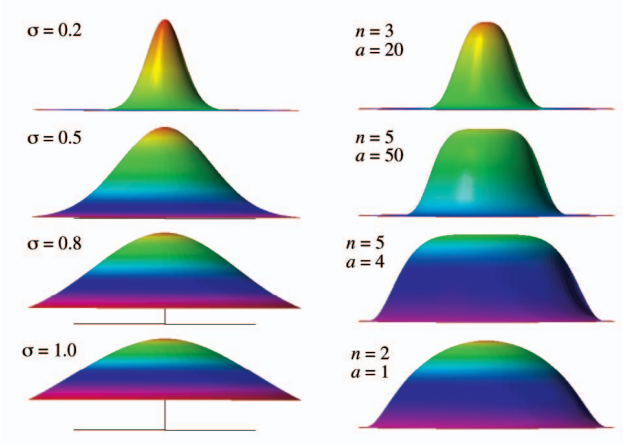


Fig. 8. Left: Gaussian $\psi_G(r)$ for $r > 1.5$. Right: Kernels $\psi_E(r)$ for $b = 1.5$.

5.2 Blending Kernels

As explained above, we can shift our focus to define a simple two-dimensional rotationally symmetric blending kernel $\psi : r \rightarrow [0, 1]$, apply it to each sample s_i scaled and oriented according to its bounding ellipse e_i in object-space, and achieve correct interpolation by a postprocess per-pixel color normalization. There are many obvious choices for blending kernels $\psi(r)$ such as hat-functions or Gaussians. However, to achieve good blending results and to provide flexibility in rendering systems, we want a blending kernel with the following properties:

1. positivity: $\forall r : \psi(r) \geq 0$,
2. smoothness: $\psi(r)$ is n times differentiable,
3. limited support: $\psi(r) = 0$ for $r > c_{\max}$,
4. control over width: $\psi(r) > 0.5$ for $r < c_{\text{mid}}$,
5. control over slope: $\frac{\partial}{\partial r}\psi(r) = s$ for $r = c_{\text{mid}}$.

A Gaussian blending kernel given by

$$\psi_G(r) = e^{-r^2/2\sigma^2} \tag{9}$$

satisfies the first two criteria; however, it does not have a limited support and very limited control over its shape. Fig. 8 shows several Gaussians $\psi_G(r)$ over a limited domain for varying parameters σ. Obviously, the shape can be made more or less narrow with varying σ. However, this single parameter also concurrently influences the slope around $\psi_G(c_{\text{mid}}) = 0.5$ as well as its support $\psi_G(c_{\max}) \leq \varepsilon$.

We propose a new family of blending kernels $\psi_E(r)$ that support *plateau*-like blending functions with variable sharp drop-off regions:

$$\psi_E(r) = e^{\frac{-a\cdot(\frac{r}{b})^n}{1-(\frac{r}{b})^n}} \quad (0 \leq r \leq b). \tag{10}$$

This blending kernel supports all desired criteria specified above. In particular, it is defined over a limited support specified by the parameter b with $\lim_{r\rightarrow b} \psi_E(r) = 0$. Furthermore, the width of the kernel adjusts with parameter a and the slope of the drop-off region with parameter n. As shown in Fig. 8, a larger a makes more narrow blending kernels, while a larger n generates plateau-like kernels.

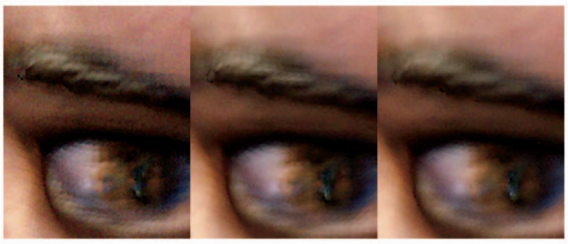


Fig. 9. From left to right: $\psi_G(r)$ with $\sigma = 0.5$; $\psi_E(r)$ with $a = 4$, $n = 5$; and $\psi_G(r)$ with $\sigma = 0.8$.

Fig. 9 illustrates the potential limitations of Gaussians at a detailed rendering level. The new blending kernel $\psi_E(r)$ with $a = 4$, $n = 5$ (and $b = 1.5$) exhibits less blurring than the Gaussian $\psi_G(r)$ with $\sigma = 0.8$ and less faceted appearing than $\psi_G(r)$ with $\sigma = 0.3$. Our new class of blending kernels has the added flexibility to achieve sharper results without faceted artifacts, while Gaussians only allow a smooth transition from faceted to blurred appearance.

6 Rendering

6.1 Overview of Algorithm

Our point blending and splatting algorithm uses hardware acceleration to efficiently render and scan convert points as α-textured polygons. Exploiting vertex-program [34] and pixel-shader [35] programmability, we present an efficient OpenGL-based rendering algorithm that performs visibility splatting and blending. Accurate blending, as discussed in Section 5, is performed by a per-pixel normalization postprocess. Our rendering algorithm performs the following steps for each frame:

1. LOD selection of all visible point samples s_i.
2. Representation of selected s_i as α-textured triangles.
3. Visibility splatting using an ε-z-buffer approach.
4. Per-pixel postprocess image normalization.

Fig. 10 illustrates our point rendering process as discussed in more detail in the following sections.

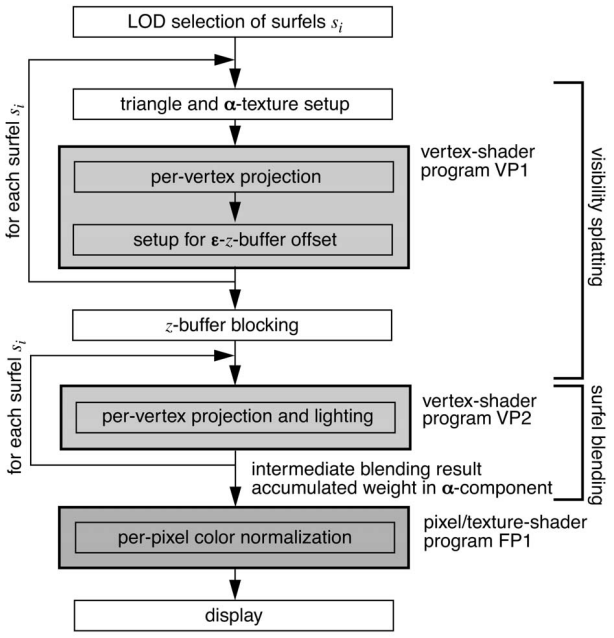


Fig. 10. Point rendering overview.

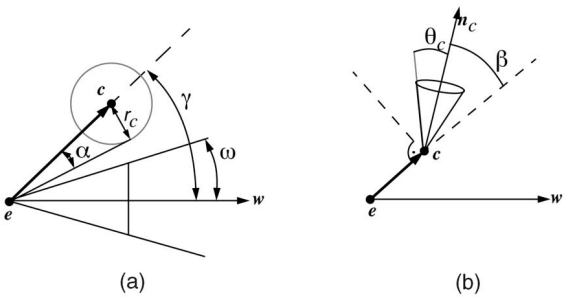


Fig. 11. (a) View-frustum culling if $\gamma - \alpha > \omega$ and (b) back-face culling if $\beta + \theta < 90°$.

6.2 LOD Selection

The LOD selection takes three view-dependent selection criteria into account: *view-frustum culling*, *back-face culling*, and *screen projection tolerance*. These criteria allow efficient back-tracking during the recursive traversal of the multiresolution hierarchy H. If, for a node $c \in H$, the bounding sphere with radius r_c does not intersect the view frustum (approximated by a viewing cone) or if the bounding normal-cone [30] with semi-angle θ_c indicates back-facing, the recursive LOD selection is stopped. We assume that the viewpoint $\boldsymbol{e}$, the viewing direction $\boldsymbol{w}$, and semi-angle ω of the viewing cone are given for each rendered frame.

As shown in Fig. 11, view-frustum culling is performed if $\gamma - \alpha > \omega$ and back-face culling is done if $\beta + \theta < 90°$. Both criteria can be computed without any expensive trigonometric functions, as shown in [36], [37].

Additionally, a screen projection error tolerance is used. A node c is selected if its elliptical disk e_c projection on screen is less than a threshold τ. Given the area A_{e_c} of the ellipse e_c, the normalized viewing direction $\boldsymbol{w}$, and the focal length d as shown in Fig. 12, the projection on screen is $A_{\text{screen}} = f(A_{e_c} \cdot d^2/|(\boldsymbol{c} - \boldsymbol{e}) \cdot \boldsymbol{w}|^2)$. The factor $f = \cos(\gamma - \theta_c)$ takes the *tilting* of the ellipse plane due to the normal-cone into account. For a given viewpoint $\boldsymbol{e}$, the visible area is maximal if $\gamma = 0° \Rightarrow \cos(\gamma) = 1$ and minimal

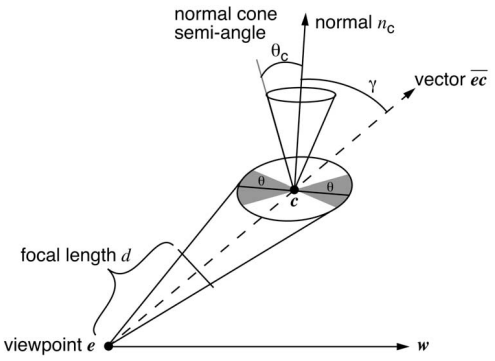


Fig. 12. Screen projection of elliptical disk.

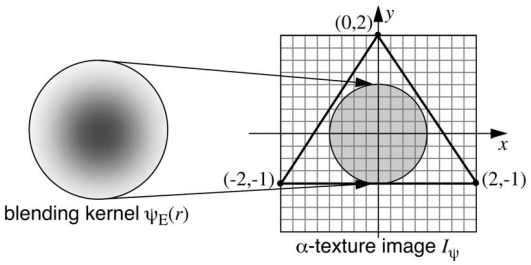


Fig. 13. Mapping of blending kernel $\psi_E(r)$ as α-texture onto a generic unit triangle.

if $\gamma = 90° \Rightarrow \cos(\gamma) = 0$. However, due to the normal variation bounded by θ_c, the maximal visible area can occur when $\cos(\gamma - \theta_c)$ is 1. More complex error metrics (see [14]) can be added to account for objects with sharp edges at the expense of processing time.

6.3 Blending

Blending of overlapping points is performed by mapping the blending kernel $\psi_E(r)$ onto the elliptical disk e_i of each visible point sample s_i. In fact, the blending kernel $\psi_E(r)$ is precomputed and rasterized in a preprocess and the result is stored in a high-resolution image I_ψ. As shown in Fig. 13, this α-texture I_ψ is mapped onto a generic triangle t_{unit} in the x, y-plane such that $\psi_E(r)$ covers a unit-size disk. For each s_i, the triangle setup stage of Fig. 10 scales the generic triangle t_{unit} in the x and in the y directions according to the major and minor axis lengths of ellipse e_i, rotates the triangle to align with the normal $\boldsymbol{n}_i$ and ellipse axis, and translates it to the point $\boldsymbol{p}_i$, resulting in triangle t_i. Thus, each point is rendered as a triangle and blending is achieved by α-texturing.

For each visible point, the reference triangle must be translated, oriented, and scaled according to the point information, yielding triangles t_i. Instead of doing this for each frame and rendering pass, it can be precomputed as early as when loading the point model from disk. This allows the use of OpenGL vertex arrays to improve rendering performance.

6.4 Visibility Splatting

Visibility splatting uses an ε-z-buffer technique similar to [6] and [12], implemented by a two-pass rendering approach. In the first pass, all selected points are rendered with lighting and α-blending disabled. Note however, that the α-test is enabled so that the α-texture I_ψ generates elliptical splats during scan-conversion of triangles t_i. During this pass, the corners of the triangle t_i are perspectively displaced by ε, as shown in Fig. 14. The depth-buffer then represents the surface perspectively offset by ε along the view-projection. A vertex program—VP1 in Fig. 10—performs the appropriate object-space to screen-space transformations and also the perspective translation of each corner of t_i by ε, as illustrated in Fig. 14.

In the second rendering pass, the depth-buffer is set to read-only and all visible points are rendered with lighting and α-blending enabled. Since the depth-buffer contains the visible surface offset by ε, the desired ε-z-buffering effect is achieved. A vertex program—VP2 in Fig. 10—with per-vertex normal, α-texture and material properties enabled is used in order to incorporate proper shading and lighting.

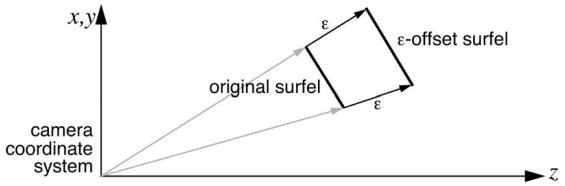


Fig. 14. Application of a depth ε-offset along view-projection for visibility splatting.

6.5 Normalization

After visibility splatting and blending, the image I contains the interpolated colors according to (8). The pixel colors contain the intermediate blending result $\boldsymbol{c}' = (\alpha \cdot R, \alpha \cdot G, \alpha \cdot B, \alpha)$ and the α component contains the accumulated blending weight w. This corresponds to the correct proportionally blended color values, however, the α values need not yet be 1 as required. To get the final color $\boldsymbol{c} = (R, G, B, 1)$, each color component of $\boldsymbol{c}'$ has to be multiplied by α^{-1}. This normalization is performed as an image postprocess stage, see also Fig. 10.

Without hardware support to perform per-pixel manipulations, this normalization step has to be performed in software. However, most graphics accelerators now offer per-pixel shading operators that can be used more efficiently. In our current implementation, we perform this normalization using an OpenGL Fragment Program [35]. In [38], [39], [40], we present an implementation using texture shaders [41].

Using a fragment program, the normalization step is very simple and can be implemented very efficiently. For every incoming pixel fragment, its color $\boldsymbol{c}' = (r, g, b, \alpha)$ is normalized to $\boldsymbol{c} = (R, G, B, 1)$ by dividing each channel by the alpha value. An ARB OpenGL fragment program [35] receives the r, g, b, α color vector as input for the incoming fragment and the supported operations allow component-wise division by α. The resulting $R, G, B, 1$ color vector is written to the output which results in the final correct color assigned to that pixel fragment in the color frame buffer.

7 Experimental Results

The blending kernel $\psi_E(r)$ introduced in Section 5 has some advantages due to its flexibility in parameterization. Fig. 15 shows a coarse checkerboard of 512×512 point samples rendered with a fairly "round" kernel on the left and with a more localized kernel on the right. Clearly, the more localized kernel exhibits less blurring effects, but still provides some antialiasing in object-space. However, aliasing artifacts are not completely avoided by this object-space blending approach.

In Fig. 16, we can see another effect of a wide plateau-like blending kernel $\psi_E(r)$ on the left compared to a narrow blending kernel on the right. The wide plateau shaped kernel provides very smooth blending while retaining

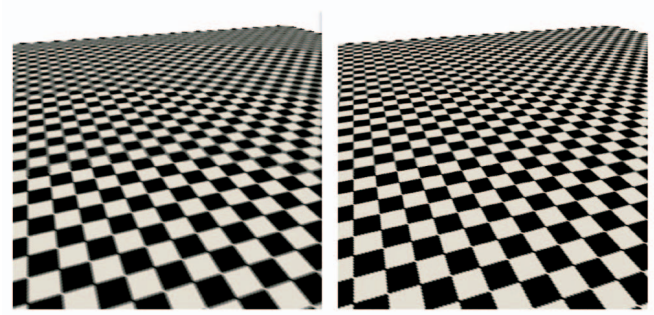


Fig. 15. Different blending kernels $\psi_E(r)$ with parameters: $a = 1.0$, $b = 1.5$, and $n = 2.0$ (left) and $a = 6.0$, $b = 1.5$ and $n = 3.0$ (right) (at a screen-projection tolerance $\tau = 0.02\%$ of the viewport size).

texture detail; the narrow kernel results in a more faceted appearance.

The performance of our point blending and splatting method was measured using several color textured models. All performance measures were taken on a 2.8GHz Pentium4 CPU with 1GB equipped with an NVIDIA GeForce FX 5900 GPU.

The data-driven multiresolution point-octree hierarchy is illustrated in Fig. 17 by translucent boxes denoting the space-partitioning structure. In comparison to more widely used region-octree hierarchies (i.e., such as in [7], [8]), the point-octree hierarchy is more adaptive to the spatial distribution of the unorganized point set. This can also be seen from Table 1 which reports much fewer nodes for the point-octree hierarchy than reported in [7] for a region-octree (i.e., for the David head model).

Table 1 also shows the memory cost of our multiresolution data structure. Without use of quantization and compression methods, our point-octree consumes an amortized 62 bytes per input point, including all hierarchy and per-point LOD attributes (coordinates, normal, color, elliptical splat definition). Furthermore, Table 1 provides the timings of the preprocess that generates the octree hierarchy and computes the elliptical splat sizes as described in Section 4. We can see that our approach achieves a performance of processing about 150,000 to 190,000 input points per second. Even multimillion point models can be efficiently processed in a matter of seconds.

The performance of our point blending, visibility splatting, and color normalization algorithm is summarized in Table 2. It lists the number of actually visible and processed points; the time for LOD selection, blending, and visibility splatting, all given in seconds; and the time for color normalization given in milliseconds. All values are averaged per frame over 1,000 frames rotating around the object at 1 degree per frame. All tests and images have been rendered at a resolution of 512×512 pixels. Apart from LOD selection, the splat blending and rendering pipeline itself is not dependent on the output image size.

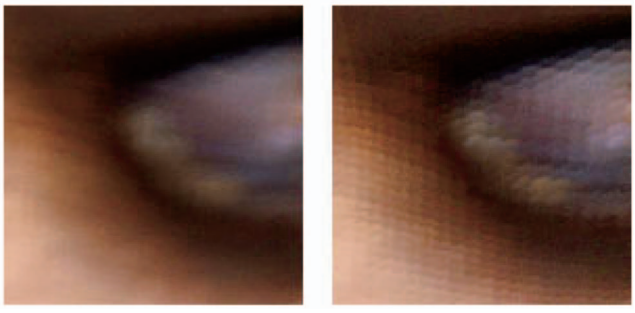


Fig. 16. Blending kernels $\psi_E(r)$ with parameters: $a = 4.0$, $b = 1.5$, and $n = 5.0$ (left) and $a = 6.0$, $b = 1.5$, and $n = 3.0$ (right) (at a screen-projection tolerance $\tau = 0.0\%$).

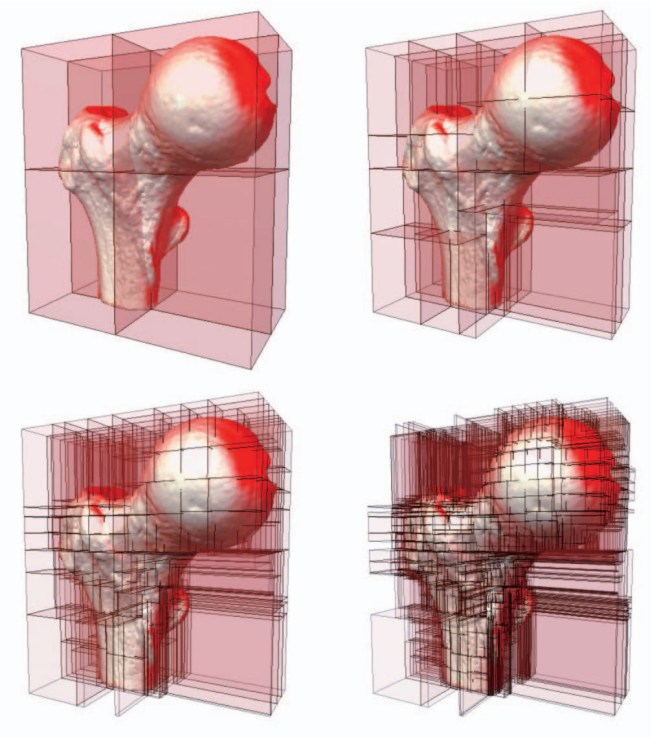


Fig. 17. Examples of different point-octree space partitioning levels.

Our point splatting pipeline achieves maximal rendering speed for models that can be fully cached as vertex arrays in the graphics card's video memory, which is the case for the Female, Balljoint, and Dragon models. In that case, our pipeline achieves rendering rates of 2.2 (Dragon) to 4.7 (Balljoint) million points per second (MPS); including LOD selection, full per-vertex illumination with two light sources, two-pass visibility splatting, as well as per-pixel accurate blending and normalization of the weighted blended colors. Balljoint exhibits a special behavior in that, for this model, our implementation is capable of splatting up to 7.3 MPS; however, the view-dependent LOD selection limits the overall rendering rate to 4.7 MPS.

The David head model with its maximal size of 2,000,606 displayed points does not fit into the graphics card's video

TABLE 1
Multiresolution Point Hierarchy File Sizes and Construction and Splat Generation Times

Model	# Points	# Nodes	Size	Time
David	2000606	802371	118.5MB	12.7s
Female	302948	121439	17.9MB	2.97s
Balljoint	137062	54992	8.1MB	0.73s
Dragon	437645	173507	25.8MB	2.41s

TABLE 2
Rendering Performance Is Given for Each Task in Seconds Used per Frame

Model	Tol. τ	# Splats	LOD	Splatting	Normal.	Total
David	0.0 %	899,523	0.187s	0.397s	0.128ms	0.584s
	0.02%	185,858	0.121s	0.163s	0.121ms	0.284s
	0.04%	172,806	0.115s	0.151s	0.122ms	0.266s
Female	0.0%	160,344	0.027s	0.047s	0.116ms	0.075s
	0.02%	60,876	0.023s	0.017s	0.127ms	0.041s
	0.04%	53,512	0.020s	0.015s	0.134ms	0.035s
Balljoint	0.0%	67,363	0.011s	0.014s	0.113ms	0.026s
	0.02%	26,356	0.011s	0.004s	0.091ms	0.019s
	0.16%	22,067	0.013s	0.003s	0.089ms	0.018s
Dragon	0.0%	221,277	0.041s	0.087s	0.124ms	0.129s
	0.02%	72,037	0.036s	0.033s	0.140ms	0.069s
	0.04%	57,655	0.031s	0.025s	0.141ms	0.056s

memory cache and exhibits a significantly lower rendering rate of 1.1 to 2.3 MPS. This is mostly due to not exploiting on-card video memory for vertex caching, but also partly due to an increased fraction of main CPU time spent in the view-dependent LOD selection stage.

Example renderings are shown in Fig. 18 for the David and Balljoint models with τ given in percentage of the viewport size. Even at a significant simplification rate, our approach produces extremely high quality renderings with smooth texture color interpolation and blending of point samples. A blending kernel $\psi_E(r)$ with parameters $n = 3$, $a = 6.0$, and $b = 1.5$ was used. The pseudocolored images in Fig. 18 show the amount of overlap measured per pixel, the redraw rate. The colors encode the number of points overlapping one single pixel and we can observe a very low overlap rate (mostly blue colors). Thus, our system efficiently avoids excessive redrawing of pixels and is generally not pixel-fill rate limited.

8 Discussion

In this paper, we have presented Confetti, a novel point-based rendering algorithm including efficient LOD splat size generation, flexible object-space blending kernels, and a point-blending and splatting rendering pipeline that exploits hardware accelerated blending.

While providing more flexibility through added shape parameters, we do not yet know of any closed form solution of our novel blending kernel for reconstruction and low-pass filtering in image space. This is in contrast to Gaussians as exploited in [10] and has the implication that we cannot directly integrate screen-space antialiasing into the blending process. In practice, however, object-space blending hardly leads to noticeable aliasing effects (see also the high-quality renderings in [13]). Furthermore, under minification, the view-dependent LOD framework naturally provides some level of antialiasing as shown in Fig. 15. In particular, no aliasing effects are noticeable when animating objects.[1] Without extensive experimentation or analytical studies, current experiments have shown that

1. See additional accompanying electronic video and image material.

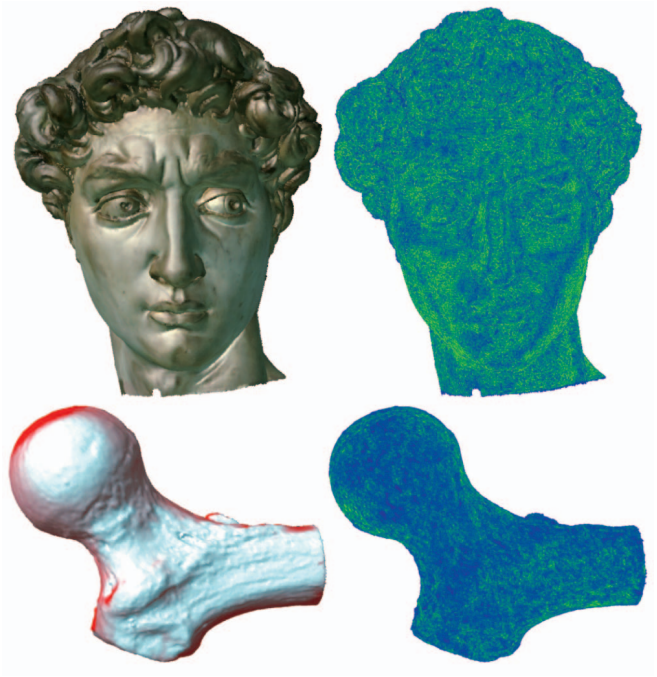


Fig. 18. Top: David at $\tau = 0.02\%$ error tolerance, rendered 380,663 points out of 2,000,606. Bottom: Balljoint at $\tau = 0.02\%$, rendered 47,492 points out of 137,062. On the right: Number of points blended per pixel (blue = 1, green = 8, and red = 16). Viewport of $1,280 \times 1,024$.

blending kernels with parameters $n = 3$, $a = 6$ to $n = 5$, $a = 4$ (and fixed support $b = 1.5$) provide good compromises. The value $n = 5$, $a = 4$ exhibited maximal smoothing while retaining the significant texture details. With reference to Fig. 8, we conclude that a medium to wide kernel with a smooth but accentuated drop-off is best. Note that these blending kernel shapes are not achievable with Gaussians.

The presented per-pixel accurate blending and normalization process has been derived from our earlier work on depth-image meshing and warping in [38], [39]. This work introduced a hardware accelerated per-pixel normalization process using texture shaders [41], which was also used in our first implementation of the presented point-rendering pipeline in [40]. With pixel shaders [35], this implementation has become trivial and others have independently developed similar techniques (i.e., in [13]).

We can directly compare our system to a QSplat implementation on exactly the same machine. Using their highest-quality circle primitive and maximum detail, QSplat achieved a consistent speed of 3.5 million points per second (MPS) for the same set of models reported in Section 7. Considering the much higher quality texture blending and rendering of our approach and the fact that this blending requires a two-pass rendering process, we compare extremely well with rendering speeds of 2.2 to 4.7 MPS. The increased CPU and GPU cost of Confetti compared to QSplat is offset to a large degree by using optimized rendering primitives such as vertex arrays and on-board geometry caching.

Hardware accelerated EWA splatting [12] reports up to 90 thousand points per second with per-pixel and 1 MPS with per-surfel normalization (without LOD selection and illumination). On a slightly faster machine with an NVIDIA

GeForce4 Ti 4600 GPU, we achieve 2.1 MPS for the Female model, including fully lit points and LOD selection cost.

The approach in [13] uses similar hardware optimizations, such as vertex arrays and caching of geometry in on-board graphics memory, as well as a similar machine with an NVIDIA GeForce FX 5800 GPU. Using point-sprites, they are able to achieve 5 to 10 MPS, not including any LOD selection cost. While slightly slower, with splatting rates of 2.2 to 4.7 MPS—and 7.3 MPS for Balljoint without LOD selection—our system achieves comparable order-of-magnitude rendering performance. Obviously, the approach in [13] benefits from the simplicity of point-sprites compared to textured triangles as used in our current implementation.

Other approaches such as [14] obtain even higher frame rates, up to 50 MPS, at the expense of strictly limiting the number of points to fit into on-board graphics memory. Furthermore, their single-pass rendering does not allow for high-quality blending. A major difference from previous methods is that the multiresolution hierarchy is transformed into a sequential point list in [14] which allows for very fast LOD selection at the expense of limited culling efficiency.

While not optimized for space usage, the Confetti multiresolution point-octree hierarchy consumes about 62 bytes amortized per input point. This compares well with view-dependent multiresolution triangle mesh data structures such as [42] (88+ bytes/vertex), [43] (138+bytes/vertex), or [36] (106 bytes/vertex). However, more optimized point hierarchies such as QSplat [7] and the octree structure in [8] use aggressive point-attribute quantization and, hence, require significantly less space. In fact, the optimized rendering data structures such as vertex arrays used in our current implementation further increase the memory footprint beyond the 62 bytes per point since multiple copies of point attributes are stored for fast rendering.

Future work will address the problem of transparent surfaces within hardware accelerated point rendering frameworks and the development of single-pass visibility splatting and point blending algorithms. Furthermore, we want to improve the parameterization of our new blending kernels such that the parameters are computed automatically or are easier to use.

ACKNOWLEDGMENTS

The authors would like to thank Roberto Lario for sharing his implementation of using the vertex cache and his help during the integration with our rendering pipeline. They also thank the Stanford *3D Scanning Repository* and *Digital Michelangelo* projects as well as *Cyberware* for freely providing geometric models to the research community.

REFERENCES

[1] M.H. Gross, "Are Points the Better Graphics Primitives?" *Computer Graphics Forum,* vol. 20, no. 3, 2001.

[2] H. Hoppe, T. DeRose, T. Duchampt, J. McDonald, and W. Stuetzle, "Surface Reconstruction from Unorganized Points," *Proc. ACM SIGGRAPH '92,* pp. 71-78, 1992.

[3] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," *Proc. ACM SIGGRAPH '96,* pp. 303-312, 1996.

[4] M. Gopi, S. Krishnan, and C.T. Silva, "Surface Reconstruction Based on Lower Dimensional Localized Delaunay Triangulation," *Proc. EUROGRAPHICS '00,* pp. 467-478, 2000.

[5] M. Levoy and T. Whitted, "The Use of Points as Display Primitives," Technical Report TR 85-022, Dept. of Computer Science, Univ. of North Carolina at Chapel Hill, 1985.

[6] J.P. Grossman and W.J. Dally, "Point Sample Rendering," *Proc. Eurographics Rendering Workshop '98.* pp. 181-192, 1998.

[7] S. Rusinkiewicz and M. Levoy, "Qsplat: A Multiresolution Point Rendering System for Large Meshes," *Proc. SIGGRAPH 2000,* pp. 343-352, 2000.

[8] M. Botsch, A. Wiratanaya, and L. Kobbelt, "Efficient High Quality Rendering of Point Sampled Geometry," *Proc. Eurographics Workshop Rendering,* pp. 53-64, 2002.

[9] H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels: Surface Elements as Rendering Primitives," *Proc. SIGGRAPH 2000,* pp. 335-342, 2000.

[10] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "Surface Splatting," *Proc. SIGGRAPH 2001,* pp. 371-378, 2001.

[11] P.S. Heckbert, "Fundamentals of Texture Mapping and Image Warping," MS thesis, Dept. of Electrical Eng. and Computer Science, Univ. of California Berkeley, 1989.

[12] L. Ren, H. Pfister, and M. Zwicker, "Object Space EWA Surface Splatting: A Hardware Accelerated Approach to High Quality Point Rendering," *Proc. EUROGRAPHICS 2002,* also in *Computer Graphics Forum,* vol. 21, no. 3, 2002.

[13] M. Botsch and L. Kobbelt, "High-Quality Point-Based Rendering on Modern GPUs," *Proc. Pacific Graphics 2003,* pp. 335-343, 2003.

[14] C. Dachsbacher, C. Vogelgsang, and M. Stamminger, "Sequential Point Trees," *Proc. ACM SIGGRAPH '03,* pp. 657-662, 2003.

[15] A. Kalaiah and A. Varshney, "Differential Point Rendering," *Proc. Rendering Techniques,* 2001.

[16] A. Kalaiah and A. Varshney, "Modeling and Rendering Points with Local Geometry," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 1, pp. 30-42, Jan.-Mar. 2003.

[17] J.D. Cohen, D.G. Aliaga, and W. Zhang, "Hybrid Simplification: Combining Multi-Resolution Polygon and Point Rendering," *Proc. IEEE Visualization 2001,* pp. 37-44, 2001.

[18] B. Chen and M.X. Nguyen, "POP: A Hybrid Point and Polygon Rendering System for Large Data," *Proc. IEEE Visualization 2001,* pp. 45-52, 2001.

[19] T.K. Dey and J. Hudson, "PMR: Point to Mesh Rendering, a Feature-Based Approach," *Proc. IEEE Visualization 2002,* pp. 155-162, 2002.

[20] L. Coconu and H.-C. Hege, "Hardware-Oriented Point-Based Rendering of Complex Scenes," *Proc. Eurographics Workshop Rendering,* pp. 43-52, 2002.

[21] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C.T. Silva, "Point Set Surfaces," *Proc. IEEE Visualization 2001,* pp. 21-28, 2001.

[22] M. Pauly, M. Gross, and L.P. Kobbelt, "Efficient Simplification of Point-Sampled Surfaces," *Proc. IEEE Visualization 2002,* pp. 163-170, 2002.

[23] M. Zwicker, M. Pauly, O. Knoll, and M. Gross, "Pointshop 3D: An Interactive System for Point-Based Surface Editing," *Proc. ACM SIGGRAPH 2002,* pp. 322-329, 2002.

[24] M. Stamminger and G. Drettakis, "Interactive Sampling and Rendering for Complex and Procedural Geometry," *Proc. Eurographics Workshop Rendering,* pp. 151-162, 2001.

[25] M. Wand, M. Fischer, I. Peter, F.M. auf der Heide, and W. Strasser, "The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes," *Proc. SIGGRAPH 2001,* pp. 361-370, 2001.

[26] G. Meenakshisundaram, "Theory and Practice of Sampling and Reconstruction for Manifolds with Boundaries," PhD thesis, Dept. of Computer Science, Univ. of North Carolina Chapel Hill, 2001.

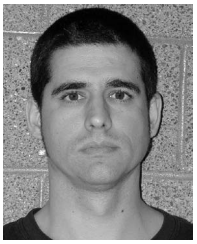
[27] T.K. Dey, J. Giesen, and J. Hudson, "A Delaunay Based Shape Reconstruction from Larga Data," *Proc. IEEE Symp. Parallel and Large Data Visualization and Graphics,* pp. 19-27, 2001.

[28] J. Nievergelt, "7 ± 2 Criteria for Assessing and Comparing Spatial Data Structures," *Proc. First Symp. Design and Implementation of Large Spatial Databases,* pp. 3-27. 1989.

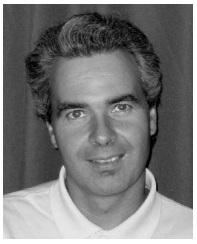
[29] H. Samet, *The Design and Analysis of Spatial Data Structures.* Reading, Mass.: Addison Wesley, 1989.

[30] L.A. Shirman and S.S. Abi-Ezzi, "The Cone of Normals Technique for Fast Processing of Curved Patches," *Proc. EUROGRAPHICS '93,* pp. 261-272, also in *Computer Graphics Forum,* vol. 12, no. 3, 1993.
[31] H. Samet, "The Quadtree and Related Hierarchical Data Structures," *Computing Surveys,* vol. 16, no. 2, pp. 187-260, June 1984.
[32] R. Pajarola, "Efficient Level-of-Details for Point Based Rendering," *Proc.s IASTED Int'l Conf. Computer Graphics and Imaging (CGIM 2003),* 2003.
[33] D.H. Eberly, *3D Game Engine Design.* San Francisco: Morgan Kaufmann, 2001.
[34] OpenGL Architecture Review Board, "ARB Vertex Program," OpenGL Vertex Program Documentation, 2002.
[35] OpenGL Architecture Review Board, "ARB Fragment Program," OpenGL Fragment Program Documentation, 2002.
[36] R. Pajarola, "Fastmesh: Efficient View-Dependent Meshing," *Proc. Pacific Graphics 2001,* pp. 22-30, 2001.
[37] R. Pajarola, M. Antonijuan, and R. Lario, "QuadTIN: Quadtree Based Triangulated Irregular Networks," *Proc. IEEE Visualization 2002,* pp. 395-402, 2002.
[38] R. Pajarola, Y. Meng, and M. Sainz, "Fast Depth-Image Meshing and Warping," Technical Report UCI-ECE-02-02, The Henry Samueli School of Eng., Univ. of California Irvine, 2002.
[39] R. Pajarola, M. Sainz, and Y. Meng, "DMesh: Fast Depth-Image Meshing and Warping," *Int'l J. Image and Graphics (IJIG),* to appear.
[40] R. Pajarola, M. Sainz, and P. Guidotti, "Object-Space Point Blending and Splatting," *ACM SIGGRAPH Sketches & Applications Catalogue,* 2003.
[41] S. Dominé and J. Spitzer, "Texture Shaders," developer documentation, 2001.
[42] H. Hoppe, "Efficient Implementation of Progressive Meshes," *Computers & Graphics,* vol. 22, no. 2, pp. 27-36, 1998.
[43] J. El-Sana and A. Varshney, "Generalized View-Dependent Simplification," *Proc. EUROGRAPHICS '99,* pp. 83-94, 1999.

Renato Pajarola received a Dr.sc.techn. degree in computer science in 1998 from the Swiss Federal Institute of Technology (ETH) Zurich. Following his dissertation, he was a postdoctoral researcher and part-time faculty member in the Graphics, Visualization, & Usability Center at the Georgia Institute of Technology for one year. Since 1999, he has been an assistant professor in computer science at the University of California, Irvine. His current research interests include real-time 3D graphics, multiresolution modeling, image-based rendering, image and geometry compression, interactive remote visualization, large scale terrain and volume visualization, as well as interactive 3D multimedia. He has published a wide range of technical papers in top journals and conferences. He has served as a reviewer and program or organization committee member for several international conferences and is a frequent referee for journal articles. He is a member of the IEEE Computer Society, ACM, and the ACM Special Interest Group on Computer Graphics.



Miguel Sainz received the degree in electrical engineering from the Polytechnical University of Catalonia, UPC, Spain in 1996 and the PhD degree in electrical and computer engineering from the University of California, Irvine in 2003. He is currently a postdoctoral researcher and part-time lecturer at the School of Information and Computer Science at the University of California, Irvine. His current research interests include image based modeling and rendering, 3D model reconstruction from images, tracking and image processing, real-time 3D graphics, and GPU programming. He is a member ot the IEEE Computer Society and the ACM.



Patrick Guidotti received the DrPhil degree in mathematics in 1996 from the University of Zürich, Switzerland. Following his dissertation, he was a postdoctoral scholar at the University of Halle-Wittenberg, Germany, and at the California Institute of Technology. Since 2001, he has been an assistant professor in mathematics at the University of California, Irvine. His current research interests include applied analysis and applied mathematics. He has published various technical papers and has served as a referee for a variety of internationally recognized journals. He is a member of the AMS and SIAM.