

# **Object-Space Blending and Splatting of Points**

*Renato Pajarola, Miguel Sainz, Patrick Guidotti*

UCI-ICS Technical Report No. 03-01  
Department of Information & Computer Science  
University of California, Irvine

February 2003

# Object-Space Blending and Splatting of Points

Renato Pajarola

Computer Graphics Lab  
Information and Computer Science  
University of California Irvine

Miguel Sainz

Image Based Modeling and Rendering Lab  
Electrical Engineering and Computer Science  
University of California Irvine

Patrick Guidotti

Mathematics  
Physical Sciences  
University of California Irvine

## Abstract

*In this paper we present 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 parametrized 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  $\alpha$ -blending as well as programmable vertex- and pixel-shaders. Experimental results show the high quality and rendering efficiency that is achieved by our approach.*

**Keywords:** point-based rendering, image-based rendering, multiresolution modeling, level-of-detail, hardware accelerated blending

## 1. Introduction

In recent years, point-based surface representations have been established as viable graphics rendering primitives [Gro01]. In particular they lead to compact multiresolution representations that can provide efficient level-of-detail (LOD) rendering of large point-sampled surfaces such as the digitally scanned statues of Michelangelo [LPC<sup>+</sup>00]. The advantage of the point-based representations over triangle meshes is that the explicit surface reconstruction step (i.e. [HDD<sup>+</sup>92, CL96, GKS00]) and the storage of mesh connectivity is avoided. Recent efforts in point-based rendering (PBR) have focused on compact and very efficient multiresolution models [RL00, BWK02], as well as on high-quality texture sampling and rendering [ZPvBG01, RPZ02].

The major challenge for a PBR method is to achieve smooth and continuous surface interpolation from point samples that are irregularly distributed over a surface. Furthermore, it must support correct visibility as well as provide an efficient rendering algorithm. In this paper we propose a novel point blending and rendering technique that is based on the direct interpolation between point samples in 3D. In contrast to previous methods, we 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 parametrized blending kernels. We also provide an efficient technique to calculate splat sizes in a multiresolution point hierarchy. Furthermore, our approach exploits hardware acceleration.

An example rendering result of our approach for the textured David statue of Michelangelo is shown in Figure 1.



**FIGURE 1.** The head of Michelangelo’s David statue rendered with  $\tau=16$  pixels screen tolerance, at 1/4 of the full resolution (510827 out of 2000606 points).

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.

The remainder of the paper is organized as follows. Section 2 briefly discusses related work on point-based rendering. Section 3 discusses how to efficiently determine the elliptical splat size for each LOD-node of a spatial-partitioning multiresolution point data structure. In Section 4 we introduce our concept of object-space point interpolation and define a new class of parametrized blending kernels with limited support. Our point rendering algorithm is described in Section 5. Section 6 presents experimental results of our approach and Section 7 concludes the paper.

## 2. Related Work

Points as rendering primitives have first been discussed in [LW85] but only much later were re-discovered as a practical approach to render complex geometric objects [GD98, RL00]. *QSplat* [RL00] is a high-performance point rendering system that uses a region-octree multiresolution data structure for efficient LOD selection and a simple point splatting method for fast rendering. The *surfels* approach [PZvBG00] generates a hierarchical orthographic sampling

of an object in a preprocess, and surfels colors are obtained by texture pre-filtering. Its rendering algorithm is a combination of approximate visibility splatting, texture mipmapping, and image filtering to fill holes between surfels. In [ZPvBG01] the basic principle of EWA texture filtering [Hec89] is applied to irregularly sampled texture information on 3D surfaces. For this EWA surface splatting a two-pass rendering method and hardware accelerated implementation is provided in [RPZ02]. Another efficient point rendering approach is presented in [BWK02] with a memory efficient multiresolution hierarchy and a precomputed footprint table for fast splatting.

Further related techniques using point representations include the integration of polygonal and point primitives in a unified multiresolution data structure [CAZ01, CN01, DH02], simplification methods [ABCO<sup>+</sup>01, PGK02] and interactive editing of point sets [ZPKG02]. Also the *randomized z-buffer* method [WFP<sup>+</sup>01] is related to PBR since it renders triangle meshes from dynamically chosen random point samples.

### 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 discrete input point set reasonably samples the surface (i.e. satisfies the Nyquist sampling criteria, or other sufficient surface sampling criteria as discussed in [Mee01]). Not unlike [PZvBG00] and [ZPvBG01, RPZ02], our approach also assumes that the points reasonably sample the surface’s color texture.

The full resolution input data set consists of point samples, surface elements (*surfels*)  $s$  with attributes for spatial coordinates  $\mathbf{p}$ , normal orientation  $\mathbf{n}$  and surface color  $\mathbf{c}$ . Furthermore, it is assumed that each surfel also contains the information about its spatial extent in object-space. This *size* information specifies an elliptical disk  $e$  centered at  $\mathbf{p}$  and perpendicular to  $\mathbf{n}$ . For correct visibility and occlusion, these elliptical surfel disks must cover the sampled object nicely without holes and thus overlap each other in object-space as shown in Figure 2. As noted also in [PZvBG00], tangential planar disks may not completely cover a surface if it is strongly bent or under extreme perspective projections. However, this is not very often noticeable in practical situations.

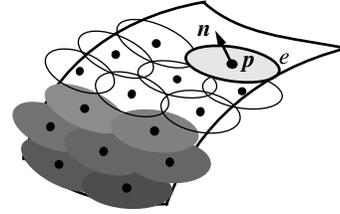


FIGURE 2. Elliptical surface elements covering a smooth and curved 3D surface.

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

In the remainder of this section we focus on discussing adequate splat size generation in a spatial-partitioning multiresolution hierarchy defined over the input point set, and we assume that the surfel splat sizes of the input data set are already given. Initial splat ellipses could be derived from locally computed Voronoi cells as in [DGH01] and [DH02], or from local neighborhood and covariance analysis as proposed in [PGK02] and outlined at the end of Section 3.4.

#### 3.2 Multiresolution hierarchy

The multiresolution point representations considered in this paper are hierarchical space-partitioning data structures [Nie89, Sam89]. Each cell or node  $c$  of such a hierarchy  $H$ , containing a set of  $k$  surfels  $S_c = \{s_1 \dots s_k\}$  has a representative sample  $s_c$  with average coordinates  $\bar{\mathbf{p}}_c = k^{-1} \cdot \sum_{i=1}^k \mathbf{p}_i$ , as well as average normal  $\bar{\mathbf{n}}_c$  and color  $\bar{\mathbf{c}}_c$  information. Furthermore, for efficient view-frustum and back-face culling each cell  $c \in H$  may also include the sphere radius  $r_c$  and normal-cone [SAE93] semi-angle  $\theta_c$  parameters bounding all surfels in  $S_c$ . Several conforming space-partitioning multiresolution hierarchies have been proposed for point-based rendering [RL00, BWK02, PGK02] which can be used with our techniques. In our work we use a point-octree [Sam84, Sam89] hierarchy which partitions the space adaptively to the sample distribution (data-driven) rather than regularly in space (space-driven) as region-octrees which have been proposed more commonly. Figure 3 illustrates the two-dimensional example of a point-quadtrees with up to 4 points stored in a leaf node.

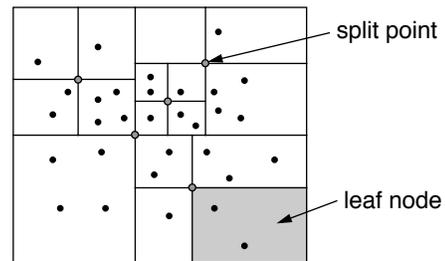


FIGURE 3. Point-quadtrees example with a fan-out of 4 also at the leaf level – up to 4 elements per leaf node.

Given  $n$  input surfels  $s_1 \dots s_n$ , a point-octree data structure can efficiently be generated by a single depth-first traversal in  $O(n \log n)$  time as illustrated in Figure 4. Given the

$k$  surfels  $S_c = \{s_1 \dots s_k\}$  of a node  $c \in H$  and the average position  $\bar{p}_c$  of the surfels  $S_c$ , the set  $S_c$  is partitioned into sets  $S_1$  to  $S_8$  according to the eight octants with respect to the split coordinate  $\bar{p}_c$ . While dividing  $S_c$  into the subsets  $S_i$  the averages  $\bar{p}_i$ ,  $\bar{n}_i$  and  $\bar{c}_i$  are computed (for  $i = 1 \dots 8$ ), and the process is repeated recursively for the eight child nodes. To compute the elliptical disk  $e_c$  the child nodes of  $c$  return their *generic homogeneous* covariance matrices  $\bar{M}_i$  to calculate  $\bar{M}_c$  and then derive the ellipse  $e_c$ . In the following Section 3.3 we introduce the concept of a generic homogeneous covariance matrix and in Section 3.4 we describe how to derive the elliptical surfel disks therefrom.

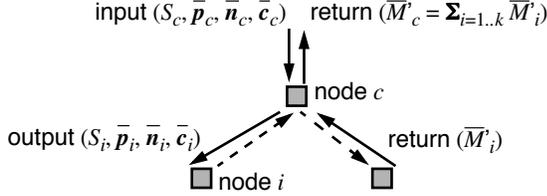


FIGURE 4. Recursive point-octree generation.

### 3.3 Generic homogeneous covariance

Given  $n$  points  $p_1, \dots, p_n \in \mathbf{R}^3$  and their average  $\bar{p}$  the covariance matrix is defined by

$$M = \frac{1}{n} \sum_{i=1}^n (p_i - \bar{p}) \cdot (p_i - \bar{p})^T. \quad (\text{EQ 1})$$

In homogeneous space with  $p'_i = (p_i^T, 1)$  we can rewrite the expression  $(p_i - \bar{p}) \cdot (p_i - \bar{p})^T$  to  $(T \cdot p'_i) \cdot (T \cdot p'_i)^T$  with the transformation matrix  $T$  denoting the translation by  $-\bar{p}$ . Thus we can revamp Equation 1 to

$$\begin{aligned} M &= \frac{1}{n} \sum_{i=1}^n (T \cdot p'_i) \cdot (T \cdot p'_i)^T \\ &= \frac{1}{n} \sum_{i=1}^n (T \cdot p'_i \cdot p'^T_i \cdot T^T) \\ &= \frac{1}{n} T \cdot \left( \sum_{i=1}^n p'_i \cdot p'^T_i \right) \cdot T^T = \frac{1}{n} T \cdot \bar{M} \cdot T^T \end{aligned} \quad (\text{EQ 2})$$

with  $\bar{M}$  denoting the new *generic homogeneous* covariance matrix of points  $p_1 \dots p_n$ . Thus from calculating  $\bar{M}$  we can derive a covariance matrix  $M$  with respect to any center  $o$  as the upper-left 3x3 sub-matrix of  $T \cdot \bar{M} \cdot T^T$  using the translation matrix  $T$  with parameters  $(-o_x, -o_y, -o_z)$  and multiplied by  $1/n$ .

In fact, we can now express the homogeneous covariance matrix of a set of points for any local coordinate system from the generic homogeneous covariance matrix as

$$M = \frac{1}{n} R \cdot T \cdot \bar{M} \cdot T^T \cdot R^T \quad (\text{EQ 3})$$

with  $T$  denoting the translation of the point set with respect to the new origin and  $R$  expressing the rotation into the new coordinate axis directions. Furthermore, from  $M = |M|^{\text{upper-left } 3 \times 3 \text{ matrix}}$  the 2D covariance matrices  $M^{x,y}$ ,  $M^{y,z}$  or  $M^{z,x}$  of the points projected into the  $x,y$ ,  $y,z$  or  $z,x$ -planes, of the new local coordinate system, can be

formed by simply removing the row and column vectors from  $M$  corresponding to the projection axis.

The introduced generic homogeneous covariance matrix now allows efficient calculation bottom-up in the multiresolution hierarchy as indicated in the previous Section 3.2. Given two different sets of points  $P = \{p_1, \dots, p_n\}$  and  $Q = \{q_1, \dots, q_m\}$  as well as their covariance matrices  $\bar{M}_P = \sum p'_i \cdot p'^T_i$  and  $\bar{M}_Q = \sum q'_i \cdot q'^T_i$  as sum of tensor products, the combined generic covariance matrix  $\bar{M}$  of the union  $P \cup Q$  is simply given by

$$\bar{M} = \bar{M}_P + \bar{M}_Q. \quad (\text{EQ 4})$$

Thus the expensive tensor product sums of all points are only computed once for the leaf nodes of the multiresolution hierarchy in  $O(n)$  time. All non-leaf nodes compute their generic homogeneous covariance matrix by component-wise addition from the child nodes using Equation 4 as suggested in Figure 4.

### 3.4 Splat-size determination

The elliptical disks  $e_c$  of nodes  $c \in H$  must cover the surface at all levels in the multiresolution representation. Based on the generic homogeneous covariance matrix we show how these elliptical disks are kept covering the surface at all LODs. The basic principle of our approach is to project the set of points  $p_1 \dots p_k$  of a node  $c$  onto the tangent plane  $\kappa_c: \bar{n}_c \cdot (x - \bar{p}_c) = 0$  defined by  $\bar{p}_c$  and normal orientation  $\bar{n}_c$ , and to calculate a bounding ellipse  $e_c$  in this tangent plane as illustrated in Figure 5.

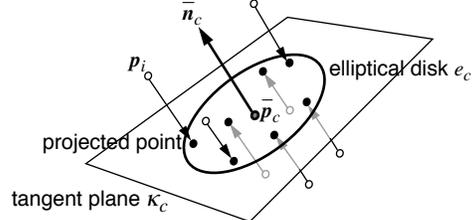


FIGURE 5. Projection of points onto tangent plane  $\kappa_c$  at position  $\bar{p}_c$  and with normal  $\bar{n}_c$ .

Using the generic homogeneous covariance matrix  $\bar{M}'_c$  we first get the elliptical distribution of points in the tangent plane  $\kappa_c$  and second we adjust the ellipse to include all points.

Let us first outline how we get the ellipse axis and its axis-ratio within the tangent plane  $\kappa_c$ . For a node  $c$  and its matrix  $\bar{M}'_c$  denoting the covariance of all points  $p_i$  represented by  $c$ , we apply a coordinate system transformation  $RT$  according to Equation 3 using a translation matrix  $T$  with the last column being  $(-\bar{p}_x, -\bar{p}_y, -\bar{p}_z, 1)^T$  from the average  $\bar{p}_c$  and a rotation matrix  $R$  with row vectors  $R_x = R_y \times R_z$ ,  $R_y = (0, -\bar{n}_z, \bar{n}_y, 0)$  and  $R_z = (\bar{n}_x, \bar{n}_y, \bar{n}_z, 0)$  given by  $\bar{n}_c$ . The resulting transformed homogeneous matrix  $M'$  then expresses the covariance in the local tangent-space coordinate system. Moreover, its upper-left 2x2 sub-matrix  $M'^{x,y}$  represents the covariance matrix of all points projected onto the tangent plane  $\kappa_c$ . We get the ellipse axis-ratio of the point distribution in  $\kappa_c$  from the eigenvalue decomposition of  $M'^{x,y}$ , and

we obtain the eigenvalues  $\lambda_1$  and  $\lambda_2$  from solving the quadratic equation

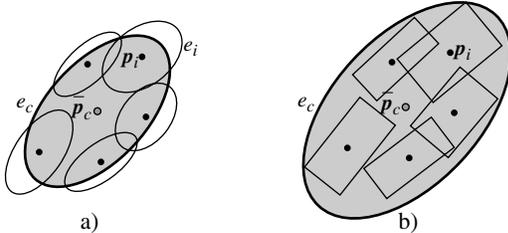
$$\lambda^2 + \text{trace}(M^{x,y}) \cdot \lambda + \det(M^{x,y}) = 0. \quad (\text{EQ } 5)$$

Furthermore, we obtain the major and minor axis orientations of the bounding ellipse in the tangent plane  $\kappa_c$  by solving

$$M^{x,y} \cdot \mathbf{v}_i = \lambda_i \cdot \mathbf{v}_i \quad (\text{EQ } 6)$$

for the eigenvectors  $\mathbf{v}_1$  and  $\mathbf{v}_2$ . Note that  $\mathbf{v}_1$  and  $\mathbf{v}_2$  are in  $\mathbf{R}^2$ , the tangent plane  $\kappa_c$ . However, with respect to the local coordinate system with  $z$ -axis perpendicular to  $\kappa_c$  we get the appropriate 3D vectors by setting  $z$  to zero,  $\mathbf{v}_i^T = (\mathbf{v}_i^T, 0)$ . The world-coordinate system ellipse axis  $\mathbf{e}_1$  and  $\mathbf{e}_2$  are obtained by applying the inverse rotation  $R^{-1}$  of the coordinate system transform as  $\mathbf{e}_i = R^{-1} \cdot \mathbf{v}_i$  and normalization to unit length. Now we have defined a planar elliptical disk  $e_c$  in the world coordinate system with center  $\bar{\mathbf{p}}_c$ , axis directions  $\mathbf{e}_1, \mathbf{e}_2$  perpendicular to  $\bar{\mathbf{n}}_c$  as well as major axis length  $a' = \lambda_1$  and minor axis length  $b' = \lambda_2$ .

The so defined elliptical disk does not yet exactly bound all points  $\mathbf{p}_i$  projected onto  $\kappa_c$  and its size must be scaled to  $a = fa'$  and  $b = fb'$ . We obtain the necessary maximal scale factor  $f$  by evaluating the ellipse equation  $f^2 = x^2/a^2 + y^2/b^2$  in the tangent plane  $\kappa_c$  spanned by  $\mathbf{e}_1$  and  $\mathbf{e}_2$  for all points  $\mathbf{p}_i$ , with  $x_i = (\mathbf{p}_i - \bar{\mathbf{p}}_c) \cdot \mathbf{e}_1$  and  $y_i = (\mathbf{p}_i - \bar{\mathbf{p}}_c) \cdot \mathbf{e}_2$ . However, since every surface element  $s_i$  represents an elliptical disk  $e_i$  and not just a single point  $\mathbf{p}_i$  we generate bounding ellipses that not only include  $\mathbf{p}_i$  but cover the entire disks  $e_i$ , approximated by bounding boxes as illustrated in Figure 6. Without this conservative measure a coarse LOD representation would not cover the surface well.



**FIGURE 6.** a) Ellipse  $e_c$  bounding only the points  $\mathbf{p}_i$  and b) conservatively bounding the disks  $e_i$ .

The outlined generation of elliptical disks cannot only be used to compute bounding surfels of nodes  $c$  of the multiresolution hierarchy  $H$ , but could also be applied in a similar way to obtain elliptical disks of the initial input point set. For this one would compute the  $k$ -nearest neighbors of each point, calculate the average normal if necessary, compute the covariance matrix of this neighborhood and get a bounding ellipse of the  $k$ -nearest neighbors as outlined above.

## 4. Point Blending

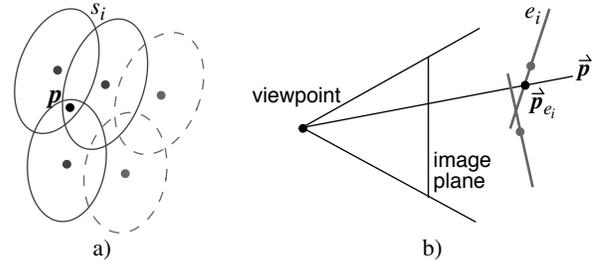
### 4.1 Continuous interpolation

Our approach of smoothly interpolating surface parameters between surfels in object-space is closely related to parametric curved surface design using weighted blending of control points such as for example Bezier or B-Spline surfaces. Given a grid of control points  $\mathbf{p}_{i,j}$  and weight functions  $B_{i,j}$  a parametric surface is given by  $s(u, v) = \sum_{i,j} B_{i,j}(u, v) \cdot \mathbf{p}_{i,j}$  (generally for  $u$  and  $v \in [0, 1]$ ). In particular, the blending functions  $B_{i,j}$  satisfy the *positivity*  $B_{i,j}(u, v) \geq 0$  and *partition-of-unity*  $\sum_{i,j} B_{i,j}(u, v) = 1$  criteria. Blending functions with *global support* have  $B_{i,j}(u, v) > 0$  for the open interval  $u, v \in (0, 1)$ , while *local support* means non-zero weights  $B_{i,j}(u, v)$  only for  $u$  and  $v$  within a sub-interval  $I_{i,j} \subset [0, 1] \times [0, 1]$  denoting the limited support of control point  $\mathbf{p}_{i,j}$ .

Similarly we interpret the interpolation of surface parameters such as color in object-space between surfels  $s_1 \dots s_n$  as a weighted sum. In fact, we need to compute the interpolated color  $\hat{\mathbf{c}}_{\hat{\mathbf{p}}}$  of a pixel  $\hat{\mathbf{p}}$  which is the perspective projection of a point  $\mathbf{p}$ . Thus Equation 7 interpolates between the visible surfels  $s_i$  whose elliptical disks  $e_i$  have a non-empty intersection with the projection  $\hat{\mathbf{p}}$  as illustrated in Figure 7.

$$\hat{\mathbf{c}}_{\hat{\mathbf{p}}} = \sum_{\forall i: \hat{\mathbf{p}} \cap e_i \neq \emptyset} \Psi_i(u_i, v_i) \cdot \mathbf{c}_i \quad (\text{EQ } 7)$$

The blending function  $\Psi_i$  of the surfel  $s_i$  is only positively defined over its *local support*, the elliptical disk  $e_i$ , and zero outside. The local parametrization of  $\Psi_i$  with respect to a pixel  $\hat{\mathbf{p}}$  is given by the intersection  $\hat{\mathbf{p}}_{e_i} = \hat{\mathbf{p}} \cap e_i$  in the plane of the elliptical disk  $e_i$ . The parameters  $u_i, v_i$  then specify the intersection point  $\hat{\mathbf{p}}_{e_i}$  expressed as linear combination of the two ellipse axis directions  $\mathbf{e}_1$  and  $\mathbf{e}_2$ , thus  $\hat{\mathbf{p}}_{e_i} = u_i \cdot \mathbf{e}_1 + v_i \cdot \mathbf{e}_2$ . Note that  $u_i$  and  $v_i$  never have to be explicitly computed because the blending function  $\Psi_i$  is implemented as an  $\alpha$ -texture mapped on the disk  $e_i$ , see also Section 5.2.



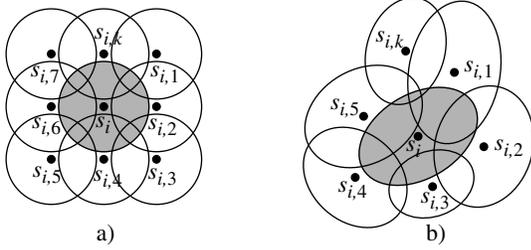
**FIGURE 7.** a) Surfels  $s_i$  overlapping  $\mathbf{p}$  (dashed surfels do not contribute). b) Side view of projection  $\hat{\mathbf{p}}$  intersecting the planar elliptical disks  $e_i$ .

In order to achieve a continuous interpolation, the blending functions  $\Psi_i$  must satisfy the *positivity*  $\Psi_i(\hat{\mathbf{p}}) \geq 0$  and *partition-of-unity*  $\sum_{\forall i: \hat{\mathbf{p}} \cap e_i \neq \emptyset} \Psi_i(\hat{\mathbf{p}}) = 1$  criteria for any given  $\hat{\mathbf{p}}$ . We can define a conforming blending function  $\Psi_i$  as a normalization of simple rotation-symmetric blending

kernels  $\psi$ . Given a surfel  $s_i$  and its overlapping neighbors  $s_{i,1} \dots s_{i,k}$  as illustrated in Figure 8 we define its conforming blending function as

$$\Psi_i(\vec{p}) = \frac{\psi_i(\vec{p})}{\psi_i(\vec{p}) + \sum_{j=1}^k \psi_j(\vec{p})}. \quad (\text{EQ 8})$$

Thus given an arbitrary, positive blending kernel  $\psi$  used for each surfel, and to achieve partition-of-unity its final contribution to a pixel is normalized by the sum of all other surfels' blending kernels contributing to that same pixel. Per-pixel visibility determination of contributing surfels is detailed in Section 5.3.



**FIGURE 8.** a) Regular grid of control points. b) Irregular set of surface elements.

For control points distributed on a regular grid in a global  $u, v$ -parameter space as shown in Figure 8 a) a solution to Equation 8 is simple to achieve and a single blending function  $\Psi$  can be obtained to be used for each control point. In our case, however, we have a point sampled geometry with surfels irregularly distributed over the surface as illustrated in Figure 8 b). Therefore, conforming blending functions according to Equation 8 have to be computed for each individual surfel resulting in a computationally intractable solution. Note that any conforming blending function  $\Psi$  with limited support is not rotationally symmetric.

The solution to Equation 8 is to separate Equation 7 into separate sums for the nominator and the denominator as

$$c_{\vec{p}} = \frac{\sum_{\forall i: \vec{p} \cap e_i \neq \emptyset} \psi_i(\vec{p}) \cdot c_i}{\sum_{\forall i: \vec{p} \cap e_i \neq \emptyset} \psi_i(\vec{p})}. \quad (\text{EQ 9})$$

Thus we can perform the interpolation of Equation 7 using a simple blending kernel  $\psi$  and get an intermediate blended color

$$c'_{\vec{p}} = \sum_{\forall i: \vec{p} \cap e_i \neq \emptyset} \psi_i(\vec{p}) \cdot c_i. \quad (\text{EQ 10})$$

Then the resulting intermediate color  $c'_{\vec{p}}$  of a pixel  $\vec{p}$  after blending all contributing surfels has a weight  $w_{\vec{p}} = \sum_{\forall i: \vec{p} \cap e_i \neq \emptyset} \psi_i(\vec{p})$  of value less than 1, thus not (yet) partitioning unity. However, this weight  $w_{\vec{p}}$  is exactly the same value as the denominator of Equation 9. Hence we can obtain the final and correct color  $c_{\vec{p}}$  by first blending the surfels according to Equation 10 to get the intermediate per-pixel result  $c'_{\vec{p}}$  and then performing a post-process per-pixel normalization by dividing the color intensity  $c'_{\vec{p}}$  by its weight  $w_{\vec{p}}$ . Note that this normalization to guarantee

partition-of-unity must be performed per pixel and its implementation is explained in Section 5.4.

## 4.2 Blending kernels

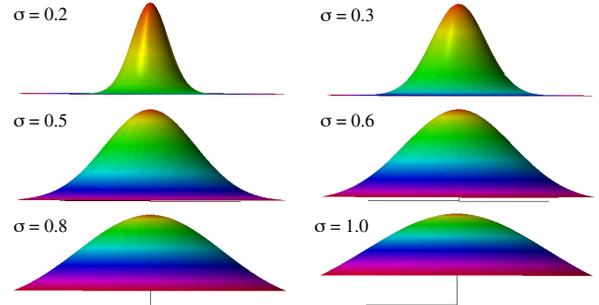
As shown in the previous section we can shift our focus to define a simple two-dimensional rotationally symmetric blending kernel  $\psi: r \rightarrow [0, 1]$ , apply it to each surfel  $s_i$  scaled and oriented according to the ellipse  $e_i$ , and achieve correct interpolation by a post-process per-pixel 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 and for future research we would like to define 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:  $\psi'(c_{\text{mid}}) = s$

A Gaussian blending kernel given by

$$\psi_G(r) = e^{\frac{-r^2}{2\sigma^2}} \quad (\text{EQ 11})$$

satisfies the first two criteria of positivity and smoothness, however, clearly does not have a limited support and very limited control over its shape. Figure 9 shows several Gaussian blending kernels  $\psi_G(r)$  over a limited domain  $r < 1.5$  for varying parameters  $\sigma$ . Obviously the kernel can be made more or less localized with varying  $\sigma$ , however, this single parameter concurrently influences the slope of the interpolation around  $\psi_G(c_{\text{mid}}) = 0.5$  and the support  $\psi_G(c_{\max}) \leq \epsilon$ .



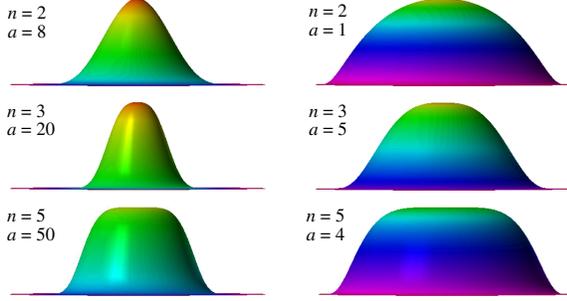
**FIGURE 9.** Gaussian blending kernels for  $r < 1.5$  and with:  $\sigma = 0.2, 0.3, 0.5, 0.6, 0.8, 1.0$  (u.l. to l.r.).

We propose a new blending kernel  $\psi_E(r)$  that supports plateau-like blending kernels with variable sharp drop-off regions:

$$\psi_E(r) = e^{-a \cdot \left(\frac{r}{b}\right)^n} \quad \text{for } 0 \leq r \leq b \quad (\text{EQ 12})$$

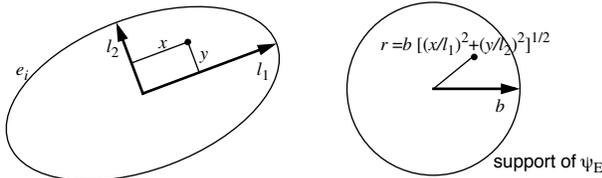
The novel blending kernel defined in Equation 12 supports all five desired control criteria specified above. In particular it is well defined over a limited support area specified by the parameter  $b$  with  $\lim_{r \rightarrow b} \psi_E(r) = 0$ . Furthermore one can adjust the width of the kernel with parameter  $a$  and sep-

arately control the slope of the drop-off region by the parameter  $n$ . As can be seen from Figure 10, higher settings of  $a$  make for a more narrow blending kernel (for a fixed  $n$ ), while high values for  $n$  generate plateau like kernels.  $\psi_E(r)$  also allows to simulate Gaussian shaped kernels if desired, for example  $\psi_E(r)$  with  $n = 2$  and  $a = 10$  is very similar to  $\psi_G(r)$  with  $\sigma = 0.3$ .



**FIGURE 10.** Smooth blending kernels  $\psi_E(r)$  for  $b = 1.5$  and with:  $n = 2$ ,  $a = 8$  and  $a = 1$ ;  $n = 3$ ,  $a = 20$  and  $a = 5$ ;  $n = 5$ ,  $a = 50$  and  $a = 4$  (u.l. to l.r.).

The kernel  $\psi_E(r)$  defined on the disk with radius  $b$  is mapped to an elliptical surfel disk  $e_i$  by simple scaling. For  $e_i$  with major and minor axis lengths  $l_1$  and  $l_2$ , a point  $(x, y)$  on  $e_i$  is mapped to  $r = b\sqrt{(x/l_1)^2 + (y/l_2)^2}$  as shown in Figure 11.



**FIGURE 11.** Mapping from elliptical surfel disk  $e_i$  to the support domain of the blending kernel  $\psi_E$ .

## 5. Rendering

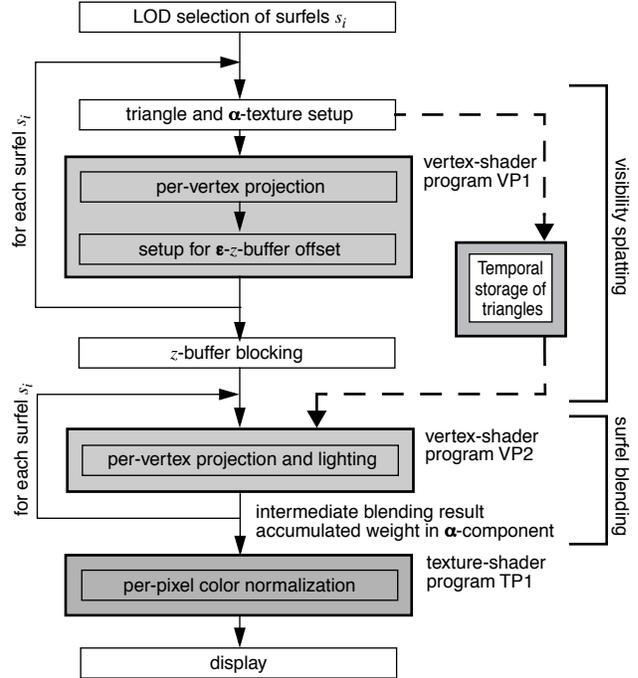
### 5.1 Overview of algorithm

Our proposed point blending and splatting algorithm uses hardware graphics acceleration to efficiently render and scan convert surfels as  $\alpha$ -textured polygons. Exploiting vertex- and pixel-shader programmability we present an efficient rendering algorithm that performs visibility splatting and blending. The blending correction is then performed by a per-pixel normalization post process. Our rendering algorithm performs the following steps for each frame:

1. View-dependent LOD selection of surfels  $s_i$  with projected screen size less than a threshold of  $\tau$  pixels.
2. Selected surfels  $s_i$  are represented as triangles oriented according to the surfel normal  $\mathbf{n}_i$  and the ellipse axis  $e_{i1}$  and  $e_{i2}$ . The blending kernel  $\psi_E(r)$  is specified by an  $\alpha$ -texture on the triangle.
3. The surfels are visibility splatted using an  $\epsilon$ -z-buffer concept such that surfels within  $\epsilon$  distance of each other are blended together according to Equation 10.

4. The intermediate blending result is normalized in an imaging post process by dividing each pixel color by its accumulated blending weight.

Figure 12 illustrates the different stages of our point rendering process. The first step of view-dependent LOD selection is not further explained in this paper as it is a straight forward depth-first traversal of a multiresolution hierarchy. Given the hierarchy  $H$  a top-down traversal decides for each node  $c \in H$  if the corresponding surfel is visible, within the view-frustum and front-facing. Furthermore, if its elliptical disk projection on screen is larger than a threshold of  $\tau$  pixels the child nodes are recursively processed.

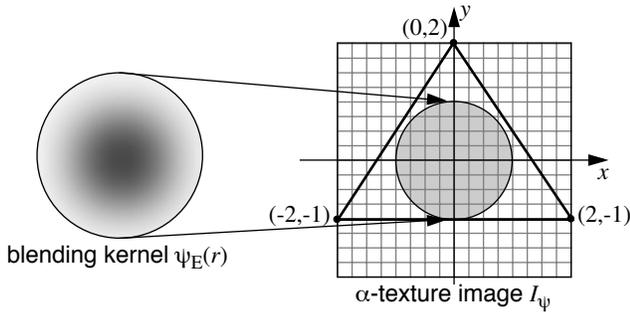


**FIGURE 12.** Point rendering overview.

Our rendering algorithm is implemented in OpenGL and takes advantage of nVIDIA's vertex-program [Ope02] and texture-shader [DS01] extensions.

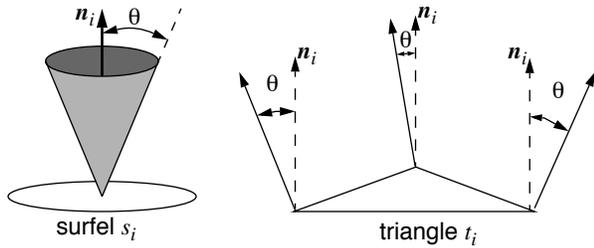
### 5.2 Blending

As mentioned earlier, blending is performed by mapping the rotationally symmetric and circular blending kernel  $\psi_E(r)$  onto the elliptical disk  $e_i$  of each visible surfel  $s_i$ . In fact, the blending kernel  $\psi_E(r)$  is precomputed and discretized in a pre-process, and the result is stored in a high-resolution raster image  $I_\psi$ . As shown in Figure 13, this  $\alpha$ -texture  $I_\psi$  is mapped onto a generic triangle  $t_{\text{unit}}$  in the  $x, y$ -plane such that the blending kernel covers a unit-size disk. For each surfel  $s_i$ , the triangle setup stage in Figure 12 scales the generic triangle  $t_{\text{unit}}$  in  $x$ - and in  $y$ -directions according to the major and minor axis lengths of ellipse  $e_i$ , rotates the triangle to align with the surface normal  $\mathbf{n}_i$  and the two ellipse axis, and then translates it to the object-space coordinates  $\mathbf{p}_i$ , resulting in triangle  $t_i$ . Thus each surfel is rendered as one triangle and blending is achieved by  $\alpha$ -texturing.



**FIGURE 13.** Mapping of blending kernel  $\psi_E(r)$  as  $\alpha$ -texture onto a generic triangle.

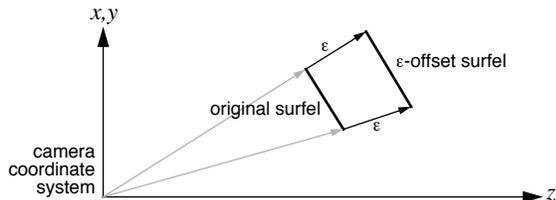
If a coarse LOD surfel  $s_i$  has a normal cone of semi-angle  $\theta$  associated with it, then the corners of the triangle  $t_i$  can have surface normals that are offset by  $\theta$  from  $n_i$  as shown in Figure 14. This allows that the elliptical splat will actually be Gouraud shaded in the graphics hardware which increases the smoothness of the point blending.



**FIGURE 14.** Triangle normals offset by normal cone semi-angle  $\theta$ .

### 5.3 Visibility splatting

Visibility splatting uses an  $\epsilon$ -z-buffer rendering concept similar to [GD98] and [RPZ02], implemented by a two-pass rendering approach. In a first pass all selected surfels  $s_i$  are rendered with lighting and  $\alpha$ -blending disabled. Note however, that the  $\alpha$ -test function is enabled such that the  $\alpha$ -texture map  $I_\psi$  indeed generates an elliptical splat during scan-conversion of the rendered triangle. During this first pass the corners of the triangle representing a surfel are displaced by  $\epsilon$  along the perspective projector as shown in Figure 15. After this first rendering pass the depth-buffer represents the rendered surface perspectively translated by  $\epsilon$  along the view-projection.



**FIGURE 15.** Application of a depth  $\epsilon$ -offset along view-projection for visibility splatting.

At the beginning of the rendering stage, and for each of the selected surfels  $s_i$ , the generic reference triangle  $t_{\text{unit}}$

needs to be translated, oriented and scaled accordingly to the surfel information yielding  $t_i$ . Since this will be repeated for the second rendering pass and since it involves expensive CPU calculations, a temporal data structure is set to store the transformed values of the vertices of each surfel triangle  $t_i$ .

Additionally, since the purpose of the first rendering pass is only to set the  $\epsilon$ -offset z-buffer, the triangles sent to the graphics card do not require any color or normal information. A simple vertex program, VP1 in Figure 12, performs the appropriate object-space to screen-space transformations of this first rendering pass. VP1 also performs the perspective translation of each vertex of  $t_i$  by  $\epsilon$  as illustrated in Figure 15.

In the second rendering pass the depth-buffer is set to read-only and all visible surfels are rendered with lighting and  $\alpha$ -blending enabled. Since the depth-buffer contains the visible surface offset by  $\epsilon$ , the desired  $\epsilon$ -z-buffering effect is achieved.

For the second rendering pass, the object-space coordinates of each surfel triangle  $t_i$  are read from the temporal data structure and sent to the standard OpenGL pipeline or a vertex program – VP2 in Figure 12 in our case – with per-vertex normal,  $\alpha$ -texture and material properties in order to incorporate Gouraud shading and lighting.

### 5.4 Normalization

After visibility splatting and blending, the resulting image  $I$  contains the interpolated pixel values according to Equation 10. The pixel color values contain the intermediate blending result  $c' = (\alpha \cdot R, \alpha \cdot G, \alpha \cdot B, \alpha)$ , the  $\alpha$  component contains the accumulated blending weight. These color values constitute the correct proportionally blended color values, however, the  $\alpha$  values need not be 1.0 as required. To get the final desired color  $c = (R, G, B, 1.0)$  each color component of  $c'$  has to be multiplied by  $\alpha^{-1}$ . This normalization is performed as an image post-process stage on the intermediate blending result, see also Figure 12.

Without any hardware extensions to perform complex per-pixel manipulations this normalization step has to be performed in software. However, widely available graphics accelerators now offer per-pixel shading operators that can be used more efficiently. In our current implementation, we perform this normalization in hardware using nVIDIA's OpenGL *Texture Shader* extension [DS01].

To compensate the illumination deficiency we perform a remapping of the  $R$ ,  $G$  and  $B$  values based on the value of  $\alpha$ . During initialization time we construct a texture encoding in  $(s, t)$  of a look-up table of transparency and luminance values respectively, from 0 to 255 possible values as shown in Figure 16. The pixels of this texture encode the new intensity for a given luminance ( $t$ ) compensated with the  $\alpha$  transparency ( $s$ ).

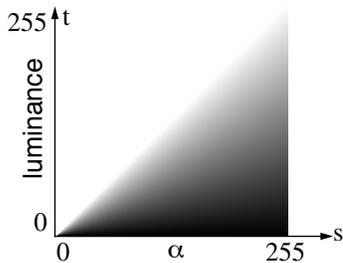


FIGURE 16. Alpha-Luminance map.

Based on this alpha-luminance map, we proceed to correct each of the  $R, G$  and  $B$  channels of every pixel. Using nVIDIA’s texture shader extension operation `GL_DEPENDENT_AR_TEXTURE_2D_NV` the graphics hardware can remap the  $R$  and  $\alpha$  by a texture lookup with coordinates  $s = \alpha$  and  $t = R$  into our alpha-luminance map. At this point, rendering a quadrilateral with the intermediate image  $I$  as texture-one and the alpha-luminance map as texture-two, and setting the color mask to block the  $G, B$  and  $\alpha$  channels will compensate the red channel by  $\alpha^{-1}$ . Note that only the  $R$  and  $\alpha$  channels are used by this dependent texture replace operation. Therefore, we need to remap the  $G$  and  $B$  channels to the  $R$  channel of two new images  $I_G$  and  $I_B$  while copying the  $\alpha$  channel as well. This is done by rendering two quads and using nVIDIA’s register combiners. Then the dependent texture replace operation is also performed on the images  $I_G$  and  $I_B$ . Thus by separating the  $RGB$  channels into three different images and using the  $\alpha R$ -dependent texture replace operation we get the corrected  $RGB$  values (in the red channel of three new images). Figure 17 illustrates this normalization process.

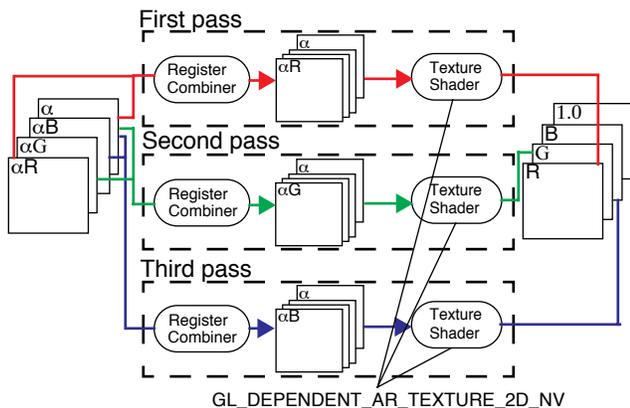


FIGURE 17. Per-pixel normalization process.

## 6. Experimental Results

The blending kernel  $\psi_E(r)$  introduced in Section 4 has many advantages due to its flexibility in parametrization. Figure 18 shows a checkerboard of  $512 \times 512$  point samples rendered with a fairly “round” kernel in Figure 18 a) and

with a more localized kernel in Figure 18 b) (see also Figure 10). Clearly the more localized kernel exhibits less blurring effects, but it nevertheless provides to some extent antialiasing in object-space.

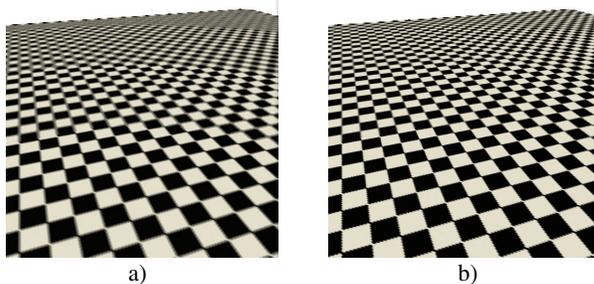


FIGURE 18. Different blending kernels  $\psi_E$  with parameters: a)  $a = 1.0, b = 1.5$  and  $n = 2.0$  and b)  $a = 6.0, b = 1.5$  and  $n = 3.0$ . (both rendered at screen-projection tolerance  $\tau = 0.02\%$  of window area)

In Figure 19 we can see the effect of a wide plateau like blending kernel  $\psi_E(r)$  in image b) compared to a more round but narrow blending kernel in image a). The wide plateau shaped kernel provides very smooth blending while retaining detail sharpness; the narrow kernel can result in an almost faceted appearance.



FIGURE 19. Blending kernels  $\psi_E$  with parameters: a)  $a = 6.0, b = 1.5$  and  $n = 3.0$  and b)  $a = 4.0, b = 1.5$  and  $n = 5.0$ . ( $\tau = 0$ )

The performance of our point blending and splatting method was measured using several color textured models. All performance measures were taken on a 1.5GHz Pentium4 CPU and nVIDIA GeForce4 Ti4600 GPU.

In Table 1 we provide timings of the pre-process that generates the octree multiresolution hierarchy and computes the elliptical splat sizes as described in Section 3. We can see that our multiresolution model and splat size generation achieves a performance of processing about 100,000 input points per second. Even multi-million point models can efficiently be processed by our method, and the splat size generation using the homogeneous covariance computation is very efficient.

Model	#Points	#Nodes	Time
David	2000606	802371	20.6s
Male	321237	126773	3.10s
Female	302948	121439	2.97s
Balljoint	137062	54992	1.37s
Dragon	437645	173507	4.26s

**TABLE 1.** Multiresolution point hierarchy construction and splat generation times.

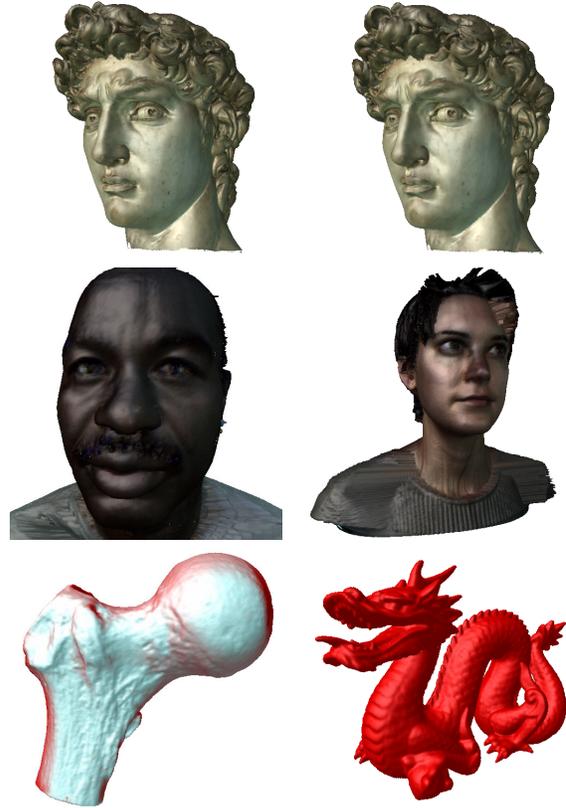
The performance of our point blending, visibility splatting and color normalization algorithm is summarized in Table 2. It lists the number of actually visible and processed splats, the time for LOD selection, the time for blending and visibility splatting, and the time for color normalization all given in seconds for one rendered frame. We can see that our approach can render up to and over 600,000 points per second. Note that unlike in other experimental results this performance measure includes LOD processing and complete per-vertex illumination, as well as a per-pixel normalization of the weighted blended colors.

Model	Tol. $\tau$	#Splats	LOD	Splatting	Normalization	Total
David	0.0%	904121	0.315s	1.386s	0.0002s	1.710s
	0.01%	454656	0.269s	0.791s	0.00019s	1.061s
	0.12%	340537	0.195s	0.612s	0.0002s	0.808s
Male	0.0%	116549	0.038s	0.160s	0.00203s	0.199s
Female	0.0%	147098	0.048s	0.217s	0.00019s	0.266s
Balljoint	0.0%	60663	0.018s	0.081s	0.00019s	0.100s
Dragon	0.01%	163238	0.070s	0.246s	0.00019s	0.317s

**TABLE 2.** Rendering performance is given for each task in seconds used per frame. Splats denotes the number of visible surfels at the given screen-space tolerance  $\tau$ . (tolerance  $\tau$  in percent of the window area size)

All tests and images have been rendered at a resolution of 512x512 pixels. Note however, that higher resolutions such as 1024x1024 pixels have not shown any noticeable change in rendering performance. Our rendering algorithm is not dependent on the output image size.

Screen captured images of all test models are given in Figure 20. Note the high quality of the David model rendered at a screen-space tolerance of  $\tau = 0.01\%$  of the window area and the reduction of visible splats by 50% compared to the zero tolerance. All models provide a smooth texture color interpolation and blending of surfels using a blending kernel  $\psi_E(r)$  with parameters  $a = 6.0$ ,  $b = 1.5$  and  $n = 3.0$ .



**FIGURE 20.** From top-left to lower-right: David at  $\tau=0$  and  $\tau=0.01$ ; Male, Female and Balljoint at  $\tau=0$ ; and Dragon at  $\tau=0.01$ . (screen-space tolerance  $\tau$  given in percent of window area)

## 7. Conclusion

We have presented a novel point-based rendering algorithm that interprets the problem of blending between surface elements as an interpolation in object-space. We have analyzed the problem of object-space point interpolation and demonstrated that correct blending can be achieved even for irregularly distributed points using simple blending kernels and a per-pixel normalization post-process. Furthermore, we have provided mathematical details on how to generate appropriate splat sizes for irregular point samples maintained in a space-partitioning multiresolution hierarchy. We have also described an efficient hardware accelerated point splatting algorithm. Our approach provides great flexibility in the shape of the desired blending kernels and is capable of efficiently generating high-quality texture color interpolation between surface samples.

Future work will address the problem of transparent surfaces within this framework and the development of a single-pass point blending and splatting algorithm. Furthermore, we want to improve the parametrization of our new blending kernels such that the parameters affect the shape of the blending function more independently and are easier to use.

## Acknowledgements

We would like to thank the Stanford *3D Scanning Repository* and *Digital Michelangelo* projects as well as *Cyberware* for freely providing geometric models to the research community.

## References

- [ABCO+01] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shackar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In *Proceedings IEEE Visualization 2001*, pages 21–28. Computer Society Press, 2001.
- [BWK02] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. Efficient high quality rendering of point sampled geometry. In *Proceedings Eurographics Workshop on Rendering*, pages –, 2002.
- [CAZ01] Jonathan D. Cohen, Daniel G. Aliaga, and Weiqiang Zhang. Hybrid simplification: Combining multi-resolution polygon and point rendering. In *Proceedings IEEE Visualization 2001*, pages 37–44, 2001.
- [CL96] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings ACM SIGGRAPH 96*, pages 303–312. ACM Press, 1996.
- [CN01] Baoquan Chen and Minh Xuan Nguyen. POP: A hybrid point and polygon rendering system for large data. In *Proceedings IEEE Visualization 2001*, pages 45–52, 2001.
- [DGH01] Tamal K. Dey, Joachim Giesen, and James Hudson. A delaunay based shape reconstruction from large data. In *Proceedings IEEE Symposium in Parallel and Large Data Visualization and Graphics*, pages 19–27, 2001.
- [DH02] Tamal K. Dey and James Hudson. PMR: Point to mesh rendering, a feature-based approach. In *Proceedings IEEE Visualization 2002*, pages 155–162. Computer Society Press, 2002.
- [DS01] Sébastien Dominé and John Spitzer. Texture shaders. Developer Documentation, 2001.
- [GD98] J.P. Grossman and William J. Dally. Point sample rendering. In *Proceedings Eurographics Rendering Workshop 98*, pages 181–192. Eurographics, 1998.
- [GKS00] M. Gopi, S. Krishnan, and C.T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. In *Proceedings EUROGRAPHICS 00*, pages 467–478, 2000.
- [Gro01] Markus Gross. Are points the better graphics primitives? Computer Graphics Forum 20(3), 2001. Plenary Talk Eurographics 2001.
- [HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings ACM SIGGRAPH 92*, pages 71–78. ACM Press, 1992.
- [Hec89] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Master’s thesis, Department of Electrical Engineering and Computer Science, University of California Berkeley, 1989.
- [LPC+00] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings SIGGRAPH 2000*, pages 131–144. ACM SIGGRAPH, 2000.
- [LW85] Marc Levoy and Turner Whitted. The use of points as display primitives. Technical Report TR 85-022, Department of Computer Science, University of North Carolina at Chapel Hill, 1985.
- [Mee01] Gopi Meenakshisundaram. *Theory and Practice of Sampling and Reconstruction for Manifolds with Boundaries*. PhD thesis, Department of Computer Science, University of North Carolina Chapel Hill, 2001.
- [Nie89] J. Nievergelt. 7±2 criteria for assessing and comparing spatial data structures. In *Proceedings of the 1st Symposium on the Design and Implementation of Large Spatial Databases*, volume 409 of *Lecture Notes in Computer Science*, pages 3–27. Springer-Verlag, 1989.
- [Ope02] OpenGL Architecture Review Board. ARB vertex program. OpenGL Vertex Program Documentation, 2002.
- [PGK02] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings IEEE Visualization 2002*, pages 163–170. Computer Society Press, 2002.
- [PZvBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings SIGGRAPH 2000*, pages 335–342. ACM SIGGRAPH, 2000.
- [RL00] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings SIGGRAPH 2000*, pages 343–352. ACM SIGGRAPH, 2000.
- [RPZ02] Liu Ren, Hanspeter Pfister, and Matthias Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Proceedings EUROGRAPHICS 2002*, pages –, 2002. also in Computer Graphics Forum 21(3).
- [SAE93] Leon A. Shirman and Salim S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. In *Proceedings EUROGRAPHICS 93*, pages 261–272, 1993. also in Computer Graphics Forum 12(3).
- [Sam84] Hanan Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 16(2):187–260, June 1984.
- [Sam89] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, Reading, Massachusetts, 1989.
- [WFP+01] Michael Wand, Matthias Fischer, Ingmar Peter, Friedhelm Meyer auf der Heide, and Wolfgang Strasser. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Proceedings SIGGRAPH 2001*, pages 361–370. ACM SIGGRAPH, 2001.
- [ZPKG02] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3D: An interactive system for point-based surface editing. In *Proceedings ACM SIGGRAPH 2002*, pages 322–329. ACM Press, 2002.
- [ZPvBG01] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. EWA volume splatting. In *Proceedings IEEE Visualization 2001*, pages 29–36. Computer Society Press, 2001.