Eurographics Symposium on Geometry Processing (2005)
M. Desbrun, H. Pottmann (Editors)

Sparse Low-degree Implicit Surfaces with Applications to High Quality Rendering, Feature Extraction, and Smoothing

Yutaka Ohtake
RIKEN

Alexander Belyaev
MPI Informatik

Marc Alexa
TU Darmstadt

Abstract

*We propose a new surface representation delivering an accurate approximation to a set of points scattered over a smooth surface by **S**parse **L**ow-degree **IM**plicits (**SLIM**). The SLIM surface representation consists of a sparse multi-scale set of nonconforming surface primitives which are blended along view rays during the rendering phase. This new representation leads to an interactive real-time visualization of large-size models and delivers a better rendering quality than standard splatting techniques based on linear primitives. Further, SLIM allows us to achieve a fast and accurate estimation of surface curvature and curvature derivatives and, therefore, is very suitable for many non-photorealistic rendering tasks. Applications to ray-tracing and surface smoothing are also considered.*

1. Introduction

Efficient approximation, representation, and processing of complex large-size signals, images, and shapes is of primary importance in many information-processing areas. Sparse approximation techniques aimed to build an economical and accurate representation of an input signal as a combination of elementary signals have become increasingly popular in signal and image processing [CDS01] (see also references therein). Sparse representations of 3D shapes and, in particular, 3D point scattered data have received so far considerably less attention although radial basis functions [SPOK95, TO99, CBC*01], partition of unity [OBA*03, OBS04b], moving least squares [ABCO*01, AK04, SOS04], and point splatting [PZ*00, RL00, KV01, BSK04] surface representation techniques fit in the framework of sparse surface approximations.

In this paper, we propose a novel sparse shape representation which approximates a scattered set of points by Sparse Low-degree IMplicits (SLIM for short). Following the general approach of [PZ*00] let us define a surface element, surfel, as the triplet

$$s = (\mathbf{c}, \rho, f(\mathbf{x})), \tag{1}$$

where $\mathbf{c}$ is the center of a ball of radius ρ and

$$f(\mathbf{x}) = 0 \tag{2}$$

delivers a local surface approximation inside the ball. Given a set of points $\mathscr{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$ sampled from a surface in $\mathbb{R}^3$ and equipped with normals, our SLIM representation consists a sparse and hierarchical set of surfels $\{s_1, \ldots, s_M\}$, $M \ll N$, and delivers an accurate multi-resolution shape adaptive approximation of the surface. Since we want to achieve an interactive real-time visualiza-

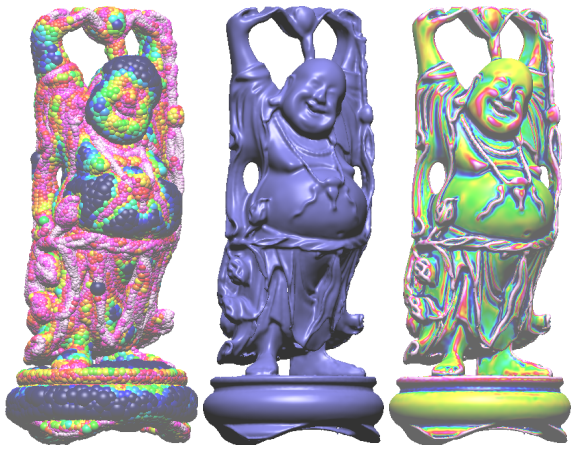

Figure 1: *A SLIM-based approximation of the Stanford Happy Buddha model consisting of only 25K cubic surfels. Left: the surfel balls are colored according to their size which decreases from blue to red. Middle: The model is shaded using the first-order surface derivatives. Left: the mean curvature map.*

tion of large-size models [LPC*00], we use low-degree polynomials (quadratic and cubic) as local approximations in (1).

The SLIM surfaces are not globally smooth (they are C^{-1} smooth in terminology of [WK04]) since they are composed by nonconforming overlapping surfels (1) (an analogy with cabbage leaves seems appropriate here). It is worth to mention here that approximating surfaces by a set of nonconforming smooth elements is now a standard FEM technique (the so-called variational crime) for numerical solving hydrodynamics and thin-plate problems [BS02].

Once the SLIM representation of a given surface is built, we achieve a fast high-quality surface rendering by blending local approximations (2) along view rays. Further we estimate surface curvatures using the same blending-along-view-rays procedure applied to derivatives of local approximations (2).

Figures 1, 2, and 3 demonstrate merits of the SLIM approach in high-quality rendering (photo-realistic and not) and accurate curvature estimation.

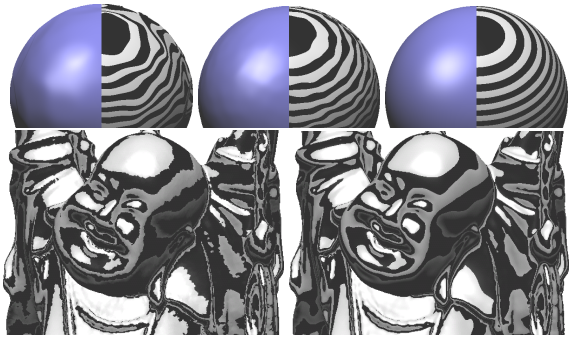

Figure 2: *Shading and reflection lines are used to reveal the quality of approximations. Top: SLIM approximations of a sphere with linear and quadratic surface elements: the sphere is approximated by 128 linear surfels (left), 450 linear surfels (middle), 105 quadratic surfels (right). Bottom: reflection lines on the Phong-shaded Stanford Buddha mesh consisting of about 1M triangles (left) and on a SLIM-based approximation of the model with 50K quadratic primitives (right).*

While most of components of our approach are rather known (our hierarchical representation of splats is similar to that of [LPC*00], our local fitting procedure is an extension of that developed in [OBS04b], the idea of using high-order local approximations for better rendering was also exploited in [KV01], various blending-along-view-rays schemes were used in [PZ*00, SJ00, PSG04], and only the idea of blending of derivatives of local approximations (2) seems completely new), it delivers a unique combination of fast high-quality rendering and interactive curvature feature detection.

In the field of interactive high-quality surface splatting, the main competitors of our approach are Differential Points of Kalaiah and Varshney [KV01] and Phong Splatting of Botsch et al. [BSK04]. Roughly speaking, both Differential Points and Phong Splatting deliver a quadratic accuracy in surface approximation. However rendering with quadratic patches (or using equivalently accurate linear patches in the space of surface normals) fails to deliver accurate results in a small vicinity of the surface curves with vanishing Gaussian curvature (the so-called parabolic lines) where a cubic accuracy approximation is required [OY93]. In contrast, quadratic and cubic patch blending procedures used within the SLIM approach are capable to deliver higher-order surface approximations.

Our simple SLIM-based procedure of estimating high-order surface derivatives leads to a robust and fast detection surface features based on curvatures and curvature derivatives. For large-size models, it can compete with such sophisticated feature detection techniques as one proposed recently in [DFR04]. This makes our approach extremely useful for various non-photorealistic rendering applications [GG01, DFRS03].

The SLIM surface rendering toolkit accompanying the paper allows the reader to verify our claims made above. Due to an out-of-core preprocessing stage and sparsity of the SLIM surface representation, practically there is no limit to the size of models to be processed with the toolkit.

In this paper, we also demonstrate that the projection-based framework of our SLIM-based shading approach (see Section 3 for details) can be easily combined with ray-tracing rendering techniques.

Finally, to stress importance of our approach to geometric modeling tasks, we consider a simple application of the SLIM-based surface representation to surface denoising.

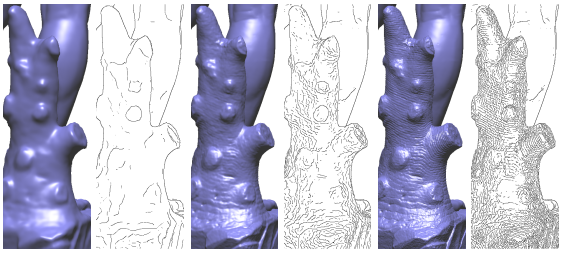

Figure 3: *Shading of - and detecting suggestive contours on - a lower part of the David statue model represented as a SLIM surface at different levels of detail.*

2. Creating SLIM surface representation

The SLIM representation of a smooth surface consists of a set of surfels (1) delivering overlapping local approximations of the surface. Our approach to generating the SLIM representation can be considered as a multi-scale extension of the method of generating an adaptive partition-of-unity surface approximation [OBS04b] and consists of three components: selecting a center $\mathbf{c}$ of a new surfel s, computing the

surfel radius ρ, determining the corresponding local polynomial approximation $f(\mathbf{x}) = 0$. Loosely speaking, we first select randomly a new surfel center $\mathbf{c}$ among hose points of $\mathscr{P}$ which are not sufficiently covered by the already constructed surfels. Then we search an optimal ρ such that the local approximation $f(\mathbf{x}) = 0$ (which depends on $\mathbf{c}$ and ρ) minimizes a certain error metric.

Local polynomial approximation. Given the two first components of a surfel s defined by (1), our task is to construct the third component, local polynomial approximation $f(\mathbf{x})$. Various schemes for local polynomial fitting were analytically and experimentally studied in [CP03, GI04]. For our purposes, the following simple procedure suffices. First the surface normal at surfel center $\mathbf{c}$ is roughly estimated by averaging the surface normals assigned to the points of $\mathscr{P}$ from the surfel ball $\{|\mathbf{p}-\mathbf{c}| < \rho\}$. Then a local coordinate system (u,v,w) with the origin at $\mathbf{c}$ is introduced, such that the plane (u,v) is orthogonal to the normal at $\mathbf{c}$. Finally, the bi-quadratic polynomial associated with surfel s is given by

$$f(\mathbf{x}) = w - \left(a_{11}u^2 + 2a_{12}uv + a_{22}v^2 + a_1u + a_2v + a_o\right),$$

where, $\mathbf{x} = (u,v,w)$ and the unknown coefficients are determined by minimizing the sum $\sum \omega_i f(\mathbf{p}_i)^2$ taken over $\mathscr{P}_s = \mathscr{P} \cap \{|\mathbf{c}-\mathbf{p}| < \rho\}$. Here ω_i are Gaussian-like weights penalizing points which are "too far" from surfel center $\mathbf{c}$. Estimating the linear and cubic local surface approximations is similar.

Optimal surfel radius. Given the surfel center $\mathbf{c}$ we determine the surfel radius ρ using a slightly modified version of the MDL-based procedure proposed in [OBS04b]. The abbreviation MDL stands for the Minimal Description Length principle, a scientific generalization of Occam's razor. Let $\varepsilon(\rho)$ denote a local L^2 error measure estimating the deviation of points $\mathscr{P}_s$ from local approximation $f(\mathbf{x})$. We consider a regularization of $\varepsilon(\rho)^2$

$$E(\rho) = \varepsilon(\rho)^2 + \lambda\,(T_{MDL}/\rho)^2, \qquad (3)$$

where λ is constant and parameter T_{MDL} is a user-specified parameter which controls the trade-off between the sparseness and approximation quality. Indeed the second term in the right-hand side of (3) prevents overfitting and penalizes the number of primitives used to approximate $\mathscr{P}$, see [OBS04b] for a statistical nature of such regularization. To determine λ in (3) we compute the smallest eigenvalue of the co-variance matrix for each point of $\mathscr{P}$ with its ten nearest neighbors and set λ equal to the arithmetic mean of the eigenvalues over all the points of $\mathscr{P}$.

The images of Figure 4 show typical behaviors of functions $\varepsilon(\rho)$ and $E(\rho)$ for surface regions of various geometric complexity. We emphasize here that the both the energies $\varepsilon(\rho)$ and $E(\rho)$ may have multiple minima. We use Brent's method [PTVF93] to find essential minimums of $E(\rho)$. The method is based on parabolic interpolation and starts from three points $\rho_l < \rho_m < \rho_r$ such that the value of $E(\rho)$ at the middle point is less than its values at the end points. Motivated by the right image of Figure 4 we also require that $\varepsilon(\rho_l)$, $\varepsilon(\rho_m)$, and $\varepsilon(\rho_r)$ be monotonically increasing:

$$E(\rho_l) > E(\rho_m) < E(\rho_r) \text{ and } \varepsilon(\rho_l) < \varepsilon(\rho_m) < \varepsilon(\rho_r). \quad (4)$$

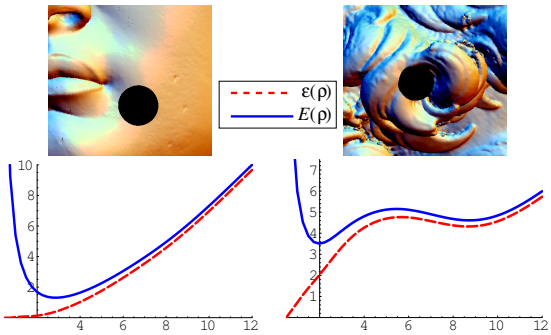

Figure 4: *Graphs of $E(\rho)$ and $\varepsilon(\rho)$ for surface regions of different geometric complexity.*

Apparently, it is not obvious how to determine the desired triplet (ρ_l, ρ_m, ρ_r) in practice. In the following, we use a sliding segment strategy for detecting essential minima of (3) and building a tree-like hierarchical surfel-based shape representation.

Multi-scale surfel-based approximation. Let us denote by L the main diagonal of the bounding box of $\mathscr{P}$, set $\rho_0 = L/10$, and generate a sequence of overlapping segments

$$\left(\rho_l^{(k)}, \rho_m^{(k)}, \rho_r^{(k)}\right) = \rho_0(g^{k+1}, g^k, g^{k-1}), \qquad (5)$$

where $g = (\sqrt{5}-1)/2$ is the golden ratio, as seen in Figure 5

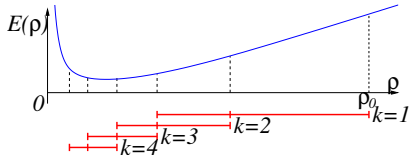

Figure 5: *Sliding intervals for minimum finding of $E(\rho)$.*

To ensure that a set of surfel regions $\mathscr{B} = \{b_j\}$, $b_j = \{|\mathbf{x} - \mathbf{c}_j| < \rho_j\}$, delivers a sufficient cover of $\mathscr{P}$, for each $\mathbf{x}$ we introduce an overlap measure

$$o(\mathscr{B}, \mathbf{x}) = \sum_{b_j \in \mathscr{B}} G_{R_j}(\|x - \mathbf{c}_j\|),$$

where $G_{R_j}(\rho) = G(\rho/R_j)$ is a Gaussian-like function whose tails are smoothly splined to zero and $R_j = \alpha\rho_j$. This additional parameter α plays an important role in our rendering scheme and will be discussed in the next section. We say that point $\mathbf{p} \in \mathscr{P}$ is γ-covered by $\mathscr{B}$ if $o(\mathscr{B}, \mathbf{p}) \geq \gamma$. We have found that $\gamma = 0.1$ works well for all models we tested.

An initial set of balls $\mathscr{B}_1$ is chosen as follows. Initially all

the points of $\mathscr{P}$ are marked as uncovered. We pick $\mathbf{c} \in \mathscr{P}$ randomly and remove from the set of uncovered points those points $\mathbf{p}_i$ for which $\{\|\mathbf{p}_i - \mathbf{c}\| < \rho_0\}$. Then we chose another uncovered point and repeat the whole procedure until no points of $\mathscr{P}$ remain uncovered.

For each selected point $\mathbf{p}$ we define a surfel $s = (\mathbf{c}, \rho, f(\mathbf{x}))$ whose radius ρ is determined by minimizing (3) with the triplet (5) with $k = 1$. Surfel s defines a node at the level $k = 1$. If both the conditions of (4) are satisfied, a value of r minimizing (3) on $[\rho_l, \rho_r]$ is found and assigned to the surfel s which is considered as a leaf node. Otherwise $\rho = \rho_r$ is assigned to the surfel. This procedure creates two families of surfels: the leaf surfels whose regions are balls $\mathscr{B}_1$ and remaining surfels which will serve as internal nodes in a tree-like structure that we are building. We call the latter internal surfels. The local approximations $f(\mathbf{x})$ of the internal surfels at level $k = 1$ are found with $\rho = \rho_r^{(1)} = \rho_0$.

On subsequent levels k, parts of the input point set $\mathscr{P}$ are covered by the balls of the sets $\mathscr{B}_1, \ldots, \mathscr{B}_{k-1}$. The remaining points are uncovered (more precisely, covered only by the regions of internal surfels). As on the first level, approximation centers are chosen randomly from the set of uncovered points until a covering set of centers for level k is constructed. The balls whose radii are determined using the triplet (5) satisfying (4) form $\mathscr{B}_k$ and their surfels are constructed. The remaining balls are the surfel regions or internal surfels whose local approximations $f(\mathbf{x})$ are constructed with $\rho = \rho_r^{(k)} = \rho_0 g^{k-1}$. The procedure is repeated until all points $\mathscr{P}$ are covered by $\mathscr{B} = \bigcup_k \mathscr{B}_k$.

At the next stage, a rendering tree-like structure of balls and their surfels is built. The internal surfels at level k are connected with surfels at the $(k+1)$-level: a link between an internal surfel $s = (\mathbf{c}, \rho, f)$ at the level k and a surfel $s' = (\mathbf{c}', \rho', f')$ (either leaf or internal one) at the $(k+1)$-level is created if $\mathbf{c}'$ lies inside the region of s, as demonstrated in the right image of Figure 6. Finally, a ball enclosing all the constructed balls is added as the root node, see the left image of Figure 6.

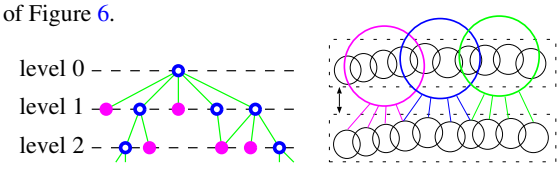

Figure 6: *Surfel/ball tree-like structure built for multi-scale rendering. Left: the leaf nodes are colored in red. Right: establishing links between two subsequent level of the hierarchy of surfels (balls).*

Figure 7 visualizes building the surfel hierarchy for the Stanford Happy Buddha model.

3. SLIM-based Shading

As mentioned in Introduction, typically the primitives of a composite implicit surface (RBF/PU/MLS) are blended in space. Usually direct rendering of such composite implicit surfaces is computationally expensive: blending simple surface primitives leads to an algebraically and geometrically complex surface for which the basic rendering problem, an accurate detection of intersections between a ray and a surface, has high computational complexity. See the left image of Figure 8.

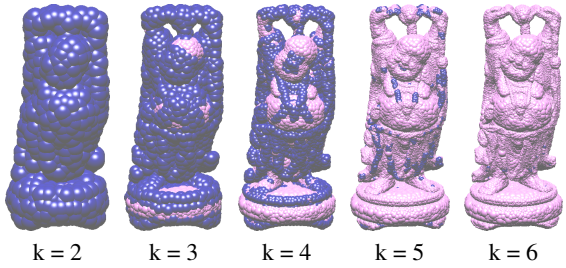

Figure 7: *Converting internal surfels (blue) into leaf surfels (pink) for the Stanford Happy Buddha model.*

Our simple, yet effective idea is to first find the intersections between a viewing ray and the few low-degree (quadratic/cubic) polynomial patches corresponding to surfels closest to the viewer and then interpolate the resulting points on the ray, as shown in the right image of Figure 8. This is in a sense similar to splatting [ZPBG01] when linear functions are used, however, here we use weights that are derived from the construction of the representation.

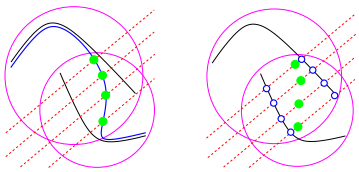

Figure 8: *Left: blending surface primitives in space greatly complicates practical detection of intersections with viewing rays. Right: blending of intersections between surface primitives and viewing rays is computationally simple.*

Generating intersections on rays. Given a ray, we first need a ray point $\hat{\mathbf{q}}$ that is expected to be close to the resulting ray-surface intersection. Once $\hat{\mathbf{q}}$ is determined, a set $\hat{\mathbf{q}}$-supported surfels $\{s_i = (\mathbf{c}_i, \rho_i, f_i(\mathbf{x})) : \|\hat{\mathbf{q}} - \mathbf{c}_i\| < \rho_i\}$ is collected. For each such surfel, intersection $\mathbf{q}_i$ between $f_i(\mathbf{x}) = 0$ and the ray is computed (if there are several intersection, we choose the one closest to $\hat{\mathbf{q}}$). Then the resulting intersection on the ray is obtained by simple averaging:

$$\mathbf{q} = \frac{\sum G_{r_i}(\|\mathbf{q}_i - \mathbf{c}_i\|)\,\mathbf{q}_i}{\sum G_{r_i}(\|\mathbf{q}_i - \mathbf{c}_i\|)}. \tag{6}$$

Differential properties of the individual implicit primitives $f_i(\mathbf{x}) = 0$ are averaged in the same way, and can be used to compute normals or curvatures. For example, the normal in $\mathbf{q}$ is set to

$$\mathbf{n} = \frac{\sum G_{r_i}(\|\mathbf{q}_i - \mathbf{c}_i\|)\,\nabla f_i(\mathbf{q}_i)}{\|\sum G_{r_i}(\|\mathbf{q}_i - \mathbf{c}_i\|)\,\nabla f_i(\mathbf{q}_i)\|}, \tag{7}$$

as demonstrated in the left image of Figure 9.

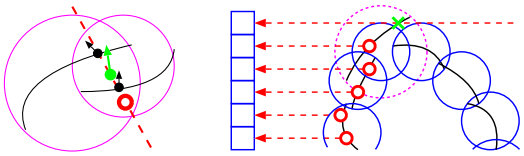

Figure 9: *Left: computing normals for SLIM-approximated surfaces. Right: illustrating our forward rendering procedure. The red circles are used to mark initial approximations* $\hat{\mathbf{q}}$*s. The green dot and green arrow denote intersection* $\mathbf{q}$ *defined by (6) and its normal* $\mathbf{n}$ *determined by (7), respectively.*

Forward rendering. Computing (6), (7) and determining $\hat{\mathbf{q}}$ can be efficiently performed using a forward rendering approach including a z-buffer. Our rendering method consists of two stages: First, all primitives are forward projected to determine a lower bound on the z value per pixel, and then the surfels supporting the front-most points are evaluated and averaged on each ray.

In the first stage, a point $\hat{\mathbf{q}}_{uv}$ for each pixel (u,v) is computed. The depth of $\hat{\mathbf{q}}_{uv}$ is initialized to the far clipping plane. Then all leaf surfels $s_i = (\mathbf{c}_i, \rho_i, f_i(\mathbf{x}))$ are projected to the z-buffer as follows:

Step 1: Project the α-shrunken surfel region $\|\mathbf{x}-\mathbf{c}_i\| < R_j$, $R_j = \alpha\rho_i$, to the screen space.
Step 2: For each pixel (u,v) in the projected region:

- **Step 2.1:** Find the first intersection $\tilde{\mathbf{q}}$ of $f_i(\mathbf{x}) = 0$ and the view-ray through (u,v). If there is no intersection, then continue .
- **Step 2.2:** If the depth of $\tilde{\mathbf{q}}$ is less than the depth of $\hat{\mathbf{q}}_{uv}$, set $\hat{\mathbf{q}}_{uv} = \tilde{\mathbf{q}}$.

Here and everywhere below we use the continue statement in the standard programming sense.

A kind of α-trimming in Step 1 is used to exclude regions that are only minimally supported by the surfels. Based on our numerical experiments, we recommend to set $\alpha = 2/3$.

The right image of Figure 9 illustrates computing $\hat{\mathbf{q}}_{uv}$.

During the second stage, we compute (q-position) and (q-normal) as follows. The variables $\Sigma\mathbf{q}_{uv}$, $\Sigma\mathbf{g}_{uv}$, and Σw_{uv} used below store sums required to evaluate (6) and (7) and are initialized to zero.

Step 1: Project the spherical support region $\|\mathbf{x}-\mathbf{c}_i\| < \rho_i$ to the screen space.
Step 2: For each pixel (u,v) in the projected support region:

- **Step 2.1:** If $\|\hat{\mathbf{q}}_{uv}-\mathbf{c}_i\| \geq \rho_i$, then continue .
- **Step 2.2:** Find closest to $\hat{\mathbf{q}}$ intersection $\mathbf{q}_{uv}$ between $f_i(\mathbf{x}) = 0$ and the corresponding view-ray through (u,v). If there is no intersection, then continue .
- **Step 2.3:** Compute the gradient $\mathbf{g}_{uv} = \nabla f_i(\mathbf{q}_{uv})$ and the spatial weight w at q_{uv}.
- **Step 2.4:** $\Sigma\mathbf{q}_{uv} += w\mathbf{q}_{uv}$, $\Sigma\mathbf{g}_{uv} += w\mathbf{g}_{uv}$, $\Sigma w_{uv} += w$.

After traversing all balls, we perform normalizations, $\mathbf{q}_{uv} = \Sigma w\mathbf{q}_{uv}/\Sigma w_{uv}$ and $\mathbf{n}_{uv} = \Sigma\mathbf{g}_{uv}/\|\Sigma\mathbf{g}_{uv}\|$, for each pixel (u,v) with $\Sigma w_{uv} \neq 0$. Finally, Phong shading is used.

View dependent refinement. So far we have simply projected the surfel regions to the screen. This might result in many (very small) surfels projected to one pixel. It is unlikely that this is necessary for a good visual result. So our tree-like structure described in Section 2 is used for implementing a view-dependent LOD refinement. We simply traverse the tree-like structure as long as

- the surfel region (ball) corresponding to the node intersects the view frustum,
- and the size of the surfel region projected on the screen space is larger than a few pixels (four pixels in our current implementation).

We pay no special attention to the contours.

LOD shading of SLIM-approximated 1mm David statue is illustrated in Figure 10.

Figure 10: *LOD shading of SLIM-approximated David statue. Left:* 730×650 *pixels, 1.7 sec. Middle:* 590×650 *pixels, 1.0 sec. Right:* 250×650 *pixels, 0.4 sec. The image heights are equal and about 650 pixels. Computations were performed on a 3.0 GHz Pentium 4 CPU.*

4. Results & Discussions

In Figures 2,11, 12, 13, and 17 we use reflection lines, suggestive contours and crest lines [DFRS03], curvature maps, and shadings for demonstrating advantages of the quadratic and cubic SLIM approximations and comparing them with each other.

Table 1 presents numbers of leaf surfels, threshold values, computational time measurements for quadratic and cubic SLIM approximations of various point datasets. Notice how compact the SLIM shape representation is.

The rendering results of our approach demonstrate that unconnected higher order primitives do represent a very good compromise between a useful modeling representation and the possibility of direct, fast, high quality visualization. In addition, differential information is readily available in every pixel, and can be used for NPR. Further, it turns out that a separate blending of surface derivatives often leads to

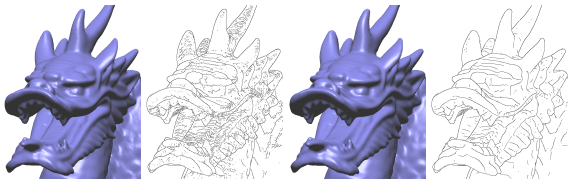

Figure 11: *Two left images: the original Phong-shaded mesh model of the Stanford dragon (871K triangles) and its suggestive contours computed in the image plane. Two right images: the model is SLIM-approximated using 44K quadratic surfels. The suggestive contours (also drawn in the image plane) are much cleaner.*

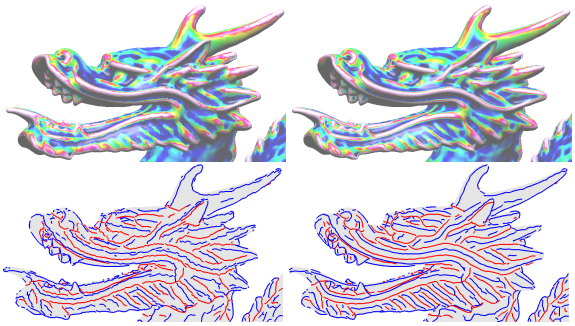

Figure 12: *Quadratic (left) vs. cubic (right) SLIM approximations. Top: coloring by $k_{\max}$, the maximal principal curvature. Bottom: blue ridges consisting of $k_{\max}$-maxima and red ravines. Using cubic SLIMs leads to a better curvature feature detection.*

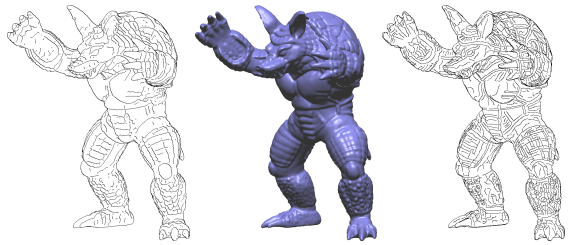

Figure 13: *High-quality suggestive contours (left) and crest lines (right) are easily detected on SLIM-approximated Armadillo (middle). These feature lines are computed in the image space as suggested by DeCarlo et al. [2003].*

a better estimation of differential surface attributes than a conventional approach consisting of surface reconstruction and then estimating differential characteristics of the reconstructed surface [OBS04a, GCO05]. It is especially true if a partition-of-unity reconstruction is used and advantages of

$$\frac{\sum w_i(x) D[f_i(x)]}{\sum w_i(x)} \quad \text{over} \quad D\left[\frac{\sum w_i(x) f_i(x)}{\sum w_i(x)}\right], \qquad (8)$$

where D is a linear differential operator, are obvious. Here $\{f_i(x)\}$ denote local surface approximations and $\{w_i(x)\}$ are blending functions. The sketch of Figure 14 and Figure 15 illustrate advantages of the left approximation in (8) over the right one (PU) for $D = \nabla$ in the blending-along-view-rays case. Figures 16 demonstrates that the same idea with D used to denote the matrix-valued operator the second-order derivatives can be applied for robust curvature estimation.

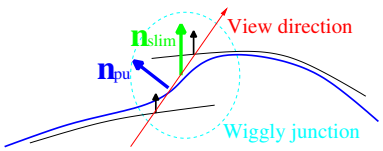

Figure 14: *Illustrating advantages of the left approximation in (8) over the right one in the blending-along-view-rays case, $D = \nabla$.*

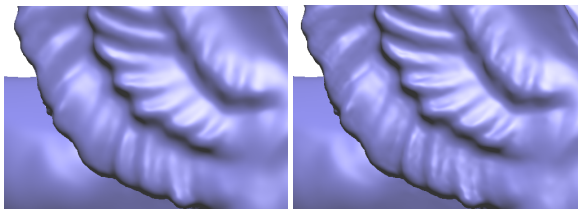

Figure 15: *Left: shading w.r.t. normals obtained via blending-along-view-rays (the left equation of (8)). Right: shading w.r.t. normals estimated from a partition-of-unity surface reconstruction (the right equation of (8)).*

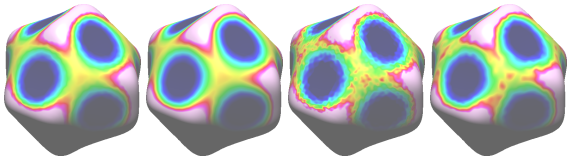

Figure 16: *Coloring by mean curvature. Blending-along-view-rays of 1st- and 2nd-order derivatives is used for quadratic (left) and cubic (middle-left) SLIM primitives. Necessary derivatives are estimated according to the right part of (8) with quadratic (middle-right) and cubic (right) surface patches.*

In Figures 12, 16, and 17 we compare approximation properties of SLIM surfaces composed of quadratic and cubic patches. Using cubic patches allows for slightly better shading results and gives a significant improvement in estimating surface curvatures and curvature derivatives. On the other hand, sometimes cubic surfels may fail to deliver an appropriate approximation at high-curvature regions due too poor estimation of surface normals, see a small defect in the upper part of the wing in the top-right image of Figures 17.

The tree-like structure described in the third part of Section 2 and containing the multi-scale surface representation can be created offline and out-of-core. Out-of-core processing is simple, because all calculation are local. The timing results of Table 1 show that even large data sets (like the

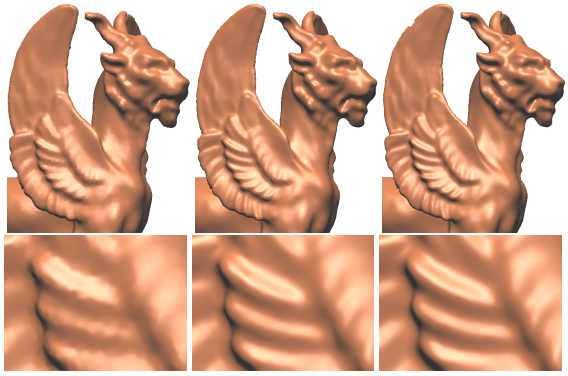

Figure 17: *The Caltech feline model (200K points) is SLIM-approximated with 31K linear surfels (left), 16K quadratic surfels (middle), and 12K cubic surfels (right) using the same thresholding: $T_{MDL} = 0.02$. Computational times for generating top-row images (approx. 450×600 pixels) are 0.45 sec. for linear elements, 0.72 sec. for quadratic ones, and 1.66 sec. for cubic surfels.*

1mm David statue model with 28 million points) can be processed in a matter of minutes, while smaller data sets are typically a matter of seconds. In general, running times are quasi-linear in the input and output, where the factor in the output is much larger due to the minimization for each ball. The resulting multi-scale representation is not only better suited for rendering, it is also much more compact.

Model	#points	#leafs	T_{MDL}	SLIM type	Timing
Dinosaur	56K	6K	0.02	quadratic	8 s
Dragon	438K	44K	0.01	quadratic	67 s
Dragon	438K	31K	0.01	cubic	86 s
Thai Statue	5M	345K	0.005	quadratic	15 m
David Statue	28M	933K	0.002	quadratic	64 m

Table 1: *Timing results for generating SLIM-approximations (timings for IO file operations are included) on a 3.0 GHz Pentium 4. It requires about 100 Mb per 1 million points. The 1 mm David statue model is processed in an out-of-core manner: the model was sliced in several parts along the longest axis of its bounding box. The slices are ρ_0-overlapped for computing local approximations correctly.*

The projection-based framework of our shading approach described in Section 3 is nicely adapted for implementing ray-tracing methods. The top image of Figure 18 demonstrates standard ray-traced rendering of a quadratic SLIM-based approximation of the Michelangelo's "Night" model (courtesy of the Digital Michelangelo Project) with two light sources. Since the original model has multiple gaps, as seen in the bottom images of Figure 18, the ray-traced image contains small white spots which, if necessary, can be easily eliminated by image processing tools.

The bottom images of Figure 18 present a visual comparison of the SLIM and QSplat [RL00] approximations. While SLIM delivers a significantly better surface approximation than QSplat, SLIM can create small bumpy defects because of its attempts to cover missed data gaps. These gaps are mostly untouched by QSplat because it uses splats of constant size.

Figure 18: *Top: standard ray-traced rendering of a quadratic SLIM-based approximation of the "Night" model with two light sources; original 11 M points are approximated by 432 K quadratic patches; it took 86 sec. for generating the image (1000×1000 pixels). Bottom-left: a zoomed fragment of the model approximated and rendered using quadratic SLIM patches. Bottom-right: the same fragment is rendered with our implementation of the Stanford QSplat Multiresolution Point Rendering System (we use our implementation instead of the original one proposed in [RL00] and available online in order to equalize shading effects for the bottom images).*

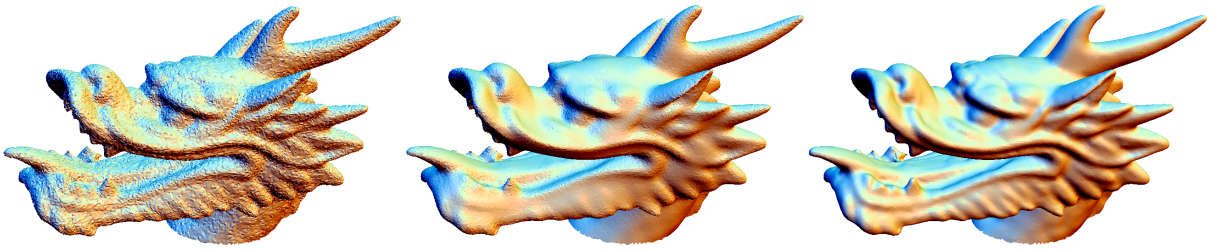

Figure 19: *Left: a noisy mesh used for testing in [JDD03]. Middle: the noisy mesh is smoothed by non-iterative, feature-preserving, bilateral filtering scheme [JDD03]. Right: the same data is smoothed by projecting the noisy points onto the corresponding SLIM surface.*

Our SLIM surface representation that we use for rendering purposes does not deliver an accurate surface approximation. Strictly speaking, our SLIM-based visualization procedure is not invariant w.r.t. rigid transformations. This is the price we pay for fast rendering and curvature feature extraction. On the other hand, we found out that the crest lines [DFRS03, OBS04a], very delicate surface features based on 1st and 2nd curvature derivatives, change only slightly when we rotate the surface.

Smoothing. There are several possibilities to use the SLIM surface representation for smoothing scattered point data (equipped with normals) and meshes. One simple idea consists of using the averaging procedure defined by (6). Given an oriented point $(\mathbf{p},\mathbf{n})$, we set $\hat{\mathbf{q}}=\mathbf{p}$, consider the ray $\mathbf{p}+t\mathbf{n}$, and use (6) to define the smoothed location $\mathbf{q}$ of the initial noisy point $\mathbf{p}$.

For a comparison we choose the non-iterative bilateral filtering mesh smoothing method of Jones et al. [JDD03] since it is simple and elegant, the authors accurately recorded their experiments, the source code and meshes used in [JDD03] are available online, and last but not least, the method was presented in a recent Siggraph paper. In Figure 19 we compare the method with our simple SLIM-based smoothing procedure by smoothing a model used in [JDD03].

A slightly more complex SLIM-based smoothing scheme can be used for processing noisy meshes. Given a triangle mesh, let us apply the projection operation described above to each triangle centroid, estimate the normals by (7) and then find positions of the new mesh vertices by minimizing the Quadric Error Metric (QEM) [GH97]. This mesh smoothing scheme is motivated by the feature sensitive surface extraction method [KBSS01] exploiting QEM. Figure 20 demonstrates performance of the scheme by smoothing a noisy mesh reconstructed from point data data acquired by an inexpensive computer-vision system [BP]. The result has a comparable quality to that obtained in [OBS02] where a computationally expensive but powerful mesh smoothing method was developed and the same noisy dataset was used for testing.

Sparsity analysis. We use the hand model mesh (courtesy of FarField Technology Ltd [Far]) to analyze sparsity properties of the SLIM approximations with linear, quadratic, and cubic primitives. To determine the approximation error we consider the projection procedure define in our first smoothing scheme described above. Then the L^2 approximation error is computed by normalized averaging of squared projections of the mesh vertices. Figure 21 delivers a visual comparison of the SLIM approximations of the same accuracy. Note that we need 7 floats to describe a linear SLIM primitive (the coordinates of the center, the support size, and the coefficients) 12 floats for a quadratic primitive, and 16 floats for a cubic one. Thus the sparsity of the quadratic and cubic SLIM approximations are nearly equal while the sparsity of the linear SLIM is 1.5 times smaller. As seen in Figure 22, even a significant increasing of the number of linear primitives is not sufficient to achieve a comparable rendering quality.

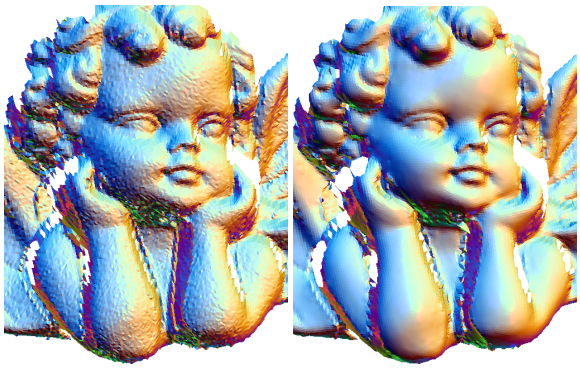

Figure 20: *Left: a noisy mesh reconstructed from data acquired by a computer-vision system [BP]. Right: the mesh is smoothed via projecting the mesh centroids onto a SLIM surface and generating new vertices by a QEM-based optimization procedure.*

However rendering with linear primitives as in [RL00] is always faster than with nonlinear ones. Indeed the rendering cost does not depend on the number of primitives. It is determined by the number of ray/implicit intersections that we have to compute and depends on the number of rays.

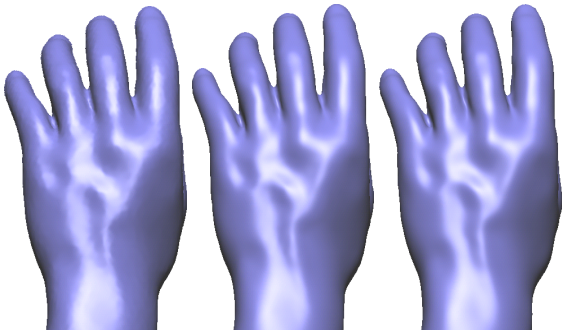

Figure 21: *The same* 0.01% L^2*-approximation accuracy is achieved with 5.4K linear primitives (left), 1.9K quadratic patches (middle), and 1.3K cubic patches (right).*

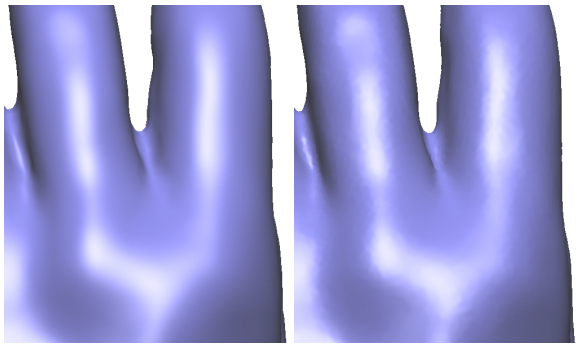

Figure 22: *Left: a zoomed part of the hand model approximated by 1.9K quadratic patches. Right: the SLIM approximation with 25K linear primitives is not enough to achieve a comparable rendering quality.*

5. Conclusion and Future Work

We have presented a new point-based surface representation and demonstrated its usefulness for various rendering and geometric modeling tasks.

The SLIM surface rendering toolkit accompanying the paper allows the reader to verify a high efficiency of our rendering approach. The source code, binaries, and some models are available from *http://www.riken.go.jp/lab-www/V-CAD/VCAD-Team/members/ohtake/slim/*

A general open problem is to construct a good approximation of a given surface with a small number of higher order elements. The large number of unknowns (position, support region, parameters of the polynomial) makes an optimal solution intractable. Particular avenues are very important though: Adding elements for representing sharp features or ellipsoidal supports are likely to enhance the system.

As the reader probably has noted, the SLIM and MLS surface representations have many similarities. Thus SLIM can serve as a bridge between various splatting techniques [RL00, SJ00, KV01, BSK04], partition-of-unity approximations [OBA*03, OBS04b], and MLS surfaces [Lev04, ABCO*01, AK04]. Equipping SLIM surfaces with necessary mathematical rigor and clarifying relations between various surface representations constitute an interesting direction for a future research.

Our current implementation performs all operations on the CPU. We would expect a considerable speed-up by performing some parts of the necessary steps on programmable hardware, however, have not exploited this option so far.

Acknowledgments

The models are courtesy of the Digital Michelangelo Project and Stanford 3D Scanning Repository (Stanford Computer Graphics Lab), Cyberware, Caltech Multi-Res Modeling Group, Caltech Vision Group, and FarField Technology Ltd. We are grateful to the authors of [JDD03] for making the source code of their mesh smoothing method available online. We would like to thank the anonymous reviewers of this paper for their valuable and constructive comments. The research of the second and third authors was supported in part by the AIM@SHAPE project (EU IST NoE 506766).

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *IEEE Visualization 2001* (2001), pp. 21–28. 1, 9

[AK04] AMENTA N., KIL Y.: Defining point-set surfaces. *ACM Transactions on Graphics 23*, 3 (2004), 264–270. Proceedings of ACM SIGGRAPH 2004. 1, 9

[BP] BOUGUET J.-Y., PERONA P.: 3D photography on your desk. www.vision.caltech.edu/bouguetj/ICCV98/gallery.html. 8

[BS02] BRENNER S. C., SCOTT L. R.: *The Mathematical Theory of Finite Element Methods*, vol. 15 of *Texts in Applied Mathematics*. Springer-Verlag, New York, 2002. Second Edition. 2

[BSK04] BOTSCH M., SPERNAT M., KOBBELT L.: Phong splatting. In *Symposium on Point-Based Graphics 2004* (2004), pp. 25–32. 1, 2, 9

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 67–76. 1

[CDS01] CHEN S. S., DONOHO D. L., SAUNDERS M. A.: Atomic decomposition by basis pursuit. *SIAM Review 43*, 1 (2001), 129–159. 1

[CP03] CAZALS F., POUGET M.: Estimating differential quantities using polynomial fitting of osculating jets. In *Symposium on Geometry Processing* (Aachen, Germany, 2003), pp. 177–187. 3

[DFR04] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S.: Interactive rendering of suggestive contours with temporal coherence. In *NPAR 2004* (2004), pp. 15–24. 2

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3 (2003), 848–855. Proceedings of ACM SIGGRAPH 2003. 2, 5, 8

[Far] FARFIELD TECHNOLOGY:. www.farfieldtechnology.com. 8

[GCO05] GAL R., COHEN-OR D.: Salient geometric features for partial shape matching and similarity. *ACM Transactions on Graphics 24* (2005). Under revision. 6

[GG01] GOOCH B., GOOCH A.: *Non-Photorealistic Rendering*. AK Peters Ltd, Publishers, Natick, Massachusetts, 2001. 2

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of ACM SIGGRAPH '97* (1997), pp. 209–216. 8

[GI04] GOLDFEATHER J., INTERRANTE V.: A novel cubic-order algorithm for approximating principal direction vectors. *ACM Transactions on Graphics 23*, 1 (2004), 45–63. 3

[JDD03] JONES T. R., DURAND F., DESBRUN M.: Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics 22*, 3 (2003), 943–949. Proceedings of ACM SIGGRAPH 2003. 8, 9

[KBSS01] KOBBELT L., BOTSCH M., SCHWANECKE U., SEIDEL H.-P.: Feature sensitive surface extraction from volume data. In *Proceedings of ACM SIGGRAPH '01* (2001), pp. 57–66. 8

[KV01] KALAIAH A., VARSHNEY A.: Differential point rendering. In *Eurographics Workshop on Rendering Techniques '01* (2001), pp. 139–150. 1, 2, 9

[Lev04] LEVIN D.: Mesh-independent surface interpolation. In *Geometric Modeling for Scientific Visualization*, Brunnett G., Hamann B., Müller H.,, Linsen L., (Eds.), Mathematics and Visualization. Springer, 2004, pp. 37–49. 9

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The Digital Michelangelo Project: 3D scanning of large statues. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 131–144. 2

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.-P.: Multi-level partition of unity implicits. *ACM Transactions on Graphics 22*, 3 (2003), 463–470. Proceedings of ACM SIGGRAPH 2003. 1, 9

[OBS02] OHTAKE Y., BELYAEV A. G., SEIDEL H.-P.: Mesh smoothing by adaptive and anisotropic gaussian filter. In *Vision, Modeling, and Visualization 2002* (2002), pp. 203–210. 8

[OBS04a] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: Ridge-valley lines on meshes via implicit surface fitting. *ACM Transactions on Graphics 23*, 3 (2004), 609–612. Proceedings of ACM SIGGRAPH 2004. 6, 8

[OBS04b] OHTAKE Y., BELYAEV A. G., SEIDEL H.-P.: 3D scattered data approximation with adaptive compactly supported radial basis functions. In *Shape Modeling International 2004* (2004), pp. 31–39. 1, 2, 3, 9

[OY93] OLANO T. M., YOO T. S.: *Precision Normals (Beyond Phong)*. Tech. Rep. 93-021, Department of Computer Science, University of North Carolina, 1993. 2

[PSG04] PAJAROLA R., SAINZ M., GUIDOTTI P.: Confetti: Object-space point blending and splatting. *IEEE Transactions on Visualization and Computer Graphics 10*, 5 (2004), 598–608. 2

[PTVF93] PRESS W. H., TEUKOLSKY S. A., VETTERLING W. T., FLANNERY B. P.: *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1993. 3

[PZ*00] PFISTER H., ZWICKER M., , BAAR J. V., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 335–342. 1, 2

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: a multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 343–352. 1, 7, 8, 9

[SJ00] SCHAUFLER G., JENSEN H. W.: Ray tracing point sampled geometry. In *Eurographics Workshop on Rendering Techniques 2000* (2000), pp. 319–328. 2, 9

[SOS04] SHEN C., O'BRIEN J. F., SHEWCHUK J. R.: Interpolating and approximating implicit surfaces from polygon soup. *ACM Transactions on Graphics 23*, 3 (2004), 896–904. Proceedings of ACM SIGGRAPH 2004. 1

[SPOK95] SAVCHENKO V. V., PASKO A. A., OKUNEV O. G., KUNII T. L.: Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum 14*, 4 (1995), 181–188. 1

[TO99] TURK G., O'BRIEN J.: Shape transformation using variational implicit functions. In *Proceedings of ACM SIGGRAPH '99* (1999), pp. 335–342. 1

[WK04] WU J., KOBBELT L.: Optimized sub-sampling of point sets for surface splatting. *Computer Graphics Forum 23*, 3 (2004), 643–652. Eurographics 2004 issue. 2

[ZPBG01] ZWICKER M., PFISTER H., BAAR J. V., GROSS M. H.: Surface splatting. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 371–378. 4