A Survey of Point-Based Techniques in Computer Graphics

Leif Kobbelt Mario Botsch

Computer Graphics Group
RWTH Aachen University
{*kobbelt, botsch*}*@cs.rwth-aachen.de*

Abstract

In recent years point-based geometry has gained increasing attention as an alternative surface representation, both for efficient rendering and for flexible geometry processing of highly complex 3D-models. Point sampled objects do neither have to store nor to maintain globally consistent topological information. Therefore they are more flexible compared to triangle meshes when it comes to handling highly complex or dynamically changing shapes. In this paper, we make an attempt to give an overview of the various point-based methods that have been proposed over the last years. In particular we review and evaluate different shape representations, geometric algorithms, and rendering methods which use points as a universal graphics primitive.

Key words: Point-sampled geometry, point-based graphics, GPU processing

1 Introduction

Triangle meshes are still the most common surface representation in many computer graphics applications. Because of their simplicity and flexibility, they replace traditional CAD surface representations, like NURBS surfaces, in many areas where processing performance matters.

The reason for this is that triangle meshes are significantly more flexible, since surfaces of any shape and topology can be represented by a single mesh without the need to satisfy complicated inter-patch smoothness conditions. The simplicity of the triangle primitive allows for easier and more efficient geometry generation and geometry processing algorithms. This is most evident for interactive graphics, where the highly optimized (and specialized) graphics hardware is able to process several millions of triangles per second. Obviously,

since the triangle primitive is mathematically much simpler compared to a NURBS patch, more of them have to be used to obtain the same approximation quality. However, if a smooth surface is to be represented by a triangle mesh (a piecewise linear surface), the approximation order is quadratic, i.e., halving the edge lengths reduces the error by a factor of 4 which means the number of triangles is inversely proportional to the approximation error. Hence, even with the weaker asymptotic behavior, a good approximation (for the typical precision requirements in graphics applications) can be achieved with a moderately fine mesh whose vertex density and distribution adapt to the surface curvature, i.e., to the shape complexity.

Although being much more flexible than NURBS, triangle meshes can also have restrictions and disadvantages in some special applications. Most algorithms working on triangle meshes require topologically consistent two-manifold surfaces. As a consequence, manifold-extraction or topology cleanup steps are necessary for mesh generation methods (Amenta et al., 1998; Bischoff et al., 2004). Maintaining the topological consistency throughout the mesh processing pipeline makes these algorithms sometimes significantly more complicated, even if the eventual target application is, e.g., visualization where a consistent topology is actually not necessary. Dynamic mesh connectivity (Welch and Witkin, 1994; Kobbelt et al., 2000) is one example where frequent topology changes occur because the mesh is locally restructured in order to avoid too much stretching after extreme deformations or to provide more degrees of freedom in certain regions of interest. In cases like this, point-based representations would allow for more flexibility when a globally consistent surface topology is not necessarily required (e.g., for rendering). Hence, the consequent next step towards simpler and more flexible geometric primitives seems to be the use of unstructured clouds of points or disks where we do not need to store and maintain globally consistent connectivity information.

From a rendering point of view, triangle meshes may not be suitable as meshes are becoming more and more complex while the typical screen resolution does not grow as fast. The steadily increasing performance of CPUs and graphics hardware, the cheap memory, and the wide availability of range-scanning devices result in the acquisition and generation of massive highly detailed geometry data — models consisting of several millions of triangles are common nowadays (Levoy et al., 2000). When the number of triangles exceeds the number of pixels on the screen, rendering such massive datasets leads to triangles whose projected area is less than one pixel. In this situation, the traditional incremental rasterization methods become inefficient because of the expensive triangle setup. Hence, points seem to qualify as a better suited rendering primitive for such highly complex models.

In this survey paper we will first discuss several point-based geometry representations (Sec. 2) and show special application scenarios where geometry

processing algorithms can greatly benefit from the flexibility offered by point-sampled geometries (Sec. 3). In Sec. 4 we will finally review various point-based rendering methods. More details on point-based geometry processing, point-based rendering, or point-based computer graphics in general can be found in (Pauly, 2003; Zwicker, 2003; Alexa et al., 2003b, 2004).

2 Point-based Representations

A point-based geometry representation can be considered a sampling of a continuous surface, resulting in 3D positions $\mathbf{p}_i$, optionally with associated normal vectors $\mathbf{n}_i$ or auxiliary surface properties like, e.g., colors or other material properties.

2.1 Neighborhoods & Normals

If normal vectors are not given, they can be estimated by analyzing the local neighborhood of each sample point. Because there is no connectivity information available, these local neighborhoods are usually constructed using either Euclidean neighborhoods or k nearest neighbors. In the first case, all samples within an ε-ball around a query point are defined to be its neighbors. This naïve method is not suited for irregularly sampled models, since either too many or too few neighbors may be found within the ε-ball. The neighborhood estimate will also get unreliable as soon as the local feature size (Amenta et al., 1998) becomes smaller than ε, e.g., if two separate surface sheets are located spatially close to each other.

In contrast, the k nearest neighbors provide a naturally adaptive neighborhood relation in such situations. If the points satisfy certain sampling criteria, like adaptation to the local feature size, then the neighborhood estimate is guaranteed to be reliable (Amenta et al., 1998; Andersson et al., 2004). Moreover, the k nearest neighborhood structure has linear complexity in the number of sample points which is similar to polygon meshes.

Both, Euclidean neighborhood and k nearest neighborhood can be computed efficiently by using some hierarchical space partitioning technique. In contrast to triangle meshes, where (consistent) neighborhood information is represented explicitly, the local neighborhoods are usually dynamically re-computed in the case of point-sampled geometries. However, in certain applications a caching of local neighborhood information may also be suitable (Linsen and Prautzsch, 2002).

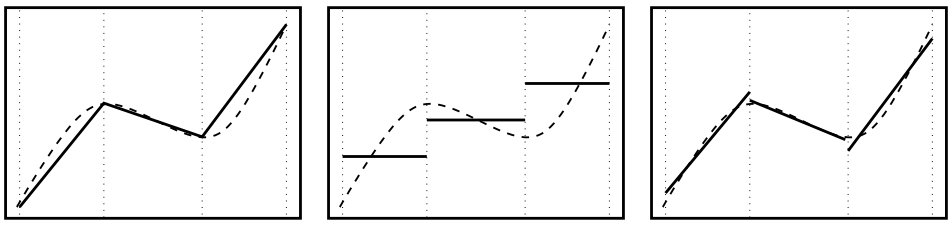

Fig. 1. Comparison of the different shape approximations: piecewise linear C^0 polygons (*left*), piecewise constant C^{-1} points (*center*), and piecewise linear C^{-1} splats (*right*). Splats provide the same approximation power as triangle meshes, but due to the C^{-1} continuity they offer the same flexibility as point clouds.

Let $\mathbf{p}_0$ be a sample point and $\{\mathbf{p}_1, \ldots, \mathbf{p}_k\}$ its k nearest neighbors. The covariance matrix

$$C := \sum_{i=0}^{k} (\mathbf{p}_i - \bar{\mathbf{p}}) (\mathbf{p}_i - \bar{\mathbf{p}})^T \in \mathbb{R}^{3\times 3},$$

with $\bar{\mathbf{p}} := \sum_{i=0}^{k} \mathbf{p}_i/(k+1)$, is symmetric and positive semi-definite. The eigenvector corresponding to the smallest eigenvalue gives an estimate for the normal direction. Since this determines the normal vector up to its sign only, a consistent orientation over all sample points has to be constructed by a propagation along a minimum spanning tree (Hoppe et al., 1992).

2.2 Purely Point-Based Representations

From an approximation point of view, a point cloud is a piecewise constant surface approximant. Hence, the resulting approximation power is linear, i.e., with an average spacing h between the samples $\mathbf{p}_i$, the approximation error with respect to each coordinate function is of the order $O(h)$ (Davis, 1975) (cf. Fig. 1). For the approximation of a geometric shape this means that the approximation error is determined by the *spacing* between the samples. As a consequence, the sampling has to be adjusted to the surface area, implying a dense sampling even in flat surface regions. In fact, the number of samples is inversely proportional to the *squared* approximation error.

A point-based representation that is mainly targeting at rendering is proposed by Grossman and Dally (1998). Instead of a completely unorganized point cloud, they use a set of depth images that are orthogonally sampled from a given input geometry. Similar to image-based approaches, this representation is also constructed from several views of an input object, but it differs in that each pixel is a surface sample containing geometric position and (view-independent) surface color.

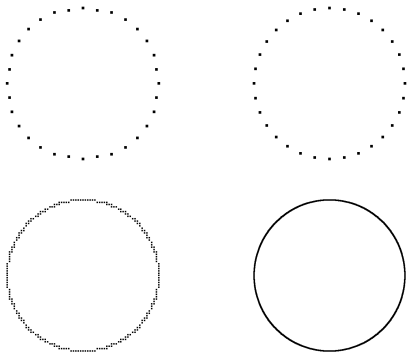

Fig. 2. Point-sampling of a circle with different quantization levels (*left*: 5 bit, *right*: 10 bit) and different sampling densities (*top*: $2\,\pi/32$, *bottom*: $2\,\pi/1024$). In the top row the approximation error between the *continuous* circle and the *discrete* point sets is dominated by the distance between samples while in the bottom row the error is dominated by the quantization. Top left and bottom right are good samples since quantization error and sampling density are of the same order thus minimizing redundancy.

If the input geometry contains a certain amount of (tolerable) noise, an exact sampling may not be necessary. Instead, Kalaiah and Varshney (2003b) proposed a non-deterministic statistical point-based representation, where a hierarchical PCA analysis partitions the geometry and its attributes (normals and colors) into a set of local Gaussian probability distributions. Reconstruction and rendering can then be performed by random sampling these probability distribution functions.

For a compact representation, the quantization precision to be used for storing the sample coordinates also has to be taken into account. Since the average sample spacing and the approximation error are of the same order, the quantization error should also be about the same magnitude in order to minimize redundancy (cf. Fig. 2) (Botsch et al., 2002). If we sample with a resolution of $n \approx h^{-1}$, we need $O(n^2)$ samples to cover a given surface. Using a quantization precision of $O(n)$ then leads to a total memory consumption of $O(n^2 \log(n))$ to store the coordinates of all sample points.

This memory cost can be reduced significantly by a hierarchical point-based representation. We start by using a regular 3D binary voxel grid to represent a point-based model, such that for each non-empty grid cell its center position is used as sample point. This sampling respects the proportionality between sample spacing and quantization precision, but it has complexity $O(n^3)$, since empty cells have to be stored as well. Recursively sub-sampling the regular grid and applying zero-tree pruning results in an adaptive octree structure. This hierarchical organization of the full cells yields a very compact point-based LoD-representation that needs about 8/3 bit per sample point. By entropy encoding, this amount is further reduced down to about 1–1.5 bit per point

(Botsch et al., 2002). Since the bit rate is independent of the sampling density h, the total memory costs in fact reduce to $O(n^2)$.

However, the insufficient object-space approximation power of purely point-based representations usually requires a very dense sampling. Therefore, *splats* seem to be a better compromise between the simplicity of the geometric primitive and the number of primitives that have to be used, as shown in the next section.

2.3 Surface Splats

Surface splats have first been proposed for rendering purposes by Zwicker et al. (2001). In order to bridge the gaps between neighboring point samples, points $\mathbf{p}_i$ are associated with a normal vector $\mathbf{n}_i$ and a radius r_i, turning them into object-space circular disks.

A locally optimal adaptation to the curvature of the underlying surface is provided by *elliptical* splats, that are defined by two tangential axes $\mathbf{u}_i$ and $\mathbf{v}_i$ and their respective radii. Optimal local approximation is achieved if the two axes are aligned to the principal curvature directions of the underlying surface and the radii are inversely proportional to the corresponding minimum and maximum curvatures. The tangent vectors can be scaled according to the corresponding ellipse radii such that an arbitrary point $\mathbf{q}$ in the plane spanned by $\{\mathbf{p}_i, \mathbf{u}_i, \mathbf{v}_i\}$ lies in the interior of the splat if it satisfies the condition

$$\left(\mathbf{u}_i^T\,(\mathbf{q}-\mathbf{p}_i)\right)^2 \;+\; \left(\mathbf{v}_i^T\,(\mathbf{q}-\mathbf{p}_i)\right)^2 \;\leq\; 1.$$

Differential geometry tells us that locally a proper ellipse is the best linear approximant to a smooth surface. In this sense splat-based representations are superior to triangle meshes. Since splats are piecewise linear surface primitives, they provide the same quadratic approximation order as triangle meshes (cf. Fig. 1). Just like for triangle meshes, the sampling density can therefore adapt to the surface curvature, such that highly detailed regions are sampled with a higher density, while flat surface regions are sampled more sparsely. Because of the higher approximation power, the quantization precision for splat coordinates has to be $O(n^2)$, n again denoting the sampling density. Since splats do not have to join continuously (like triangles in a mesh), but are C^{-1} continuous instead, they still provide the same topological flexibility as pure point clouds (cf. Figs. 5, 13).

Representing sharp features, like edges or corners in technical datasets, is a well studied problem for triangle meshes. Because the surface is no longer differentiable, the approximation power breaks down to linear order. Addi-

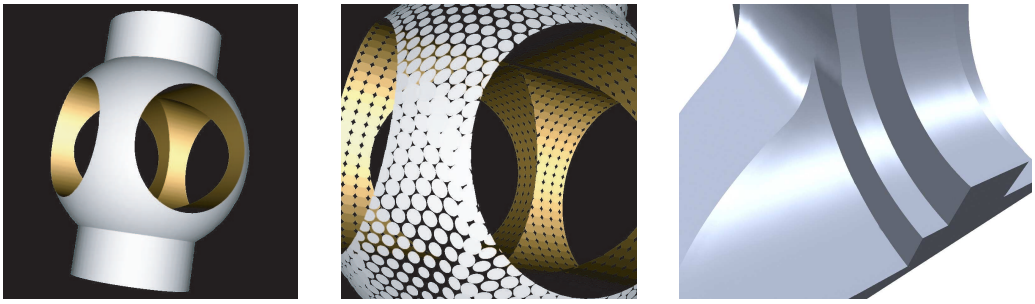

Fig. 3. In order to represent sharp features in technical datasets, each splat can be clipped against one or two clipping lines. The left example was generated using CSG operations, the right example shows a splat-based approximation of the well-known fandisk model.

tionally, alias artifacts are introduced by insufficient sampling, that cannot be removed by increasing the sampling density (Kobbelt et al., 2001). In order to remove these artifacts and reduce normal noise, the sampling has to be aligned to the principal curvature directions (Botsch and Kobbelt, 2001). If surface splats are to represent sharp features, all splats that sample these features have to be clipped against one (*edges*) or two (*corners*) clipping lines that are specified in their local tangent frames (cf. Fig. 3) (Pauly et al., 2003b).

2.4 Moving Least-Squares Surfaces

The *moving least-squares* (MLS) surfaces of Levin (1998, 2003) provide an approximating or interpolating surface for a given set of point samples by local higher order polynomials and have first been applied to point-based methods by Alexa et al. (2001, 2003a). An MLS surface S is defined in terms of the MLS projection operator $\Psi : \mathcal{B} \to \mathbb{R}^3$, that projects points from a vicinity $\mathcal{B}$ of the MLS surface onto the surface itself. Hence, the surface $\mathcal{S}$ is defined by all fix points of this operator that project onto themselves:

$$\mathcal{S} := \{\mathbf{x} \in \mathcal{B} \ : \ \Psi(\mathbf{x}) = \mathbf{x}\} = \text{range}(\Psi).$$

The domain $\mathcal{B}$ of this projection operator is the neighborhood of the input points $\mathbf{p}_i$ and can be defined, e.g., by a union of balls centered at the $\mathbf{p}_i$:

$$\mathcal{B} = \bigcup_i \left\{\mathbf{x} \in \mathbb{R}^3 \mid \|\mathbf{x} - \mathbf{p}_i\| < r_{\mathcal{B}}\right\}.$$

The actual computation of the projection $\mathbf{r} \mapsto \Psi(\mathbf{r})$ is split into three steps:

(1) Find a local *reference plane* $H_{\mathbf{r}} = \left\{\mathbf{x} \in \mathbb{R}^3 \ : \ \mathbf{n}^T\mathbf{x} = \mathbf{n}^T\mathbf{q}\right\}$ by minimizing

the non-linear energy functional

$$E_{MLS}(\mathbf{q}, \mathbf{n}) := \sum_i \left(\mathbf{n}^T \mathbf{p}_i - \mathbf{n}^T \mathbf{q}\right)^2 \theta\left(\|\mathbf{p}_i - \mathbf{q}\|\right)$$

for any $\mathbf{n}, \mathbf{q} \in \mathbb{R}^3$ with $\mathbf{n} = \mathbf{n}(\mathbf{q}) = \frac{\mathbf{r}-\mathbf{q}}{\|\mathbf{r}-\mathbf{q}\|}$, where $\theta : \mathbb{R} \to \mathbb{R}$ denotes a smooth, positive, and monotonically decreasing weight function.

(2) Find a local bivariate *polynomial approximation* $g : H_{\mathbf{r}} \to \mathbb{R}^3$ by a weighted least-squares fit to the points $\mathbf{p}_i$ in the neighborhood of $\mathbf{q}$. Let $\mathbf{q}_i$ be the orthogonal projection of $\mathbf{p}_i$ onto $H_{\mathbf{r}}$, (x_i, y_i) its local 2D coordinates, and $f_i := \|\mathbf{q}_i - \mathbf{p}_i\|$ its height over $H_{\mathbf{r}}$. Then the error

$$\sum_i \left(g(x_i, y_i) - f_i\right)^2 \theta\left(\|\mathbf{p}_i - \mathbf{q}\|\right)$$

is to be minimized.

(3) The projection of $\mathbf{r}$ is finally defined by $\Psi(\mathbf{r}) := \mathbf{q} + g(0,0)\mathbf{n}$.

The typical choice for the decreasing weight function θ is a Gaussian $\theta(d) = e^{-d^2/h^2}$, resulting in C^∞ continuous MLS surfaces. The parameter h of the Gaussian corresponds to the globally estimated sample spacing h and can be used to control the degree of smoothing. For irregularly sampled point sets this parameter could be chosen locally instead, as proposed by (Pauly et al., 2002). The approximation power of MLS surfaces is conjectured to be $O(m+1)$, with m being the degree of the local polynomial approximant g.

The drawback of MLS surfaces is the computationally involved non-linear optimization problem for finding the reference frame $H_{\mathbf{r}}$. A slightly different but considerably simpler projection approach is proposed by Alexa and Adamson (2004), where they also take a correct normal computation into account in order to properly define implicit surfaces from point cloud data (Adamson and Alexa, 2003, 2004). Their projection procedure repeatedly projects a given point $\mathbf{x}$ onto local reference planes that are defined by a weighted average position

$$\mathbf{a}(\mathbf{x}) := \frac{\sum_i \theta\left(\|\mathbf{x} - \mathbf{p}_i\|\right) \mathbf{p}_i}{\sum_i \theta\left(\|\mathbf{x} - \mathbf{p}_i\|\right)}$$

and a normal computed by either a weighted least-squares fit

$$\mathbf{n}(\mathbf{x}) := \text{argmin}_{\|n\|=1} \sum_i \left\|\mathbf{n}^T \left(\mathbf{x} - \mathbf{p}_i\right)\right\|^2 \theta\left(\|\mathbf{x} - \mathbf{p}_i\|\right)$$

or by a weighted averaging of input normals $\mathbf{n}_i$

$$\mathbf{n}(\mathbf{x}) := \frac{\sum_i \theta\left(\|\mathbf{x} - \mathbf{p}_i\|\right) \mathbf{n}_i}{\left\|\sum_i \theta\left(\|\mathbf{x} - \mathbf{p}_i\|\right) \mathbf{n}_i\right\|}.$$

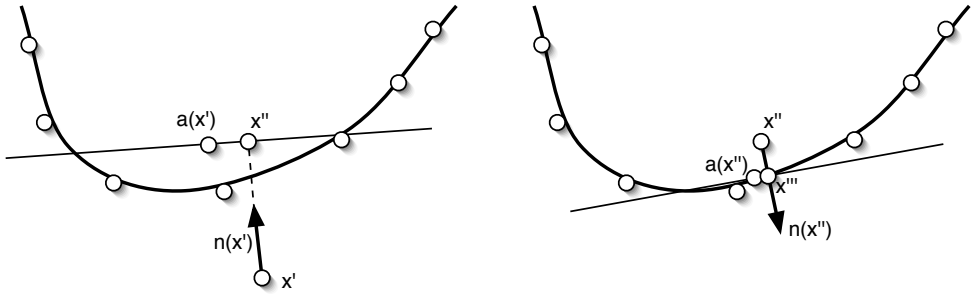

Fig. 4. Simplified MLS projection procedure. In each step the current approximation $\mathbf{x}'$ is updated by projecting it orthogonally onto the reference plane given by $\mathbf{a}(\mathbf{x}')$ and $\mathbf{n}(\mathbf{x}')$.

The iterative projection procedure is then proposed as follows (cf. Fig. 4):

(1) Initialize $\mathbf{x}' \leftarrow \mathbf{a}(\mathbf{x})$.
(2) Compute $\mathbf{n} \leftarrow \mathbf{n}(\mathbf{x}')$ and $\mathbf{a} \leftarrow \mathbf{a}(\mathbf{x}')$.
(3) If $\left\|\mathbf{n}^T\left(\mathbf{a}-\mathbf{x}'\right)\right\| < \varepsilon$ return $\mathbf{x}'$.
(4) Else project $\mathbf{x}' \leftarrow \mathbf{x}' + \mathbf{n}\mathbf{n}^T\left(\mathbf{a}-\mathbf{x}'\right)$ and go back to (2).

If this iteration converges, it yields a point $\mathbf{x}$ on the MLS surface. However, this simplified projection procedure does not result in an orthogonal projection, but it can be enhanced to do so (Alexa and Adamson, 2004).

In contrast to the algorithmic construction of MLS surfaces using the projection operator, Amenta and Kil (2004) give an explicit definition of MLS surfaces in terms of critical points of the energy function E_{MLS} along lines determined by a vector field. In this work they also discuss stability issues of the traditional projection operator for points not being sufficiently close to the surface and propose an alternative projection technique.

MLS surfaces can be used to define a smooth surface from a set of points, but they are also a versatile tool to generate additional sample points on a point-sampled surface, e.g., for up- or down-sampling a model, for low-pass filtering it, or for mapping points back onto the initial surface after local restructuring.

MLS surfaces are a valuable and efficient tool for point-based geometry processing. However, rendering them, e.g. by sufficient up-sampling, is quite involved and does not map to graphics hardware (see Sec. 4). Hence, in most interactive applications, usually a combination of surface splats and MLS surfaces is used.

3 Point-based Geometry Processing

Several concepts and algorithms known from triangle meshes have been transfered to point-based geometry representations by simply replacing the manifold neighborhood relation implied by the triangle connectivity with the more general k nearest neighborhood relation for point clouds. However, several recently developed techniques are especially designed for point-sampled geometries and actually exploit the additional flexibility offered by them. Since no connectivity information or topological consistency has to be maintained, local re-sampling or restructuring is much easier than it would be using triangle meshes.

Point-based models are usually acquired by range scanning or image-based reconstruction methods. In both cases, the point samples contain a certain amount of noise due to physical measurement errors, that is to be reduced by low-pass filtering techniques. Using a decomposition into patches and local height-field approximations, Pauly and Gross (2001) transfered Fourier-based spectral methods to point-sampled geometries. Their method allows for mathematically well-defined frequency analysis and filtering. While the frequencies could also be used to find feature regions of the geometry, a more robust feature detection method was presented by Pauly et al. (2003a). Local covariance analyses on multiple scales robustly find feature regions that can be connected to feature lines based on a minimum spanning tree approach.

Scanned data usually contains more artifacts than just measurement noise. Occlusions in concave areas or specular surface materials may lead to holes or insufficient sampling, requiring some kind of hole-filling mechanism. Difficult material reflectances or textured models can also lead to outliers that have to be detected and removed before further processing. These two and several other intuitive tools for post-processing scanned data have recently been been presented by Weyrich et al. (2004).

Because real-world objects are captured with as much detail as possible, the resulting models are usually highly complex and have to be simplified in order to be of suitable complexity for further processing. Pauly et al. (2002) transfered several mesh decimation methods — vertex clustering, incremental decimation using error quadrics, and remeshing by particle simulation — to point-sampled surfaces. These methods show the same behavior as their mesh counterparts, i.e., greedy incremental decimation gives good results, but the point distribution can greatly be equalized by several point repulsion iterations. However, both methods do not allow to prescribe an exact global error for the decimation, and they are not taking the full splat geometry into account, but rather rely on the proper distribution of splat *centers*.

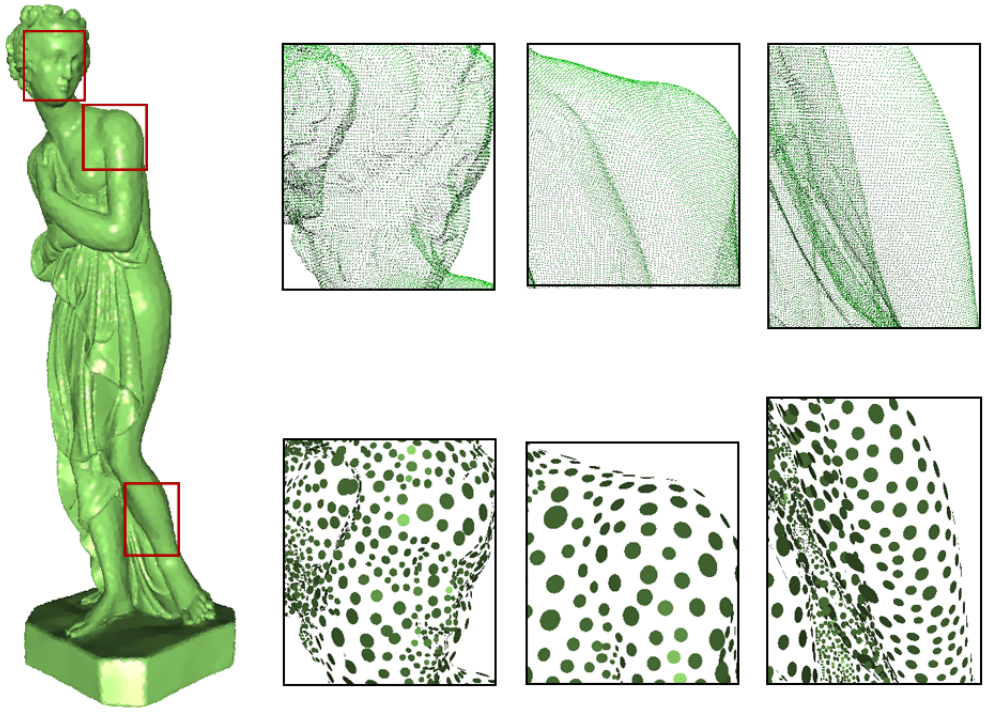

Fig. 5. Optimized sub-sampling of the Iphigenie (*left*, 350k points) using 30k circular splats (*right*). While respecting a prescribed error tolerance, a global optimization yields a very regular splat distribution.

In contrast, Wu and Kobbelt (2004) propose a simplification method that is especially designed for splat-based surface approximations. Taking the full piecewise linear geometry of surface splats into account, their method provides an exact global error bound as well as a high-quality splat distribution (cf. Fig. 5). In a first step, a patch, respectively a splat, is grown from each input sample, such that it does not violate the error tolerance. From this set of candidate splats, a subset that fully covers the input geometry is chosen in a greedy manner. Finally, a global optimization of the patch selection improves the splat distribution, similar to a repulsion-based particle simulation. Because all candidate splats have been constructed to stay within the error tolerance, the coverage tests can be done by simple set operations. This technique exploits both the improved local approximations by elliptical splats as well as the added flexibility that comes from not having to maintain a globally consistent topology, in order to obtain a considerably better approximation compared to triangle meshes (cf. Fig. 12, left, and Fig. 13).

The biggest advantage of point-based representations over polygonal meshes is that they can easily be restructured without the need to take care of manifold conditions. Hence, applications requiring frequent geometry re-sampling will benefit most from point-based methods. One prominent example is PointShop3D of Zwicker et al. (2002), that is a PhotoShop-like tool for manipulating point-sampled models (cf. Fig. 6). Since the point samples can represent surface

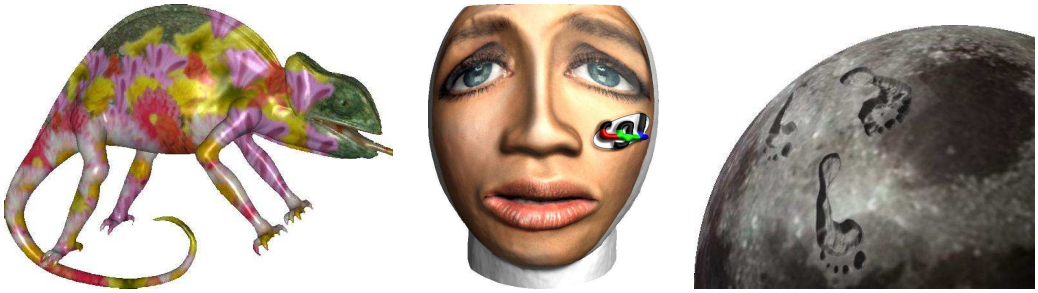

Fig. 6. PointShop3D offers various manipulations of point-sampled models, like texture painting (*left*), low-distortion parameterization and displacement mapping (*center*), and carving (*right*).

positions as well as material properties or textures, these values can easily be modified on a per-point basis. Tools like sculpting and carving change the point positions, texture smoothing or painting modifies the point colors. In order to be able to paint arbitrary fine details onto the model or to fill up gaps after displacing points, local re-sampling is heavily used. Another very recent approach for painting point-sampled surfaces has been proposed by Adams et al. (2004). Targeting at realistic painting, their method provides haptic feedback and physically simulated paint transfer. Again, this method allows the user to paint arbitrarily fine detail on the surface by locally up-sampling the affected regions on the fly in order to provide a sufficient sample resolution.

The simultaneous point-sampling of geometry, normals, and appearance attributes greatly simplifies these painting methods. In contrast, for triangle meshes there is no such tight coupling between geometry and appearance, requiring the use of a texture atlas for painting the mesh. To enable the painting of fine details, this texture atlas has to be refined dynamically (Carr and Hart, 2004), what is considerably more complicated as the natural point-sampling approach.

Another challenging topic is shape deformation. Among the many different approaches to model freeform shapes, volumetric freeform deformation seems to be best suited for the interactive modification of point-sampled geometries. These deformation techniques first generate a displacement function $d : \mathbb{R}^3 \rightarrow \mathbb{R}^3$. Then every surface sample $\mathbf{p}_i$ (e.g. a mesh vertex or a splat center) is shifted by $\mathbf{p}_i \mapsto d(\mathbf{p}_i)$.

When applying an extreme deformation to a triangle mesh, certain triangles exhibit strong stretching, leading to numerically and visually undesirable triangles that have to be removed by local restructuring (Welch and Witkin, 1994; Kobbelt et al., 2000; Lawrence and Funkhouser, 2003). For point-based models, the problem is even worse, since the model can tear apart under too much stretch. However, the splats exhibiting too much stretching are easily detected by looking at the Jacobian of the displacement function d. Splitting

Fig. 7. Shape modeling with point-sampled models. Frequent local re-sampling accounts for stretching and allows for extreme deformations (*left*). A combination of free-form deformation, CSG operations, and displacement mapping was used to create the mug (*right*).

those splats into two, followed by a local tangential relaxation (using particle repulsion and MLS projections), solves the dynamic restructuring problem for point clouds and allows for extreme deformations of point-based models (cf. Fig. 7) (Pauly et al., 2003b). In a similar sense, oriented particles have been used to represent deformable (parametric or implicit) surface models that can easily be stretched, split, or joined, while a repulsion-based particle simulation achieves a uniform and sufficiently dense surface sampling throughout the modeling process (Szeliski and Tonnesen, 1992; Witkin and Heckbert, 1994).

Constructive Solid Geometry (CSG) is a very common technique for building complex models by boolean combinations of simpler ones. Since this requires inside/outside tests w.r.t. the given solids, an implicit representation of the models is best suited (Hoffmann, 1989). Such a volumetric surface representation can easily be derived from a point-sampled surface by observing the (direction of) the shift vector $\mathbf{p}_i - \Psi(\mathbf{p}_i)$ induced by the MLS projection operator relative to the oriented normal vector.

Since the intersection of two smooth objects may result in sharp feature curves, these have to be sampled properly in order to avoid aliasing artifacts. Pauly et al. (2003b) proposed a simple technique to snap nearby sample points to the sharp features which is based on a local Newton-type iteration method. The sharp features are then represented by clipped splats (cf. Fig. 8, left). Another very efficient method for interactive CSG computations was proposed by Adams and Dutré (2003) (cf. Fig. 8, right). In a following paper they also implemented their approach on the GPU in order to exploit hardware acceleration for CSG (Adams and Dutré, 2004).

Solving PDEs on triangle meshes is a very important topic and has been widely used for mesh smoothing, mesh editing, deformable models, and many other geometry processing tasks. When transferring these methods to point-sampled models, the missing geodesic neighborhood information again has to be replaced by the spatial k nearest neighbors. These can be used to define

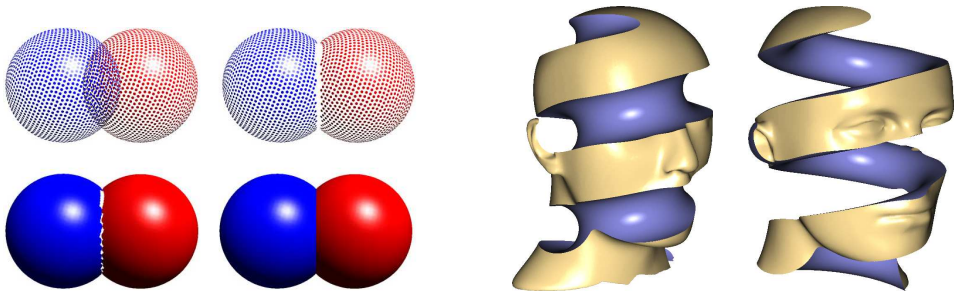

Fig. 8. The left part shows the different steps for point-based CSG operations: classification (*top row*) and intersection curve sampling (*bottom row*). Examples of difference and intersection of a head and a helix are shown on the right.

an approximate tangent space on the point models such that local Delaunay triangulations define the coupling between neighboring points. This allows for the straightforward discretization of differential operators and the derivation of a corresponding stiffness matrix. This approach has been proposed by Clarenz et al. (2004), where they present PDE-based segmentation, texture impainting, texture synthesis, and geometric smoothing as applications of their framework.

Even physically based modeling and animation of volumetric point-sampled models has been proposed very recently (Müller et al., 2004). Since both the surface and the volume of the model are represented by point samples, local restructuring is easily possible, allowing for arbitrarily large distortions from the original shape. Exploiting the flexibility of point-based methods even complicated topology changes, like splashing water, can be simulated.

4 Point-based Rendering

The final stage for interactive geometry processing applications is the efficient rendering of point-sampled geometries. Points have first been proposed as universal rendering primitives by Levoy and Whitted (1985). Instead of deriving a rendering algorithm for each geometry representation, they propose to subdivide each representation into a sufficiently dense set of sample points. The rendering of these points can then be performed by one specially tuned technique.

Since we want to generate continuous (i.e. hole-free) images by rendering a discrete set of surface samples, methods for closing the holes and gaps in-between the samples have to be found. This can be done by image-space reconstruction techniques (Grossman and Dally, 1998; Pfister et al., 2000) or by object-space re-sampling. The techniques from the latter category dynamically adjust the sampling rate, such that the density of projected points meets the pixel resolution. Since this depends on the current viewing parameters, the re-sampling

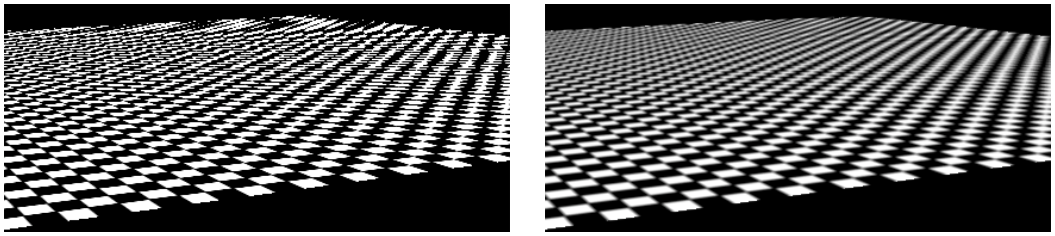

Fig. 9. A checkerboard texture rendered with surface splats: without (*left*) and with EWA filtering (*right*).

has to be done dynamically for each frame. Examples are dynamic sampling of procedural geometries (Stamminger and Drettakis, 2001), the randomized z-buffer (Wand et al., 2001), and the rendering of MLS surfaces (Alexa et al., 2001, 2003a; Fleishman et al., 2003).

In contrast to this, surface splatting (Zwicker et al., 2001) renders splats, i.e., object-space disks or ellipses, instead of points only. In this case, the mutual overlap of splats in object-space guarantees a hole-free rendering in image-space. On the other hand, the naïve rendering of inter-penetrating splats results in shading discontinuities (cf. Fig. 12). Therefore, Zwicker et al. (2001) proposed a high quality anisotropic anti-aliasing method that resembles the anisotropic EWA texture filtering of Heckbert (1989). Each splat is assigned a radially symmetric Gaussian filter kernel, such that a continuous surface signal in object-space is reconstructed by a respectively weighted averaging of splat data (e.g. colors, normals). Combining these object-space reconstruction kernels with a band-limiting image-space filter results in their high quality *EWA splatting* framework (cf. Fig. 9). If both the object-space and the image-space filters are Gaussians, and if the projection is locally (i.e., per splat) approximated by an affine mapping, then these two filters can be combined into one single Gaussian, enabling a quite efficient implementation. Nevertheless, being a purely software-based implementation, this approach is computationally too expensive for displaying highly complex models, since it achieves a splat rate of about 1M splats/sec on current hardware.

Botsch et al. (2002) therefore proposed the use of hierarchical representations for more efficient rendering. Their octree-based point clouds (see Sec. 2) can be displayed by rendering either one point or one splat for each non-empty leave cell. They showed that modelview and projection transformations of a leaf cell center $\mathbf{p}$ can be implemented by an incremental summation during a depth-first traversal of the octree, followed by a final de-homogenization step. Since points sharing a common octree path also share terms of these sums, it turns out that each point transformation requires just about 4 scalar additions and 2 divisions in total. Combining this with pre-computed tables for lighting and splat footprints results in a splat rate of 5M splats/sec. However, this implementation is still completely software-based and puts a high load on the CPU, blocking it from other geometry processing tasks.

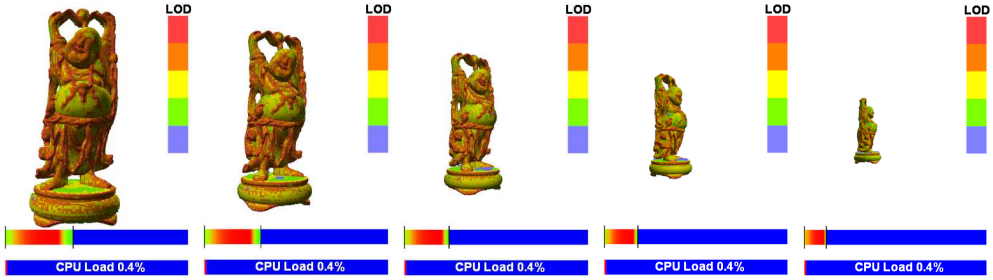

Fig. 10. A sequentialization of a LoD hierarchy results in the rendering of continuous segments of a linear point list. This, and the delegation of fine-grained LoD control to the GPU, results in a hierarchical LoD rendering with almost no CPU load.

In the last years, the increasing efficiency and programmability of modern graphic cards (Lindholm et al., 2001; Mark et al., 2003) triggered the development of hardware-based splatting methods. Targeting at the efficient visualization of the models acquired during the Digital Michelangelo project (Levoy et al., 2000), Rusinkiewicz and Levoy (2000) proposed a hierarchical rendering method based on a pre-computed bounding-sphere tree structure.

As this recursive tree traversal is not efficient enough to keep current GPUs busy, Dachsbacher et al. (2003) proposed a sequentialization of the LoD tree structure that corresponds to a breadth-first re-ordering. Rendering the model at a certain level of detail then requires to process a continuous sub-segment of this linear point list directly by the GPU (cf. Fig. 10). Transferring the point list into video memory once and implementing the LoD selection in the vertex shaders of the GPU results in a low CPU load and an impressive splat rate of more than 50M splats/sec. The drawback of this approach is that the rendering primitives used for each splat are un-filtered image space squares only, hence the quality could be improved considerably by using Gaussian filtering and elliptical splat primitives.

To remove shading discontinuities and achieve a visually smoother rendering, a Gaussian filter kernel should be assigned to each splat also in the case of hardware-accelerated rendering. The difficulty is that when two splats are projected to the same pixel, only closely overlapping splats should be blended, while in other cases when the z-distance between the splats is above a certain threshold, the front-most splat should overwrite the splats behind. An implementation of this functionality requires a fuzzy depth test or a more general a-buffer (Carpenter, 1984). Both are not available on current GPUs, therefore multiple rendering passes have to be used instead. The first one does so-called *visibility splatting*, i.e., it fills the z-buffer by rendering (without lighting) all objects slightly shifted away from the viewer by ε. The second pass renders all splats with Gaussian blending turned on, but it does not alter the z-buffer, thereby blending only those splats that differ by less than ε in depth (Rusinkiewicz and Levoy, 2000). This process accumulates a weighted sum of colors $(\sum_i \alpha_i rgb_i, \sum_i \alpha_i)$ in each RGBA pixel, such that a per-pixel division

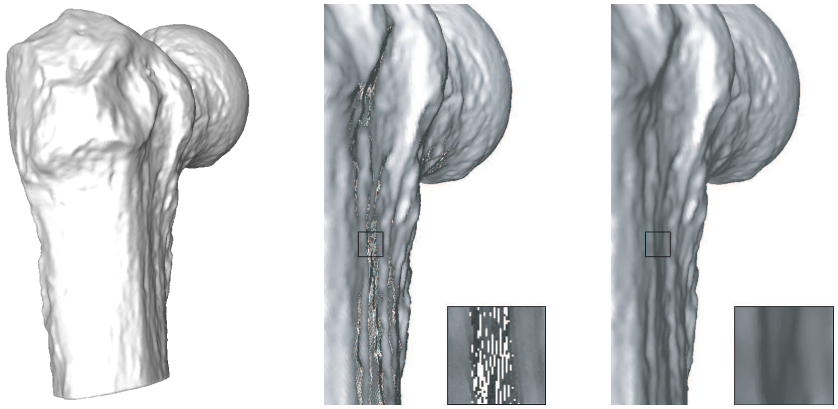

Fig. 11. Previous local affine approximations to the projective mapping caused holes for extreme viewing angles (*center*). These holes can be closed either by an affine approximation that correctly maps the outer splat contour (*right*) or by per-pixel ray-casting that exactly computes the inverse projective mapping.

by its accumulated alpha value calculates the correct weighted average. This per-pixel normalization can efficiently be done by rasterizing a window-sized quad that is textured by the result of the second rendering pass (Botsch and Kobbelt, 2003; Guennebaud and Paulin, 2003).

Using the pixel shaders of current graphics hardware allows the rasterization of elliptical splats by rendering just one vertex per splat. Computing the projected size in a vertex shader triggers the rasterization of an image space square. A fragment shader processes each of its pixels and constructs the elliptical shape by discarding pixels outside the ellipse. An implementation using circular object-space splats and two-pass Gaussian filtering was presented by Botsch and Kobbelt (2003), achieving a splat rate of 10M splats/sec.

However, this method, like most others, uses an affine approximation to the projective mapping in order to compute the splat's footprint in image-space. This simplification may lead to holes in the image, when the model is viewed from extremely flat angles and is shifted away from the main viewing axis (cf. Fig. 11). This problem was addressed first by the perspective accurate splatting of Zwicker et al. (2004), using a different affine approximation to the projection, such that the outer splat contour (instead of the splat center) is correctly mapped. Although being slightly incorrect in the splat's interior, this method effectively avoids holes in the image. As the projection is still approximated affinely, the full EWA splatting framework can be integrated into this approach. Per-pixel correct projections can be achieved performing a local ray casting in the fragment shaders, as proposed by Botsch et al. (2004).

Most point-based rendering methods use a constant normal vector for lighting each individual splat, leading to results comparable to flat shading for triangle meshes (cf. Fig. 12, center left). Blending by Gaussian reconstruction kernels smoothes out the shading artifacts and is hence comparable to Gouraud

Fig. 12. Comparing different rendering techniques for a simplified model of a scanned statue: naïve splatting without filtering is comparable to flat shading (*center left*), Gaussian blending corresponds to Gouraud shading (*center right*), Phong splatting yields the same quality as Phong shading for triangle meshes (*right*).

shading: the shading varies smoothly, but the image may appear blurry (cf. Fig. 12, center right). The use of non-constant normal fields and per-pixel lighting significantly improves the visual quality (Zwicker et al., 2001; Kalaiah and Varshney, 2001, 2003a). For triangle meshes, Phong shading achieves superior rendering quality by computing per-pixel lighting based on interpolated normal vectors of the triangle's vertices in object space. This concept is not directly applicable to point-based rendering since access to the normal vectors of neighboring points or splats is not possible due to the lack of explicit connectivity information. In order to achieve per-pixel normals while maintaining the simplicity of splat-based representations, Botsch et al. (2004) assign a linear normal field to each splat individually. These normal fields are derived by least squares fitting to a given dense set of sample normals. Rendering such Phong-splats can easily and efficiently be implemented on the GPU and yields high visual quality comparable to Phong shading for triangle meshes (cf. Fig. 12, right).

In order to represent sharp features by point-sampled geometries, Pauly et al. (2003b) proposed to clip splats against clipping lines defined in their local tangent frames (cf. Fig. 3). This representation can easily be rendered by integrating a per-pixel clipping test into the fragment shaders, as proposed by Zwicker et al. (2004) and Botsch et al. (2004).

The presented techniques exploit programmable shaders of current GPUs to make hardware acceleration directly available for point-based rendering. The resulting rendering approaches are very efficient and provide a high visual quality. Since elliptical splats represent an optimal approximation to the local surface geometry and since the C^{-1} continuity of splats provides further degrees of freedom, splatting techniques lead to superior visual rendering results

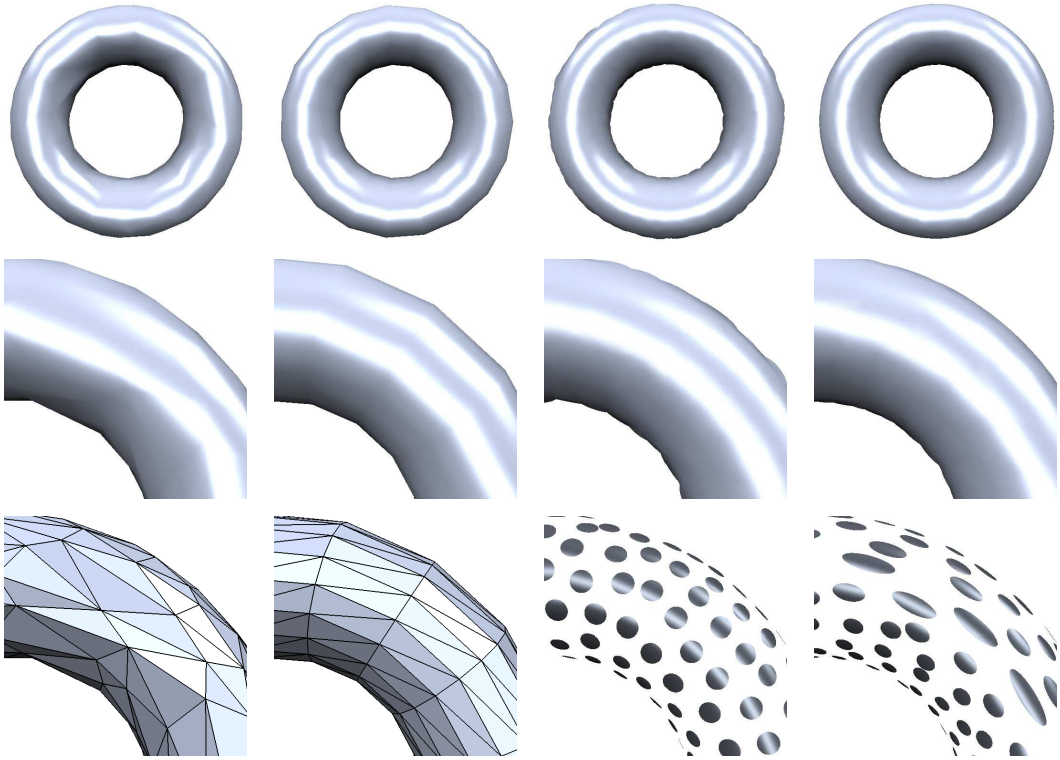

Fig. 13. Comparing different shape approximations consisting of about 730 geometry primitives: irregular triangle meshes (*left*), regular aligned triangle mesh (*center left*), circular splats (*center right*), elliptical splats (*right*). The respective approximation errors to the continuous torus are 0.64%, 0.58%, 0.20%, and 0.14%.

compared to triangle rendering with the same number of linear primitives (cf. Fig. 13). Nevertheless, since current graphics hardware is highly optimized and specialized for triangle rendering, points or splats still cannot keep up with triangles in terms of effective rendering performance.

5 Conclusion

Point-based representations have proven to be a valuable alternative to polygonal meshes in several special applications. In particular when highly complex or dynamic models are to be processed and a proper surface topology is not necessarily required, the simplicity of point-based representations leads to simpler and more efficient algorithms.

Elliptical surface splats provide the same asymptotic approximation power as triangle meshes, but achieve a locally better approximation since the axes of elliptical splats can be aligned to the principal curvature directions of the surface. Moreover, splats offer more flexibility since they do not require C^0 continuity. Hence, given a certain budget of surface primitives, splats usu-

ally yield better approximation results compared to triangles (cf. Fig. 13). Although current graphics hardware is optimized for triangle rendering, programmable shaders enable the very efficient implementation of high quality point-based rendering techniques whose performance is already coming close to the effective polygon rate.

In the future, new types of graphics hardware architectures could exploit the particular advantages of points as a general 3D graphics primitive by supporting adaptive object space sampling techniques in order to reduce the redundancy of point-based representation and the overdraw rate for screen pixels.

Acknowledgments

We thank Mark Pauly and Marc Alexa for helpful comments and the following people for providing us images from their papers: Matthias Zwicker (Fig. 3, 6, 9), Mark Pauly (Fig. 7, 8), Bart Adams (Fig. 8), Carsten Dachsbacher (Fig. 10), and Jianhua Wu (Fig. 5).

References

Adams, B., Dutré, P., 2003. Interactive boolean operations on surfel-bounded solids. In: Proc. of ACM SIGGRAPH 03. pp. 651–656.

Adams, B., Dutré, P., 2004. Boolean operations on surfel-bounded solids using programmable graphics hardware. In: Proc. of Symp. on Point-Based Graphics 04. pp. 19–24.

Adams, B., Wicke, M., Dutré, P., Gross, M., Pauly, M., Teschner, M., 2004. Interactive 3D painting on point-sampled objects. In: Proc. of Symp. on Point-Based Graphics 04. pp. 57–66.

Adamson, A., Alexa, M., 2003. Approximating and intersecting surfaces from points. In: Proc. of Eurographics Symposium on Geometry Processing 03. pp. 245–254.

Adamson, A., Alexa, M., 2004. Approximating bounded, non-orientable surfaces from points. In: Proc. of Shape Modeling International 04.

Alexa, M., Adamson, A., 2004. On normals and projection operators for surfaces defined by point sets. In: Proc. of Symp. on Point-Based Graphics 04. pp. 149–155.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C. T., 2001. Point set surfaces. In: Proc. of IEEE Visualization 01. pp. 21–28.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C. T., 2003a. Computing and rendering point set surfaces. IEEE Transactions on Visualization and Computer Graphics 9 (1), 3–15.

Alexa, M., Dachsbacher, C., Gross, M., Pauly, M., van Baar, J., Zwicker, M., 2003b. Point-based computer graphics. In: Eurographics 2003 Tutorial Notes.

Alexa, M., Gross, M., Pauly, M., Pfister, H., Stamminger, M., Zwicker, M., 2004. Point-based computer graphics. In: SIGGRAPH 2004 Course Notes.

Amenta, N., Bern, M., Kamvysselis, M., 1998. A new Voronoi-based surface reconstruction algorithm. In: Proc. of ACM SIGGRAPH 98.

Amenta, N., Kil, Y., 2004. Defining point-set surfaces. In: Proc. of ACM SIGGRAPH 04.

Andersson, M., Giesen, J., Pauly, M., Speckmann, B., 2004. Bounds on the k-neighborhood for locally uniformly sampled surfaces. In: Proc. of Symp. on Point-Based Graphics 04. pp. 167–171.

Bischoff, S., Pavic, D., Kobbelt, L., 2004. Automatic restoration of polygon models. Preprint.

Botsch, M., Kobbelt, L., 2001. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. In: Proc. of Eurographics 01. pp. 402–410.

Botsch, M., Kobbelt, L., 2003. High-quality point-based rendering on modern GPUs. In: Proc. of Pacific Graphics 03.

Botsch, M., Spernat, M., Kobbelt, L., 2004. Phong splatting. In: Proc. of Symp. on Point-Based Graphics 04.

Botsch, M., Wiratanaya, A., Kobbelt, L., 2002. Efficient high quality rendering of point sampled geometry. In: Proc. of Eurographics Workshop on Rendering 02.

Carpenter, L., 1984. The a-buffer, an antialiased hidden surface method. In: Proc. of ACM SIGGRAPH 84. pp. 103–108.

Carr, N. A., Hart, J. C., 2004. Painting detail. In: Proc. of ACM SIGGRAPH 04.

Clarenz, U., Rumpf, M., Telea, A., 2004. Finite elements on point based surfaces. In: Proc. of Symp. on Point-Based Graphics 04. pp. 201–211.

Dachsbacher, C., Vogelgsang, C., Stamminger, M., 2003. Sequential point trees. In: Proc. of ACM SIGGRAPH 03.

Davis, P., 1975. Interpolation and Approximation. Dover Publications.

Fleishman, S., Cohen-Or, D., Alexa, M., Silva, C. T., 2003. Progressive point set surfaces. ACM Transactions on Graphics 22 (4).

Grossman, J. P., Dally, W. J., 1998. Point sample rendering. In: Proc. of Eurographics Workshop on Rendering 98. pp. 181–192.

Guennebaud, G., Paulin, M., 2003. Efficient screen space approach for hardware accelerated surfel rendering. In: Proc. of Vision, Modeling, and Visualization 03.

Heckbert, P. S., 1989. Fundamentals of texture mapping and image warping. Master's thesis, University of California at Berkley.

Hoffmann, C. M., 1989. Geometric and solid modeling: an introduction. Morgan Kaufmann Publishers Inc.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W., 1992. Sur-

face reconstruction from unorganized points. In: Proc. of ACG SIGGRAPH 92. pp. 71–78.

Kalaiah, A., Varshney, A., 2001. Differential point rendering. In: Proc. of Eurographics Workshop on Rendering Techniques 2001.

Kalaiah, A., Varshney, A., 2003a. Modeling and rendering points with local geometry. IEEE Transactions on Visualization and Computer Graphics 9(1), 30–42.

Kalaiah, A., Varshney, A., 2003b. Statistical point geometry. In: Proc. of Eurographics Symposium on Geometry Processing 03. pp. 107–115.

Kobbelt, L., Bareuther, T., Seidel, H.-P., 2000. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In: Proc. of Eurographics 00.

Kobbelt, L., Botsch, M., Schwanecke, U., Seidel, H.-P., 2001. Feature sensitive surface extraction from volume data. In: Proc. of ACM SIGGRAPH 01. pp. 57–66.

Lawrence, J., Funkhouser, T., 2003. A painting interface for interactive surface deformation. In: Proc. of Pacific Graphics 03. pp. 141–150.

Levin, D., 1998. The approximation power of moving least-squares. Mathematics of Computation 67 (224), 1517–1531.

Levin, D., 2003. Geometric Modeling for Scientific Visualization. Springer, Ch. Mesh-independent surface interpolation, pp. 37–49.

Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., Fulk, D., 2000. The digital Michelangelo project: 3D scanning of large statues. In: Proc. of ACG SIGGRAPH 00. pp. 131–144.

Levoy, M., Whitted, T., January 1985. The use of points as display primitives. Tech. rep., CS Departement, University of North Carolina at Chapel Hill.

Lindholm, E., Kilgard, M., Moreton, H., 2001. A user-programmable vertex engine. In: Proc. of ACM SIGGRAPH 01. pp. 149–158.

Linsen, L., Prautzsch, H., 2002. Fan clouds - an alternative to meshes. In: Proc. of Dagstuhl Seminar on Theoretical Foundations of Computer Vision. pp. 39–57.

Mark, W. R., Glanville, R. S., Akeley, K., Kilgard, M. J., 2003. Cg: A system for programming graphics hardware in a C-like language. In: Proc. of ACM SIGGRAPH 03.

Müller, M., Keiser, R., Nealen, A., Pauly, M., Gross, M., Alexa, M., 2004. Point based animation of elastic, plastic and melting objects. In: Proc. of Eurographics/ACM SIGGRAPH Symp. on Computer Animation 04.

Pauly, M., 2003. Point primitives for interactive modeling and processing of 3D geometry. Ph.D. thesis, ETH Zürich.

Pauly, M., Gross, M., 2001. Spectral processing of point-sampled geometry. In: Proc. of ACM SIGGRAPH 01.

Pauly, M., Gross, M., Kobbelt, L., 2002. Efficient simplification of point-sampled surfaces. In: Proc. of IEEE Visualization 02.

Pauly, M., Keiser, R., Gross, M., 2003a. Multi-scale feature extraction on

point-sampled surfaces. In: Proc. of Eurographics 03.
Pauly, M., Keiser, R., Kobbelt, L., Gross, M., 2003b. Shape modeling with point-sampled geometry. In: Proc. of ACG SIGGRAPH 03.
Pfister, H., Zwicker, M., van Baar, J., Gross, M., 2000. Surfels: Surface elements as rendering primitives. In: Proc. of ACM SIGGRAPH 00. pp. 335–342.
Rusinkiewicz, S., Levoy, M., 2000. QSplat: a multiresolution point rendering system for large meshes. In: Proc. of ACM SIGGRAPH 00. pp. 343–352.
Stamminger, M., Drettakis, G., 2001. Interactive sampling and rendering for complex and procedural geometry. In: Proc. of Eurographics Workshop on Rendering 01. pp. 151–162.
Szeliski, R., Tonnesen, D., 1992. Surface modeling with oriented particle systems. In: Proc. of ACM SIGGRAPH 92.
Wand, M., Fischer, M., Peter, I., Meyer auf der Heide, F., Straßer, W., 2001. The randomized z-buffer algorithm: interactive rendering of highly complex scenes. In: Proc. of ACM SIGGRAPH 01. pp. 361–370.
Welch, W., Witkin, A., 1994. Free form shape design using triangulated surfaces. In: Proc. of ACM SIGGRAPH 94.
Weyrich, T., Pauly, M., Heinzle, S., Keiser, R., Scandella, S., Gross, M., 2004. Post-processing of scanned 3D surface data. In: Proc. of Symp. on Point-Based Graphics 04. pp. 85–94.
Witkin, A., Heckbert, P., 1994. Using particles to sample and control implicit surfaces. In: Proc. of ACM SIGGRAPH 94.
Wu, J., Kobbelt, L., 2004. Optimized subsampling of point sets for surface splatting. In: Proc. of Eurographics 04.
Zwicker, M., 2003. Continuous reconstruction, rendering, and editing of point-sampled surfaces. Ph.D. thesis, ETH Zürich.
Zwicker, M., Pauly, M., Knoll, O., Gross, M., 2002. PointShop 3D: An interactive system for point-based surface editing. In: Proc. of ACM SIGGRAPH 02.
Zwicker, M., Pfister, H., van Baar, J., Gross, M., 2001. Surface splatting. In: Proc. of ACM SIGGRAPH 01. pp. 371–378.
Zwicker, M., Räsänen, J., Botsch, M., Dachsbacher, C., Pauly, M., 2004. Perspective accurate splatting. In: Proc. of Graphics Interface 04.