Eurographics Symposium on Point-Based Graphics (2004)
M. Alexa, S. Rusinkiewicz, (Editors)

Real-Time Point Cloud Refinement

G. Guennebaud and L. Barthe and M. Paulin†

IRIT - CNRS - Université Paul Sabatier - Toulouse - France

Abstract

Splatting-based rendering techniques are currently the best choice for efficient high quality rendering of point-based geometries. However, such techniques are not suitable for large magnification, especially when the object is under-sampled. This paper improves the rendering quality of pure splatting techniques using a fast dynamic up-sampling algorithm for point-based geometry. Our algorithm is inspired by interpolatory subdivision surfaces where the geometry is refined iteratively. At each step the refined geometry is that from the previous step enriched by a new set of points. The point insertion procedure uses three operators: a local neighborhood selection operator, a refinement operator (adding new points) and a smoothing operator. Even though our insertion procedure makes the analysis of the limit surface complicated and it does not guarantee its G^1 continuity, it remains very efficient for high quality real-time point rendering. Indeed, while providing an increased rendering quality, especially for large magnification, our algorithm needs no other preprocessing nor any additional information beyond that used by any splatting technique.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Viewing algorithms

1. Introduction

Owing to the absence of topological information, point clouds give us a simple and powerful surface representation for complex geometries where the accuracy mainly depends on the number of points. However, real-time visualization of such data sets requires additional information such as normal vector, texture color, and an estimation of the local sampling density. From these additional attributes, a continuous image of the point cloud can be reconstructed using an image-based filtering technique, by adjusting the sampling density on the fly or by using the so-called surface splatting technique [ZPvBG01]. In the latter case, each point is represented by an oriented disk (a *surfel*) in object space [PZvG00]. Rendering is then equivalent to a resampling process where surfels are blended with a Gaussian distribution in the image space. In this paper, we call such a point cloud a *surfel set*. Currently, for high quality and efficient point-based rendering, a splatting approach is doubtless the best choice since such approaches are supported by modern GPUs [BK03, GP03]. Whereas a surfel set describes a continuous texture function [ZPvBG01], from the geometric point of view it is a simple set of oriented overlapping disks. Hence, in the case of an under-sampled surface, visual artifacts appear on the silhouette and in areas of high curvature (figure 1 left). Moreover,

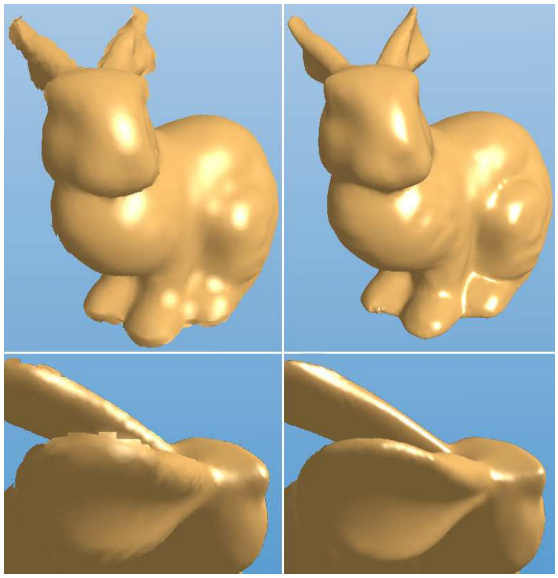


Figure 1: *Left: rendering of an undersampled bunny with a pure high quality splatting technique. Artifacts on silhouette and specular reflexions are clearly visible. Right: same model with our dynamic up-sampling algorithm enabled.*

† e-mail: {guenneba | lbarthe | paulin}@irit.fr

effects at pixel frequency such as reflections (i.e. specular reflections and environment maps) can not be properly handled by large splats (figure 9). Thus, for high quality rendering, the use of a pure splatting based approach is limited to relative small magnification.

Although a point set intrinsically describes a smooth surface, the geometry itself is discontinuous. This can be compared to polygonal meshes where the geometry is only C^0 continuous even though we may intend the mesh to describe a smooth (G^1) surface. In order to overcome the continuity problem of polygonal meshes, several methods have been developed. Among these, subdivision surfaces perform the refinement of a coarse mesh into a finer one and several iterations generate a sequence of incrementally refined meshes which converges to a smooth surface [Cla78, DS78, ZS00, WW02]. More specifically, interpolatory subdivision schemes [DLG90, ZSS96, Kob96] are well suited when we desire smooth interpolation of the mesh vertices. Following the same idea, a point set could be refined in order to maintain local point density and hence improve rendering quality. Unfortunately, owing to the lack of topological information, subdivision operators for meshes cannot be directly applied to point sets.

On the other hand, several *consolidation* methods have been proposed. By *consolidation* we mean the process of *extrapolating* a continuous surface from the point set. Most consolidation methods are based on an implicit representation. For example, in [HDD*92], a triangular mesh is built from a signed distance function defined on a volumetric grid. Others are based on radial basis functions (RBF) that reconstruct a C^n implicit surface from a scattered point set [CBC*01]. However, owing to the global support of RBFs, such approaches need an expensive preprocessing step since the coefficients of the RBFs are computed by solving a large linear system. This problem is partially overcome by local approaches [OBA*03, TRS04], but they remain too expensive for real-time applications.

In [Lev01], Levin introduces a smooth point-based representation called moving least squares (MLS) surface. The surface is defined implicitly by a local projection operator. Based on this representation, several methods for down-sampling [ABCO*03, PGK02] and up-sampling [ABCO*03, PKKG03] point sets have been proposed. However, these up-sampling methods are not suitable for real-time applications since the computation of the local projection operator is a non-linear optimization problem. Moreover, methods used for the generation of a locally uniform sampling are expensive to evaluate since they are based on either a local Voronoi diagram or a particle simulation [Tur92]. In [ABCO*03] Alexa et al. present an interactive rendering technique based again on the MLS surface representation. In a preprocessing step, a bivariate polynomial is computed for each point of the reference point set. During rendering, additional points can be dynamically sampled from these polynomials. In addition to the need for preprocessing, the drawbacks of this up-sampling approach are that it does not support discontinuities or texture colors, it requires much memory for storing the polynomials, and it generates oversampling because of the overlapping of polynomials patches.

In [SD01], Stamminger and Drettakis render complex procedural geometry with a dynamic $\sqrt{5}$ sampling algorithm. While their sampling scheme is fast to evaluate, its extension to the smooth up-sampling of general point-based geometries is difficult.

In order to increase the rendering quality of surfel sets, we present a new up-sampling method inspired by subdivision surfaces. The main features of our algorithm are:

- **speed**: real-time processing is our major constraint.
- **simplicity**: easy to implement and adapted to further hardware optimizations.
- **smoothness**: the visualized surface looks smooth.
- **locally uniform sampling**: avoiding oversampling is a fundamental issue, especially for hardware splatting approaches that are limited by the precision of the color buffer.
- **globally adaptive sampling**: only areas that need accurate sampling are refined.
- **suitable for discontinuities**: our method handles boundaries and sharp creases.
- **no preprocessing**: our system takes as input an unstructured point set with per point normal, texture color and radius. This set of attributes is the minimum information needed for all point based rendering techniques. Because our algorithm does not need any preprocessing, it is well suited for handling deformable models.

Note that our real-time constraint limits our choices for the design of the interpolation methods so, while the approach presented here increases the rendering quality of a pure splatting technique, we cannot guarantee its G^1 continuity.

2. Overview

Our algorithm takes, as input, a regular point set $P^0 = \{p_i\}$ defining a smooth surface. We assume that we also know, for each point $p_i \in P$, its normal $\vec{n}_i$, its texture color and the local density described by a scalar radius r_i. The radius, r_i, of each surfel has to be large enough to provide a splatting rendering without holes, and it must be less than or equal to the maximum distance between the i^{th} surfel and its neighbors. The initial point set, P^0, is up-sampled by inserting additional points yielding the new set P^1 with $P^0 \in P^1$. In a similar fashion to subdivision surfaces, the up-sampled point set describes a new surface that is used for the next refinement step. At each refinement step, the number of points approximately quadruples, increasing the resolution by a factor of two (figure 2). Hence, the radius of surfels are divided by two at each step. By repeating the refinement step we construct a sequence $P^0, P^1, \ldots$ of point sets with $P^l \subset P^{l+1}$.

Our up-sampling algorithm can be described by a selection operator Ψ and an interpolation operator Φ. The selection operator (see section 3.1 and figure 4) takes a point

$p \in P^l$ and defines the set $\Psi(p)$ of point subsets $\Psi_i(p)$ around p from which a single new point will be inserted:

$$\Psi : P^l \longrightarrow \mathcal{P}(\mathcal{P}(P^l)) \qquad (1)$$
$$\Psi : p \longmapsto \{\Psi_0(p), ..., \Psi_m(p)\}$$

with $\mathcal{P}(E)$ the power set of the set E: $\mathcal{P}(E) = \{e | e \subset E\}$. The operator Φ (section 3.2) inserts a single new point by interpolation of the points of $\Psi_i(p)$. Hence for each $\Psi_i(p)$, a new point is added to P^{l+1}:

$$\Phi : \mathcal{P}(P^l) \longrightarrow \mathbb{R}^3 \qquad (2)$$

and the up-sampled point set P^{l+1} of P^l is defined as follows:

$$P^{l+1} = P^l \cup \{\Phi(\Psi_i(p)) | \Psi_i(p) \in \Psi(p), \forall p \in P^l\} \qquad (3)$$

For convenience, attributes of points (normals, colors, etc) do not appear in these definitions. As mentioned in section 4, the global subdivision process must be slightly modified to avoid redundancy. However, before describing the global subdivision algorithm (section 4), we first present in detail the refinement procedure around a single point $p \in P^l$, by describing the local operators Ψ and Φ.

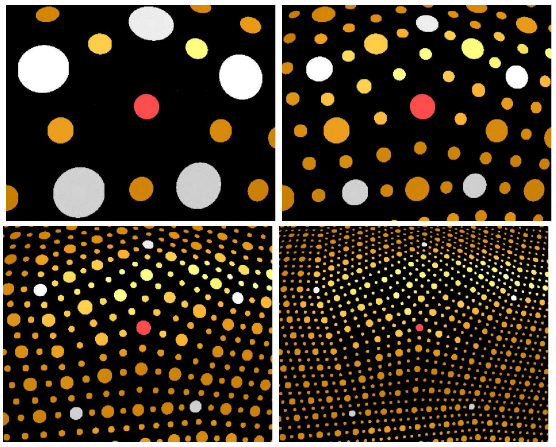
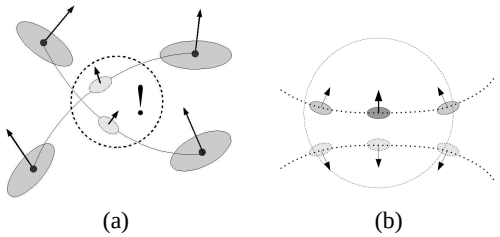


Figure 2: *Illustration of the refinement procedure. On the top left, the initial points (from the bunny model) are visualized with large white surfels. The smaller points have been introduced by a single refinement step. The red point comes from the interpolation of five points. From left to right and top to bottom, one refinement step is performed on the input points (comming from the previous refinement step and visualized with large surfels).*

3. Local Up-Sampling

3.1. The Selection Operator, Ψ

Our up-sampling scheme is based on the idea of adding a new point for each pair of neighbor samples. However, whatever the accuracy of the neighbor relation, this basic idea is insufficient because a subset of $k \geq 4$ points that are all in the neighborhood of one another generates $\frac{1}{2}k(k-3)$ points near their center (figures 3a, 4b). In such cases, the obvious choice is to insert only a single new point.



(a) (b)

Figure 3: *(a) Four surfels are all in the neighborhood of one another. Interpolating points two by two leads to oversampling and incoherency. (b) The query ball intersects two disjoint components of the surface.*

From a given point $p \in P$ and its neighborhood $N_p \subset P$ (section 3.1.1), the selection operator Ψ must define a set of subsets of points in N_p for which a single new point must be inserted. This is done by building a local set of polygons, called a *polygon fan*, from the implicit triangle fan defined by the neighborhoods (section 3.1.2). This construction is similar to the *fan cloud* representation of Linsen et al. [LP02].

Hence, the robustness of Ψ to generate a local uniform sampling typically depends on the definition of the neighborhood. To perform a complete neighbor selection, N_p must enclose the current point, and it must not select samples which are not in the first ring neighborhood. Moreover, in order to be sample-order independent and to be able to solve the global duplication problem (section 4), the neighbor relation must be symmetric. Hence, simple *k-nearest* neighborhoods cannot be used. More sophisticated neighborhoods based on the Voronoi diagram or BSP are too selective to be used in our case since they can remove samples that are actually in the first neighborhood ring [FR01]. For these reasons, we define our own neighborhood, based on distance and minimum angle criteria.

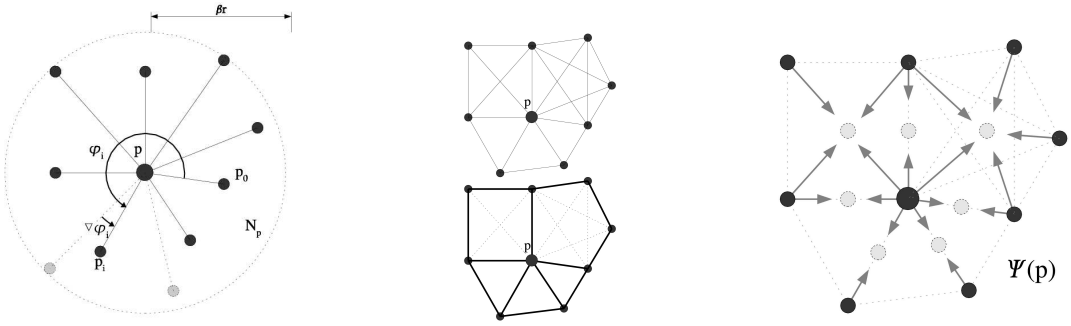
3.1.1. Local Neighborhood

The computation of the neighborhood N_p of a given point $p \in P$, of radius r and normal $\vec{n}$ is performed in two steps. First, we compute the subset $\widetilde{N}_p \subset P$ such that each point $p_i \in \widetilde{N}_p$ is in the sphere of center p and radius βr. In order to avoid problems owing to fine "features" (figure 3b), we also remove from $\widetilde{N}_p$ points for which the angle between normals $\vec{n}_i$ and $\vec{n}$ is greater than a given crease angle threshold θ (see section 3.2.2 for more details on creases).

$$\widetilde{N}_p = \{p_i \in P \mid \|p_i - p\| \leq \beta r,\ \vec{n} \cdot \vec{n}_i > cos(\theta)\} \qquad (4)$$

Since the radius r should be slightly smaller than the maximum distance between p and its neighbors, a value of β in $[1,2]$ ensures to find all neighbors.

In the second step, points p_i are projected onto the tangent plane of p and sorted such that their projections q_i form increasing angles $\varphi_i = \widehat{q_0 p q_i}$. Finally, we compute the subset $N_p \subset \widetilde{N}_p$ by removing neighbors that are not close enough to the point p. This is done by removing the farthest point



(a) (b) (c)

Figure 4: *Illustration of the local selection operator Ψ applied to a point p. (a) Computation of the neighborhood N_p. After sorting neighbors with increasing angles, the sample p_{i-1} is removed because it is too close to p_i according to an angle-distance criterion. (b) Computation of the polygon fan. Light lines represent the neighbor relations. (c) The result of the local selection operator yields the insertion of 7 new points.*

between p_i and p_{i-1} if they are too close to each other according to an angle criterion: $\nabla\varphi_i = \varphi_i - \varphi_{i-1} < \tau$. Since the projection onto the two-dimensional tangent plane reduces the angle between two consecutive neighbors, the angle threshold τ must be small. Experimentation shown that $\tau = \frac{\pi}{8}$ is a reasonable choice.

3.1.2. Local Polygon Fan

Remember that the basic principle of our up-sampling method is to add a new point for each pair of neighbors. However, before adding a point for each edge (p, p_i) with $p_i \in N_p$ we must detect whether any other pair $(p_j, p_k) \in {N_p}^2$ is in *interaction* with the current edge (p, p_i) as illustrated in figures 3a and 4b. Hence, in this section we explain how to compute the polygon fan around the point p from its neighborhood $N_p = \{p_0, ..., p_m\}$.

We consider the current subset $H_0 = \{p, p_0\}$. A polygon is built from this subset by adding iteratively into H_0 the successors p_j of p_0 while p_j is a neighbor of all points of H_0. At the end of this insertion, the set $H_0 = \{p, p_0, ..., p_l\}$ describes a polygon which is the first of the polygon fan. We restart the construction with $H_1 = \{p, p_l\}$ and it is repeated until all neighbors are taken into account. This procedure produces a *polygon fan* (figure 4b) that completely defines the selection operator $\Psi(p)$. Note that these polygon fans can contain holes and degenerated polygons (edges). Finally, the set $\Psi(p) = \{\Psi_i(p)\}$ is the union of all polygons H_k such that $|H_k| \geq 4$ and all $\{p, p_j\}$ such that (p, p_j) is an edge of the final polygonal fan (figure 4c). Hence a new point is inserted for each outgoing edge from p and each polygon that have a minimum of 4 vertices.

3.2. The Interpolation Operator, Φ

We have designed our interpolation operator to be as efficient as possible without the need for preprocessing. Most smooth interpolation methods need a relatively large neighborhood but, in our case, computing a neighborhood larger than one ring is too expensive. Hence we choose to perform interpolation only with the small input set $S = \Psi_i(p)$ given by the selection operator. Even though the simple set S is not enough to perform a globally smooth interpolation, we can still interpolate the points of S locally with a cubic curve or a bicubic patch, using their normal information. This allows us to insert a new point which lies on this curve or patch.

We decompose the interpolation operator Φ as an *insertion* operator inserting a new point at the center of gravity (*Cog*) and a *smoothing* operator $\widetilde{\Phi}_k$ such that:

$$Cog(\{p_0, ..., p_k\}) = \frac{1}{k}\sum_{i=0}^{k} p_i \tag{5}$$

$$\Phi(S) = Cog(S) + \widetilde{\Phi}_{|S|}(Cog(S), S) \tag{6}$$

where $|S|$ denotes the cardinality of the set S. Since the new point is inserted at the center of gravity of the set S, the texture color of the new sample is calculated as the simple average of the texture colors of all points in S. After describing the smoothing operator $\widetilde{\Phi}_k$ in the next subsection we discuss discontinuity issues in section 3.2.2.

3.2.1. Normal Based Smoothing, $\widetilde{\Phi}_k$

As mentioned above, our interpolation method is based on the construction of a local surface made up of bicubic Bézier patches (triangular and quadrilateral) with the help of the given normals. Our construction is similar to *PN triangles* of Alex Vlachos [VPBM01]. However, the construction of such a surface with G^1 continuity is too expensive for our real-time constraint. In fact, we do not need to build an explicit set of Bézier patches since only a few new points are added. For instance, no sample are inserted into triangles. Moreover, the computation of all patches at each step compensates partially the fact that adjacents patches are only C^0 continuous. Our method provides good results with only a few computations. Let $k = |S|$ be the number of points from which a new sample is interpolated. Depending on the value, k, we have different cases:

- $k = 2$: interpolation by a cubic Bézier curve

- $k = 3$: owing to the refinement operator, no new point is inserted in a triangle (figure 4c)
- $k = 4$: interpolation by a bicubic Bézier patch
- $k \geq 5$: irregular case

Cubic Point-Normal Interpolation, $\widetilde{\Phi}_2$

The smoothing operator displaces the inserted point $c = Cog(S)$ on an interpolation curve. As suggested in [Far02], the interpolation of two oriented points p_{i_0}, p_{i_1} ($S = \{p_{i_0}, p_{i_1}\}$) with normals $\vec{n}_{i_0}, \vec{n}_{i_1}$ is based on the construction of a cubic Bézier curve $B(u)$. We take $B(0.5)$ for the position of the inserted point, i.e. the smoothing operator is defined as $\widetilde{\Phi}_2(c, \{p_{i_0}, p_{i_1}\}) = B(0.5) - c$ (figure 5). The extremities b_0, b_3 of the curve are p_{i_0} and p_{i_1} and we must take b_1 (resp. b_2) in the tangent plane of p_{i_0} (resp. p_{i_1}). Since there are an infinite number of solutions, we take one which is both convenient to compute and of reasonable shape. Let b_1' be the projection of the point p_{i_1} into the tangent plane of p_{i_0}. We take b_1 such that $\overrightarrow{b_0 b_1} = \nu \overrightarrow{b_0 b_1'}$. The ν scalar defines the *velocity* of the curve and it must be close to $\frac{1}{3}$ for visually good results [VPBM01]. Let T_i be the projection operator:

$$T_i(q) \; = \nu * (q - ((p_i - q) \cdot \vec{n}_i)\, \vec{n}_i) \tag{7}$$

Hence we have:

$$\widetilde{\Phi}_2\,(c, \{p_{i_0}, p_{i_1}\}) = \frac{3}{8}\,(T_{i_0}(p_{i_1}) + T_{i_1}(p_{i_0})) \tag{8}$$

In order to compute the normal $\vec{n}$ of the new point $p = \widetilde{\Phi}_2\,(c, \{p_{i_0}, p_{i_1}\})$ we first compute the curve tangent $\dot{B}(0.5)$ and we take a perpendicular vector. Again, there are an infinite number of solutions, and a reasonable choice is to take the normal which is in the plane of normal $\vec{n}_{plane}$:

$$\begin{aligned} \vec{n}_{plane} &= (\vec{n}_{i_0} + \vec{n}_{i_1}) \wedge (p_{i_1} - p_{i_0}) \\ \vec{n} &= \vec{n}_{plane} \wedge \dot{B}(0.5) \end{aligned} \tag{9}$$

Bicubic Point-Normal Interpolation, $\widetilde{\Phi}_4$

When a point has been inserted from four surfels, its displacement can be computed from a bicubic Bézier patch $B(u,v)$. Again, we take $\widetilde{\Phi}_4(c, \{p_{i_0}, ..., p_{i_3}\}) = B(0.5, 0.5) - c$ as the smoothing displacement vector. The position of the 4 corner Bézier points are $p_{i_0}, ..., p_{i_3}$. The 8 control points at the boundary of the patch are computed as in the previous case. For the 4 interior Bézier points the simpler solution is to take the *zero twists* method [Far02]. We have, for the corner point p_{i_0}:

$$\begin{aligned} b_{00} &= p_{i_0} \\ b_{01} &= b_{00} + T_{i_0}(p_{i_1}) \\ b_{10} &= b_{00} + T_{i_0}(p_{i_3}) \\ b_{11} &= b_{00} + T_{i_0}(p_{i_1}) + T_{i_0}(p_{i_3}) \\ &= b_{00} + 2T_{i_0}(\tfrac{p_{i_1} + p_{i_3}}{2}) \end{aligned}$$

This *zero twists* solution makes the evaluation of $B(0.5, 0.5)$ very efficient since, after simplifications, we do not need to compute the position of the boundary Bézier points:

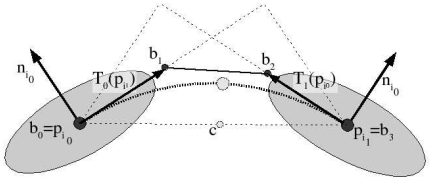


Figure 5: *Contruction of a cubic Bézier curve from two oriented surfels. One sample is added at the middle of the curve.*

$$\widetilde{\Phi}_4(c, \{p_{i_0}, ..., p_{i_3}\}) = \frac{3}{16} \sum_{j=0}^{3} 2T_{i_j}(\frac{p_{i_{j+1}} + p_{i_{j+3}}}{2}) \tag{10}$$

The normal is given by the cross product of the two tangents of the Bézier patch.

Generalised Point-Normal Interpolation, $\widetilde{\Phi}_k, k \geq 5$

While it is possible to construct patches with an arbitrary number of edges [Far02], we propose here a simpler method. Indeed, such *irregular* cases appear only during the first refinement step and with a small frequency. By extension to the regular two previous cases, we compute the displacement of the inserted point c from the interpolation of k surfels, with $k \geq 5$, as follow:

$$\widetilde{\Phi}_k(c, \{p_{i_0}, ..., p_{i_{k-1}}\}) = \frac{3}{4k} \sum_{j=0}^{k-1} 2T_{i_j}(c) \tag{11}$$

The computation of the normal cannot be generalized in the same manner. A reasonable solution is to take the average of the k normals resulting of the k cross products: $(p_{i_{j-1}} - p) \wedge (p_{i_j} - p)$.

While we give a generalized case for polygons with $k \geq 5$ edges, in practice we never met cases with $k > 5$. This is principally due to both the minimum angle criterion in our neighborhood definition that forces the selection of mostly regular polygons and the regularity of the input point set. Indeed, we notice that the robustness of our method directly depends of the sampling regularity. It is also possible to avoid such cases by splitting polygons into triangles and quads, but by doing so, several points will be inserted in the polygon, yielding to a less uniform sampling. Moreover, this can introduce more oscillations since all vertices do not participate equally in the interpolation. The refinement of a such case is illustrated figure 2.

3.2.2. Discontinuities

Discontinuities such as boundaries and creases can be handled easily by our approach. Indeed, boundaries do not need special treatment if we assume that the boundary lines pass through the exact centers of the boundary surfel. Creases can be handled in the same way by using an explicit crease line representation as in [PKKG03]. Sharp features are handled by pairs of *half-surfels* with the same position but different normals along the crease line. Hence, the crease angle θ used

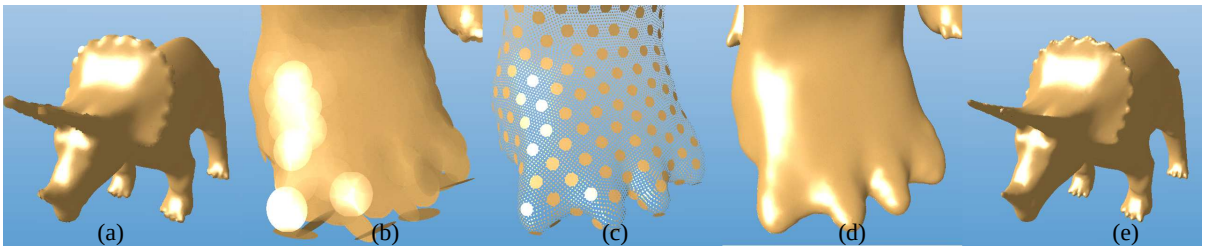


Figure 6: *Illustration of our algorithm on the Triceratops models.(a) Rendering of the given point cloud (16k points).(b) A close view without refinement.(c) Illustration of the refinement.(d) The same close view after three refinements. (e) A global view of the Triceratops with up-sampling enabled.*

in the neighborhood definition (section 3.1.1) discards surfels that are in the other face and the crease case is the same as the boundary case. More details on the rendering of sharp features can be found in [PKKG03, ZRB*04].

However, if creases are not explicitly represented by two *half-surfels* in the reference point cloud geometry, it is possible to detect such creases in the first subdivision step using the crease angle θ and generate two half-surfels at the intersection of the tangent planes instead of using our *normal based interpolation*. It is also possible to use, in a preprocessing step, a more robust and automatic feature detection algorithm [PKG03].

4. Global Up-Sampling Algorithm

In the previous section we have shown how a given point neighborhood is refined into several points with local uniformity. However, the direct subdivision of a point set P^l to P^{l+1} with the basic formulation (equation 3) generates multiple duplicated samples: points generated from k samples appear k times in P^{l+1}. In order to avoid these duplications, we first remove from P^l the current processed point. Hence, P^{l+1} is computed as follow:

```
for each p ∈ P^l do
    P^{l+1} ← P^{l+1} ∪ {Φ(S_i)|S_i ∈ Ψ(p)}
    P^l ← P^l − {p}
done
```

Another problem is the overlaping polygons (figure 7) which is inherent to the independance of the neighborhood computations. Such overlaping polygons can also appear when processed surfels have been removed from the neighborhood of the current surfel. Overlaping polygons yield to the insertion of very closed samples and hence a non-uniform sampling. We solve this problem by storing for each point p a *black list* L of indices containing the list of *wrong* neighbors. This list L is used during the local neighborhood compution of the point p by removing from the coarse neighborhood $\widetilde{N}_p$ the list of points indexed by L:

$$\widetilde{N}_p \leftarrow \widetilde{N}_p - \{p_j \in P | j \in L\}$$

These *black lists* are updated as points are processed. After sorting $\widetilde{N}_p$ with the angle criterion, we update the *black list* of each neighbor $p_j \in \widetilde{N}_p$ by adding all $p_k \in \widetilde{N}_p$ into L_j if and only if the edge (p_k, p_j) is into the polygon defined by N_p and p_k is not the successor or the predecessor of p_j. To be efficient, the *black lists* must be as short as possible. Hence, we had two other simple conditions: p_k must be into $\widetilde{N}_{p_j}$ and $j < k$ (if we assume that points are processed along their index). The number of selected neighbors thus decreases dramatically during the refinement procedure, significantly increasing the performance.

Note that neighbor lists are not all stored into memory. The neighborhood is computed for a given surfel only when this surfel is refined and deleted straight away. During the refinement procedure, the main memory comsumption is due to the *black lists* (an average of 3 indices by surfel).

5. Real-Time Rendering

Our refinement procedure has been designed to improve the rendering quality of point based geometry in the context of real-time applications. Our rendering method is simple and our refinement process is added on top of a hardware accelerated EWA splatting algorithm [GP03]. We assume that the point set is stored into a simple octree that allows classical optimizations such as hierarchical visibility culling and multiresolution rendering. In the case of magnification, all points of visible cells are refined with our algorithm while the density is not high enough. A more accurate selection of visible areas could be done using the method presented in [GBP04]. In order to improve performance, the result of each refinement is stored in a cache and cells are sub-

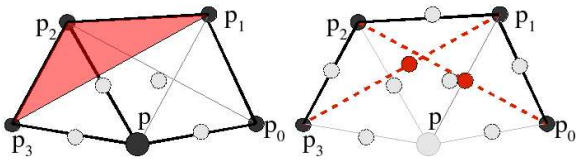


Figure 7: *Left: the point p is refined and 4 new points are inserted while the red triangle overlaps two other polygons. Right: the point p is removed and the points p_0 is refined. A wrong new point is inserted between p_0, p_2. Then, p_1 is refined and another wrong point is inserted between p_1, p_3. This problem is solved during the refinement of p by inserting p2 into the black list of p_0 and p_3 into the black list of p_1.*

divided only if the desired level does not already exist in the cache. Hence, owing to temporal and spatial coherency, only a few samples need to be refined at each frame and real-time can be achieved. Since the splatting process can be entirely performed by the graphics hardware [GP03], GPU and CPU tasks can be efficiently organized. So, we recommend that the rendering of available surfels starts before the up-sampling procedure, hence absorbing a large part of the up-sampling overhead.

Figure 8: *A close view of the cameleon model (90k pts). Left: EWA splatting. Right: after two refinement steps.*

6. Implementation and Results

We have implemented an experimental point-based rendering system based on our hardware accelerated EWA splatting algorithm presented previously in [GP03]. A critical time-consuming part of our up-sampling algorithm is the search of the neighborhood $\widetilde{N}_p$ (section 3.1.1). Fast closest-points queries are classically performed using a *kd*-tree data structure. However, in our case, a simple 3D grid is well suited since it is faster to compute and update. Indeed, we have shown that we have to remove the current processed point from the current point set (section 4), but in fact we only remove its index from the query grid data structure. Moreover, the reduction of the number of elements in the grid increases the speed of the search.

We have tested our unoptimized implementation on a 2GHz AMD Athlon system with 512Mb of memory and a nVidia GeforceFX 5900 graphic card. Some results of our up-sampling method are shown figures 1,6 and 8. Both of these images show the low quality of a fully optimized splatting technique on the model's silhouette and on under-sampled geometry. Images rendered after multiple refinements show a real improvement in quality. Our interpolatory method is visually compared to the modified butterfly scheme figure 10. Whereas our method can provide some local normal discontinuities, it generates less oscillations than the butterfly [ZSS96].

Figure 9: *Left, splatting with reflexion lines from a spherical environment map on the bunny model (3k pts). Right, same model with our dynamic up-sampling algorithm enabled (187k pts).*

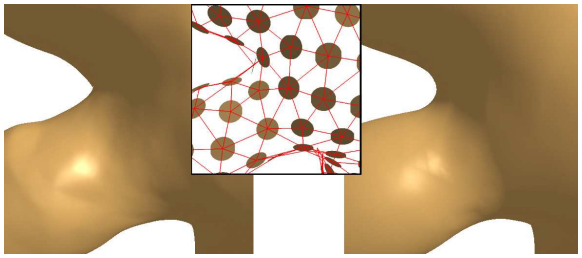


Figure 10: *An implicit surface sampled with 300 pts (center) is visualized after five refinement steps with the modified butterfly algorithm (left) and our method (right).*

Table 1 shows the raw performance of our up-sampling method. We are able to process approximately 250k points in one second, yielding a point generation performance of 1M points per second (because each iteration quadruples the number of points). Such performance is sufficient for the rendering algorithm described above. Indeed, due to spatial and temporal coherency, the number of samples which have to be refined per frame rarely exceeds 15k, and hence we can keep the frame rate above 25 fps. With regard to the cost of each part of our algorithm, searching the grid takes approximately 40% of the time while interpolation takes 30%. The remaining 30% is used for the rest of the processing, essentially for sorting and selecting neighbors.

	Bunny		Triceratops	
# iter.	# points	Time (s)	# points	Time (s)
0	3k	-	16k	-
1	11k	0.01	63k	0.07
2	46k	0.039	256k	0.28
3	187k	0.160	1M	1.05
4	750k	0.79	—	—
5	3M	2.9	—	—

Table 1: *Raw performances of our up-sampling algorithm on two complete models.*

7. Conclusions and Future Work

We have presented a fast and easy to implement up-sampling algorithm for oriented point-clouds. We present both a refinement scheme and a smoothing operator. While the smoothing operator is not G^1 continuous, our results show that we significantly improve the quality of pure splatting techniques.

Our method is also useful in less time-critical applications. Because the refinement and smoothing are totally independent, for such less time-critical applications, it would

be possible to use more robust existing interpolation methods. For instance, we could use the projection procedure of the MLS surface representation as the smoothing operator. An alternative would be to use the gradient of a precomputed RBF implicit surface.

As future work we will further optimize our software implementation and try a partial hardware implementation with GPU features available in upcoming graphics cards. The simplicity of our interpolation method is a good starting point. We will also attempt to improve the quality by making our smoothing operator more robust and G^1 continuous. Finally, it would be interesting to integrate a more sophisticated selection of points that have to be refined by taking into account, in addition to the local density, the local curvature and the silhouette.

Acknowledgements

We would like to thank Neil Dodgson from the University of Cambridge for proof-reading the paper.

References

[ABCO*03] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Computing and rendering point set surface. *IEEE Transaction on Visualization and Computer Graphics* (January 2003). 2

[BK03] BOTSCH M., KOBBELT L.: High-Quality Point-Based Rendering on Modern GPUs. In *11th Pacific Conference on Computer Graphics and Applications* (2003), pp. 335–343. 1

[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. 2

[Cla78] CLARK E. C.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design 10*, 6 (1978), 350–355. 2

[DLG90] DYN N., LEVIN D., GREGORY J.: A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transaction on Graphics, 9 (2)* (1990), 160–169. 2

[DS78] DOO D., SABIN M.: Analysis of the behaviour of recursive subdivision surfaces near extraordinary points. *Computer Aided Design 10*, 6 (1978), 356–360. 2

[Far02] FARIN G.: *CAGD a pratical guide*, 5 ed. Academic Press, 2002. 5

[FR01] FLOATER M. S., REIMERS M.: Meshless parameterization and surface reconstruction. *Comp. Aided Geom. Design 18* (2001), 77–92. 3

[GBP04] GUENNEBAUD G., BARTHE L., PAULIN M.: Efficient point selection using temporal coherency. In *Proceedings of Eurographics 2004, to appear* (2004). 6

[GP03] GUENNEBAUD G., PAULIN M.: Efficient screen space approach for Hardware Accelerated Surfel Rendering. In *Vision, Modeling and Visualization* (2003), IEEE Signal Processing Society. 1, 6, 7

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *Proceedings of ACM SIGGRAPH 92, Computer Graphics Proceedings* (1992). 2

[Kob96] KOBBELT L.: Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Proceedings of Eurographics 1996* (1996). 2

[Lev01] LEVIN D.: Mesh-independent surface interpolation. In *Advances in Computational Mathematics* (2001). 2

[LP02] LINSEN L., PRAUTZSCH H.: Fan clouds - an alternative to meshes. In *Dagstuhl Seminar 02151 on Theoretical Foundations of Computer Vision - Geometry, Morphology and Computational Imaging* (2002). 3

[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 (July 2003), 463–470. 2

[PGK02] PAULY M., GROSS M., KOBBELT L. P.: Efficient simplification of point-sampled surfaces. In *Proceedings of the 13th IEEE Visualization Conference* (2002), pp. 163–170. 2

[PKG03] PAULY M., KEISER R., GROSS M.: Multi-scale feature extraction on point-sampled models. In *Proceedings of Eurographics 2003* (2003). 6

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. In *Proceedings of ACM SIGRAPH 2003, Computer Graphics Proceedings* (2003), pp. 641–650. 2, 5, 6

[PZvG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000, Computer Graphics Proceedings* (2000), pp. 335–342. 1

[SD01] STAMMINGER M., DRETTAKIS G.: Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the 12th Eurographics workshop on Rendering* (2001), pp. 151–162. 2

[TRS04] TOBOR I., REUTER P., SCHLICK C.: Multiresolution reconstruction of implicit surfaces with attributes from large unorganized point sets. In *Shape Modeling International 2004, to appear* (2004). 2

[Tur92] TURK G.: Re-tiling polygonal surface. In *Proceedings of ACM SIGGRAPH 92, Computer Graphics Proceedings* (1992). 2

[VPBM01] VLACHOS A., PETERS J., BOYD C., MITCHELL J. L.: Curved pn triangles. In *Proceedings of the 2001 symposium on Interactive 3D graphics* (2001). 4, 5

[WW02] WARREN J., WEIMER H.: *Subdivision methods for geometric design: A constructive approach*. 2002. 2

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings of ACM SIGGRAPH 2001, Computer Graphics Proceedings* (2001), pp. 371–378. 1

[ZRB*04] ZWICKER M., RÄSÄNEN J., BOTSCH M., DACHSBACHER C., PAULY M.: Perspective accurate splatting. In *Graphics Interface 2004, to appear* (2004). 6

[ZS00] ZORIN D., SCHRÖDER P.: Subdivision for modeling and animation. In *SIGGRAPH 2000 Course Notes* (2000). 2

[ZSS96] ZORIN D., SCHRÖDER P., SWELDENS W.: Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of ACM SIGGRAPH 1996, Computer Graphics Proceedings* (1996), pp. 189–192. 2, 7