Efficient Refinement of Dynamic Point Data

B. Solenthaler †, Y. Zhang, R. Pajarola
Visualization and MultiMedia Lab, Department of Informatics, University of Zurich

Abstract
Particle simulations as well as geometric modeling techniques have demonstrated their ability to process and render points interactively. However, real-time particle-based fluid simulations suffer from poor rendering quality due to low surface particle resolutions. Surfaces appear blobby, surface details are lost, and features like edges are degraded due to smoothing effects. This paper presents a novel point refinement method for irregularly sampled, dynamic points coming from a particle-based fluid simulation. Our interpolation algorithm can handle complex geometries including splashes, and at the same time preserves features like edges. Point collisions are avoided resulting in a nearly uniform sampling facilitating surface reconstruction techniques. No point preprocessing is necessary, and point neighborhoods are dynamically updated reducing computation and memory costs. We show that our algorithm can efficiently detect and refine the surface points of a fluid and we demonstrate the improvement of rendering quality and applicability to real-time simulations.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: – Animation and Virtual Reality I.3.5 [Computational Geometry and Object Modeling]: – Curve, surface, solid, and object representations

1. Introduction
Point representations have been used successfully in geometric modeling and in physically based particle systems. The lack of topological and connectivity information simplifies modeling interaction effects (eg. [MKN*04, MHTG05, MSKG05, PKA*05, SSP07]) as well as geometric manipulations (eg. [ZPKG02, AD03, PKKG03, PKG06]). However, it comes at a cost, as neighborhood information has to be computed. Nevertheless, [MCG03] succeeded in interactively simulating and rendering particle-based fluids and demonstrated its applicability to virtual reality simulators and 3D games [MST*04, Age05]. However, due to the real-time constraint, the number of particles has to be low which causes a loss of visual quality. Surface details are smoothed out as a result of surface reconstruction techniques and bumps related to the coarse particle distribution are visible. Using a point splatting approach as rendering technique is also not feasible as under-sampled geometries show artifacts at the silhouette and blur due to large splat radii. It would be desirable to improve the visual quality of low resolution particle simulations while still running at interactive frame rates.

The presented approach is to use an upsampling algorithm on the surface particles to optimize the visual appearance of particle simulations. Such a technique avoids the overhead of running a high resolution physical simulation which is substantial as the number of physical particles increases disproportionally with the desired number of surface particles. Even worse, the Courant condition requires smaller time steps for higher resolutions ( [Mon89]) resulting in a computational effort of physical simulations increasing quadratically with the number of desired surface particles.

For static point geometries, refinement methods have already been demonstrated (e.g. [PGK02, ABCO*03, GBP04, GBP05]). Nevertheless, these methods cannot be applied to points evolved by a Smoothed Particle Hydrodynamics (SPH) simulation [Mon92] due to the real-time constraint as well as the challenging properties of such dynamic SPH particles. The challenges thereby are irregularly distributed particles, low-quality normals, as well as complex surfaces including splashes and isolated particles. Another issue is the robustness of the algorithm used for surface particle detection and upsampling in order to avoid visible artifacts resulting from the dynamic nature of particle simulations.

1.1. Our Contributions
To reveal all details present in (low resolution) particle-based fluid simulations (for interactive and real-time applications), we propose an efficient upsampling method applicable, but

† e-mail: {solenthaler, zhang}@ifi.uzh.ch, pajarola@acm.org

© The Eurographics Association 2007.
not limited to irregularly sampled, dynamic point data. The main features of our method are:

- **Low computational costs**: No point preprocessing is necessary, new points are added efficiently, and neighborhoods are updated dynamically instead of being determined from scratch each refinement step, reducing computation and memory costs.
- **Irregular initial sampling**: SPH particles are often irregularly distributed during the simulation. The refinement procedure effectively copes with this problem, and holes are generated only if indicated by the physics simulation.
- **Uniform sampling after refinement**: Point collisions are detected and avoided resulting in a nearly uniform sampling.
- **Details and sharp features**: Our interpolation yields many details of surfaces and splashes, and preserves features like edges.
- **Splashes and isolated particles**: Although isolated particles and particles in splashes possess low-quality normals, curvature and connectivity are preserved in a plausible way.

1.2. Related Work

Currently, many particle-based fluid simulation models are based on SPH, which was developed in astrophysics [Mon92, Mon05]. SPH has been successfully used in graphics to simulate soft objects [DC96], lava [SAC99], multiple fluids [MSKG05], phase changes [MKN+04, KAG+05, SSPO7] and interaction with solid objects [MKI03, MST+04]. It was also used for fluid control [TKPR06] and with adaptively sized particles [APKG07]. A real-time SPH simulation was demonstrated in [MCG03].

To reconstruct the surface from a set of fluid particles several techniques have been proposed, all without upsampling the surface points of a fluid. An efficient approach is presented in [MCG03] where they render the isosurface of a color field defined by the particles. A grid-based level-set simulation guided by particles is presented in [PTB03], where they succeeded in achieving high visual quality but at the expense of computation time. [ZBO5] presented a reconstruction technique using an implicit function defined on the center of mass of a local neighborhood of the particles leading to very smooth surfaces. Unfortunately, this method suffers from artifacts in concave regions where they propose to remove in a post processing step. In [SSPO7], a method is presented to detect and avoid these errors on the fly. Another extension has been presented in [APKG07], where particle-to-surface distances are used for the reconstruction of surfaces from adaptively sized particles.

In geometry processing, a surface reconstruction based on the use of Radial Basis Functions with global support is presented in [CBC01], whereas [OBA+03, TRS04] reduce the support by local approaches. MLS surface reconstruction [Lev03] has shown to be successful in surface editing [PKKG03], raytracing [AA03], and up- and downsampling [PGK02, ABCO+03]. A sphere fit MLS improving the stability of the projection in low-sampled and curved regions has been presented in [GG07]. However, since the projection procedure of the MLS is quite expensive, it is unsuitable for upsampling a set of points in real-time. Real-time upsampling restricted to static, uniformly sampled point data was presented in [GBP04]. [GBP05] overcomes this weakness and presented a method which is able to fill large holes of static point clouds.

2. Particle-Based Physics Simulation

We briefly describe the particle-based fluid simulation model SPH which we use to demonstrate our upsampling method. SPH is an interpolation method for particle systems, where viscosity and pressure force fields are directly derived from the Navier-Stokes equations. The fluid is discretized using particles which carry field quantities $A$. At any position $r$ in space, these quantities can be evaluated by summing up the weighted contributions of the neighboring particles $p_j$:

$$A(r) = \sum_{j} \frac{m_j}{\rho_j} A_j W(r - r_j, r),$$

where $m_j$ and $\rho_j$ are the mass and the density of particle $p_j$, respectively. The radial symmetric weighting kernel $W(r, r)$ has a finite support determined by the support radius $r$. In our simulation, we use the physics equations and kernel functions presented in [MCG03] and [MSKG05]. For a more detailed description about SPH we refer to [Mon92, Mon05].

3. Refinement

3.1. Surface Particle Detection

Since we are interested in visualizing the fluid surface, we only want to refine surface particles. The detection of free surface particles is a difficult problem and has shown to be a critical step in our refinement procedure as erroneously detected and erroneously undetected particles can lead to surface artifacts. In the particle simulation literature, several methods to detect surface particles have been proposed, often based on the number of neighbors in the support radius. These techniques do not satisfy our needs as particles are partly erroneously detected as surface, especially when high pressure forces are involved. This is ascribed to the irregular particle distribution and therefore to the non-constant number of particles in the neighborhoods.

We propose a method to detect surface particles which is based on the distance from the particle to the normalized center of mass $\mathbf{cm}$ of its neighborhood $N$. This criterion still investigates the particle distribution in the neighborhood of a particle but is independent of the actual number. See Figure 1 for an example. For each particle $p_i$, this distance $d_{i, cm}$ is calculated by

$$d_{i, cm} = \frac{\sum_j (r_i - r_j)m_j}{\sum_j m_j}.$$
A surface particle is defined by having a $d_{cm}$ exceeding a certain threshold. In order to avoid oscillations between being surface and not being surface, we use a slow reaction for surface particles to turn into the state of not being surface. This is achieved by using two different thresholds, where a lower one is used for particles which are already marked as surface. Isolated particles have to be treated separately, where they are defined by having an empty neighborhood.

3.2. Point-Normal Interpolation

3.2.1. Initial Neighborhood

We use the surface particles as initial point set $P_0$ for the refinement procedure (Figure 1). Similar to [GBP04] we insert additional points which yield a new point set $P_1 \supseteq P_0$. The new point set $P_1$ is then used for the next refinement step. This procedure can be repeated until the desired point resolution is reached. During one refinement step, an additional point is inserted between each point $p_i$ and its neighbors $p_j$, where it is avoided to generate points which are too close to points already created in order to achieve a nearly uniform sampling (Section 3.3). Figure 2 illustrates the points of the individual refinement steps. The neighborhood $N^\text{surf}_{p_i}$ of each point $p_i \in P_0$ is inherited from the physics simulation, where $N^\text{surf}_{p_i} \subseteq N_i$; if two surface particles are visible to each other in the physics calculation, meaning that they interact with each other, the pair is refined. Otherwise, if two particles have a distance larger than the support radius and do not interact with each other, no additional point is added. That is, the initial visibility radius $r_0$ used in the refinement procedure is equal to the support radius $r$ used in the physics. Note that $r_0$ is the same for all points. In the following refinement step, the radius is reduced as described in Section 3.2.3.

The point normals are determined either in the physics by using the method proposed in [MCG03] or by using a transformation invariant homogeneous covariance analysis as described in [Paj03]. The latter is more expensive but leads to higher quality normals which is crucial when using point splatting as rendering technique (see Figure 9).

3.2.2. Spherical Interpolation

If a new point is added, its position and normal have to be determined. For real-time applications, it is important that we get these values at low computational costs. Nevertheless, surface details should be preserved as far as possible. While standard Bezier schemes are efficient ([GBP04, GBP05]) they often show insufficient uniformity of upsampled points. We therefore propose a spherical interpolation method which sets a new point (child $c_{1,2}$) onto a sphere which is defined by the two points being refined (parents $p_1$ and $p_2$), as illustrated in Figure 3. The child point is set onto the perpendicular bisector of $d = r_1 - r_2$, where the displacement from $d$ onto the sphere is called $h$. The position of the child $r_c$ is given by

$$r_c = \frac{r_1 + r_2}{2} + h n_c,$$

where $n_c$ is the normal of $c_{1,2}$ as defined in Equation 14.

Each parent computes a displacement $h_1$ and $h_2$, respectively, and the final $h$ is a function of $h_1$ and $h_2$. There are different possibilities to determine $h$, one is to take the average, another one is to take the absolute minimum:

$$h = \frac{h_1 + h_2}{2},$$

$$h = \min(|h_1|, |h_2|).$$

The first one (illustrated in Figure 3) leads to a smooth surface whereas the second one preserves edges and corners more accurately. We prefer the second one and our results are all computed using this approach.

To facilitate the calculation of $h_1$ and $h_2$, we determine the slopes $\frac{s_1}{t_1}$ and $\frac{s_2}{t_2}$. The displacements are given by

$$h_1 = \frac{s_1}{t_1} \frac{||d||}{2},$$

$$h_2 = \frac{s_2}{t_2} \frac{||d||}{2},$$

where $s$ and $t$ are defined by

$$s_1 = ||n_1 \cdot a||,$$

$$t_1 = n_1 \cdot b + ||n_c||,$$

and

$$s_2 = ||n_2 \cdot a||,$$

$$t_2 = n_2 \cdot b + ||n_c||.$$
where $\rho$ is inserted. As we want to reach a uniform sampling after the refinement procedure, it is important to avoid adding points too close to existing points.

3.2.2. Radius in the Next Refinement Step

For efficiency reasons, it is desirable to limit $|h|$ to $h_{\text{max}}$. We choose $h_{\text{max}}$ so that 6 points may still refine to a perfect sphere assuming this is a reasonable requirement to the resolution of an underlying simulation. For an illustration in 2d see Figure 4. In this situation, $h_{\text{max}}$ is given by

$$h_{\text{max}} = \rho - \frac{r_f}{2} = \frac{r_f}{2}(\sqrt{2} - 1),$$

where $\rho$ is the sphere radius. As we know $h_{\text{max}}$, the radius $r_{i+1}$ of the next refinement step is always well-defined, since it is required that the child is just in the neighborhood of its parent to avoid parasitic holes. Therefore, $r_{i+1}$ is defined by

$$r_{i+1} = \sqrt{h_{\text{max}}^2 + \left(\frac{r_f}{2}\right)^2} = \frac{r_f}{2}\sqrt{4 - 2\sqrt{2}}$$

which means that in each refinement step the radius is reduced by a factor of $\approx 0.54$.

3.3. Point Collision Avoidance

As mentioned above, for each pair of points a new point is inserted. As we want to reach a uniform sampling after the refinement procedure, it is important to avoid adding points too close to existing points.

3.3.1. Collision Detection

We use a collision distance $\beta$ which defines a sphere around a point which has to be empty. The collision volume of a child point for the case $h = h_{\text{max}}$ is two-dimensionally illustrated in Figure 5 as red circle. If any other point lies in the same volume the child point is rejected and not added. In each refinement step, $\beta$ is adjusted proportionally to $r_f$. We use $\beta = 0.2r_f$ which we have determined heuristically to be adequate to approach a uniform sampling. There are three different collision configurations, see Figure 6 for an illustration. (for simplicity reasons, we refer to the different refinement steps as $s_0, s_1, s_2, \ldots, s_n$, and a point which is added in step $s_i$ is called $p^{s_i}$):

1. A point created in the current refinement step $s_i$ is closer than $\beta$ to its parents.
2. A point created in the current refinement step $s_i$ is closer than $\beta$ to a (non-parent) point which was created in one of the last refinement steps $s_{j<i}$.
3. A point created in the current refinement step $s_i$ is closer than $\beta$ to another point which was also created in the current step $s_i$.

The first type where a child collides with its parents can be avoided by not refining a pair of points which are closer than $r_{i+1}$. This is valid as long as $\beta < 0.5r_{i+1}$, which is the case when using $\beta = 0.2r_f$. In this situation, the pair gets refined in the next refinement step $s_{i+1}$.

Since both parents know their neighboring points added in $s_{j<i}$ we can avoid collisions of type 2 by taking the intersection set $I$ of both parent neighborhoods $N_1, N_2$, which is $I = N_1 \cap N_2$. As can be seen in Figure 5, the blue region $B$ corresponds to the critical volume which has to be checked when $h$ is not known. This volume encloses the red region $R$ (collision volume) and is enclosed by the intersection set $I$: $R \subseteq B \subseteq I$. Therefore, all collisions of type 2 can be found by distance checking the child with all $p^{s_{j<i}} \in I$. Since the number of points in $I$ is relatively small, the fact that $I$ is larger than $R$ is not very time critical.

$\copyright$ The Eurographics Association 2007.
Collision avoidance algorithm. The red points were added in by using a smaller collision distance and almost all collisions are avoided. Nevertheless, closer than thus $S$ is inside that region (except at points in the green region).

### Figure 6: Refined points without and with using our point collision avoidance algorithm. The red points were added in $s_0$ and the blue points in $s_1$.

However, this is different for the third type, since the number of $p^h$ is growing disproportionally. It turned out that an efficient way to solve type 3 collisions is to register an added child point at all points $p^{ri+1} \in N_1 \cup N_2$ if point is inside that region (except at points in the green region). Dashed black: $R_1, R_2$: Registered point sets of $p_1$ and $p_2$.

### Figure 5: Red: Collision region $R$ for a child point in the case where $h = h_{\text{max}}$. Blue: Critical region $B$ which has to be checked for collisions when $h$ is not known. Dashed yellow: A new point $\psi$ registers at all points $p^{ri+1} \in N_1 \cup N_2$ if point is inside that region (except at points in the green region). Dashed black: $R_1, R_2$: Registered point sets of $p_1$ and $p_2$.

As can be seen in Figure 5, it cannot be guaranteed that all collisions are found. A new child does not register in the green region which is outside of the union $U = N_1 \cup N_2$. But since this volume is very small and is getting smaller or even disappears the smaller $h$ is, this method is a good approximation and almost all collisions are avoided. Nevertheless, by using a smaller collision distance $\beta$ or reducing $h_{\text{max}}$ it is possible to detect and avoid all collisions.

#### 3.3.2. Collision Handling

When we detect that a child point $c$ collides with an existing point $p$ and therefore is rejected, we move $p$ into the direction of $c$ in order to achieve that $p$ is more equally distanced to the nearby points. This is done by averaging the positions and normals:

$\mathbf{r}_p = \frac{w_p \mathbf{r}_p + w_c \mathbf{n}_c}{w_p + w_c}$  \hspace{1cm} (17)

$\mathbf{n}_p = \frac{w_p \mathbf{r}_p + w_c \mathbf{n}_c}{\|w_p \mathbf{r}_p + w_c \mathbf{n}_c\|}$  \hspace{1cm} (18)

where $w$ is a weight which is assigned to a point at the time of creation. We chose this weight to be proportional to the distance from its parents, where a high weight is assigned to all $p \in R_0$ in order to preserve the original surface geometry. After a collision is detected, the weight of the point which is moved in the process is adjusted as well:

$w_p = w_p + w_c$.  \hspace{1cm} (19)

This averaging positively affects the uniformity of the points after the refinement.

#### 3.4. Neighborhood Update

To execute the next refinement step $s_{i+1}$, the new neighborhoods defined by $r_{i+1}$ have to be determined for each existing point. Instead of using a search data structure and recomputing the neighborhoods in each step, we iteratively update the neighbors of each point which is less expensive concerning computation time and memory usage while maintaining correctness.

Neighborhood updates are done simultaneously to the registering described in Section 3.3 as in both steps we have to access all points $p^{ri+1} \in U$. In order to save computation time, the distances which were already calculated in the collision test are reused, and neighborhoods are not updated after the last refinement step. We distinguish between three different neighbor relations (note that these are symmetric since the same $r$ is used for all points):

\[ \text{Figure 7: A point created by two parent points outside } U \text{ are always outside the red region } V. \text{ As } V \text{ fully encloses the child’s neighborhood } N_c \text{ of the next refinement step (blue) all candidate neighbor points are known and neighborhoods can be updated correctly.} \]
1. Two points created in one of the last refinement steps $s_{j<i}$ are neighbors.
2. A point created in one of the last refinement steps $s_{j<i}$ and a point created in the current refinement step $s_i$ are neighbors.
3. Two points created in the current refinement step $s_i$ are neighbors.

The update of type 1 neighborhoods is straightforward: as $N_i^{psi} \subset N_i^{h}$ and the distance to each neighbor is already known from the last refinement step, we can update the neighborhoods cheaply by comparing the distance to $r_{i+1}$.

Neighbor relations of type 2 are generated after having checked a child point for collisions. When a child does not collide with any other point and therefore is accepted, it is added to the neighborhood of all $p^{psi} \in U | d \leq r_{i+1}$, and vice versa.

Additionally, the child has to be added to all points $p^{psi} | d \leq r_{i+1}$ (type 3 relations). Figure 7 illustrates that it is sufficient to know all $p^{psi} \in U$ to find all neighbor candidate points $p^{h}$, as long as each point $p^{psi}$ knows the added children where it was involved as a parent, and therefore correct neighborhoods can be guaranteed. As we know $h_{max}$, $r_{i+1}$, and the maximal distance between two parent points $d_{max} = r_i$, we can show geometrically that it is not possible that two points $p^{h}$ both outside of $U$ are refined yielding a new point which is closer than $r_{i+1}$ to the child $c_{i,2}$. All points created by such parents would lie outside of the red dashed region $V$ illustrated in Figure 7. As can be seen, $V$ completely encloses the neighborhood $N_{i+1}^{c_{i,2}}$ defined by $r_{i+1}$ (illustrated in blue), even in the most extreme case where $h_{child} = h_{max}$: $V \supseteq N_{i+1}^{c_{i,2}}$. This means that all neighbor candidate points $p^{h}$ are known by the child and can be distance checked.

4. Results and Discussion

We have tested our new refinement method on different irregularly sampled point scenes and rendered them either using the raytracing approach presented in [SSP07] or point splatting. All timings are given for an Intel Core2 2.66 GHz.

Three frames of a splashing column simulation consisting of 3k physics particles are shown in Figure 8. During the whole simulation, a point generation rate between 415k and 845k points per second is achieved (Table 1). A simulation sequence running at interactive rates is demonstrated in Figure 9, where the upsamplled fluid with 14k surface points is running at 11fps (41 time steps per second) and the initial fluid with 1k surface points at 17fps (58 time steps per second). These timings include all computational costs (physics, normals, refinement, and visualization).

The effect of upsampling a textured fluid can be seen in Figure 10. Whereas the initial fluid appears blurry, the upsamplled fluid is much sharper and detail-conserving. The introduced smoothing related to the original texture is visu-
5. Conclusion

We have presented a refinement method which is suitable to efficiently improve the visual quality of low resolution particle-based fluids used in interactive and real-time applications. Our algorithm is able to robustly detect and refine surface points, particularly if the input points are irregularly sampled dynamic points coming for example from an SPH fluid simulation. No preprocessing is necessary and due to our fast neighborhood update we can reduce computation and memory costs. Our interpolation method can handle complex surfaces and splashes, and features like edges can be preserved. Point collisions are avoided yielding a nearly uniform point distribution, which also supports surface reconstruction techniques. Our algorithm can generate up to 34k points per frame at 25fps which makes it suitable to apply the technique to the computationally expensive field of fluid simulation. As was shown, our algorithm improves the visual quality of the surface while preserving surface details. As future work, we intend to integrate a more robust normal computation, as the low quality normals currently used adversely affect the reconstructed surfaces. Additionally, we intend to use more sophisticated methods to optimally select the points which are going to be refined in order to reduce the computation costs even further.
References


[Age05] Ageia: Physics, Gameplay and the Physics Processing Unit. White paper, 2005. 1


[Age05] Ageia: Physics, Gameplay and the Physics Processing Unit. White paper, 2005. 1


[ZB05] Zhu Y., Bridson R.: Animating sand as a fluid. ACM Trans. Graph. 24, 3 (2005), 965–972. 2