Interactive Water Streams with Sphere Scan Conversion

Rama Hoetzlein*
Department of Computer Science
Media Arts & Technology Program
University of California Santa Barbara

Tobias Höllerer†
Department of Computer Science
Media Arts & Technology Program
University of California Santa Barbara

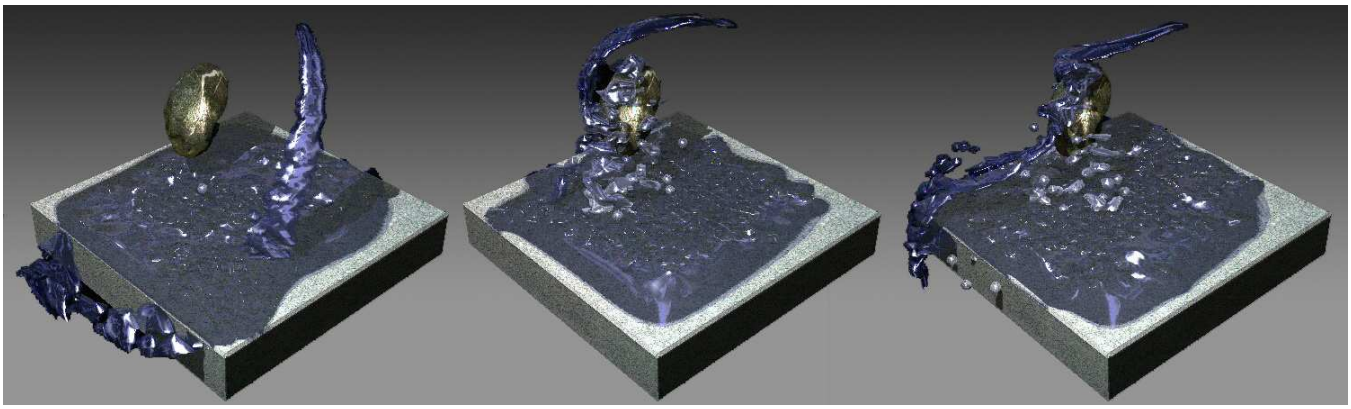

Figure 1: Interactive water simulation of 2500 particles at 75 fps with surface extraction by sphere scan conversion on the CPU and rendering of shadow and environment maps on the GPU.

Abstract

Fluid simulations require efficient dynamics, surface extraction and rendering in order to achieve real time interaction. We present a novel technique for the surface extraction of stream-shaped fluid simulations represented as particles. Typical surface extraction methods for particles combine implicit function evaluation with the marching cubes algorithm. In our approach, we dynamically update vertex positions in pre-generated geometry to efficiently construct and render fluid surfaces. Cylinders are wrapped to water streams composed of particles, with simulation and polygonization on the CPU, and shadows and lighting on the GPU. While limited to stream-shaped fluids, our technique is significantly faster than marching cubes, scales well with resolution and number of particles and, unlike point-based rendering, produces true 3D polygonal surfaces.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1 Introduction

Early efforts in Computational Fluid Dynamics focused on accuracy, stability and performance of simulations. However recent advances in methods, and modern graphics hardware, have allowed for the simulation of fully interactive fluids at interactive rates. The performance bottleneck for real time fluids now shifts from simulation to surface extraction and rendering.

The potential of real time fluid simulation presents a range of novel opportunities in interactive game design, acceleration of film production, and in encouraging new directions for fluid dynamics research in engineering. Stable grid-based techniques have allowed for interactive animation of smoke [Stam 2003]. Free surface flows, for liquids such as water, require additional computational resources to track and polygonize the fluid surface. Smoothed particle hydrodynamics, originally using in astrophysics [Gingold and Monaghan 1977], has become a valuable technique for liquid simulation [Monaghan 1992] [Desbrun and Gascuel 1996].

Recently, GPU implementations of smoothed particle hydrodynamics have permitted a much larger number (up to 60k) of simulated particles in real time [Harada et al. 2007] [Zhang et al. 2007]. In these examples either point-based rendering or marching cubes is used to perform surface extraction offline. While point-based rendering can achieve one million points per second with modern hardware, for dynamic data this requires normals to be re-evaluated per frame, and point-based methods to produce shadow maps [Botsch et al. 2005]. Marching cubes can be efficiently computed on GPUs, but typically from static grid-based data volumes [Tatarchuk et al. 2008]. For realistic interactive simulations to become a reality, our efforts therefore focus on alternatives for direct surface polygonization of point-based fluids in motion.

1.1 Related Work

Early methods for fluid simulation avoided surface extraction with the use of 2D height-field techniques [O'Brien and Hodgins 1995]. Recent applications of height-fields have permitted interactive simulation of ocean waves [Hinsinger et al. 2002] and object interactions [Yuksel et al. 2007]. Other techniques combine height-fields with particles to approximate splashing [Iwasaki et al. 2006]. To correctly render flowing and splashing water, however, requires true

*e-mail: rch@umail.ucsb.edu
†e-mail: holl@cs.ucsb.edu

3D surfaces extracted from fluid motion.

The established technique for surface extraction is marching cubes [Lorensen and Cline 1987] [Bloomenthal 1994], and may be used either for grid-based or particle-based fluids [Müller et al. 2003]. In the later case, the surface is reconstructed by blending implicit functions [Blinn 1982]. The performance of basic marching cubes for particle data sets is O (Pn^3). By evaluating the implicit function only near the surface, the performance can be O(Pn^2). However, run time is still proportional to both the number of particles and grid resolution. Techniques such as the Fast Marching Method take advantage of particle locations to track the surface temporally [Triquet et al. 2001]. Recently, GPU implementations improve performance but do not greatly improve the scalability of the marching cubes to large numbers of particles [Goetz et al. 2005]. This is partly due to the need to generate new topology (i.e. vertex-face connectivity) for every frame.

Our contribution focuses on a novel real-time technique for direct polygonization of point-based data sets by taking advantage of the tubular shape of typical water streams. We present a technique for polygonizing stream-shaped fluids such as water hoses, facuets, and fountains by matching individual streams with pre-generated cylinders. Our method can represent fluids beyond height-field techniques without using marching cubes for surface extraction, allowing for shadow maps, environment maps, and other lighting effects applied to the resulting polygonal surfaces. By taking advantage of the fixed topology of cylinders, we send face connectivity to the GPU once, then create a representational surface by deforming only vertex positions.

2 Methodology

A novel surface extraction technique is presented by fitting and deforming cylinders around stream-shaped fluid volumes. Multiple cylinders are used to wrap independent streams in a fluid simulation. Our technique is motivated by the fact that geometry deformation is faster than generating new topology every frame (as is done with marching cubes), and consists of five basic steps: 1) Pregenerate cylinders, 2) Bin sort particles into slices along a selected V-axis, 3) Group particles into local clusters, 4) Deform cylinders to wrap each group, and 5) Render deformed cylinders. Cylinders were selected to represent the surface as most water streams have a similar overall shape. This technique is therefore best suited to rendering flowing water such as faucets, water hoses, fire hydrants, and fountains, and can be combined with conventional height-field techniques to achieve interactive simulations.

2.1 Sphere Scan Conversion

We begin by pre-generating and discretizing a set of cylinders (Figure 2). Cylinders are pre-tesselated in the U and V directions, and vertex-face connectivity is sent to the GPU for all cylinders just once. The steps of bin sorting, grouping and cylinder deformation occur on every frame (Figure 3). In general, for each group of fluid particles representing a connected stream, we scan the particles as spheres along the V-axis and deform the U-contour of each cross-sectional plane of a cylinder. In Section 2.2 we discuss the grouping algorithm used to isolate fluid streams, and in Section 2.3 we define a radial blending function used to describe the surface contour.

As a pre-processing step, a hash table sort is used to assign fluid particles to cross sectional bins in linear time. Using preallocation for memory management, our implementation requires kP insertions total, where P is the number of particles. For each particle there is some constant overhead (k) due to the fact that a sphere

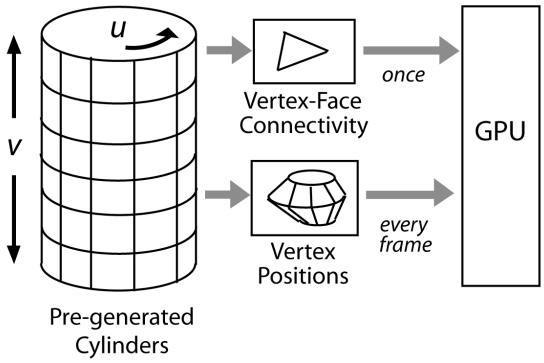

Figure 2: Pre-generated cylinders are discretized with vertex-face connectivity sent to the GPU once. Deformed vertex positions are then sent to the GPU on every per frame.

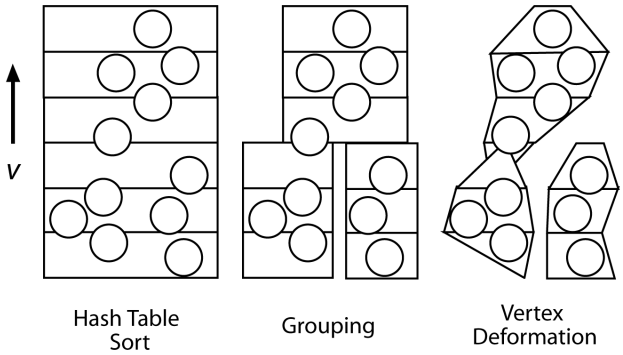

Figure 3: Steps of sphere scan conversion consisting of cylinder generation, hash table sort, grouping, vertex deformation and rendering.

may intersect multiple planes, thus each fluid particle may reside in several bins.

2.2 Grouping

Grouping assigns each fluid particle to a local stream or cluster. Rather than tracking groups temporally, we regroup spheres dynamically for each frame. The maximum number of groups is limited by the number of pre-allocated cylinders. In our evaluations, we used 150 as an upper limit. The radial fountain of Figure 12 uses 94 groups on average over all frames, while the more streamlike simulations average about 35 groups. Distance is the primary critera for group assignment, along with stream shape and coherence.

Grouping is also used to prevent undercuts in fluid streams. An undercut results when a single stream of particles intersects the same cross section twice, the simplest example of which is an arch. A single cylinder, scanning from top to bottom in V, cannot properly capture this situation. Therefore the grouping algorithm is designed to avoid this using a blocking mechanism. An example is provided in Figure 4. We define G_i as the group to which particle p_i is assigned. Input consists of the particles, bins and preallocated cylinders. Output is a set of groups, where each group is assigned to a single cylinder.

Each group data structure maintains minimum and maximum extents in V, particle count, group centroid, and the cylinder mesh associate with that group. As mentioned, groups and meshes are

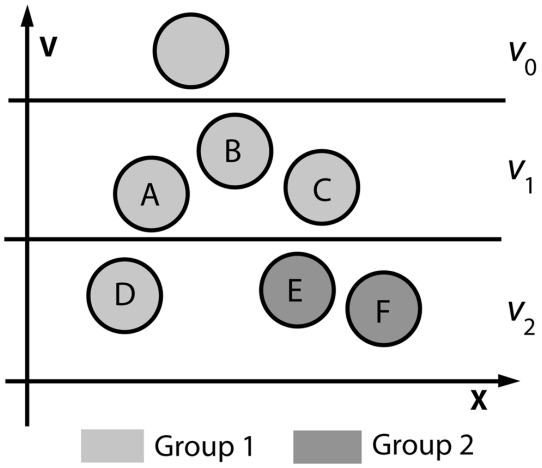

Figure 4: Particle D is the first of bin v_2 so, according to the grouping algorithm, bin v_1 is examined for nearby particles. D is assigned to group 1 due to proximity to A. E is inserted next. E is too far from D, but close enough to C. However, a cylinder cannot be stretched down both paths simultaneously, therefore D has blocked group 1 to prevent undercutting. E is inserted as an isolated particle. Finally, F is added to form a new group with E.

preallocated from a memory pool. Thus, creating a new group in the algorithm above simply involves assigning a unique ID from the pool. Groups do not maintain a list of their particles. Rather, particles are tagged to a group based on the tags of neighboring particles. This allows particles to efficiently regroup each frame. At most, group assignment involves searching for nearby particles in the current and next higher bin per particle.

The pseudocode for the grouping step follows:

```
For each bin v_0 to v_n
  For each p_j in bin v_i
    For each p_k in the current bin v_i.
      If |p_j − p_k| < ε Then
        If p_k is isolated Then
          Create new group wih p_j and p_k. Exit p_k loop.
        If p_k is assigned Then
          Add p_j to group G_k. Exit p_k loop.
    End For
    For each p_k in bin v_{i−1} (bin at next higher z)
      If |p_j − p_k| < ε Then
        If p_k is isolated Then
          Create new group wih p_j and p_k. Exit p_k loop.
        Else p_k is part of group G_k
          If G_k is blocked Then
            Set p_j as isolated. Exit p_k loop.
          Else G_k is open Then
            Add p_j to group G_k, block G_k. Exit p_k loop.
    End For
    Clear all group blocks
  End For
End For
```

Grouping, while fast, is the limiting performance bottleneck. In the worst case, particle P_j must check all particles in the current bin v_i and next bin up, v_{i-1}. Let k be the average number of particles per bin, which is equivalent to P / V_{res} for a uniformly falling stream. Therefore, the total number comparisons required is 2 k P in the worse case, which is O (P^2 / V_{res}). However, once a particle is assigned to a group, which happens as soon as a particle is within an ϵ distance of an existing particle, it can exit the inner loop. Thus, in practice, the average run-time is determined by the threshold distance ϵ and by the coherence of the water stream. See Section 3 for results.

Notice this technique also generates a set of isolated particles. Where marching cubes typically polygonizes isolated spheres at grid resolution, this allows those spheres to be rendered with a static tesselation at a much lower resolution without loss of quality.

Grouping results in discrete boundaries between fluid streams not present in marching cubes. This, and the dependence on the V-axis, are limitations of our technique. However, at interactive rates the boundaries between groups are not noticable. The method described is able to efficiently extract a surprising range of fluid behaviors, with resulting surfaces similar in shape to marching cubes.

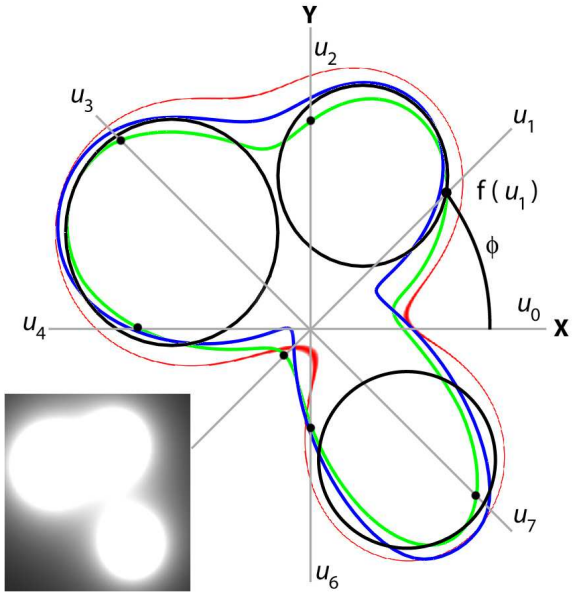

Figure 5: Two functions of theta: smoothed circle maxima (green) and blended gaussians (blue), compared to an implicit surface contour (red) plotted at $f(x, y) = 0.95 \pm 0.01$ for input spheres of different radii. Dots represent evaluation of the circle maxima function f(u_i) at the u-section angles u_i.

2.3 Cross-Section Deformation

To deform a given cross-section of a cylinder, we consider the plane of the section and the particles present in a particular bin. For each cross-section, a sphere defined by each particle will intersect the plane to produce a set of circles. Using the centroid of the circles as the origin, we define a *radial distance function* in polar coordinates (θ, φ) that smoothly approximates the target curve bounding this set of circles. Rather than sample an implicit function at a subset of points in the plane, $f(x, y) \rightarrow \mathbb{R}$, as marching cubes does, we seek a function that directly specifies a parametric curve, $r(\theta) \rightarrow \mathbb{R}^2$. Grouping has eliminated distant spheres, so only those locally associated with the centroid will contribute to the curve. We use the metaballs implicit function as a baseline. The radial function should ideally match this function at a specific threshold. It should be continuous, with smooth transitions based on distance and circle size.

The two functions we experimented with are *smoothed circle*

maxima and *blended gaussians*. The former function computes the maximum of intersections between the unit vectors in the direction of theta $\vec{U}_\theta$ and the circles C_i. The result is discontinuous, however, so to smooth the curve, a 3-point iterative smoothing kernel with between ten and 100 iterations is used.

$$r(\theta) = \max\left[intersect\left(C_i, \vec{U}_\theta\right)\right] \quad (1)$$

An alternative is to use a set of gaussians centered on the circles, with heights h_i given by maximum circle distance and widths w_i based on distance and size:

$$\begin{aligned} gauss_i(\theta) &= h_i\, e^{-w_i\,[\vec{U}_\theta \cdot \vec{C}_{dir}]} \\ h_i &= C_{dist} + C_{radius} \\ w_i &= k\, C_{dist}^2 / C_{radius} \end{aligned} \quad (2)$$

The gaussians cannot be summed, however, as this would produce a radial function that extends well beyond the circle maxima. Instead, the gaussians are blended using a technique similar to linear alpha-blending in 1D.

$$r(\theta) = \text{blend}\,[gauss_i(\theta)] \quad (3)$$

The blend function is listed in Appendix A.

Figure 5 shows these two radial distance functions and an implicit surface plotted at $f(x, y) = 0.95 \pm 0.014$ for comparison. Blended gaussians are slightly more efficient (using lookup tables for the exponential) since there is no need to iteratively smooth the result. Using circle maxima produced more accurate results with better details, but require several iterations to smooth. In our interactive animations we use circle maxima with ten smoothing steps. While metaballs allow spheres to smoothly disconnect, these functions are radially defined so that spheres in the same group remain connected. We resolve this by allowing spheres to move from group to group outside of a fixed distance [see Section 2.2], which can produce discontinuities along the V-axis.

Unlike marching cubes, the radial functions are only evaluated at U discrete points per group. These points map directly to pregenerated parameterized cylinders, with no intermediate geometry. As multiple groups may exist at a given V, we find U_{res} can be usually be much smaller than V_{res} without greatly affecting visual detail.

3 Results

Sphere scan conversion was used to simulate falling water streams with object interactions and splashing. A basic particle simulator was implemented with attention placed on surface extraction rather than fluid dynamics. Our method, as determined in early experiments with SPH, should lend itself well to smoothed particle hydrodynamics of stream-shaped fluids. Comparisons are made to both marching cubes and to live video of falling water.

In order to incorporate real time water into a useful real-world application, such as a game, it will be necessary to carefully balance fluid simulation on the CPU and GPU with other real time demands. To demonstrate our technique, we implement particle motion and surface extraction on the CPU, while leaving the GPU to render shadows and environment maps.

We expect that sphere scan conversion is most likely to be used for water stream effects in games, such as hoses, faucets or fountains with character-water interactions. For large-scale effects marching cubes produces more accurate results, but scalability issues of GPU-based marching cubes make it unlikely to be a viable solution for interactive gaming in the near future.

Particles	T_{ss} (ms)	T_{mbmc} (ms)		F_{ss}	F_{mbmc}	
		64^3	128^3		64^3	128^3
1000	8	410	970	7k	6k	27k
2000	13	2070	3170	9k	14k	31k
3000	18	4560	6580	10k	22k	32k
4000	21	7240	9220	12k	27k	30k
5000	26	19360	22930	13k	32k	35k
6000	28	26690	49340	11k	35k	42k
7000	30	33440	66210	12k	37k	45k
8000	36	40430	120400	13k	38k	48k

Table 1: Performance of our technique is compared to marching cubes generated from a metaballs implicit function per frame. The columns show the number of particles P, surface extraction for sphere scanning in milliseconds T_{ss}, number of faces generated F_{ss}, and metablob-marching cubes surface extraction T_{mbmc} for grid sizes of 64^3 and 128^3, with number of faces generated F_{mbmc}. For sphere scanning, U_{res} and V_{res} are 20 and 120 respectively. Rendering time with vertex buffer objects ranges from 3 to 4 ms for the number of faces generated.

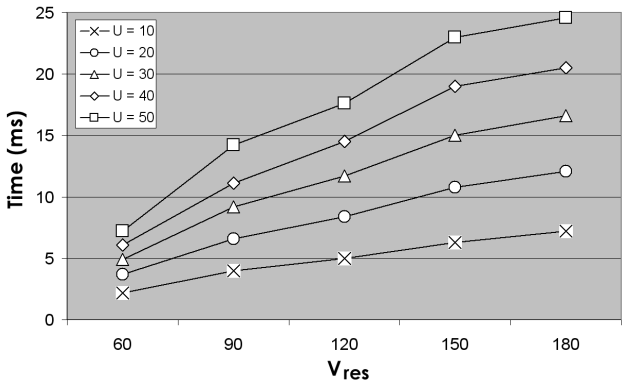

Figure 6: Sphere scan conversion performance with varying U and V. The number of particles is fixed at 2000.

3.1 Comparison to Marching Cubes

We visually compare results from sphere scan conversion and marching cubes and evaluate both for performance. A dynamic implicit function using metaballs at fluid particle locations is evaluated with marching cubes per frame. Our simulations are performed on a P4 3.2 ghz CPU with a GeForce 8800 GTX graphics card running OpenGL.

Performance comparisons of our method are made to a marching cubes implementation by Mizerski [Mizerski 2007], which tracks surface cells using an infinite support kernel for metaball evaluation. Values for T_{mbmc} in Table 1 are therefore an order of magnitude larger than recent work in this field. An infinite support radius, shown in Table 1, requires 2000 ms for 2000 particles. Müller gives a result of 200 ms (5 fps) for 2000 particles using a CPU marching cubes method with a finite support radius [Müller et al. 2003]. Compared to both cases, our method requires only 14 ms for the same number of particles, and is therefore 10x to 100x times faster than marching cubes methods on the CPU.

Scalability is discussed in comparison to modern GPU-based techniques. Tatarchuck performs efficient parallel marching tetrahedra on volumetric data using CUDA, achieving 6.3 ms on a 64^3 grid [Tatarchuk et al. 2008]. As Tatarchuck polygonizes from a static grid-based volume rather than dynamic points, direct com-

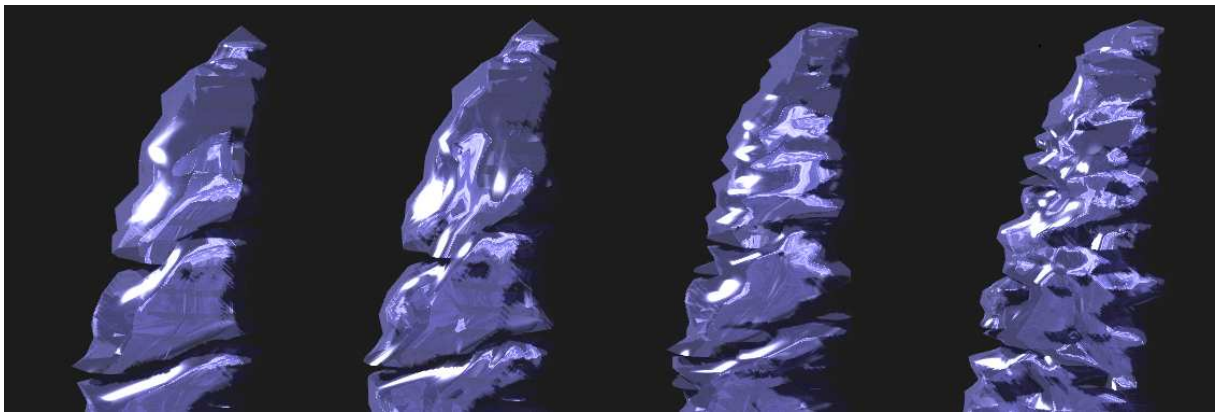

Figure 7: Surface generated for different values of U and V. From left to right, a) U=20, V=60, b) U=60, V=60, c) U=20, V=120, d) U=60, V=120. All surfaces were generated with 2000 particles.

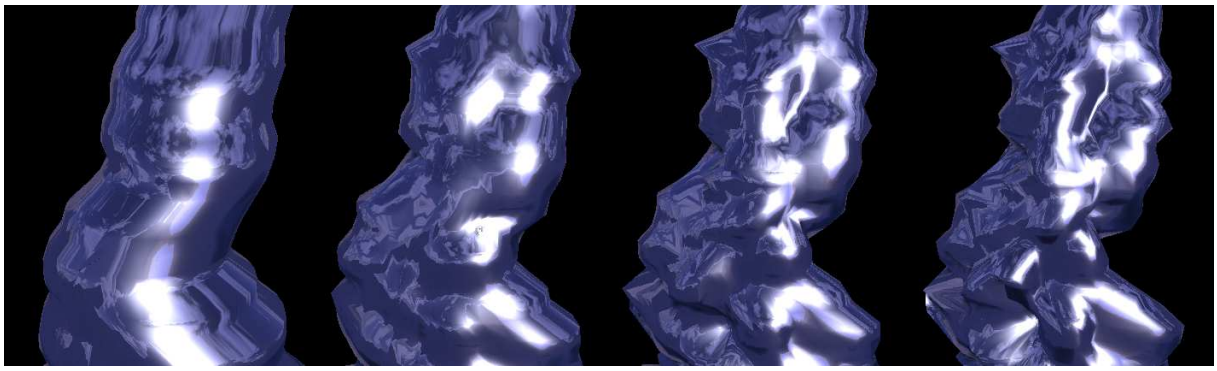

Figure 8: Surfaces generated for various values of the guassian smoothness parameter k. Resolutions of 40 and 180 were used for U and V respectively. The values are 0.1, 1.0, 2.5 and 5.0 from left to right. Results are similar for variations in the smoothing iterations with the blended maxima method.

parisons with scanning spheres in terms of number of particles is not possible. In our application, the input volume must be recomputed per frame using metaballs near the dynamic particle locations, leading to a theoretical run-time of metaball-marching methods of O(Pn^2). An integrated metaball-marching method on the GPU would allow such comparisons.

Zhou et al. use an octree representation with level-order traversal to reconstruct point-based surfaces [Zhou et al. 2007]. They polygonize 300k points at 190 ms with the GPU using an octree data structure and a Laplacian implicit function evaluated using a conjugate gradient solver. While the GPU improves execution times, these methods are theoretically more expensive than the grouping and radial function evaluations of our technique. In addition, to be applicable to fluids represented by smoothed particle hydrodynamics, direct reconstruction methods must be extended to avoid interior particles.

We imagine a complete GPU-based implementation of our method will outperform marching cubes as we show that 3D implicit function evaluation can be an unnecessary intermediate step. In Section 2.2 we found a worst case run-time for scanning spheres as O(P^2 / V). Once a particle has been tagged to a group, however, the next particle may be considered. Thus, the grouping epsilon ϵ and inter-particle distance determine actual run time. Average performance is based on particle coherence, which is typically very high for stream-based fluids. In practice, we observed linear performance relative to the number of particles up to 8000 (Table 1).

Visual comparisons to marching cubes can be found in Figures 10 and 11. The surfaces produced by scan conversion are similar to those generated by marching cubes. While marching cubes blends separating streams continuously, one limitation of our method is that clear boundaries are present between groups. However, these discontinuities occur only along the V-axis, and at interactive rates the differences are usually not noticeable.

Cylinder resolution is the primary parameter in surface quality. Our method simulates 8000 particles at 28 fps (36 ms) with U and V resolution of 20 and 120. Figure 6 demonstrates how scan conversion scales with cylinder resolution using 2000 particles, and Figure 7 gives a visual comparison for selected values of U and V. Other parameters which affect output include grouping distance and radial smoothness. The grouping distance ϵ determines proximity of connected clusters, and we find this is best set to 1.5 to 2 times the sphere diameter. Smaller values result in a simulation with more isolated particles, while larger values cause undesirable bridges to form between groups. The tension parameter *k* for gaussian blending, or smoothing iterations *I* for circle intersections, adjusts the smoothness of the radial distance function resulting in more or less "blobby" looking surfaces (Figure 8).

Unlike marching cubes, which is bounded on all three axes, scan conversion is unbounded in the horizontal plane (perpedicular to V-axis). This may be particularly useful in certain situations, such as water interactions with game characters in open spaces. However, we notice that scan conversion is least successful when many

particle lies in the same cross sectional plane, in which case there is insufficient horizontal resolution to capture details. Thus our current implementation is ideally suited to flowing water streams with a dominant axis rather than resting water. It should be a straightforward extension to locally orient the V-axis of the cylinder coordinate system to the real-world dominant axis of the water stream, providing better support for horizontal streams such as fire hydrants.

As particles splash and scatter they frequently become isolated. In this common situation, marching cubes will polygonize each sphere at grid resolution while our system identifies isolated particles and renders them using pre-tesselated sphere primitives.

Live video of falling water streams was also obtained. Water from a garden hose was filmed at night with a single light source, and also in sunlight at 24 fps. We found that falling water in sunlight casts sharp shadows. In these conditions the contrast of the shadow is often more visible than the water itself. Scan conversion produces closed 3D surfaces, lending themselves to shadows, caustics and Z-buffering. Shadow and environment maps are present in all our real time examples since the GPU is free to take on these computations in real time.

Resting water in our examples, such as pools or lakes, are simulated using common height-field techniques [Nishidate and Nikishkov 2005]. Interactions between falling streams and these surfaces are simulated by checking particle position against the height-field and producing an impulse force in the 2D wave equations as needed. While splashes cannot be generated from the pool, this allows water streams to flow and splash into the pool.

Comparisons with live video can be found in Figure 13. The higher detail of real water is apparent when the stream splashes. The motion of specular highlights with our technique appears to match live water very well during laminar flow.

3.2 Rendering

Rendering is efficient and straightforward. The vertex-face connectivity for each cylinder is constructed in the pre-allocation stage and transmitted to the GPU once. Only vertex positions within the Z-min and Z-max extents of each group are updated per frame. These extents are also used to specify the start and end face of the polygon buffer to be rendered. While somewhat wasteful of memory, as the cylinder is predefined over a large Z domain, this improves performance significantly. Vertex normals are updated per frame by taking cross products of edges and averaging among multiple faces. In the future, bin sort, cylinder distortion and normal updates may be GPU accelerated.

Unlike point-based or surface splatting techniques our method generates closed, 3D surfaces which can be used for advanced rendering such as caustics, shadows and environment maps. Modern point-based methods can achieve similar effects at the expense of GPU fill rate. Botsch el. al. render over one million particles at 62 ms (16 fps) with phong shading and shadows [Botsch et al. 2005]. However, this is achieved with pre-computed gradient buffers on static data sets. For dynamic point sets it is necessary to recompute normals per particle using the local neighborhood of each particle. To our knowledge, this has only been done with NVIDIA's recent PhysX fluid demo, with 60k particles at 25 fps without shadows on the GPU.

4 Conclusions and Future Work

We have presented a novel technique for the surface polygonization of point-based data sets without the use of marching cubes. Sphere scan conversion scales well with surface resolution and number of particles, and is able to polygonize stream-shaped fluids at interactive rates using the CPU. A complete solution would implement both smoothed particle hydrodynamics and sphere scan conversion on the GPU. This would ideally be compared to a GPU implementation of marching cubes, with timing measurements that separate fluid dynamics from implicit function evaluation and polygonization. Overall, we expect scanning spheres to perform well when implemented on the GPU.

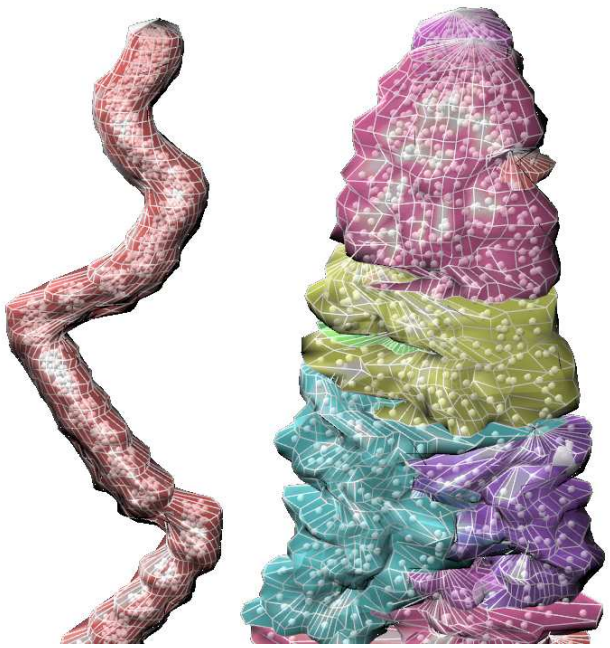

Figure 9: Generated surface mesh for narrow laminar flow (left) and a broad fountain (right), showing mesh geometry, and underlying particles as spheres. Color indicates grouping. Continuous streams are represented with a single cylinder. Simulated with 2000 particles.

The most significant limitation of our technique is the presence of a dominant axis. We are making promising first steps in transferring the main insights from our algorithm to a more general SPH-based water volume visualization in order to render arbitrary surfaces without a preferential axis.

Of the five steps in our technique, only grouping is not easily adapted to GPU parallelism. This could be investigated further. Rendering is already GPU-optimized by making use of vertex buffer objects, and cylinder deformation may be parallelized by independently deforming cylinder v-sections using vertex shaders. In the current system, we transmit only updated vertex position information to the GPU while face connectivity remains static. In the future, geometry shaders may allow adaptive resolution of the surface.

With increasingly powerful hardware, the possibility of real time fluid simulation is becoming a reality. While advances are being made rapidly in fluid dynamics, with the present technique we hope to encourage novel solutions to surface extraction and rendering as well. To our knowledge, our method presents the first real-time rendering of water streams at interactive rates with shadows, environment mapping and interactive control of the stream.

Acknowledgments

Thanks to Zachary Carter for particle system and collision detection. This research was funded in part by an NSF IGERT grant on Interactive Digital Multimedia (#DGE-0221713).

A Gaussian Blending

Gaussian blending is designed to produce a contour that approximates the maximum extents of the input circles in polar coordinates. Summation cannot be used, as this would greatly exceed the maxima, so we adopt a 1D blending strategy based on 2D alpha-blending.

For each circle C_i
  For each θ
    $gauss_i(\theta) = e^{-w_i\,[\vec{U}_\theta \cdot \vec{C}_{dir}]}$
    $gdist_i(\theta) = gauss_i(\theta)\,(C_{dist} + C_{radius})$
    If $gdist_i(\theta) > r_{i-1}(\theta)$ then
      $b = (gdist_i(\theta) - r_{i-1})/gdist_i(\theta)$
      $a = \alpha_{i-1}(1-b) + (1 - gauss_i(\theta))b$
    Else
      $a = \alpha_{i-1}$
    End if
    $r_i(\theta) = r_{i-1}(\theta)a + gdist_i(\theta)(1-a)$
    $\alpha_i(\theta) = max(\alpha_{i-1}(\theta) + gauss_i(\theta), 1)$
  End for
End for

At the gaussian peak, for a circle beyond the previous maxima, the function $r_i(\theta)$ is extended to the new maxima (now the farthest circle). If the circle is less than the previous maxima, $\alpha = \alpha_{i-1} = 1.0$ at the peak point, so the function $r_i(\theta)$ takes on the previous maxima $r_{i-1}(\theta)$. At intermediate points, the circle is blended into the contour without exceeding the maximum circle extents.

References

BLINN, J. F. 1982. A generalization of algebraic surface drawing. *ACM Trans. Graph. 1*, 3, 235–256.

BLOOMENTHAL, J. 1994. An implicit surface polygonizer. In *Graphics Gems IV*, P. Heckbert, Ed. Academic Press, Boston, 324–349.

BOTSCH, M., HORNUNG, A., ZWICKER, M., AND KOBBELT, L. 2005. High-quality surface splatting on today's gpus. In *Eurographics Symposium on Point-Based Graphics*, IEEE/Eurographics.

DESBRUN, M., AND GASCUEL, M.-P. 1996. Smoothed particles : A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation '96*, 61–76.

GINGOLD, R. A., AND MONAGHAN, J. J. 1977. Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society 181* (Nov.), 375–389.

GOETZ, F., JUNKLEWITZ, T., AND DOMIK, G. 2005. Real-time marching cubes on the vertex shader. In *Eurographics 2005 Short Presentations*.

HARADA, T., TANAKA, M., KOSHIZUKA, S., AND KAWAGUCHI, Y. 2007. Real-time particle-based simulation on GPUs. In *SIGGRAPH '07: ACM SIGGRAPH 2007 posters*, ACM Press, New York, NY, USA, 52.

HINSINGER, D., NEYRET, F., AND CANI, M.-P. 2002. Interactive animation of ocean waves. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*.

IWASAKI, K., ONO, K., DOBASHI, Y., AND NISHITA, T. 2006. Point-based rendering of water surfaces and splashes simulated by particle-based simulation. In *Proceedings of NICOGRAPH 2006*.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 163–169.

MIZERSKI, M., 2007. Marching cubes implementation. http://www.math.ubc.ca/~mizerski/, accessed Sept. 2007.

MONAGHAN, J. J. 1992. Smoothed particle hydrodynamics. In *Annu. Rev. Astron. Astrophysics*, 30:543.

MULLER, H., AND STARK, M., 1993. Adaptive generation of surfaces in volume data.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 154–159.

NISHIDATE, Y., AND NIKISHKOV, G. 2005. Fast water animation using the wave equaton with damping. In *International Conference on Computational Science*, vol. 2, 232–239.

O'BRIEN, J. F., AND HODGINS, J. K. 1995. Dynamic simulation of splashing fluids. In *Computer Animation '95*, 198–205.

STAM, J. 2003. Real-time fluid dynamics for games. In *J. Stam. Real-time fluid dynamics for games. Proceedings of the Game Developer Conference, March 2003.*

TATARCHUK, N., SHOPF, J., AND DECORO, C. 2008. Advanced interactive medical visualization on the gpu. *J. Parallel Distrib. Comput. 68*, 10, 1319–1328.

TRIQUET, F., MESEURE, P., AND CHAILLOU, C. 2001. Fast polygonization of implicit surfaces. *WSCG'2001 (Plzen, Czech Republic) 2* (feb), 283–290. http://www.wscg.zcu.cz.

YUKSEL, C., HOUSE, D. H., AND KEYSER, J. 2007. Implementing wave particles for real-time water waves with object interaction. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, ACM Press, New York, NY, USA, 14.

ZHANG, Y., SOLENTHALER, B., AND PAJAROLA, R. 2007. GPU accelerated SPH particle simulation and rendering. In *SIGGRAPH '07: ACM SIGGRAPH 2007 posters*, ACM Press, New York, NY, USA, 9.

ZHOU, K., GONG, M., HUANG, X., AND GUO, B. 2007. Highly parallel surface reconstruction. In *Microsoft Technical Report, MSR-TR-2008-53*.

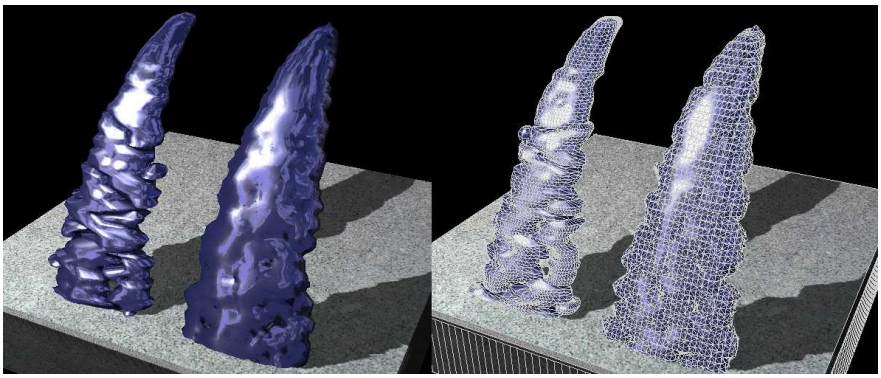

Figure 10: Comparison of sphere scan conversion (left) to marching cubes (right) with 2000 particles, a grid size of 180^3, and cylinder resolution of U=20 and V=180. Group boundaries are visible in still images, but hardly noticable at 75 fps. Opaque shading is used to emphasize details.

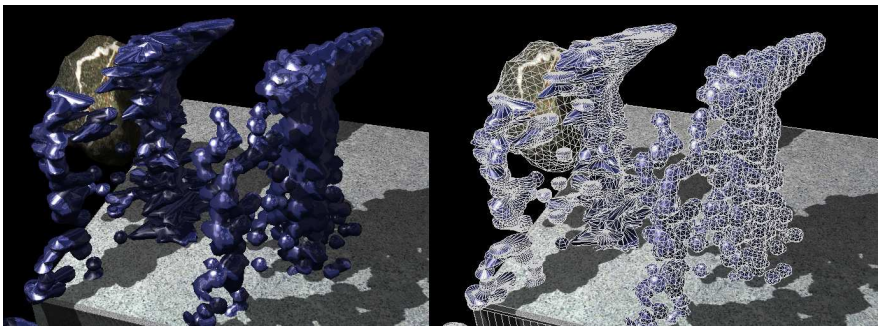

Figure 11: A more complex comparsion showing object interaction and undercut surfaces.

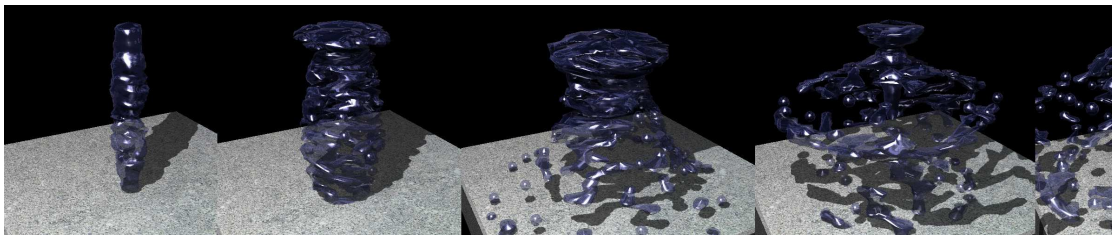

Figure 12: Simulation of a radial fountain using sphere scan conversion with varying flow velocity and 2000 particles. Images were rendered with transparency, shadows and environment mapping. Notice that isolate spheres are detected and rendering individually.

Figure 13: Comparison of live video (left frames) to sphere scan conversion (right frames)) of a water stream. Simulations run at 75 fps with 2000 particles. Custom GPU shaders were developed to match the lighting conditions. Frame rate was found to be a significant factor in the percieved realism of the simulation.