Adaptive Surface Reconstruction for SPH using 3-Level Uniform Grids

Gizem Akinci Nadir Akinci Edgar Oswald Matthias Teschner
gakinci,nakinci,oswald,teschner@informatik.uni-freiburg.de
University of Freiburg
Georges Koehler Allee 052
79110 Freiburg Germany

ABSTRACT

The marching cubes algorithm is a popular method for constructing surfaces from SPH data sets. In order to preserve all of the surface details in high curvature regions and to prevent potential temporal coherence artifacts, the resolution of the underlying uniform MC grid should be set up sufficiently high. However, this requirement unnecessarily increases the resolution in relatively flat regions where the surface can be constructed with lower resolutions without changing the quality. Accordingly, excessive number of triangles are generated, the memory consumption increases dramatically, and the performance decreases. In this paper, we present a 3-level grid structure which adapts its cells according to the curvature of the fluid surface. In contrast to widely-used octrees, we propose a simple to construct yet efficient hierarchical uniform grid structure. Mesh blocks from different resolution cells are seamlessly stitched by closing cracks with new triangles which establish only 0.15% to 0.6% of overall number of triangles in average. Experiments show that in contrast to the single level low resolution uniform grid approach, the presented method reconstructs fine details properly with a comparable performance; while it produces similar results with less number of triangles, up to four times better memory consumption and up to 60% better performance when compared to the single level high resolution uniform grid approach.

Keywords

particle-based fluids, surface reconstruction, marching cubes, multi-level uniform grids, surface curvature, cracks

1 INTRODUCTION

Generating triangle meshes using the marching cubes algorithm (MC) [LC87] is a common approach for both static point clouds and dynamic particle data. However, the chosen grid resolution restricts the user since surface details are reconstructed properly only by using high resolution grids at the expense of performance, memory footprint and storage. This issue has prompted many researchers to investigate adaptive mesh refinement techniques, e.g. by using octrees [WvG92, SFYC96, WKE99, VT01, LY04, JU06, Man10]. Although being efficient in terms of memory consumption and storage reduction, the construction of adaptive structures is not straightforward, and they lead to cracks in between different resolution cells which need to be handled carefully. The use of uniform grids, on the contrary, is motivated by the simplicity of the structure which is advantageous for especially dynamic scenes since it allows fast rebuilding of the data structure. To the best of our knowledge, most of the presented adaptive approaches focus on static scenes, e.g. medical visualizations or CAD models, and there are only few researchers who aim to efficiently rebuild those data structures for dynamic scenes, e.g. [ZGHG11].

Our contribution. In this paper, we present a memory efficient and performance friendly multi-level uniform grid structure for Smoothed Particle Hydrodynamics (SPH) surface reconstructions which allows for fast rebuild in dynamic scenes.

In our technique, the particle data set is covered by an axis aligned bounding box (AABB) which is initially subdivided uniformly with coarse (level-1) cells. These cells are utilized to extract the narrow-band region where the surface is actually defined. Depending on the surface curvature, coarse surface cells are subdivided by one (level-2) or two more levels (level-3). This allows for preserving the surface details even on highly turbulent, high curvature parts by using less triangles, since relatively flat parts are treated as level-2 parts. According to our experiments, the explained data structure is sufficient for a proper surface reconstruction of any fluid data set; and the fourth level is not necessary since it does not improve the quality further but

 ISBN 978-80-86943-74-9

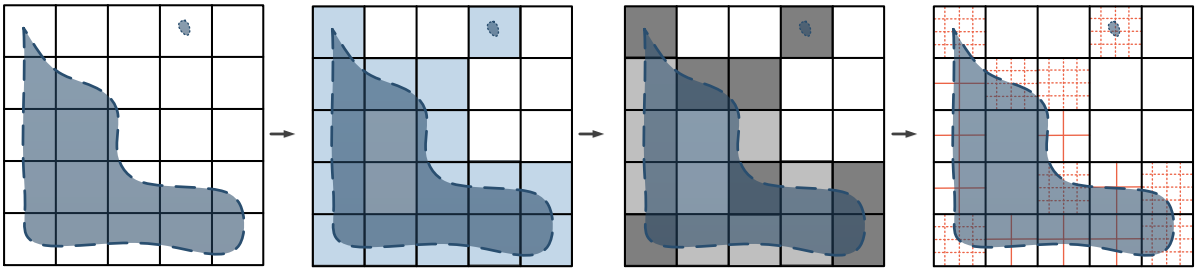


Figure 1: The 3-level adaptive grid. (a) The bounding box of the fluid is used to generate the coarse level uniform grid (level-1). (b) Cells that contain the fluid surface (dashed blue line) are extracted. These surface cells are shown in blue color. (c) Surface cells with low surface curvature are marked as level-2 (light gray cells) and the other surface cells are marked as level-3 (dark gray cells). Cells around the isolated pieces are always considered as level-3. (d) Level-2 cells are subdivided by one more level (straight red lines) while level-3 cells are subdivided by two more levels (dashed red lines).

reduces the performance significantly. The described three levels are illustrated in Fig. 1. We close the arisen cracks eventually by creating new triangles in between different resolution mesh blocks.

Experiments show that when compared to high resolution single level uniform grids, the presented method produces similar results with up to four times better memory consumption and up to 60% better computation time. Two different test scenarios are discussed in Sec. 4, where the computation time and memory consumption data are given for all scenes.

2 RELATED WORK

There exist various approaches that address the visualization of surfaces for fluids or unorganized point data sets that can also be applied to particle based fluids, e.g. implicit surface tracking [WH94], explicit surface tracking [Mul09, BB09], surface splatting [ZPvBG01, ALD06, vdLGS09], screen space meshes [MSD07], rendering using raycasting [MSD07, FAW10, GSSP10] or surface generation using voronoi diagrams [RS09]. Our approach contributes to the field of generating enclosed triangulated fluid surfaces using the marching cubes approach with an adaptive mesh generation technique.

One way to generate an adaptive mesh is to use octree structure which was addressed by many researchers, e.g. [WvG92, SFYC96, WKE99, VT01, LY04, JU06, Man10]. While being very efficient in terms of memory consumption and storage reduction, the construction of octrees is time consuming. Dynamic update of octrees, e.g. shrinking or enlarging the grid and updating child cells, in particular can be time consuming, as one can probably end up with total rebuild of the octree; which makes them less feasible for dynamic scenes. However, we aim a data structure which allows for a fast total rebuild. Furthermore, random access to octree cells usually take logarithmic time, which causes additional overhead. In addition, octrees produce many different resolution cells which means that two leaf cells may differ more than one level. In such a case, crack handling gets difficult, which is not an issue for our data structure. A similar discussion can be found in [Bri03] where Bridson proposes an alternative grid-based method that focuses on the narrow-band region by using only one level of detail.

As mentioned earlier, if levels of two neighboring cells differ, cracks arise in the corresponding transition faces. There exist different approaches which address this problem. Using simple crack patching [SFYC96, VKSM04], points that reside on the high resolution edge of one cell are projected on the low resolution edge of the neighboring cell. However, this technique produces T-vertices which may lead to visual artifacts during rendering. Filling cracks with new triangles is another popular method for handling cracks. Westermann et al. [WKE99] proposed a method where cracks are fixed by replacing coarse triangles by fans of triangles. Ju and Udeshi [JU06] prevented cracks by adding new polygons using a hybrid method that uses the marching cubes and dual contouring. It is stated in [JU06] that the mesh size can get too large by using this approach after newly added triangles. Later, Lengyel [Len10] presented transition cells method where new cells are added in between two different resolution cells for generating new triangles. The newly generated transition cells have to be checked for many cases which is a time consuming process.

Although conventional adaptive structures reduce the storage requirements efficiently, more effective results can be obtained by incorporating hashing. However, this is a challenging task and only few approaches have been proposed so far which address either hashing of octrees [Sigg06] or hashing of multi-level grids for ray-tracing, e.g. [CPJ10, LD08]. We also get assistance from hash maps for keeping our newly generated fine resolution grid points in each coarse cell (see Sec. 3.2),

 ISBN 978-80-86943-74-9

while we leave our top level coarse grid un-hashed for a straightforward implementation.

Memory footprint can be reduced by also applying dynamic tubular grids (DT-grid) [NM04] or run-length encoding (RLE) [HWB04] schemes, which are rather challenging to implement in comparison to alternative grid- and tree-based data structures.

Parallelization of the surface reconstruction algorithm is another topic which has been gaining attention in recent years. Zhou et al. [ZGHG11] proposed a parallelization technique for octrees which runs entirely on GPU. Later, Akinci et al. [AIAT12] presented a parallel method which reconstructs the surface of particle-based fluids in the narrow-band region and runs either on CPU or GPU. Memory footprint is still an open issue to be improved in this method since it sticks to single level uniform grids. While being very efficient in terms of performance, the incorporation of hashing in both approaches is challenging due to the potential thread safety issues.

The quality of the reconstructed surface depends not only on the resolution of the underlying grid but also on the preferred scalar field computation method. Within the context of these techniques, Zhu and Bridson [ZB05] proposed the signed distance field approach which alleviates former bumpiness issues but suffers from artifacts in concave regions. Solenthaler et al. addressed this issue in [SSP07]. In [YT10], Yu and Turk presented an anisotropic kernel approach which results in high quality, less bumpy surfaces while being computationally expensive. Recently, Bhattacharya et al. [BGB11] proposed a surface reconstruction technique which is based on a level set method. This method outputs surface approximations and performs smoothing steps that finally generate rough surfaces which causes to lose the details of the input particle set. With these issues in mind and based on the comparisons given in [AIAT12] and [AAIT12], we decided to use the method of Solenthaler et al. [SSP07] in all of our test scenes.

To the best of our knowledge, the efficient implementation of adaptive structures for particle-based fluids is limited with the work of Zhou et al. [ZGHG11] which uses octrees unlike our method. In this paper we present a method which allows for fast rebuild of the grid and is suitable for dynamic scenes in terms of both memory efficiency and performance.

3 SURFACE RECONSTRUCTION USING 3-LEVEL GRIDS

In this section, we present an adaptive surface reconstruction method for SPH using 3-level uniform grid.

According to our experiments, in order to obtain proper surface reconstructions, MC grid cell size should not be larger than $2r$, with r being the radius of the fluid particles, i.e. the half of the particles' equilibrium distance. However, even $2r$ cell size can be insufficient to preserve all of the surface details. In such a case, experiments show that the cell size of r preserves all surface features appropriately. However, using such a small cell size throughout the whole scene causes a trade-off between the surface quality and the performance-memory consumption. Besides, it is clear that on relatively flat regions, the cell size of $2r$ is already sufficient for obtaining proper results. Therefore, we reconstruct the fluid surfaces using these two resolutions in different regions depending on the surface details.

We initialize our surface reconstruction steps by extracting the narrow-band region where the surface is actually defined. Performing this operation does not require a high resolution grid. We initially create our grid using the cell size of $4r$, extract the narrow-band region using the coarse cells in this resolution, and subdivide the found surface cells using the surface curvature information.

We extract the surface cells using the method of Akinci et al. [AIAT12]. We refer the reader to [AIAT12] for details, however, we briefly outline the steps here. According to this method, any grid point that is in the proximity of a surface particle is defined as a surface point. Therefore, we traverse through all surface particles, define an AABB which spans $4r$ length in x, y, z directions around each surface particle and mark all grid points that reside in this AABB as surface points. Each surface point is used to mark the corresponding cell whose lower left corner is initiated in this point as a surface cell. Different from [AIAT12], we mark the cells as splash cells instead of surface cells if they are in the neighborhood of splash particles, i.e. particles whose number of neighbors are less than a pre-defined value which is 3 in our test scenes. By doing this, we are easily able to exclude such cells from the curvature computation in future steps and gain performance since they are directly subdivided as level-3 high resolution cells.

Surface particle extraction is performed in the preprocessing step using the smoothed color field method of Muller et al. [MCG03]. The same method is also used to compute particles' normals which are required for the curvature computation in the following step (see Sec. 3.1). After refining the cells, the scalar field is computed over all grid cells (see Sec. 3.2), the cracks are handled by adding new triangles and different resolution meshes are stitched seamlessly (see Sec. 3.3).

3.1 Curvature Computation

According to our criterion, a coarse surface cell can be subdivided as either level-2 or level-3 which depends on the surface curvature inside the cell. We approximate the surface curvature using the method described

 ISBN 978-80-86943-74-9

in [IAAT12]. Hence, we firstly compute the curvature for each surface particle that resides in this cell with the help of its neighboring particles as:

$$\kappa_i = \sum_j \kappa_{ij} = \sum_j (1 - \hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j) \mathrm{W}(\mathbf{x}_{ij}, h) \qquad (1)$$

where j stands for the neighboring particles, $\hat{\mathbf{n}}$ is the normal of any particle and W is the kernel function described as:

$$\mathrm{W}(\mathbf{x}_{ij}, h) = \begin{cases} 1 - \|\mathbf{x}_{ij}\| / h & \text{if } \|\mathbf{x}_{ij}\| \leq h \\ 0 & \text{else} \end{cases} \qquad (2)$$

with $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$.

Finally, the curvature of any surface cell is approximated as:

$$\kappa_{cell} = \left(\sum_{\text{for all } i} \kappa_i \right) / N \qquad (3)$$

where i and N represent the surface particles and the total number of surface particles inside the cell, respectively.

If the curvature is smaller than the predefined threshold, we mark the cell as low resolution level-2 cell. Otherwise, the cell is marked as high resolution level-3 cell. In the remainder of the paper, we will call any coarse surface cell as level-2 or level-3 cell depending on its refinement level.

3.2 Scalar Field Computation

We initialize the second step of our method with cell refinement. Therefore, we traverse through all coarse surface cells. If the cell is marked as a level-2 cell, we generate 8 new cells in the coarse cell which correspond to 27 new grid points. A similar approach is followed for level-3 cells with 64 new cells and 125 new grid points. A 2-dimensional illustration of this refinement is shown in Fig. 2.

At this part of our implementation, we need a data structure to keep our newly generated grid points. This structure should support easy access to the required grid point. For this aim, we compute an *id* for each grid point and use hash map structure which allows for easy data access by querying the *id*s that are the keys of the hash map. This structure also stores the corresponding scalar values as the data part of each entry. The *id* of a grid point is computed using its position $\mathbf{GridPos_{l-3}} = (gpx, gpy, gpz)$ and number of potential grid points X_{l-3} and XY_{l-3} in x- and xy-directions, respectively, in a virtual, high resolution level-3 grid throughout the whole scene. $\mathbf{GridPos_{l-3}}$ can be written as:

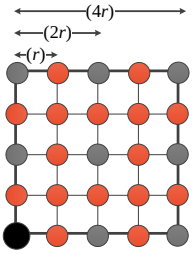


Figure 2: A 2-dimensional illustration of the cell refinement. The thick black line shows the outline of the coarse surface cell whose initial coarse point is colored in black. The gray circles demonstrate level-2 points and both gray and red circles demonstrate level-3 points that are added after the cell refinement.

$$\mathbf{GridPos_{l-3}} = 4 \cdot \mathbf{vc} + \mathbf{offset} \qquad (4)$$

where **vc** is the position of the coarse initial grid point of the cell in the coarse grid. The 3-dimensional $\mathbf{offset} = (a, b, c)$ vector of any new grid point describes the position of this point in the cell. a, b and c values are defined within the range of [0,5). Thus, these values start with 0 and depending on the refinement level, they increase by 1 or 2 for the high resolution and low resolution cells, respectively (see Fig. 3-(a)). Furthermore, X_{l-3} and XY_{l-3} are computed as:

$$\begin{aligned} X_{l-3} &= 4 \cdot ncx + 1 \\ Y_{l-3} &= 4 \cdot ncy + 1 \\ XY_{l-3} &= X_{l-3} \cdot Y_{l-3} \end{aligned} \qquad (5)$$

with *ncx* and *ncy* being the number of cells in x and y directions in our coarse grid. Finally, the *id* of a grid point is computed as:

$$id = gpx + X_{l-3} \cdot gpy + XY_{l-3} \cdot gpz \qquad (6)$$

After computing the *id* of any grid point, we check whether it is already stored in the hash map. In such a case, we do not compute the scalar value since this information is already kept together with the *id*. Otherwise, we compute the scalar value of the grid point using the improved signed distance field approach of Solenthaler et al. [SSP07]. At this step, it is important to identify the particles which contribute to the scalar value computation of each grid point. Particles within the influence radius of each coarse grid point can be easily retrieved by using the method of Akinci et al. [AIAT12]. However, since the influence region is changed for the newly added fine points, we check all the particles that lie also in the influence region of the final grid point in the cell, i.e. the fine point with **offset**(4,4,4), which is shown with yellow color in Fig. 3. The computed scalar value can now be stored in our hash map with its corresponding *id*.

 ISBN 978-80-86943-74-9

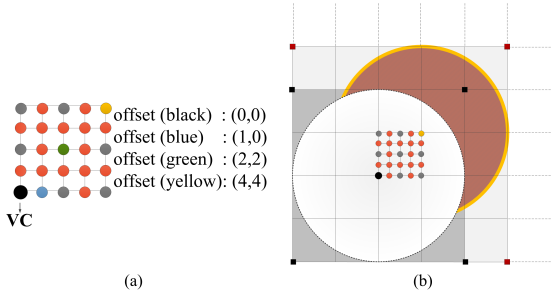


Figure 3: (a) The coarse initial grid point with position **vc** is illustrated with a black circle. Offset values for some fine grid points are given in 2-D which is similar for the 3-D version. (b) The white, large circle shows the influence region of the coarse initial surface grid point. However, this region is not sufficient for newly added fine grid points. The red region which is the influence region of the yellow point in the figure should also be checked for influencing particles.

3.3 Triangulation

Triangulation is the most challenging part of our method, since we need a special treatment for handling the cracks which arise in between different resolution cells. As mentioned before, various crack handling techniques have been proposed in the literature. Many of these methods, however, are either computationally expensive, or produce T-vertices which cause visual artifacts, or produce large number of triangles and cause memory inefficiency. Therefore, we present a method for an efficient crack closing. We perform this step gradually by ensuring the scalar field continuity first and covering the cracks with new triangles afterwards.

3.3.1 Scalar Field Continuity

The scalar field that is computed over our multi level grid is not continuous, i.e. the scalar field continuity is broken on the faces of neighboring different resolution cells. Therefore, similar to [OR97], we adjust the values of intermediate grid points on such transition cell faces by sub-sampling the original data (see Fig. 4). So as to carry out this task in our implementation, we traverse through all level-2 cells and identify the cells that have a level-3 cell neighbor. If any level-2 cell meets this condition, we adjust the values in the corresponding face.

Even though this approach alleviates the scalar field discontinuity, cracks can still occur due to the numerical sampling problems which are generally observed after the computations around the middle grid point of the face, e.g. the pink grid point in Fig. 4. Those cracks are covered with triangles in our method which is explained in the following section.

3.3.2 Crack Problem

At the final step of our method, we close the cracks with new triangles for a proper visualization.

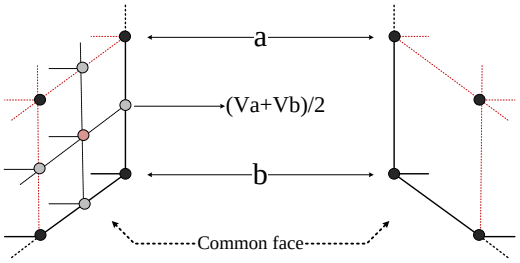


Figure 4: The scalar field continuity is ensured on transition cell faces by sub-sampling the original data.Va and Vb are the scalar values of grid points a and b, respectively.

In the beginning of our implementation, we apply standard marching cubes algorithm to all level-2 cells. Subsequently, we visit each level-3 cell and handle them sub-cell by sub-cell (see Fig. 5-(a)). We keep the face information of each sub-cell relative to the coarse cell in which it lies. For instance, the sub-cell which is shown as the first sub-cell in Fig. 5-(a), shares its near, left and bottom faces with the coarse cell that hosts it. Then, we check whether any of these shared faces has a level-2 cell neighbor using the cell neighborhood information of the coarse cell. As soon as the condition is met, we determine the edges (see Fig. 5-(b)) which have a potential surface intersection point on itself that will be computed by the marching cubes algorithm. It is sufficient to check the scalar values of the end points for each edge to determine potential intersections. According to our criterion, if there is an intersection on at least one of the inner edges (e8...e11), then there exists a crack at that face (see Fig. 6). Therefore, we keep the intersected edge information of each sub-cell face.

Once all of the required information is gathered for the crack handling, we apply marching cubes algorithm to all eight level-3 fine cells of the sub-cell. Later, we check each face of our sub-cell to see if there is any previously marked intersection on inner edges. In this case, we firstly create a crack array which is necessary to keep the intersected edge ids, and then we perform the following steps: 1) Go through outer edges

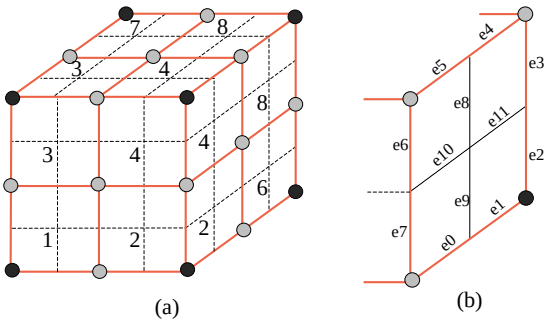


Figure 5: (a) Eight sub-cells of a level-3 cell are illustrated by red cells. (b) Edges are illustrated with their ids for the right transition face of sub-cell-6.

 ISBN 978-80-86943-74-9

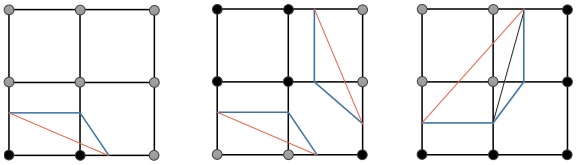


Figure 6: Three of the possible configurations for crack formation. Gray and black colored points have scalar values that are either less or more than the specified isovalue, respectively. Blue lines are generated on high resolution face while red lines are the results of low resolution face. Either one or two cracks can arise (left and middle) or only one crack can cause two triangles (right).

(e0...e7). Push the id of the first found edge which has an intersection point into the crack array. 2) For the found outer edge, check neighboring inner (NI), opposing inner (OI) and neighboring outer (NO) edges in its quarter in order to find a new intersection (see Fig. 7). Push the id of the found edge into the crack array. 3) Jump to the neighboring quarter in which the currently found edge lies and continue to search for a new intersection point by checking neighboring outer, opposing outer (OO) and neighboring inner edges. Push the id of the found edge into the crack array. 4) Follow either step 2 or 3 for any newly found outer or inner intersected edge, respectively.

Instead of simply traversing the edges of each quadrant in any ordering, we follow the edges in an order explained above since this method simplifies having a complete crack array which consists of the intersection points in an order that is easy to follow for triangle generation. For instance, if we have only 3 points in this array, we simply create a new triangle using these points as triangle corners. If we have four points, then we order the triangles' corners as c1, c2, c4 and c2, c3, c4 where c1...c4 are the entries of the crack array.

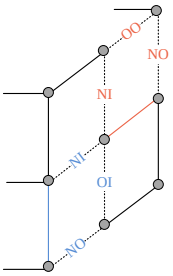


Figure 7: The illustration of an inner (red line) and an outer edge (blue line) together with their neighboring and opposing edges.

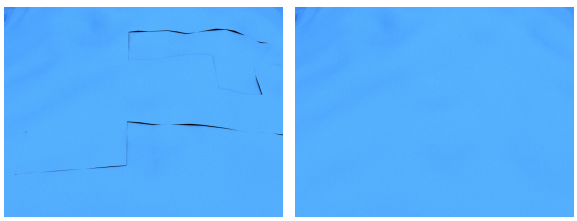


Figure 8: The fluid surface is shown before (left) and after (right) crack handling.

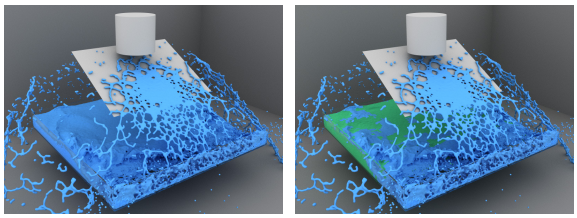


Figure 9: The splash scene with up to 500k particles. The seamlessly stitched mesh is shown on the left; while blue and green regions on the right image shows the high and low resolution parts, respectively.

In order to prevent redundant computations on intersected edges, we follow the technique discussed by Akinci et al. [AIAT12], in which grid points store each computation information for the corresponding three edges that leaves this point. This technique aims single level uniform grids and we pursue a slightly different approach for our multi level grid. We store the information in separate hash maps for level 2 and level 3 fine cells whose keys are again the grid point *id*s, and data part of each entry is composed of 3 integers that are the intersection points of corresponding edges. We firstly finish the triangulation for both level 2 and level 3 fine cells, and fill their hash maps accordingly. Later, we check the level 2 side of the transition cell faces to see if there is any marked intersection on the corresponding edge. For each transition sub-cell face of the level 3 cell, we check the fine edges of level 2 side and use the information on the associated sub-cell edge. This method is helpful to store the mesh in an indexed format (e.g. Wavefront OBJ). These indexes are also used for an easy retrieval of the corners of the newly generated triangles that fill cracks. At the end of this step, we obtain a surface mesh which contains seamlessly stitched two different resolution mesh blocks (see Fig. 8).

4 RESULTS

In this section, we demonstrate the utility of our approach with two different test scenes. Average per frame timings, the number of level-2, level-3, patch triangles and the memory consumption information of both test scenes can be found in Table 1. The demonstrated surface reconstructions were run single-threaded on Intel Xeon X5680 CPU with 24GB RAM. For the fluid simulation, we employed the PCISPH method [SP09]. For all examples in the paper, we performed rendering using mental ray [mental].

In our first experiment, we show a scene with a rotated plane along which the pouring water scatters (see Fig. 9). The scene was simulated using up to 500k particles.

 ISBN 978-80-86943-74-9

scene	#particles	t_{sf}	t_{tri}	t_{total}	$\#tri_{l-2}$	$\#tri_{l-3}$	$\#tri_p$	MEM[GB]
Splash	up to 500k	25.5 sec	4.5 sec	30 sec	140k	1.1m	4000	1
Sink	up to 2.5m	18	2	20	295k	76k	587	1.25

Table 1: Average per frame timings in seconds for each scene. t_{sf}, t_{tri} and t_{total} represent the scalar field computation, triangulation and total timing, respectively. $\#tri_{l-2}, \#tri_{l-3}$ and $\#tri_p$ show the number of level 2, level 3 and patch triangles in order.

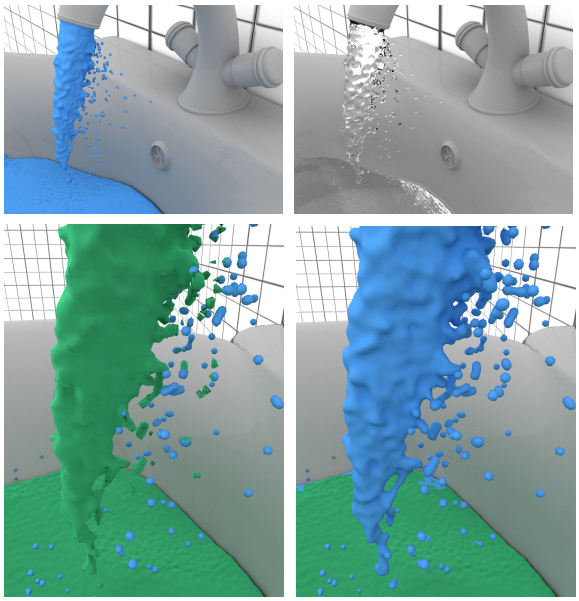


Figure 10: The sink scene with up to 2.5 million particles. Opaque and transparent renderings of the seamlessly stitched mesh are shown in the top row. In the bottom row, results with two different curvature thresholds: 4 (left) and 3 (right) are shown. Green and blue regions represent the low and high resolution parts, respectively.

This test scene shows the increase in high resolution parts and so the computation time (see Table 1) due to the particle scattering and the dominance of high turbulence regions, see Fig. 9-right.

Our next scenario is the Sink scene which consists of up to 2.5 million fluid particles (see Fig. 10 and 11). Using this scene, we firstly experimented with the curvature threshold. The curvature threshold should be chosen by taking the nature of the scene into consideration, e.g., the scene can be turbulent and dominated by high curvature regions or vice versa. Thus, the preferred threshold should guarantee sufficient resolution everywhere. In our experiments, we used 1.5 for the splash scene and 3 for the sink scene. Fig. 10 shows the difference between using two different thresholds on the sink scene. While the value of 3 guarantees a sufficient resolution on high curvature, turbulent regions, e.g. pouring water from the tap; the value of 4 cannot provide sufficient resolution on those parts. Secondly, we compare our method with low and high resolution single level uniform grid approaches which use the cell size of $2r$ and r, respectively. As shown in Fig. 11-left, the low resolution single level approach is not able to preserve the details of the pouring water and isolated pieces; and it creates under-tessellated structures. However, it is sufficient to construct the surface properly in the sink where the fluid surface is relatively flat. On the contrary, high resolution single level uniform grid preserves the details properly as can be seen in the right hand side but this resolution is not necessary for the fluid in the sink. Fig. 10 shows that our method uses low resolution inside the sink and high resolution for the pouring water. Finally, we produce a surface which is of similar quality in comparison to the high resolution single level uniform grid (see Fig. 11-middle). Table 2 and Fig. 12 illustrates the test results which show that even we introduce many steps for setting up the grid structure and crack handling, our method runs up to 60% faster than the high resolution single level uniform grid approach with up to 4 times better memory consumption. When compared to low resolution single level uniform grid approach, our approach preserves all of the surface details that yields a proper surface visualization, with an acceptable performance and memory overhead.

method	t_{total}	$\#tri_{total}$	MEM[GB]
Uniform low res.	7	310k	1.2
Our method	20	375k	1.25
Uniform high res.	34.5	1.2m	4

Table 2: The comparison of our method with single level uniform grid approaches on the sink scene. t_{total}, $\#tri_{total}$ and MEM[GB] represent average per frame computation time, number of triangles and memory consumption, respectively.

When we compare the splash scene and the sink scene, we see that the computation time and the memory consumption is nearly the same for both scenes, even though the sink scene was simulated with larger number of particles. The reason of this similarity is that the particles do not spread around the scene too much in the sink scene in contrast to the splash scene. Therefore, both the coarse grid resolution and the number of surface cells in the narrow band region of the splash scene is slightly larger than the sink scene. Besides, the splash scene is dominated by high resolution parts while the sink scene is dominated by low resolution parts. These points clarify that the computation time depends not only on the resolution of the scene but also on the nature of the scene.

 ISBN 978-80-86943-74-9

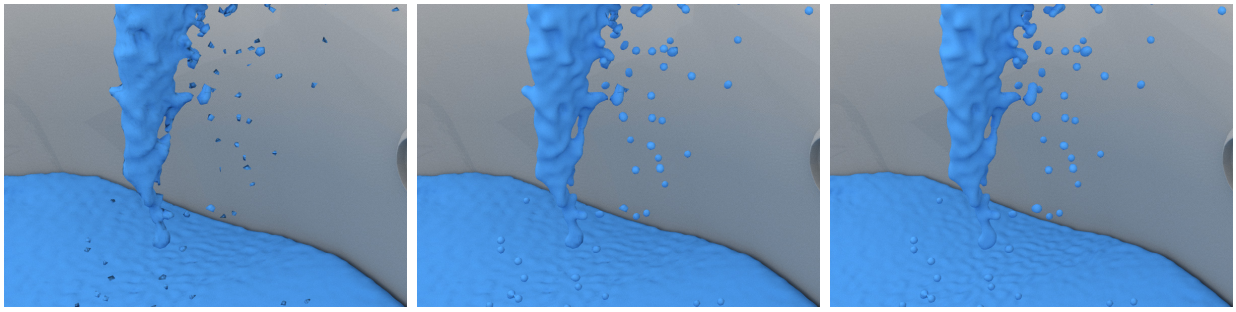
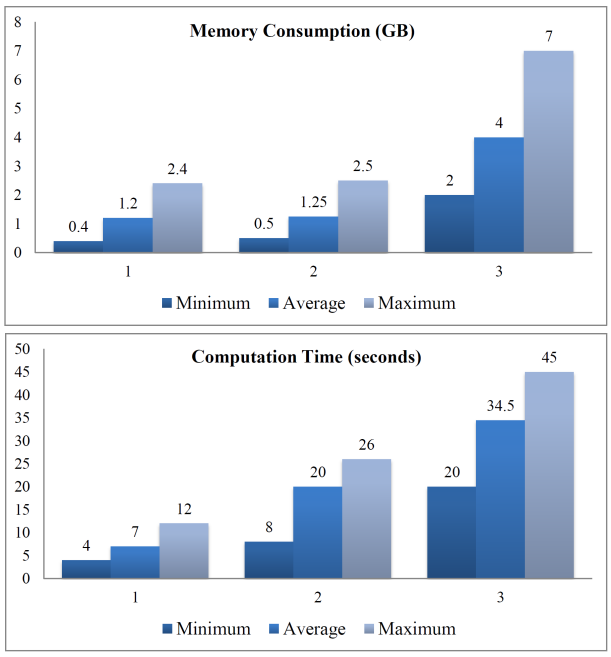


Figure 11: The comparison of the sink scene with single level uniform grid approaches. The low resolution single level uniform grid (left) neither preserve details nor reconstruct the surface properly as our approach (middle) or the high resolution single level uniform grids (right) do. On the other hand, we achieve a similar quality that high resolution single level grids produce with up to four times better memory consumption and up to %60 better performance.



1: Uniform low resolution
2: Adaptive grid
3: Uniform high resolution

Figure 12: Two graphs represent the average per frame memory consumption and the computation time of the Sink scene using our method or high/low resolution uniform grid approaches.

5 CONCLUSION AND FUTURE WORK

In this paper, we presented an adaptive surface reconstruction technique for SPH by using a 3-level uniform grid. The presented grid adapts its cells depending on the surface curvature, which allows for generating less triangles in flat regions but preserving all surface details in curvature regions. Consequently, the number of triangles, memory consumption and the computation time decrease as we produce similar quality results in comparison to high resolution single level uniform grids.

In the future, we would like to consider using hashing for also our top level coarse grid so as to prevent unnecessary data storage for unused cells.

Parallel algorithms have been gaining attention in recent years for high performance computing. However, the incorporation of parallelization in our method is a challenging task due to potential thread safety issues that arise in hashing. Another direction for future work can be addressing these issues and parallelizing our algorithm.

6 ACKNOWLEDGEMENTS

We thank reviewers for their helpful comments. This project is supported by the German Research Foundation (DFG) under contract numbers SFB/TR-8 and TE 632/1-2. The sink model is courtesy of www.turbosquid.com. We also thank NVIDIA ARC GmbH for supporting this work.

7 REFERENCES

[AIAT12] Akinci, G., Ihmsen, M., Akinci, N. and Teschner, M., Parallel surface reconstruction for particle-based fluids. Computer Graphics Forum (2012), 31: pp. 1797-1809.

[AAIT12] Akinci G., Akinci N., Ihmsen M. and Teschner, M., An efficient surface reconstruction pipeline for particle-based fluids. Proc. VRIPHYS, Darmstadt, Germany, pp. 61-68, Dec. 6-7, 2012.

[ALD06] Adams B., Lenaerts T., Dutre P.: Particle Splatting: Interactive Rendering of Particle-Based Simulation Data. Tech. Rep. CW 453, Katholieke Universiteit Leuven, 2006.

[BB09] Brochu T. and Bridson R.: Robust Topological Operations for Dynamic Explicit Surfaces. SIAM Journal on Scientific Computing 31, 4 (2009), 2472-2493.

[BGB11] Bhattacharya H., Gao Y. and Bargteil A. W., A Level-set method for skinning animated particle data. In roceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA) (2011).

[Bri03] Bridson R. E., Computational aspects of dynamic surfaces. PhD Thesis, Stanford University. 2003.

[CPJ10] Costa V., Pereira J. and Jorge J., Multi-Level Hashed Grid Construction Methods. WSCG (2010), Czech Republic.

[FAW10] Fraedrich R., Auer S., Westermann R.: Efficient High-Quality Volume Rendering of SPH Data. IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2010) (2010), 1533-1540.

[GSSP10] Goswami P., Schlegel P., Solenthaler B., Pajarola R.: Interactive SPH Simulation and Rendering on the GPU. In Proceedings of the Eurographics / SIGGRAPH Symposium on Computer Animation (SCA) (Aire-la-Ville, Switzerland, Switzerland, 2010), pp. 55-64.

[HWB04] Houston B., Wiebe M. and Batty C. 2004. RLE sparse level sets. In Proceedings of the SIGGRAPH Conference on Sketches and Applications.ACM.

[IAAT12] Ihmsen M., Akinci N., Akinci G. and Teschner M., Unified spray, foam and bubbles for particle-based fluids, The Visual Computer , vol. 28, no. 6-8, pp. 669-677, 2012, doi:10.1007/s00371-012-0697-9

[JU06] Ju T. and Udeshi T., Intersection-free contouring of an octree grid. In Proceedings of Pacific Graphics (2006).

[LD08] Lagae A. and Dutré P., Compact, fast and robust grids for ray tracing. In ACM SIGGRAPH 2008 talks (SIGGRAPH '08). ACM, New York, NY, USA, , Article 20 , 1 pages.

[LC87] Lorensen W. and Cline H., Marching cubes: A high resolution 3D surface construction algorithm. In SIGGRAPH 1987: Proceedings of the 14th annual conference on Computer graphics and interactive techniques (New York, NY, USA, 1987), ACM Press, pp. 163-169.

[LY04] Lee H. and Yang H. S., Real-time Marching-cube-based LOD Surface Modeling of 3D Objects, ICAT 2004, 2004-12-00.

[Len10] Lengyel E., Transition cells for dynamic multiresolution marching cubes. Journal of Graphics, GPU, and Game Tools (2010), 15:2, pp. 99-122.

[Man10] Manson, J. and Schaefer, S., Isosurfaces over simplicial partitions of multiresolution grids. Computer Graphics Forum (2010), 29(2): pp. 377-385.

[mental] Nvidia ARC, 2011. mental ray 3.9 [software]. URL: http://www.mentalimages.com/products/mental-ray/aboutmental-ray.html.

[MSD07] Muller M., Schirm S., Duthaler S.: Screen Space Meshes. In Proceedings of ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA) (Aire-la- Ville, Switzerland, Switzerland, 2007), pp. 9-15.

[MCG03] Muller M., Charypar D. and Gross M., Particle-based fluid simulation for interactive applications. In SCA 2003: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 154-159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Mul09] Muller M.: Fast and robust tracking of fluid surfaces. In Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (New York, NY, USA, 2009), SCA 2009, ACM, pp. 237-245.

[NM04] Nielsen M. B. and Museth K. Dynamic Tubular Grid: An efficient data structure and algorithms for high resolution level sets. Linkoping Electronic Articles in Computer and Information Science 9, 001

[OR97] Ohlberger M. and Rumpf M., Hierarchical and adaptive visualization on nested grids. Computing, 59 (4): 269-285, 1997.

[RS09] Rosenthal, P. and Linsen, L. (2009), Enclosing Surfaces for Point Clusters Using 3D Discrete Voronoi Diagrams. Computer Graphics Forum, 28: 999-1006. doi: 10.1111/j.1467-8659.2009.01448.x

[SFYC96] Shekhar R., Fayyad E., Yagel R., and Cornhil J. F., Octree-based decimation of marching cubes surfaces. In Proceedings of the 7th conference on Visualization '96 (VIS '96), Roni Yagel and Gregory M. Nielson (Eds.). IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 335-ff..

[Sigg06] Sigg C., Representation and rendering of implicit surfaces. PhD Thesis, ETH Zurich. 2006.

[SSP07] Solenthaler B., Schlafli J. and Pajarola R., A unified particle model for fluid-solid interactions. Computer Animation and Virtual Worlds 18, 1 (2007), pp. 69-82.

[SP09] Solenthaler B. and Pajarola R.: Predictive-corrective incompressible SPH. In SIGGRAPH 2009: ACM SIGGRAPH 2009 Papers (New York, NY, USA, 2009), ACM, pp. 1-6.

[VKSM04] Varadhan G., Krishnan S., Sriram T., and

 ISBN 978-80-86943-74-9

Manocha D., Topology preserving surface extraction using adaptive subdivision. In Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing (SGP '04). ACM, New York, NY, USA, pp. 235-244.

[VT01] Velasco F. and Torres J. C., Cell Octrees: A New Data Structure for Volume Modeling and Visualization. In Proceedings of the Vision Modeling and Visualization Conference 2001 (VMV '01), Thomas Ertl, Bernd Girod, Heinrich Niemann, and Hans-Peter Seidel (Eds.). Aka GmbH, pp. 151-158.

[vdLGS09] Van Der Laan W. J., Green S., Sainz M.: Screen Space Fluid Rendering with Curvature Flow. In Proceedings of the 2009 symposium on Interactive 3D graphics and games (New York, NY, USA, 2009), ACM, pp. 91-98.

[WKE99] Westermann R., Kobbelt L. and Ertl T., Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. The Visual Computer 15 (1999), pp. 100-111.

[WH94] Witkin A. and Heckbert P.: Using Particles to Sample and Control Implicit Surfaces. In SIGGRAPH 1994: Computer Graphics and Interactive Techniques 28 (New York, NY, USA, 1994), ACM, pp. 269-277.

[WvG92] Wilhelms J. and Van Gelder A., Octrees for faster isosurface generation. ACM Trans. Graph. 11, 3 (July 1992), pp. 201-227.

[YT10] Yu J. and Turk G., Reconstructing surfaces of particle-based fluids using anisotropic kernels. In Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '10). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 217-225.

[ZB05] Zhu Y. and Bridson R., Animating sand as a fluid. In SIGGRAPH 2005: ACM SIGGRAPH 2005 Papers (New York, NY, USA, 2005), ACM Press, pp. 965-972.

[ZGHG11] Zhou K., Gong M., Huang X., and Guo B., Data-parallel octrees for surface reconstruction. IEEE Transactions on Visualization and Computer Graphics 17, 5 (2011), pp. 669-681.

[ZPvBG01] Zwicker M., Pfister H., Van Baar J., Gross M.: Surface splatting. In SIGGRAPH 2001: Proceedings of the 28th annual conference on Computer graphics and interactive techniques (New York, NY, USA, 2001), ACM Press, pp. 371-378.

 ISBN 978-80-86943-74-9