HAL
open science

# Real-Time Marker Level Set on GPU

Xing Mei, Philippe Decaudin, Bao-Gang Hu, Xiaopeng Zhang

## To cite this version:

Xing Mei, Philippe Decaudin, Bao-Gang Hu, Xiaopeng Zhang. Real-Time Marker Level Set on GPU. International Conference on Cyberworlds, Sep 2008, Hangzhou, China. pp.209-216, 10.1109/CW.2008.18 . inria-00336647

## HAL Id: inria-00336647

## https://inria.hal.science/inria-00336647

Submitted on 4 Nov 2008

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Real-Time Marker Level Set on GPU

Xing Mei[1], Philippe Decaudin[2], Baogang Hu[1], Xiaopeng Zhang[1]
[1]LIAMA/NLPR, CASIA, Beijing, China
[2]Evasion, INRIA, Grenoble, France
xmei@nlpr.ia.ac.cn, philippe.decaudin@antisphere.com, hubg@nlpr.ia.ac.cn, xpzhang@nlpr.ia.ac.cn

## Abstract

*Level set methods have been extensively used to track the dynamical interfaces between different materials for physically based simulation, geometry modeling, oceanic modeling and other scientific and engineering applications. Due to the inherent Eulerian characteristics, interface evolution based on level set usually suffers from numerical diffusion, sharp feature missing and mass loss. Although some effective methods such as Particle Level Set (PLS) and Marker Level Set (MLS) have been proposed to tackle these difficulties, the complicated correction process and the high computational cost pose severe limitations for real-time applications. In this paper we provide an efficient parallel implementation of the Marker Level Set method on latest graphics hardware. Each step of the MLS method is fully mapped on GPU with an innovative combination of different computation techniques. Relying on GPU's parallelism and flexible programmability, the method provides real-time performance for large size 2D examples and moderate 3D examples, which is significantly faster than previous CPU-based methods.*

## 1. Introduction

Dynamic interface evolution between different materials is an important research topic in computational physics, computer vision, computer graphics and many other fields. The level set method, first introduced by Osher and Sethian [24], is a useful numerical technique for simulating such processes. The basic idea is to represent the interface as a zero level set of a discrete signed distance function. The interface motion is then determined by the level set advection with a problem-related velocity field [23, 30]. The continuous level set field captures large interface deformation and topology change automatically, and provides normal and curvature information easily without explicit surface reconstruction. These advantages make the level set method very popular in numerous applications such as fluid animation [8, 9], geometry processing [22], virtual surgery [14] and flow visualization [6, 35].

Pure level set method suffers from numerical diffusion because all the computation is performed on finite Eulerian grids. Sharp geometry features are gradually smoothed and distorted during the evolution, and the interface keeps shrinking due to the large mass loss. This is a severe limitation for level-set based applications such as fluid simulation and flow visualization. Several techniques have been proposed to address the problem, such as the Coupled Level Set and Volume of Fluid method (CLSVOF) [33], the Particle Level Set method (PLS) [7, 8], the semi-Lagrangian contouring method [1] and the recent Marker Level Set method (MLS) [19, 20]. However, these methods are computationally expensive for two reasons: First, the level set itself is embedded into a grid with higher dimensions, and many iterations over the grid at each time step can be costly. Second, significant computation cost has to be spent on the level set correction process. Although adaptive representations such as the local level set [26], the octree decomposition [18] and the Hierarchical RLE level set [15] help to alleviate the computation load, the performance is still limited for real-time applications.

In this paper, we present a fast and accurate level set method which runs totally on GPU. The method is based on the basic Marker Level Set framework [19, 20]: a set of surface particles move along with the interface, and their location information is used to correct the level set field. MLS has several distinguished advantages over other methods such as CLSVOF and PLS, which will be detailed in the next section. We propose some modifications to the original MLS method so that the data and each computation step can be efficiently mapped on GPU for hardware acceleration. The modifications include a level set correction process based on particle coupling, a robust re-initialization step and a simple particle adjustment strategy. The level set data, the surface particles are packed into GPU resources and get updated in parallel by algorithms running on the GPU. An efficient level set correction scheme is designed to couple the marker particles into the Eulerian grid. Ex-

perimental results demonstrate that the GPU-based MLS method is both accurate and efficient. The numerical diffusion is successfully removed from the level set field. Large size 2D examples and moderate 3D examples can be handled interactively within the system, which makes GPU-based MLS more attractive for real-time applications than previous CPU-based methods.

## 2. Related Work

As mentioned in the introduction, several effective techniques have been introduced to solve the numerical diffusion problem. CLSVOF [33] suggests coupling the level set method with the volume of fluid method for better mass conservation. Since both methods are Eulerian, sharp interface corners still get smoothed and distorted on relatively coarse grids. More methods try to incorporate some Lagrangian information into the level set framework. The semi-Lagrangian contouring method [1] proposes to implicitly track the surface mesh with a distance field. The new distance field is computed from the surface mesh extracted at the previous time step, which is more accurate than pure Eulerian advection. However, the distance computation and the mesh extraction at each step bring significant computational cost. And the contouring method is difficult to fit into the GPU computation framework.

Both PLS [7, 8, 14] and MLS [19, 20] share the same idea: a set of Lagrangian particles are maintained to correct the distorted level set field. In PLS, the particles are placed in a narrow band around the interface. Each particle is treated as a massless sphere with position and radius information for the correction of the level set. A complicated particle reseeding strategy is included to deal with the particles that escape from the interface. While in MLS, the particles are placed exactly on the interface. The correction process requires only the position information of the particles. Therefore MLS maintains much less particles and employs a simpler correction process than PLS. The accuracy of MLS is also comparable with PLS. These advantages make MLS a better choice for real-time applications.

With rapid development in hardware, more and more researchers are turning to highly parallel GPU for computation acceleration (GPGPU) [25]. The level set method is a good candidate for GPU computation, since it is defined on a regular Eulerian grid. Segmentation based on 2D and 3D level set has been successfully ported to GPU in a series of papers [17, 27, 31]. In these applications, the level set is driven by the local curvature. Numerical diffusion is not a serious issue. However, if the level set is used to visualize a turbulent flow field or to track the free surface of the water, numerical diffusion will lead to visual artifacts and degrade the results of the simulation. Less work has been done on developing an accurate level set method on GPU. In 2007, Cuntz *et al*. propose the first parallel implementation of the particle level set method on latest graphics hardware [6]. The implementation strictly follows the PLS framework, which means a lot of particles are maintained and periodically reseeded. A hierarchical distance computation algorithm [5] is included for level set re-initialization and particle reseeding, which turns out to be the most complicated part of the method. As discussed above, our method is based on the MLS framework due to its several advantages over PLS, which yields a simpler implementation with comparable accuracy and efficiency.

Recently the CUDA toolkit, a general GPU computation library provided by NVIDIA, has become popular in scientific computing applications. The library provides a convenient development environment for researchers who are not familiar with the GPU architecture, but it lacks the direct control on the graphics pipeline. Since the computation results usually need to be visualized immediately for graphics applications, traditional GPGPU framework [3, 13, 25] is still a better choice.

## 3. Marker Level Set Method

Before going into the MLS method, we first describe the necessary data and notations:

- the scalar level set field $\phi$, which defines the interface as its zero set
- the surface particle set $P$, which defines a set of particles lying on the interface
- the external velocity field $\vec{V}$, which is provided by the specific application

$\phi$ and $\vec{V}$ should be defined everywhere in the computation domain $\Omega$. Then the level set evolution problem can be stated as: Let $\phi_t$, $P_t$, $\vec{V}_t$ be the data at time $t$, $\Delta t$ be the time interval, how to get the updated level set field $\phi_{t+\Delta t}$ and the updated particle set $P_{t+\Delta t}$ at time $t + \Delta t$.

At each time step, the Marker Level Set method performs the following five steps:

1. Level set advection
2. Surface particle advection
3. Level set correction with the surface particles
4. Level set re-initialization
5. Particle addition and deletion

Step 1, 3, 4 update the level set field, while step 2 and 5 deal with the particles. We briefly discuss the proper algorithms and the possible problems for each step in turn,

which serves a starting point for the GPU implementation. We refers the readers to the original MLS papers for mathematical details, proper parameter settings and surface particle set initialization [19, 20].

In step 1, the level set field $\phi$ is advected by the velocity field $\vec{V}$. The advection equation $\frac{\partial \phi}{\partial t} + \vec{V} \cdot \nabla \phi = 0$ can be solved with first order semi-Lagrangian method [32] or second order BFECC algorithm [29]. Since the accuracy of the method is mainly determined by the surface particles, semi-Lagrangian advection would be enough for step 1.

In step 2, the surface particles $P$ are advected with the same velocity field $\vec{V}$. The position of each particle is updated with the second-order Runge-Kutta algorithm for accurate results.

The level set correction process (step 3) is the most important part of the MLS method: the particles are assumed to be still on the surface after the advection, so their position information is used to correct the distorted level set field. For a cell $(i, j, k) \in \Omega$, the correction equation for $\phi(i, j, k)$ is given as follows:

$$\phi^{new}(i, j, k) = \phi(i, j, k) - \sum_{p \in N_{i,j,k}} w(p)\phi(p) \Big/ \sum_{p \in N_{i,j,k}} w(p) \quad (1)$$

where $N_{i,j,k}$ is a small neighborhood of cell $(i, j, k)$, $p$ is a surface particle in $N_{i,j,k}$, $\phi(p)$ is the interpolated level set value at the particle position, and $w(p)$ is a weight value related to the distance between the particle $p$ and the cell $(i, j, k)$. From eqn. (1) we can see that to correct the level set value $\phi(i, j, k)$, all the surface particles in $N_{i,j,k}$ must be collected. However, despite some recent progress [11, 34], performing an efficient neighboring search on GPU is still a challenging task. Therefore we need to define the correction process in a different way: for each particle $p$, its contribution to all the neighboring cells is computed and stored respectively at these cells. The contribution of all the particles are accumulated for the final correction of the level set field. This idea leads to an efficient GPU-friendly coupling process, which will be detailed in Section 4.

After the advection, the level set ceases to be a signed distance function. The uneven local gradient will distort the interface, which brings considerable numerical errors. Therefore in step 4, a re-initialization process is performed to guarantee the gradient smoothness around the interface. The re-initialization can be realized with the fast marching method [30] or with the PDE-based method [26]. As stated in [6], the fast marching method can not be parallelized on GPU since it is basically a sequential method. Therefore we employ the PDE-based re-initialization method [26]. One known disadvantage of the PDE-based methods is that the interface tends to move slightly during the re-initialization due to the discretization error. The sub-cell fix proposed by Russo and Smereka [28] helps to alleviate the problem, but the computation cost increases since special care mush be taken for the cells near the interface. We provide a simple solution to get around the problem: we turn off the re-initialization process and keep the level set value unchanged for those cells near the interface. Therefore the gradient smoothness around the interface is achieved without destroying the accurate results obtained in step 3. The similar idea has been adopted for BFECC level set advection [29].

In the final step, the surface particle set $P$ is adjusted with the new level set field. The particles escaping from the surface are deleted from $P$. For those regions where the particle density is too sparse, some new particles are generated and added to $P$. Removing and creating particles are difficult on GPU. Although some data compaction and expansion schemes such as HistoPyramids [37] have been proposed to deal with the data arrays with changing length on GPU, we still maintain a particle set with fixed length for better computation efficiency. If a particle is drifting away from the interface, we first try to move the particle towards the interface in the normal direction. This can be done efficiently with the accurate local level set field. If the adjusted particle is still far away from the interface, we mark this particle as 'escaped' and do not use it for level set correction. To avoid the regions with sparse particle density, enough surface particles need to be generated before running the simulation. In practice we provide a user-specified parameter to control the number of the particles per unit surface area. The proper parameter is achieved by running the simulation multiple times.

If we use $\phi^*$ and $P^*$ to represent the intermediate updated data, the data flow in five steps from time $t$ to time $t + \Delta t$ can be summarized as follows:

1. $\phi^* \leftarrow LevelSetAdvection(\phi_t, \vec{V_t})$;
2. $P^* \leftarrow ParticleAdvection(P_t, \vec{V_t})$;
3. $\phi^{**} \leftarrow LevelSetCorrection(\phi^*, P^*)$;
4. $\phi_{t+\Delta t} \leftarrow LevelSetReInitialization(\phi^{**})$;
5. $P_{t+\Delta t} \leftarrow ParticleAdjustment(P^*, \phi_{t+\Delta t})$

## 4. Parallel Implementation on GPU

### 4.1. Overview

The basic GPU computation framework relies on the two important features of the graphics hardware: the render-to-texture extension and shader programming ability. The data is packed into textures and sent to the GPU memory. Then a quad is drawn parallel to the image plane. For each pixel in the quad, a fragment is generated by the hardware rasterizer and processed by a customized fragment shader. The fragment shader takes care of the computation for each pixel,

which is the kernel of the process. Finally, the output data is written into a new texture, which will be used for the computation in the next pass. At first the framework only works well for 2D textures, which means 3D data must be re-organized and stored in a tiled 2D texture for parallel computation. Recently this limitation has been removed from the Shader Model 4.0 GPUs by the new render-to-volume feature. For a detailed introduction to GPU computation, see [3, 13, 25].

We first pack the necessary data into several textures: level set $\phi$ is a scalar field in the 3D domain $\Omega$, so we represent it with a single channel 3D texture $T_\phi$. Another 3D texture $T_c$ with the same size as $T_\phi$ is preserved for level set correction. The particle set $P$ is actually a 1D array which stores the position information for all the surface particles. $P$ is re-organized and packed into a 2D texture $T_P$ with four channels, since 2D textures are better candidates than 1D textures for parallel computation. Besides $T_\phi$, $T_c$ and $T_P$, the 3D velocity field texture $T_{\vec{V}}$ is provided for the advection of the level set and the particles. All the textures are in 16-bit floating point format, which reduces the bandwidth requirements and still preserves the computation accuracy.

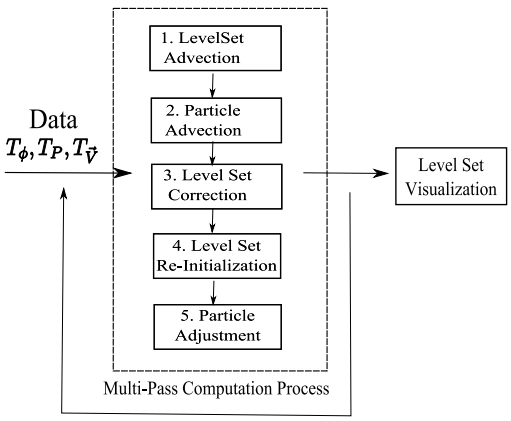


**Figure 1. An overview of the MLS implementation on GPU**

Following the data flow described in Section 3, the MLS method is implemented as a multi-pass computation process on GPU, as shown in Figure 1. Each step takes in some textures from previous steps and updates them in one or several passes. For step 1, 2 and 5, the implementation follows the basic GPU computation framework: the textures get updated in parallel with specific fragment shaders following the governing equations. Only one pass is needed respectively for these steps. The level set correction step (step 3) is the most important part of the computation, which can be efficiently implemented in two passes (Section 4.2). For level set re-initialization, the PDE needs to be iterated over the level set grid multiple times to get a narrow band with smooth gradient around the interface. In practice, six passes are enough for step 4. In total eleven passes are needed for a complete computation cycle. Then the updated level set $T_\phi$ is directly used for visualization (Section 4.3).

## 4.2. Level Set Correction

As discussed in Section 3, to correct the level set value in a grid cell, all the surface particles that stay close to the cell need to be collected. Instead of performing a neighborhood search which is difficult to map on GPU, the level set correction step is implemented as a particle-to-grid coupling process within two passes. In the first pass, each surface particle adds its contribution to the neighboring grid cells within its supporting domain. The contribution from all the particles is accumulated in a new grid $T_c$. In the second pass, $T_c$ is used to correct the level set field $T_\phi$. Therefore the accuracy and the efficiency of the correction process rely on how the particle information is coupled into the 3D grid in the first pass.

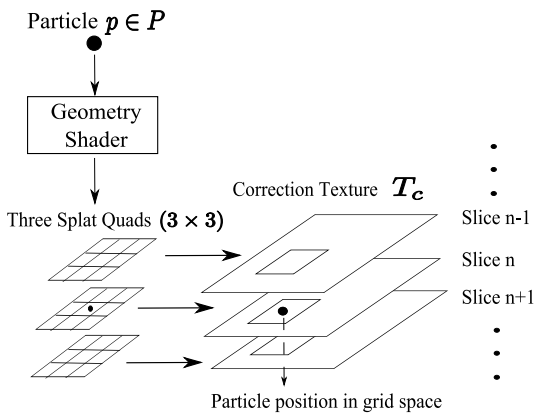


**Figure 2. The particle-to-grid coupling process**

Several methods have been proposed for the particle coupling problem, which basically share the same idea: render several splats around the particle position in the grid to scatter the particle information into the grid. The difference lies in the selection of the splat geometry: point sprites, quads [16] and points [6]. In PLS, the particle is used to correct the level set value of its eight direct neighboring cells. Therefore Cuntz *et al.* [6] suggests that rendering, for each particle, eight individual points into two subsequent slices for the correction step. However, this method doesn't work well with MLS since more neighboring cells (at least 27 ) need to be corrected for each surface particle. Rendering many points would not be efficient. Therefore we still use quads as the proper splat geometry.

The coupling pass is illustrated in Figure 2. For each surface particle, its grid position is retrieved from the particle texture $T_p$ in the vertex shader. The particle is sent to the geometry shader. We first check if the particle is an 'escaped' one or not. If yes, we discard the particle; if not, we generate three 2D splat quads with the particle in the geometry shader. Each quad's size is $3 \times 3$ so that the three quads cover the 27 neighboring cells in the grid. Then these quads are rendered into three subsequent slices of the correction texture $T_c$ at a time. The particle's contribution is computed in the fragment shader. The blending mode of this pass is set to 'Additive' to collect the contribution of all the particles in $T_c$.

Our implementation relies on two important features of the Shader Model 4.0 hardware: the geometry shader and the render-to-volume extension. The use of the geometry shader allows us to generate the splat quads on the fly, while previous methods tend to store the splats into a predefined vertex buffer. For large size particle set, much less geometric primitives are sent to the vertex shader in our approach. The use of the render-to-volume extension allows us to directly render the splat quads into correct slices of a 3D texture in one draw, without storing the volume in a tile 2D texture like previous methods. The adoption of the two new features leads to efficient coupling and interactive results, as shown in Section 5.

### 4.3. Level Set Visualization

The updated level set field $\phi$ is stored in a 3D volume texture.We employ a ray-casting method in [10] for fast volume visualization. Rays are cast into the texture volume, and their intersection points with the zero level set are found with a quick searching algorithm. The intersection points, which represent the evolving interface, are then shaded in the pixel shader using Phong lighting.

Our main framework can be easily adapted for 2D level set computation. The texture resources for the level set and the velocity field are changed into 2D formats. And the corresponding computation step 1, 3 and 4 now follow the 2D GPU computation framework instead. The only difference between 2D and 3D level set lies in the visualization part. The zero set is a set of contour lines in the 2D space. Traditional volume rendering techniques are no longer applicable. We develop a fast GPU version of the classic CONREC contouring algorithm [2] to visualize the 2D zero set.

## 5. Experimental Results

In this section we test the GPU-based MLS method with several 2D and 3D examples. These typical examples have been extensively used in computational physics and fluid simulation for evaluating the accuracy of the interface tracking methods [8, 7, 14, 19, 23]. The test platform is a Pentium IV 2.40GHz PC equipped with a NVIDIA GeForce 8800 GTX graphics card. See the accompanying video at http://xmei.info/demo/GPUMLS.avi.

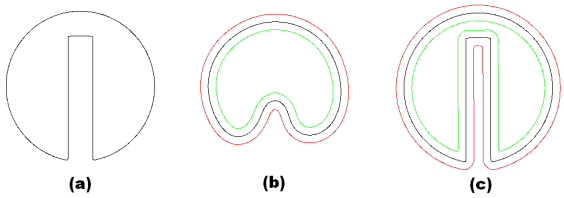


**Figure 3. 2D Zalesak's disk test: (a) the original disk (b) disk after $360°$ rotation with pure level set (c) disk after $360°$ rotation with GPU-based MLS. Adjacent $\pm 4.0\Delta x$ contour lines are given in red and green.**

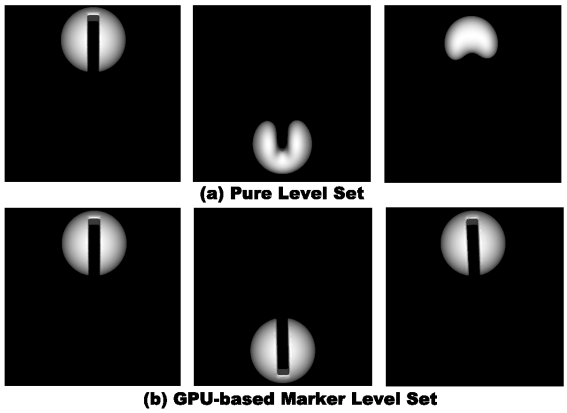


**Figure 4. 3D Zalesak's sphere test: (a) $180°$(middle) and $360°$(right) rotation with level set (b) $180°$(middle) and $360°$(right) rotation with GPU-based MLS**

We first test the 2D Zalesak's disk problem: the rigid-body rotation for a slotted disk in a constant-vorticity velocity field. The grid resolution is $256 \times 256$. The advected results for pure level set method and MLS after $360°$ rotation are given in Figure 3. With the level set method, the sharp corners of the disk get badly smoothed, and the area of the disk shrinks because of the mass loss. Both problems are solved with the GPU-based MLS method: the disk stays as a rigid-body and all the sharp features are well preserved during the rotation. The similar results for 3D Zalesak's sphere problem are presented in Figure 4. The 3D grid size is $128 \times 128 \times 128$. Again the numerical diffusion is successfully removed from the rigid-body rotation with GPU-based

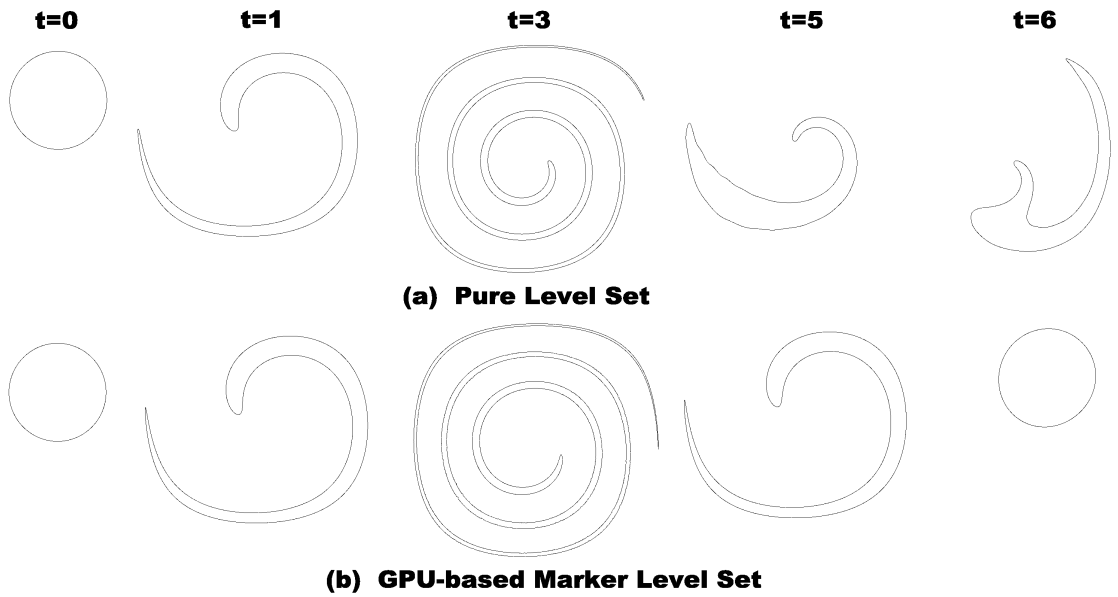


**Figure 5. 2D Vortex test: (a) interface evolution with level-set (b) interface evolution with GPU-based MLS**

MLS method.

Then we test with two difficult examples where the interface would be greatly stretched and distorted by the underlying velocity field. The first example is a 2D vortex-in-a-box problem. A disk in the computation domain is gradually stretched and dragged around the domain center in a vortex-based velocity field. After reaching a maximum stretching time $t_s$, the direction of the velocity field is reversed. The disk should return to its original position after another time interval $t_s$. In this case, $t_s = 3$. We use a grid resolution $1024 \times 1024$ to show our framework is available for large size 2D applications. The interface evolution results for level set and GPU-based MLS are given in Figure 5. In both cases, the interface performs well before the velocity field is reversed. This is partly related to the fact that we are using a high-resolution grid. However, with pure level set method, the interface gets much distorted during the reverse process, while GPU-based MLS successfully recovers the disk at the original location when $t = 6$. The second example is a 3D deformation test which is first proposed by Enright *et al*. [7]. The grid resolution is $128 \times 128 \times 128$. The velocity field is defined by combining two deformation processes in two orthogonal directions. Like the 2D vortex test, the velocity field is reversed at time point $t_s$. We set $t_s = 1$. A small sphere is put into one corner of the computation domain to test the algorithm. The deformation results are presented in Figure 6. At the reverse time point $t = 1$, the interface gets very thin and greatly stretched. For level set method, the interface breaks up due to the large mass loss. The rest of the interface gradually disappears during the deformation. For GPU-based MLS, the thin interface at $t = 1$ is well kept due to the coupling of the surface particles. The sphere returns to its original position without obvious mass loss.

| Test | Grid Size | Num. Particles | Mass Loss | FPS |
|---|---|---|---|---|
| 2D Disk | $256^2$ | 2232 | 0.1% | 430 |
| 2D Vortex | $1028^2$ | 9560 | 1.5% | 36 |
| 3D Sphere | $128^3$ | 49272 | 0.9% | 28 |
| 3D Deformation | $128^3$ | 97708 | 1.8% | 24 |

**Table 1. Performance results for the four tests. For each test, the grid size, the number of the surface particles, the total mass loss and the framerates are recorded.**

The performance results for the four tests are presented in Table 1. For each test, we record the number of the surface particles used in the simulation, the mass loss and the framerates. For 2D Disk and 3D sphere tests, the mass loss can be limited to a very small value with relatively few particles. However, for 2D Vortex and 3D deformation tests where the interface gets much distorted, more particles are needed for good mass conservation. GPU-based MLS runs at interactive framerates for all the four tests. For the sphere test, the performance is 28 fps, which is comparable with

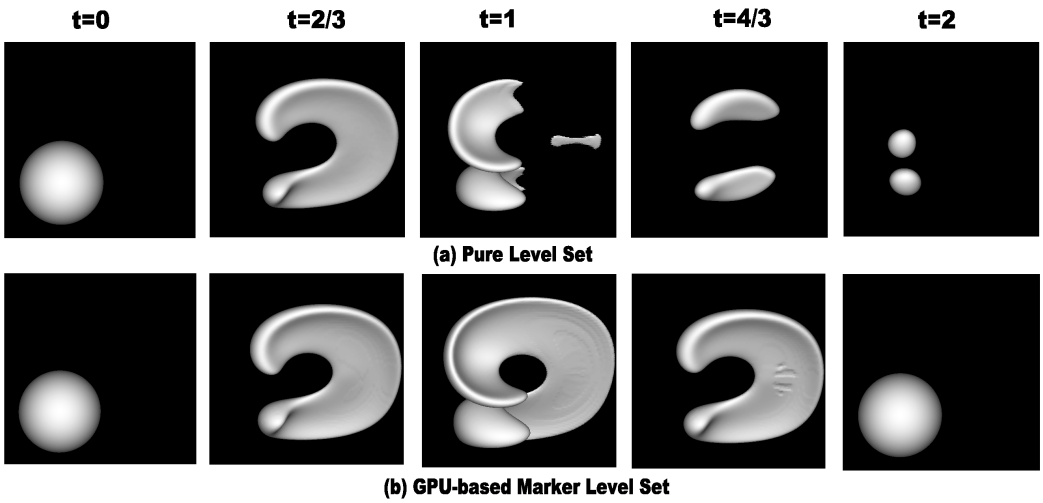


**Figure 6. 3D Enright deformation test: (a) interface evolution with level-set (b) interface evolution with GPU-based MLS**

| Test | Level set Advection | Particle Advection | Level set Correction | Level set Re-initialization | Particle Adjustment | Rendering, etc |
|---|---|---|---|---|---|---|
| 2D Disk | 3% | 2% | 6% | 11% | 2% | 76% |
| 2D Vortex | 2% | 1% | 4% | 19% | 4% | 70% |
| 3D Sphere | 2% | 2% | 26% | 58% | 2% | 10% |
| 3D Deformation | 1% | 1% | 42% | 44% | 3% | 9% |

**Table 2. The running time percentage for each step of GPU-based MLS in the four tests**

the results reported in [6]: 14.74 fps on a GeForce 8800 GTS graphics chip. The time percentage for each step of MLS in the four tests is presented in Table 2. For 2D cases the visualization part takes most of the time, while for 3D cases, the computation part is more demanding. The iterative re-initialization process is the most time-consuming step in the computation framework, while the running time for the level correction step is closely related to the number of the surface particles used in the simulation. Finally we provide a brief performance comparison with an open source CPU-based PLS library [21]: for the 2D $256 \times 256$ disk example, the PLS library runs at 36 fps, while for the 3D $128 \times 128 \times 128$ sphere example, the PLS library runs at 0.91 fps. Our method benefits a lot from the GPU acceleration.

## 6. Conclusions and Future Work

We have presented a fast and accurate level set method for real-time applications. Relying on a modified MLS framework, our method produces accurate interface evolution results without numerical diffusion. Each step of the modified MLS is well designed to be fully mapped to GPU with specific computation techniques. Our GPU-based MLS can handle large size 2D grids and moderate 3D grids efficiently. The interface evolution process can be viewed at interactive framerates. These features are usually difficult for previous CPU-based methods.

As a future work, we would like to integrate our method into an existing GPU fluid solver for free surface water simulation. Although a lot of work has been done on GPU-based fluid simulation [4, 12, 36], current simulators still suffer from mass loss, inefficient poisson solvers, pressure oscillation with non-staggered grids and other problems. We expect GPU-based MLS would help improve the simulation results.

## Acknowledgements

Xing Mei is supported by LIAMA NSFC 60073007 and by a grant from the European Community under the Marie-Curie project VISITOR MEST-CT-2004-8270. Philippe Decaudin is supported by a grant from the European Community under the Marie-Curie project REVPE MOIF-CT-

2006-22230.

## References


[1] A. W. Bargteil, T. G. Goktekin, J. F. O'brien, and J. A. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25(1):19–38, 2006.

[2] P. Bourke. CONREC: A contouring subroutine. *Byte: The Small Systems Journal*, 12(6):143–150, 1987.

[3] I. Buck. Mapping computational concepts to gpus. In *GPU Gems 2*, chapter 32, pages 509–519. Addison-Wesley, 2005.

[4] K. Crane, I. Llamas, and S. Tariq. Real-time simulation and rendering of 3d fluids. In *GPU Gems 3*, chapter 30, pages 633–673. Addison-Wesley, 2007.

[5] N. Cuntz and A. Kolb. Fast hierarchical 3d distance transforms on the gpu. In *EG'07 Short Papers*, pages 93–96, 2007.

[6] N. Cuntz, R. Strzodka, and A. Kolb. Real-time particle level sets with applications to flow visualization. Technical report, Uinversity Siegen, 2007.

[7] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-lagrangian particle level set method. *Computer and Structures*, 83(6-7):479–490, 2005.

[8] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21(3):736–744, 2002.

[9] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proc. SIGGRAPH '01*, pages 23–30, 2001.

[10] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[11] T. Harada, S. Koshizuka, and Y. Kawaguchi. Smoothed particle hydrodynamics on gpus. In *Proc. CGI '07*, 2007.

[12] M. Harris. Fast fluid dynamics simulation on the gpu. In *GPU Gems*, chapter 38, pages 637–666. Addison-Wesley, 2004.

[13] M. Harris. Mapping computational concepts to gpus. In *GPU Gems 2*, chapter 31, pages 493–508. Addison-Wesley, 2005.

[14] S. E. Hieber and P. Koumoutsakos. A lagrangian particle level set method. *Journal of Computational Physics*, 210(1):342–367, 2005.

[15] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical rle level set: A compact and versatile deformable surface representation. *ACM Trans. Graph.*, 25(1):151–175, 2006.

[16] A. Kolb and N. Cuntz. Dynamic particle coupling for gpu-based fluid simulation. In *Proc. 18th Symposium on Simulation Technique*, pages 722–727, 2005.

[17] A. E. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proc. VIS '03*, page 11, 2003.

[18] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph.*, 23(3):457–462, 2004.

[19] V. Mihalef, D. Metaxas, and M. Sussman. Textured liquids based on the marker level set. *Computer Graphics Forum*, 26(3):457–466, 2007.

[20] V. Mihalef, M. Sussman, and D. Metaxas. The marker level set method: a new approach for computing accurate interfacial dynamics. *submitted to Journal of Computational Physics*, 2007.

[21] E. Mokhberi and P. Faloutsos. A particle level set library. Technical report, University of California, Los Angeles, 2006.

[22] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr. Level set surface editing operators. *ACM Trans. Graph.*, 21(3):330–338, 2002.

[23] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2003.

[24] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[25] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[26] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A pde-based fast local level set method. *Journal of Computational Physics*, 155(2):410–438, 1999.

[27] M. Rumpf and R. Strzodka. Level set segmentation in graphics hardware. In *Proc. ICIP '01*, pages 1103–1106, 2001.

[28] G. Russo and P. Smereka. A remark on computing distance functions. *Journal of Computational Physics*, 163(1):51–67, 2001.

[29] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac. An unconditionally stable MacCormack method. *Journal of Scientific Computing*, In Press, 2007.

[30] J. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge Unviersity Press, 1999.

[31] A. Sherbondy, M. Houston, and S. Napel. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *Proc. VIS '03*, page 23, 2003.

[32] J. Stam. Stable fluids. In *Proc. SIGGRAPH '99*, pages 121–128, 1999.

[33] M. Sussman. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *Journal of Computational Physics*, 187(1):110–136, 2003.

[34] J. S. Venetillo and W. Celes. Gpu-based particle simulation with inter-collisions. *The Visual Computer*, 23(9-11):851–860, 2007.

[35] R. Westermann, C. Johnson, and T. Ertl. A level-set method for flow visualization. In *Proc. VIS '00*, 2000.

[36] E. Wu, Y. Liu, and X. Liu. An improved study of real-time fluid simulation on gpu. *Journal of Computer Animation and Virtual World*, 15(3-4):139–146, 2004.

[37] G. Ziegler, A. Tevs, C. Theobalt, and H. P. Seidel. Gpu-based data compaction using histopyramids. In *Proc. VMV '06*, pages 137–141, 2006.