

Implementation of Level Set Methods in Cactus Framework

Carbunescu Razvan Corneliu
Computer Science
Center for Computation and Technology at
Louisiana State University
LSU Campus, Baton Rouge
302 Johnston Hall, Louisiana 70803

Faculty Advisor: Dr. Gabrielle Allen

Abstract

Level set methods are a general and powerful technique to represent an object's boundary by the means of an implicit function that has a specific value on the boundary. This property permits the easy modeling of interfaces between different fluids that interact under prescribed/evolving conditions or that act under specified constraints. The Cactus Code is a general framework for developing computational solutions for a varied class of partial differential equations (PDEs). Amongst many Cactus applications are: numerical relativity, astrophysical flows and computational fluid dynamics (CFD). The purpose of this research work is to test and implement level set methods for solving problems with evolving surfaces within the Cactus framework. Once this method is implemented in Cactus it can be further used and upgraded to implement free surface flow and multiphase CFD problems making it easier for other scientists to conduct research in this field.

To benchmark and validate the implementation, a simple problem of a notched disc moving in a rotational velocity field (known as Zalesak's Problem) is solved using Cactus. A simple implementation of a specific level set method must maintain constant volume of the disc after a full rotation. The results show the importance of reinitializing the signed distance function which is used to represent the interface of the notched disc. The shape of the notched disc changes to decrease sharp corners and becomes smoother at the end of a full rotation. Although the disc preserves its volume throughout the rotation by using this simple level set method, its shape changes drastically. Implementing advanced versions of simple level set methods^{8,15} that preserve the shape of the interface over arbitrary velocity fields is deferred for future research. The results of Zalesak's Problem on different grid resolutions and under different flow conditions will be presented to validate as well as verify the correct implementation of the algorithm.

1 Introduction

1.1 what are level set methods?

Level set methods^{1,2,4,5,7,9} present a technique of representing an object and its boundary by the use of an implicit function. This representation allows for the easy manipulation of that object in given/evolving velocity fields by the use of partial differential equations. Since the use of implicit functions¹ is basic to the principle of level sets it is important to understand the concept.

Imagine a 2-dimensional plane and a circle of radius r centered at the origin within that plane. The circle divides the plane into 2 regions which are connected by the points on the curve (in our case the circle) which is named the interface. If we were to consider a way of representing this mathematically it makes sense to try to represent this with a function which is 0 on the interface and has different values for the separate regions. By choice we generally select a function which has a negative value inside and positive outside. A example for this is the function $f(x,y) = x^2 + y^2 - r^2$ which has a negative values inside, 0 on the interface, and positive outside. Similarly we can imagine a 3-dimensional space and a closed surface within that space. It similarly divides the space into 2 regions: inside the surface and outside with the surface representing the interface. By choosing a similar function Φ , which has a value of 0 on the interface and negative values inside and positive outside, we will choose an implicit function which can

represent this surface and the regions defined.

The reason for selecting these types of functions to represent surfaces is the ease with which they can be manipulated by the use of partial differential equations (PDEs) to evolve the surfaces. Computational Fluid Dynamics (CFD) use level set methods^{11,12,13,14} exactly because of this ability which allows scientists to easily modify fluid representations within different types of velocity fields, whether these are driven by the mean curvature of a fluid surface, a given motion in the normal direction or a simple external velocity field. The implicit function usually chosen to best represent these fluids is the signed distance function (SDF)¹. This is a function which is 0 on the interface -d inside the surface and +d outside the surface where d is the shortest distance from the point to the surface.

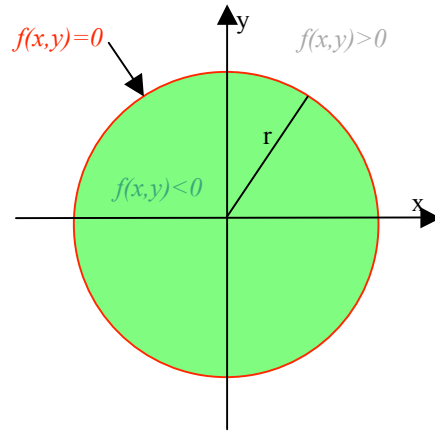


Figure 1. An implicit definition by the use of the function $f(x,y) = x^2 + y^2 - r^2$

1.2 what is the Cactus framework?

“Cactus is a framework for developing portable, modular applications, in particular, although not exclusively, high-performance simulation code”.³ Basically Cactus^{16,17} presents an easy way of writing code so that it can be used quickly and independently of a particular architecture or the need for parallelization. Although many different frameworks exist that each benefit from particular knowledge of certain aspects or fields of research Cactus is unique because of its extraordinary portability which has allowed it to be successfully tested on most known architectures. Cactus consists of two separate parts: the “flesh” which just provides the basic interaction with the computational environment; and thorns which implement different concepts. An example of a thorn is PUGH which is a default memory allocator or CartGrid3D which creates a grid with the given specification. All thorns which are generally related reside within collections known as arrangements. One other great advantage of Cactus is the fact that each thorn generates an implementation which is independent of the underlying code. This permits the user to specify any thorn which provides this implementation and allows him at any moment to substitute this thorn for any other that provides the same implementation. Also, another important aspect is the fact that Cactus is open source so it can be acquired by anyone without problem and edited, with some knowledge of the code, of course, to fit any other problems desired.

1.3 why implement level set methods in Cactus?

As previously stated, level set methods represent an important part of Computational Fluid Dynamics, which of course have many different important applications^{12,14}. Therefore the introduction of a CFD toolkit within Cactus would offer many advantages to scientists and researchers within this field. The introduction of a thorn to handle level set methods is only obvious then to facilitate the creation of this toolkit. Although the thorns which are presented here are maybe not as complex as required for a scientific program, they embed a simple example and implementation of the concept. The concept and problem were implemented in Cactus also to take advantage of the possibilities of the framework, namely the modular approach which allowed for the problem to be split into two separate thorns: one which handles the initialization of a function and velocity field with given values and another thorn to evolve this function in the given velocity field and to do the very important reinitialization of the Signed Distance Function.

2 Problem Description

2.1 notched disc or Zalesak's Problem

After writing the initial code within Cactus there is an obvious need for a test case to verify and validate the correctness of the algorithm as well as provide an example for the thorn. In this frame of reference a simple problem known as "Zalesak's Problem" was chosen to test the implementation of level sets within Cactus. This problem deals with a notched disc or sphere that evolves within a rotational velocity field. The goal of the problem is to best preserve the volume and shape of the notched disc. The shape of the notched disc should modify though within this implementation because of the simple implementation of level set methods which try to preserve volume and not shape. Under these simple level set methods the evolution will tend to 'blur out' sharp edges or corners within the interface and give a smoother picture of the disc. This tendency should be minimized as the size of the grid increases or the size of the timestep decreases thus providing greater accuracy and better results. Zalesak's Problem presents a great test case for the implementation of level set methods because it allows for testing of the way in which that particular implementation deals with sharp edges while trying to preserve volume and shape.

2.2 reinitialization importance and scheme

One important problematic aspect with level set methods that is also pointed out by Zalesak's Problem is the importance of reinitialization^{4,5,6,7} of the SDF. The problem with level set methods is that after applying the partial differential equations on the function (which is represented by values at the grid points) for a number of timesteps the function's values begin to differ significantly from the SDF which the function is suppose to represent. This requires the reinitialization of the SDF by recalculating the values of the function starting from the approximate values of the interface from the grid. This is very important for the calculation because differences in the function from the SDF can bring about large errors within the simulation.

For this task there exist different methods of recalculating the SDF. One method presented is the evolution in the normal direction of the interface with a speed of one and considering the crossing time (the time the new interface reaches a point on the grid) as the value of the SDF at that point. While this method seems to provide a very accurate approximation is also requires high amounts of simulation time. Another, less time intensive, approach for the computation of the values of the SDF is known as the 'Fast Marching Method'^{1,2,4,5}. The way this particular method works is by creating a list with the point on the interface and continuously adding to that list the closest point to the 'band of points' already selected within the rest of the Grid. This process continues until the entire grid has been covered and the values in the list represent the new values of the SDF. In the implemented code a variation of this technique was used to recalculate the SDF.

2.3 other papers discussing Zalesak's Problem

Another main reason why Zalesak's Problem was used to test the implementation of level set methods within Cactus is the fact that this problem has been very well documented and there exist many similar papers^{4,5,6,7,10,15} to which the results of the tests could be compared to. Although the implementation given within Cactus does not benefit from the same high-order discretization for the spatial derivatives (the partial differential equations with regard to the x, y and z directions) it should still provide data comparable with these other papers and should also present the same general properties.

2.4 implementation within Cactus

The implementation of level set methods within Cactus was made into two separate thorns: ID_NotchedDisc and NotchedDisc. The first thorn, named ID_NotchedDisc, supplies the simulation and implementation with the initial data needed for a run. It creates the initial data on the disc, whether it is a notched disc or simple disc (first test case) and tests the initial data parameters for errors like, for example, a notch bigger than a disc. It also creates the velocity field and initializes the values of limits which are admissible in the simulation.

The second thorn, named NotchedDisc, uses the initial data and other information about the type of simulation to evolve the desired function in the velocity. This thorn also has implemented the reinitialization algorithm and performs it whenever the required parameter is reached.

The equation, which was solved, was a simple PDE equation that represents the motion of an incompressible object within an externally generated velocity field:

$$\frac{\partial \Phi}{\partial t} + \nabla \cdot \Phi \mathbf{a} = 0 \quad \text{where} \quad \mathbf{V} = u, v, w \quad \text{and} \quad \nabla \cdot \Phi = \frac{\partial \Phi}{\partial x} + \frac{\partial \Phi}{\partial y} + \frac{\partial \Phi}{\partial z}$$

The PDEs were discretized by using first order accurate upwind schemes, which, for positive values of the velocity field, would have the following values:

$$\frac{\partial \Phi}{\partial t} = \frac{\Phi^{n+1} - \Phi^n}{\Delta t} \quad \frac{\partial \Phi}{\partial x} = \frac{\Phi^n - \Phi^n}{\Delta x} \quad \frac{\partial \Phi}{\partial y} = \frac{\Phi^n - \Phi^n}{\Delta y} \quad \frac{\partial \Phi}{\partial z} = \frac{\Phi^n - \Phi^n}{\Delta z}$$

The values for the PDEs change within the upwind scheme to account for the direction of the information flow and might cause a change in the discretization of the spatial derivatives:

$$\text{if } u > 0 \quad \frac{\partial \Phi}{\partial x} = \frac{\Phi^n - \Phi^n}{\Delta x} \quad | \quad \text{if } u < 0 \quad \frac{\partial \Phi}{\partial x} = \frac{\Phi^n - \Phi^n}{\Delta x}$$

Due to the nature of discretizing the PDEs there exists a limit which must be imposed for all the values of the velocity field and of the parameters which appear within the simulation. These restrictions are imposed because of the fact that information from one grid cell can not pass more than one space between grid points within one timestep. This is mathematically represented by a factor known as the Courant number which has the form:

$$C = \max(u, v, w) \frac{\Delta t}{\max(\Delta x, \Delta y, \Delta z)}$$

In order for the simulation of the flow to be stable and produce a correct output it is required that this Courant number be less than or equal to one for all values within the given velocity field. If these values are exceeded numerical errors appear that have a great affect on the simulation and usually cause for the simulation to 'explode', which means the appearance of very large numbers that destroy the data within the grid.

Even without the problem of an unstable grid the simulation still encounters numerical errors due to the first order schemes and the reinitialization algorithm, which are also the cause of the loss in volume.

The reinitialization algorithm implemented within Cactus also presents small differences from the regular Fast Marching Method scheme. As was previously stated the Fast Marching Method works by creating a band of points with known values and by continuously adding the closest point to that band and calculating its distance to the interface by comparing it to all the points in the band. Starting from this algorithm the following was created.

First the implemented algorithm selects the points in the interface and puts them in a list. Then it initializes a matrix with the same dimensions as the grid with either the points position if it is on the interface or with 0 if it is not on the interface; this matrix will represent the 'source', or closest point on the interface, of the selected point. Then, starting with the first point in the list, we select all the points neighboring the selected point that do not have a 'source' point. We then proceed to calculate the minimum value to the interface from that point by comparing the distance between the sources of the neighbors and the selected point plus the value of the function at the sources and selecting the smallest distance. The source of the currently selected point then becomes the source of the chosen neighbor and the point is added to the list. This is done for all the neighbors of the initially selected point in the list until all neighbors are done then the algorithm passes on to the next point in the list. The algorithm continues until the list fills up with all the points in the grid and then copies the new values to the grid.

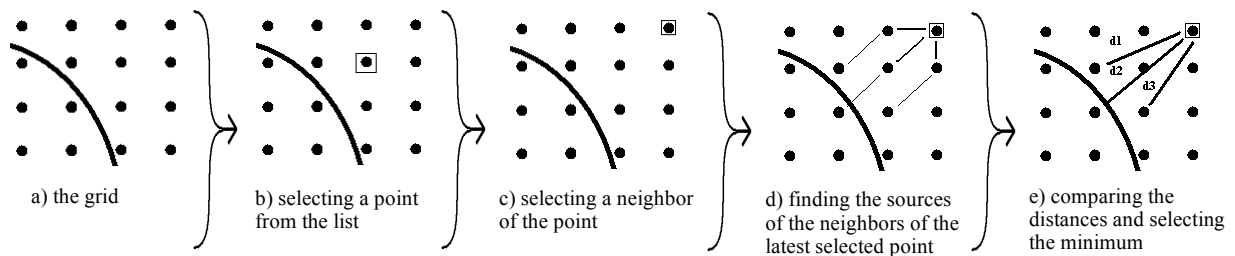


Figure 2. A view into the reinitialization algorithm

3 Simulation Results

The Cactus implementation was tested with a notched disc problem with the following characteristics: grid domain $[-20, 20] \times [-20, 20]$, radius of the disc 15, notch length 25 and notch width 5. Other characteristics of the simulation varied in order to better see the results: grid size 50x50, 100x100, 200x200 and 400x400; reinitialization every 1 timestep, 2 timesteps, 5 timesteps, 10 timesteps, 20 timesteps and no reinitialization; timesteps of 0.0005, 0.001, 0.005, 0.01 and 0.05.



























































































































VOLUME LOSS on 50 size grid							VOLUME LOSS on 100 size grid						
dt \ reinit	1	2	5	10	20	None	dt \ reinit	1	2	5	10	20	None
0.0005							0.0005						
	38.428%	38.428%	38.285%	38.142%	38.142%	79.241%		21.036%	21.001%	20.966%	20.862%	20.758%	42.107%
0.001							0.001						
	38.428%	38.428%	38.285%	38.142%	38.000%	79.000%		20.653%	20.653%	20.584%	20.514%	20.479%	42.072%
0.005							0.005						
	36.607%	36.285%	36.285%	36.714%	37.383%	77.897%		19.095%	19.026%	18.991%	18.991%	18.991%	42.017%
0.01							0.01						
	35.085%	34.517%	34.517%	34.943%	49.715%	75.284%		16.283%	16.109%	15.970%	N/A	N/A	N/A
0.05							0.05						
	N/A	N/A	N/A	N/A	N/A	N/A		N/A	N/A	N/A	N/A	N/A	N/A
VOLUME LOSS on 200 size grid							VOLUME LOSS on 400 size grid						
dt \ reinit	1	2	5	10	20	None	dt \ reinit	1	2	5	10	20	None
0.0005							0.0005						
	10.356%	10.356%	10.243%	10.138%	10.042%	17.853%		5.259%	5.209%	5.144%	5.079%	5.035%	7.568%
0.001							0.001						
	10.103%	10.060%	9.911%	9.798%	9.711%	17.714%		4.983%	4.933%	4.839%	4.787%	4.757%	7.349%
0.005							0.005						
	8.047%	7.916%	N/A	N/A	N/A	N/A		N/A	N/A	N/A	N/A	N/A	N/A
0.01							0.01						
	N/A	N/A	N/A	N/A	N/A	N/A		N/A	N/A	N/A	N/A	N/A	N/A
0.05							0.05						
	N/A	N/A	N/A	N/A	N/A	N/A		N/A	N/A	N/A	N/A	N/A	N/A
Initial	Initial											Initial	Initial
													
50 grid	100 grid											200 grid	400 grid

Figure 3. Table data with the Volume loss and final picture of the notched disc

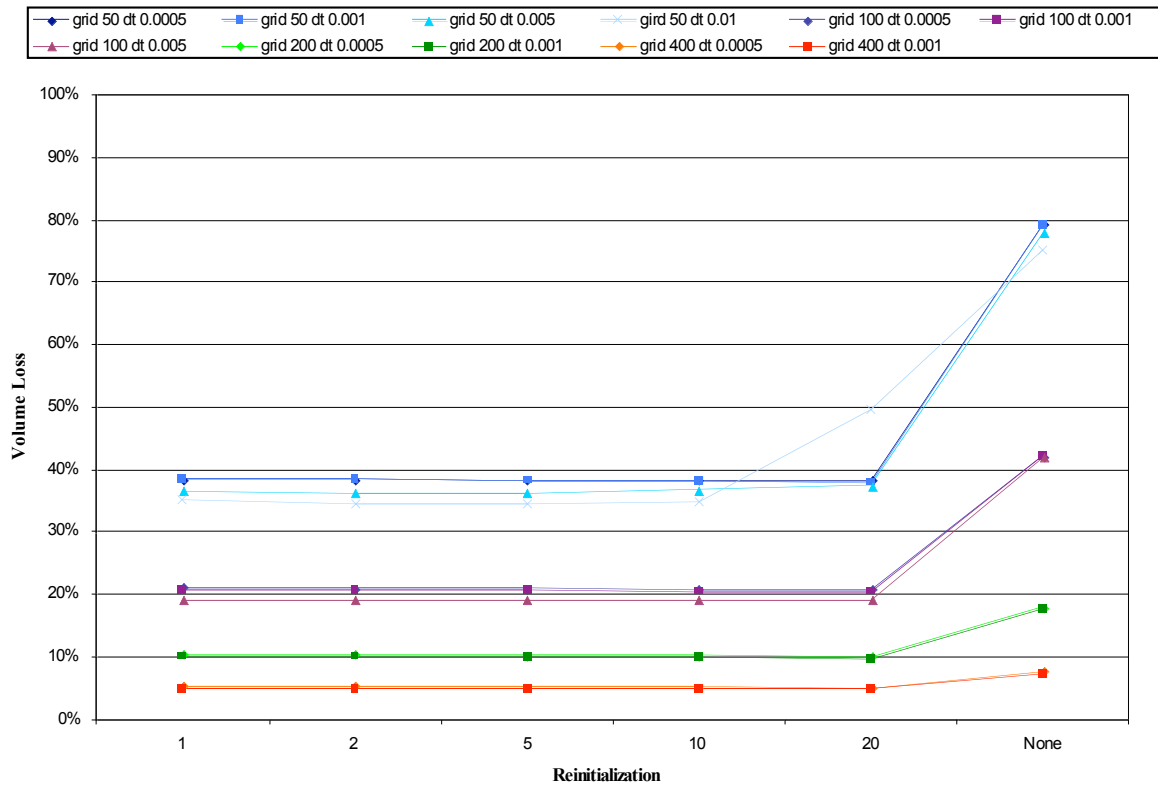


Fig 4. Volume loss comparison chart

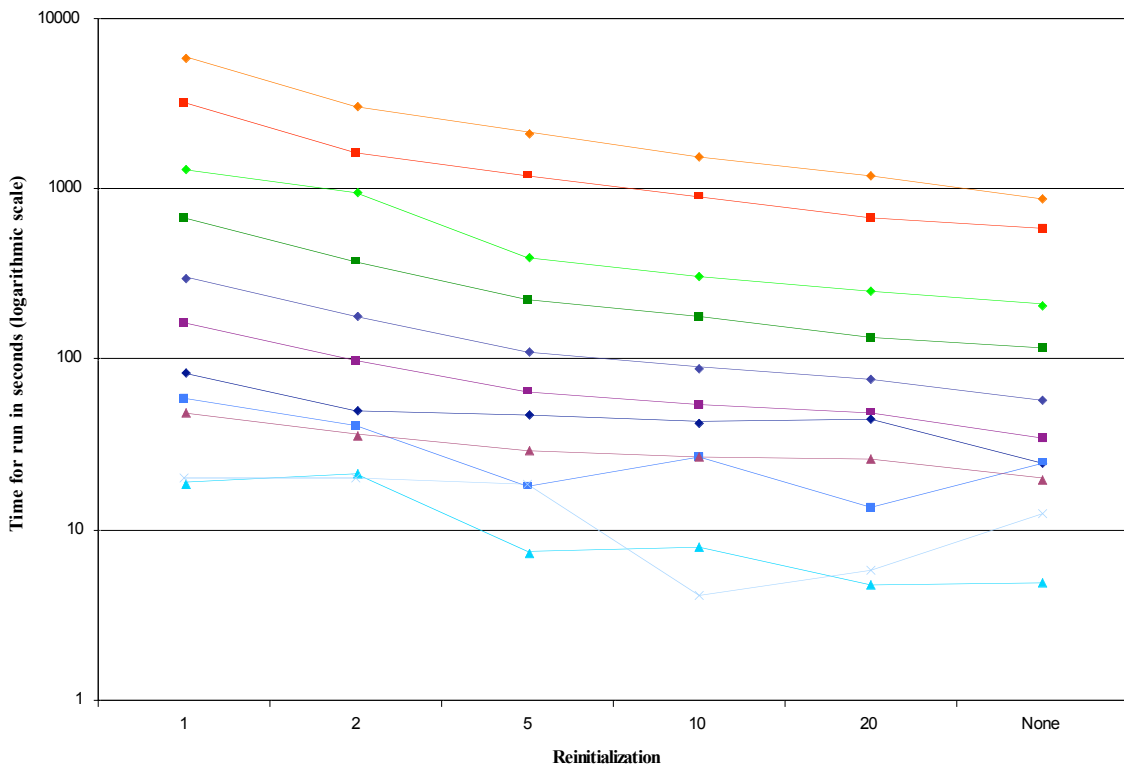


Fig 5. Time per simulation comparison chart

The results presented above may also suffer errors due to the method with which the area was calculated. For every negative value within the grid a square with an area of $\Delta x \Delta y$ was added. The results confirmed most of the concepts presented above.

The results showed that as the gridsize increases much better results appear but this is done at the cost of time as both the number of points increases and the timestep decreases as a requirement of the Courant factor. If the timestep does not decrease inversely proportional to the gridsize the Courant number increases and that can cause the simulations to 'blow up' (while the simulations with grid size 50 and timestep .01 give useful results the simulations for gridsizes 100, 200 and 400 with the same timestep blow up).

For every grid size and timestep there was a significant difference between the simulations done with and without reinitialization. The simulations showed the importance of reinitialization as these simulations presented about 50% less loss in volume after the introduction of reinitialization. One unexpected result was the appearance of slightly better solutions with reinitialization at lower time intervals but this could be explained by cancellations of the errors introduced by the first order accurate scheme and the reinitialization scheme.

4 Conclusions

The implementation of level set methods within Cactus has been successfully done even though it is only a first step since the final implementation must provide full volume conservation. The need for higher order accurate schemes is obvious from the errors which appear within the calculation at low grid sizes with 79-34% loss in volume. Also the very high time achieved by the relatively high grid size, ~6000 seconds from figure 5 which is approximately 1 hour and 40 minutes shows the need for parallelization of this algorithm in order to reduce the time on any given simulation.

5 Present and future work

Currently the status of the implementation of level set methods within Cactus has become more complex to include greater customization of the functions and to allow for greater flexibility. The implementation itself has been slowly modified and has been brought closer to a parallel version which should be available soon. The partial derivatives have also been implemented with higher accuracy 2nd order with the comment that these properties have not yet been tested. Although this is an improvement from first order the discretizations will be increased to present up to 5th order accuracy in the future.

After these problems are successfully included into Cactus the next step will be to pass on from externally generated flows to other types of motions, for example motion within the normal direction or curvature driven flow. Also more advanced level set methods like particle level set methods will be implemented to prevent changes in the shape of the fluids which appear in normal level set methods.

6 Acknowledgments

I would first like to acknowledge the excellent support provided by Dr. Gabrielle Allen and the Center for Computation and Technology at Louisiana State University by providing me with all necessary computational devices. I would especially like to thank Yaakoub El Khamra for all the help provided with Cactus in debugging the sometimes difficult and cryptic thorns that I wrote, Mayank Tyagi for helping me with my knowledge of level set methods and explaining all the theoretical concepts within this problem and Kathryn Traxler for the help in organizing the NCUR effort.

Second I would like to acknowledge the financial support provided Dr. S.S. Iyengar and the department of Computer Science at LSU. Financial support was also provided by the local Association for Computer Machinery (ACM) organization at LSU and the Student Government at LSU.

References

1. Stanley Osher and Ronald Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Applied Mathematical Sciences 153, (2003, Springer-Verlag New York, Inc.)
2. J.A. Sethian, *Level Set Methods and Fast Marching Methods*, (2002, Cambridge University Press)
3. Yaakoub El Khamra, *Introduction to Cactus*, www.cct.lsu.edu/personal/yye00/cactus_tutorial.pdf
4. M. Hermann, *A domain decomposition parallelization of the Fast Marching Method*, (2003, *Annual Research*

Briefs, Stanford: Center for Turbulence Research)

5. Douglas Enright, Frank Losasso and Ronald Fedkiw, *A Fast and Accurate Semi-Lagrangian Particle Level Set Method*, 2004
6. Mark Sussman and Emad Fatemi, *An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow*, SIAM J. SCI. COMPUT. Vol.20, No.4, pp. 1165-1191
7. Todd F. Dupont and Yingjie Liu, *Back and forth error compensation and correction methods for removing errors induced by uneven gradients of level set function*, J. Comput. Phys., v190/1, pp311-324, 2003
8. Ruben Scardovelli and Stéphane Zaleski, *Direct numerical simulation of free-surface and interface flow*, *Annual Reviews Fluid Mech.* 1999. 31:567-603
9. J.A. Sethian and Peter Smereka, *Level Set Methods for Fluid Interfaces*, *Annual Reviews Fluid Mech.* 2003. 35:341-72, doi: 10.1146/annurev.fluid.35.101101.161105
10. Mark Sussman, Emad Fatemi, Peter Smerka and Stanley Osher, *An Improved Level Set Method for Incompressible Two-Phase Flows*, *Computers and Fluids*, vol.27, Nos 5-6 (1998), pp 663-680
11. Alexandru Telea and Anna Vilanova, *A Robust Level-Set Algorithm for Centerline Extraction*, IEEE TCVG Symposium on Visualization (2003)
12. Rüdiger Westermann, Christopher Johnson and Thomas Ertl, *A Level-Set Method for Flow Visualization*, *Proceedings of IEEE Visualization'00*, pages 147-152, 550. IEEE, 2000
13. Yootai Kim, Raghu Machiraju and David Thompson, *Rough Interface Reconstruction Using the Level Set Method*, *Proceedings of IEEE Visualization 2004*, pp 251-258, 2004
14. Ken Museth, David E. Breen, Ross T. Whitaker and Alan H. Barr, *Level Set Surface Editing Operators*, 2002 ACM 1-58113-521-1/02/0007
15. Ian Mitchell, Doug Enright, Ronald Fedkiw and Joel Ferziger, *The particle Level Set Method*, 2004
16. *Cactus 4.0 Users' Guide* <http://www.cactuscode.org/old/Guides/Stable/UsersGuide/UsersGuideStable.pdf>, 2004
17. *Cactus 4.0 Reference Manual* <http://www.cactuscode.org/old/Guides/Stable/ReferenceManual/ReferenceManualStable.pdf>, 2004