Local Level Set Method in High Dimension and Codimension

Chohong Min*

chohong@math.ucla.edu
Department of Mathematics
University of California, Los Angeles
Los Angeles, CA 90095

18th November 2003

Abstract

A new method is presented for capturing an interface of arbitrary dimension and codimension. The method significantly reduces the computational expense of the level set method especially in high codimensions by localizing the level set method near the interface. A high dimensional tree structure is used for the localization. An interface is implicitly represented as the zero level set of a vector valued function. By using a Lipschitz stable interpolation and semi-Lagrangian scheme, our method is stable under both the maximum norm and the Lipschitz semi norm. Due to this stability, the method does not need to reinitialize a level set function. Several numerical examples with high codimension are successfully tested.

1 Introduction

The *level set method* in [10] has been a successful tool for simulating a moving interface of codimension one, because of its simplicity and efficiency. Its successful applications include multiphase flows [4, 7], surface reconstructions [5, 6], and image processing [3, 2]. One drawback of the level set method is its high computational expense because it expands the domain of computation from the interface to a grid of one higher dimension. To reduce the cost, two main approaches have been employed. One approach is to restrict the domain locally at the interface [13, 25]. The other approach uses a tree structure, a so called *quadtree* in $\mathbb{R}^2$ and an *octree* in $\mathbb{R}^3$. This is global, but puts more weight near the interface [12, 1].

*Research supported by ONR Grant N00014-03-1-0071 and NSF Grant DMS-0312222

The level set method of codimension one [10] was theoretically extended to simulate a moving interface of arbitrary codimension [9], where an interface is implicitly represented as the zero level set of a scalar valued function, and applied e.g., to a medical active contouring of codimension two [14]. But because of instability of isosurfacing, the method in [9] was modified in [26] such that an interface is represented as the zero level set of a vector valued function. The modified method applied to a moving curve in $\mathbb{R}^3$ with geometry dependent speed [26] and recently to a capturing one dimensional wavefronts in $\mathbb{R}^3$ and two dimensional wavefronts in $\mathbb{R}^5$ [15].

As the codimension of an interface becomes larger, a drawback of the level set method also becomes larger by its expanding the domain of computation from the interface to a grid of higher dimension. It is the purpose of this paper to introduce a numerical method overcoming this drawback, while keeping the simplicity and versatility of the level set method. Our method stems from [12] in using a tree structure and a semi-Lagrangian scheme, and generalizes [12] in capturing an interface of arbitrary codimension whose initial level set function is Lipschitz continuous.

The key concept of our method is to put more weight near the interface and save memory and effort away from the interface by using a dyadic tree structure. Another key concept is to make the building blocks in our method Lipschitz stable, because the sampling process in our method heavily relies on the Lipschitz constant of a vector valued function.

An sampling algorithm is presented in Section 2 that puts more weight near the interface from a vector valued function. In Section 3, an interpolation algorithm reconstructs a Lipschitz continuous function from the sampled function. In Section 4, the evolution algorithm of a level set function is introduced that is a combination of the sampling algorithm in Section 2 and the interpolation algorithm in Section 3. For the visualization of the level set of a sampled discrete function, an isosurfacing of a sampled function is discussed in Section 5. A practical implementation of our method is given in Section 6. Our method is tested with several examples in Section 7.

2 Sampling

A sampling process is presented in this Section, which constructs a discrete vector-valued function from a Lipschitz-continuous vector-valued function. The process is hierarchically executed on a nested grid from a coarse grid to finer grids.

Let a vector valued function $\vec{\phi} : \mathbb{R}^d \to \mathbb{R}^m$ be Lipschitz continuous in the following norm

$$Lip(\vec{\phi}) = \sup_{x \neq y \in \mathbb{R}^d} \frac{\left\| \vec{\phi}(x) - \vec{\phi}(y) \right\|_\infty}{\|x - y\|_2} < \infty$$

Let $\Gamma \subset \mathbb{R}^d$ be the zero level set of $\vec{\phi}$. A d-cube G^0 is chosen, which is big

enough to contain Γ. G^0 is decomposed into 2^d number of subcubes of equal size. Let G^1 be the set of the subcubes. By a recursive subdivision of G^0, a set of nested grids $\{G^l\}_{l=0,\ldots,l_{max}}$ is generated. Figure 1 illustrates a set of nested grids in $\mathbb{R}^2$. Each G^l consists of 2^l number of cubes, and G^{l+1} is a refinement of G^l. Given a $C \in G^l$, let us denote its parent in G^{l-1} by $prnt(C)$, the set of children cubes of C in G^{l+1} by $children(C)$, the diagonal size by $diag(C)$, the set of neighboring cubes by $ngbd(C)$, and the set of its vertices by $vert(C)$. Finally $trivert(C)$ denotes the set of all centroids of C of dimension k, $0 \leq k \leq d$: when $C = [0,1]^d$, $vert(C) = \{0,1\}^d$ and $trivert(C) = \left\{0, \frac{1}{2}, 1\right\}^d$.

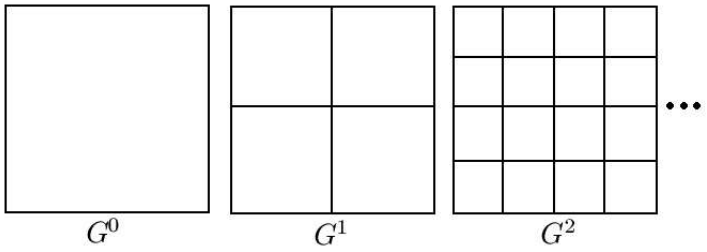

Figure 1: nested grids of $\mathbb{R}^2$

With a constant $L \geq Lip(\vec{\phi})$, Algorithm 1 samples a discrete function $R\vec{\phi}$: $D \to \mathbb{R}^c$, $D \subset \mathbb{R}^d$ from the continuous function $\vec{\phi} : \mathbb{R}^d \to \mathbb{R}^c$.

Algorithm 1 Sampling

1. $C = G^0$ and $D = \emptyset$
2. $D = D \cup trivert(C)$
3. *if* $\eta(C)$ *is true*
4. *for all* $C' \in children(C)$
5. go to 2 with $C = C'$

The splitting condition $\eta(C)$ is given by

$$\eta(C) : \min_{v \in trivert(C)} \left\| \vec{\phi}(v) \right\|_\infty \leq L \cdot \frac{diag(C)}{4}$$

Algorithm 1 can be simply expressed as "*split any cube satisfying* η". Here follows a proposition explaining the properties of the sampling algorithm.

Proposition 2.1 *D contains trivertices of every C intersecting Γ.*

Proof Let $w \in C \cap \Gamma$, then there exist a $v \in trivert(C)$ such that $\|v - w\|_2 \leq \frac{diag(C)}{4}$, for the maximum distance in a cube is its diagonal size. By the as-

sumption that $L \geq Lip(\vec{\phi})$,

$$\begin{aligned} \left\| \vec{\phi}(v) \right\|_\infty = \left\| \vec{\phi}(v) - \vec{\phi}(w) \right\|_\infty &\leq Lip \cdot \|v - w\|_2 \\ &\leq L \cdot \tfrac{diag(C)}{4} \end{aligned}$$

□

Therefore $\eta(C)$ is true. If $C \cap \Gamma \neq \emptyset$, then $prnt(C) \cap \Gamma \neq \emptyset$, because $prnt(C)$ contains C. Also every ancestor of C intersects Γ and satisfies the splitting condition η. Since the algorithm 1 recursively constructs from an ancestor to its successor satisfying η, D includes the trivertices of C.

By the above Proposition, the highest resolution is guaranteed near Γ. If every component ϕ_i of $\vec{\phi}$ is a signed distance function to its zero level set $\left\{ x \in \mathbb{R}^d \mid \phi_i(x) = 0 \right\}$, then the splitting condition will imply that the distance of C is smaller than one fourth of its size. Therefore the memory space in a grid G^l is about the order of Γ which is $(d-c)$ dimensional. Let N be the number of grid points in $G^{l_{max}}$, then $\left| D \cap G^l \right| = O(N^{d-c})$. Combining all nested grids from $l = 0, \cdots, log(N)$, the total memory size, $|D|$ will be $O(N^{d-c} \cdot log(N))$. Compared to the memory size $O(N^d)$ of the uniform sampling, algorithm 1 significantly saves memory, while maintaining its highest resolution near the interfaces by Proposition 2.1.

3 Interpolation

In Section 2, we discussed a sampling procedure from continuous functions to discrete functions. Here we discuss the reverse, an interpolation procedure from discrete to continuous. On a uniformly sampled function, piecewise polynomial interpolation has been one of the best ways to achieve both accuracy and efficiency. However its direct use on a dyadic grid would lead to Lipschitz instability. Figure 2 shows a case of Lipschitz instability when piecewise multi-linear interpolation is used on a dyadic grid in $\mathbb{R}^2$. Here we define the Lipschitz constant of a discrete function $\vec{\psi} : D \to \mathbb{R}^m$ with $D \subset \mathbb{R}^d$ as

$$Lip(\vec{\psi}) = \max_{x \neq y \in D} \frac{\left\| \vec{\psi}(x) - \vec{\psi}(y) \right\|_\infty}{\|x - y\|_2}$$

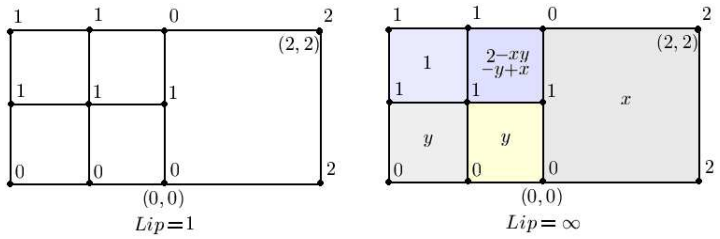

Figure 2: Lipschitz instability of multi-linear interpolation on a dyadic grid

In Section 4, we shall introduce a semi-Lagrangian scheme for local level set evolution, which is a combination of the dyadic sampling, approximate solution operator, and the interpolation. The whole process of the scheme needs to be Lipschitz stable. For that purpose, we present a Lipschitz stable interpolation on a dyadic grid.

3.1 Complete Barycentric Subdivision

Let $\vec{\psi} : D \to \mathbb{R}^c$ with $D \subset \mathbb{R}^d$ be a dyadically sampled function. Since the sampling algorithm recursively subdivides the cube G^0, the domain D is the collection of tri-vertices of some cubes $\{C_i\}$, which is a subdivision of G^0. We refine the subdivision $\{C_i\}$ of G^0 into a triangulation of G^0 by applying the complete barycentric subdivision [19, 21]. Let us give the centroids of each k-dimensional face of C_i the label k, $0 \le k \le d$. A *pulling* of a point on a subdivision is an algorithm refining the subdivision by the point [21, 22]. By pulling the centroids in order of nonincreasing label, the cubic subdivision $\{C_i\}$ is refined to a simplicial subdivision, triangulation, $\{T_j\}$. Figure 3 shows a cases in $\mathbb{R}^2$.

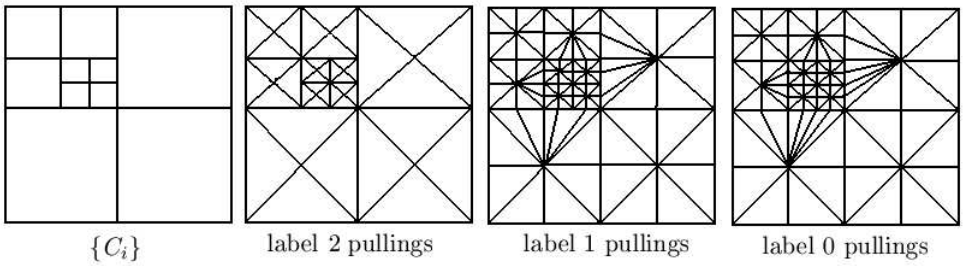

Figure 3: Complete Barycentric Subdivision in $\mathbb{R}^2$

3.2 Barycentric Interpolation

In Section 3.1, the domain $D \subset \mathbb{R}^d$ of $\vec{\psi} : D \to \mathbb{R}^m$ was triangulated into simplices $\{T_j\}$. Based on the triangulation, we define an interpolation, or a

prolongation $\mathcal{P}\vec{\psi} : \mathbb{R}^d \to \mathbb{R}^c$ by

$$\mathcal{P}\vec{\psi}(x) = \begin{cases} \mathcal{P}\vec{\psi}\,|_{T_j}\,(x) & , x \in T_j \\ \mathcal{P}\vec{\psi}(x^*) & , x \neq G^0 \end{cases}$$

Here x^* denotes the nearest point in G^0 to x, which is well defined because G^0 is convex. Figure 4 shows an example in $\mathbb{R}^2$. $\mathcal{P}\vec{\psi}\,|_{T_j}$ is defined as the unique linear interpolant of $\vec{\psi}$ on the simplex T_j. The interpolation $\mathcal{P}$ is Lipschitz stable by the following proposition.

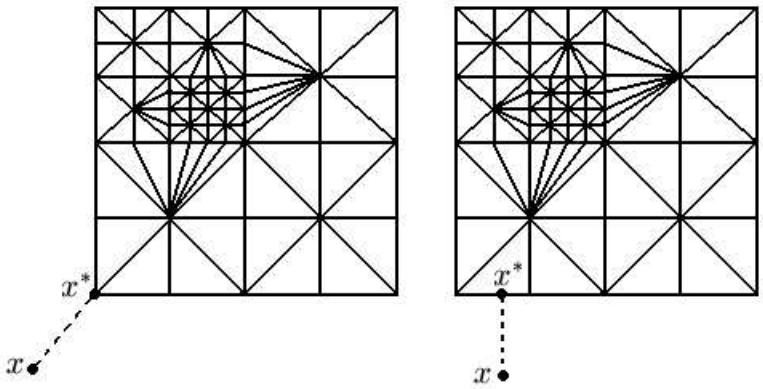

Figure 4: Projection of x outside into x^* inside the domain

Proposition 3.1 $Lip(\mathcal{P}\vec{\psi}) = Lip(\psi)$

Proof Since $\mathcal{P}\vec{\psi}$ is a first order polynomial on each T_j, $Lip\left(\mathcal{P}\vec{\psi}\,|_{T_j}\right) \leq Lip(\psi)$. On $T_j \cap T_{j'}$, $\mathcal{P}\vec{\psi}$ has two possible definitions $\mathcal{P}\vec{\psi}\,|_{T_j}$ and $\mathcal{P}\vec{\psi}\,|_{T_{j'}}$. Since $\{T_j\}$ is a triangulation, $T_j \cap T_{j'}$ is also a simplex. Since there exits only one linear interpolant on a simplex, the two definitions must be the same. Therefore $\mathcal{P}\vec{\psi}$ is continuous on G^0. Since $Lip\left(\mathcal{P}\vec{\psi}\,|_{T_j}\right) \leq Lip(\psi)$ and $\mathcal{P}\vec{\psi}$ is continuous on G^0, $Lip\left(\mathcal{P}\vec{\psi}\,|_{G^0}\right) \leq Lip(\psi)$.

For any $x, y \in \mathbb{R}^d$, $\|x^* - y^*\|_2 \leq \|x - y\|_2$, since G^0 is a cube. And,

$$\left\|P\vec{\psi}(x) - P\vec{\psi}(y)\right\|_\infty = \left\|P\vec{\psi}(x^*) - P\vec{\psi}(y^*)\right\|_\infty \leq Lip\left(\mathcal{P}\vec{\psi}\,|_{G^0}\right) \cdot \|x - y\|_2$$

so, $Lip(\mathcal{P}\vec{\psi}) \leq Lip(\vec{\psi})$. Since the domain of $\vec{\psi}$ is a subset of the domain of $P\vec{\psi}$, $Lip(\mathcal{P}\vec{\psi}) = Lip(\vec{\psi})$.□

4 Evolution

In this Section, we present an algorithm numerically capturing an interface in high dimension and codimension. Let us assume an interface $\Gamma \subset \mathbb{R}^d$ of codimension m moving with velocity $\vec{V} : \mathbb{R}^d \to \mathbb{R}^d$. It can be implicitly represented as the zero level set of $\vec{\phi}(x,t) : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^m$, each component of the $\vec{\phi}$ satisfies the so-called level set equation

$$\frac{\partial}{\partial t}\phi_i + \vec{V} \cdot \nabla \phi_i = 0$$

for $i = 1, \ldots, m$. With a uniform grid, the level set equation is usually discretized with higher order accurate ENO or WENO schemes in space and the Runge-Kutta methods in time. But in a dyadic grid, neighboring points may not exist, which makes it hard to apply conventional finite difference schemes. Because of that, a semi-Lagrangian scheme has been used for discretizing the level set equation on a dyadic grid when codimension is one [12]. We extend the techniques of [12] to higher dimension and codimension.

Given $\vec{\phi}^n : D^n \to \mathbb{R}^c$, the next level function $\vec{\phi}^{n+1} : D^{n+1} \to \mathbb{R}^c$ is given by

$$\vec{\phi}^{n+1} = \mathcal{R}\mathcal{S}\mathcal{P}\vec{\phi}^n,$$

where $\mathcal{P}$ is the interpolation operator in Section 3, S is an approximate solution operator, and $\mathcal{R}$ is the dyadic sampling operator in Section 2. Via the characteristic curve of the level set equation, $\mathcal{S}$ is equivalent to an ODE solver to the backward characteristic ODE, $\dot{x} = -\vec{V}(x,t)$. Now we define $\mathcal{S}$ as the Euler scheme for an ODE solver, and we have

$$(\mathcal{S}\mathcal{P}\vec{\phi}^n)(x) = (\mathcal{P}\vec{\phi}^n)(x - \Delta t \vec{V}(x, t^n))$$

To safely dyadically sample $\vec{\phi}^{n+1}$ from $\mathcal{S}\mathcal{P}\vec{\phi}^n : \mathbb{R}^d \to \mathbb{R}^m$, we need an estimate on the Lipschitz constant of $\mathcal{S}\mathcal{P}\vec{\phi}^n$, which is given in the following proposition.

Proposition 4.1 *If* $Lip(\vec{\phi}^n) \le L$, *then*

$$Lip(\mathcal{S}\mathcal{P}\vec{\phi}^n) \le L \cdot \left(1 + \Delta t \cdot \max_{1 \le i \le d} \sup_{x \in \mathbb{R}^d} \left\|\nabla v^i(x, t^n)\right\|_2\right)$$

Proof By the definition of S and P,

$$\begin{aligned}
\left\|\mathcal{S}\mathcal{P}\vec{\phi}^n(x) - \mathcal{S}\mathcal{P}\vec{\phi}^n(y)\right\|_\infty &= \left\|P\vec{\phi}^n\left(x - \Delta t\vec{V}(x,t^n)\right) - P\vec{\phi}^n\left(y - \Delta t\vec{V}(y,t^n)\right)\right\|_\infty \\
&\le Lip\left(P\vec{\phi}^n\right) \cdot \left\|x - y + \Delta t\left(\vec{V}(x,t^n) - \vec{V}(y,t^n)\right)\right\|_2 \\
&\le Lip\left(\vec{\phi}^n\right) \cdot \left(\|x-y\|_2 + \Delta t \cdot \left\|\vec{V}(x,t^n) - \vec{V}(y,t^n)\right\|_2\right) \\
&\le L \cdot \left(\|x-y\|_2 + \Delta t \cdot \sqrt{d} \cdot \left\|\vec{V}(x,t^n) - \vec{V}(y,t^n)\right\|_\infty\right)
\end{aligned}$$

Applying the mean value theorem to each component v^i of $\vec{V}$,

$$\left\| \vec{V}(x,t^n) - \vec{V}(y,t^n) \right\|_\infty \leq \max_{1\leq i\leq d} \sup_{x\in\mathbb{R}^d} \left\| \nabla v^i(x,t^n) \right\|_2 \cdot \|x-y\|_2$$

The proposition follows from the above two inequalities. □

The Lipschitz estimate of the above Proposition is a worst-case analysis. Practically the following weak estimate was enough in the test examples in Section 7.

$$L^{n+1} = L^n \cdot \left(1 + \frac{\Delta t}{8} \cdot \max_{1\leq i\leq d} \sup_{x\in\mathbb{R}^d} \left\| \nabla v^i(x,t^n) \right\|_2 \right)$$

Here, L^n and L^{n+1} denote the constant in the splitting condition η of $\vec{\phi}^n$ and $\vec{\phi}^{n+1}$ in Section 2.

5 Isosurfacing and Visualization

Given a dyadically sampled function $\vec{\psi} : D \to \mathbb{R}^m$ with $D \subset \mathbb{R}^d$, we discuss its isosurfacing and visualization in this Section. Since D is uniform near $\left\|\vec{\psi}\right\|_\infty$ is small, by Proposition 2.1, it would be efficient to apply isosurfacing algorithms in uniform grid to the zero isosurfacing of $\vec{\psi}$, such as [16] or [20]. Since $\vec{\psi}$ is dyadically sampled by algorithm 1, D is the set of tri-vertices of some cubes $\{C_i\}$. Each C_i is subdivided into 2^d number of subcubes $C_{i,j}$, $1 \leq j \leq 2^d$; for example, $[-1,1]^d = \cup_{a\in\{0,1\}^d} \left(a + [-1,0]^d\right)$. Based on the isosurfacing algorithm of [16], we have the isosurfacing algorithm 2 of $\vec{\psi}$.

Algorithm 2 Simplicial Isosurfacing of $\vec{\psi} : D \to \mathbb{R}^c$

```
1.   Γ = ∅
2.   C = G^0
3.   if C is branch
4.       for each C' ∈ children(C)
5.           go to 2 with C = C'
6.   else
7.       for each subcubes C_j of C, 1 ≤ j ≤ 2^d
8.           for each simplex S_k in the Kuhn Triangulation of C_j, 1 ≤ k ≤ d!
9.               Triangulate the intersection of S_k and {ψ |_{S_k} = 0}
10.              S_k ∩ {ψ |_{S_k} = 0} = ∪T_l
11.              Γ = Γ ∪ (∪_l T_l)
```

For details of Kuhn triangulation and the triangulation of the intersection of a simplex and hyperplanes, see [16]. So, the zero isosurface Γ of $\vec{\psi}$ is constructed as the union of $(d-m)$ dimensional simplices.

When $d = m+1$, $\Gamma \subset \mathbb{R}^d$ is a curve and can be visualized as a union of line segments under a projection map $\pi(x_1,\ldots,x_d) = (x_1,x_2,x_3)$. Also when

$d = m+2$, Γ is a surface in $\mathbb{R}^d$ and can be visualized as a union of triangles under the projection π. Both cases are visualized using usual 3D graphic libraries like OpenGL. To give shadings to the triangulations, the oriented normal vector field $\vec{n}$ of $\pi(\Gamma) \subset \mathbb{R}^3$ is given in [16] :

$$\vec{n} = \begin{vmatrix} \nabla\psi_1 & \nabla\psi_1 \cdot e_4 & \cdots & \nabla\psi_1 \cdot e_d \\ \vdots & \vdots & & \vdots \\ \nabla\psi_c & \nabla\psi_c \cdot e_4 & \cdots & \nabla\psi_c \cdot e_d \end{vmatrix}$$

Here e_j denotes the canonical base of $\mathbb{R}^d$ and $\nabla\psi_i$ is given by divided difference formula for each simplex [16], and the determinant is expanded with cofactors.

$$\begin{aligned} \vec{n} &= \nabla\psi_1 \cdot \begin{vmatrix} \nabla\psi_2 \cdot e_4 & \cdots & \nabla\psi_2 \cdot e_d \\ \vdots & & \vdots \\ \nabla\psi_c \cdot e_4 & \cdots & \nabla\psi_c \cdot e_d \end{vmatrix} \\ &\vdots \\ &+ (-1)^c \cdot \nabla\psi_m \cdot \begin{vmatrix} \nabla\psi_1 \cdot e_4 & \cdots & \nabla\psi_1 \cdot e_d \\ \vdots & & \vdots \\ \nabla\psi_{c-1} \cdot e_4 & \cdots & \nabla\psi_{c-1} \cdot e_d \end{vmatrix} \end{aligned}$$

Since determinants vanish when two columns are equal, $\vec{n} : \mathbb{R}^d \to \mathbb{R}^3$ and $\vec{n}$ is the oriented normal vector field of $\pi(\Gamma)$[16].

6 Implementation

We discuss an implementation of our local level set method which consists of dyadic sampling in Section 2, barycentric interpolation in Section 3, and evolution in Section 4. Since the evolution is a combination of dyadic sampling and barycentric interpolation, it is enough to discuss dyadic sampling and barycentric interpolation.

6.1 Dyadic Sampling

Since the sampling algorithm 1 recursively subdivides the coarsest cube $G^0 \in \mathbb{R}^d$, 2^d branched tree is a natural choice for the data structure of the dyadic sampling. Possible implementations of this tree structure are thoroughly discussed in [23, 24]. Among them, *region based tree* and *matrix based tree* are the most appropriate for our purpose. The region based tree is fast for interpolation but requires more memory for construction, while the matrix based tree is slow for interpolation but requires less memory for construction. Each point in a region based tree is multiply defined, possibly 2^d times, so it is hard to modify the tree. But, each point in matrix-based tree is singly defined, so it becomes easy to modify. Our algorithms, which consists of dyadic sampling and barycentric

interpolation, do not need any modification, but only need new creations of dyadically sampled functions. For these reasons, we take a region-based tree as a choice of the tree implementation.

Unlike the uniform sampling, we can not predict the memory size of the dyadic sampling before doing it. Therefore the programming languages with dynamic memory allocations are preferred, such as C++ or Java. We have chosen C++ for better performance. An implementation of region-based tree is given in the followings when $d = 5$ and $m = 3$.

```
struct Node {
    int i1,i2,i3,i4,i5;
    int size;
    bool is_leaf;
    Node* prnt;
};


struct Leaf : public Node {
    float vs1[3][3][3][3][3];
    float vs2[3][3][3][3][3];
    float vs3[3][3][3][3][3];
};


struct Branch : public Node {
    Node* children[2][2][2][2][2];
};
```

6.2 Barycentric Interpolation on a Cube

Any cube $C \subset \mathbb{R}^d$ is affinely isomorphic to $[-1,1]^d$ under translations and scalings. So, we need only to discuss barycentric interpolation on $[-1,1]^d$, and the general case will follow by the affine isomorphism. The set of tri-vertices of $[-1,1]^d$ is $\{-1,0,1\}^d$. Given $\vec{\psi} : \{-1,0,1\}^d \to \mathbb{R}^c$, its barycentric interpolation $\mathcal{P}\vec{\psi} : [-1,1]^d \to \mathbb{R}^c$ is given by the following algorithm, which is an extension of that found in [17]. Given $x \in [-1,1]^d$, coordinates of x are sorted with a permutation J of $\{1, \cdots, d\}$ such that $1 \geq \left|x_{J(1)}\right| \geq \cdots \geq \left|x_{J(d)}\right| \geq 0$. Define $P_0 = \vec{0} \in \mathbb{R}^d$ and $P_i = P_{i-1} + sgn\left(x_{J(i)}\right) \cdot e_{J(i)}$ for $i = 1, \cdots, d$, where e_j is the canonical j^{th} unit vector and $sgn(y)$ is the signum of y whose possible values are $\{-1,0,1\}$. Then a barycentric decomposition of x is given:

$$\begin{aligned} x &= \sum_{i=1}^{d} x_i e_i \\ &= \sum_{i=1}^{d} x_{J(i)} e_{J(i)} \\ &= \sum_{i=1}^{d} \left[x_{J(i)} \cdot sgn\left(x_{J(i)}\right) \cdot e_{J(i)}\right] \\ &= \sum_{i=1}^{d-1} \left(\left|x_{J(i)}\right| - \left|x_{J(i+1)}\right|\right) \cdot P_i + \left|x_{J(d)}\right| \cdot P_d + \left(1 - \left|x_{J(1)}\right|\right) \cdot P_0 \end{aligned}$$

Since $P\vec{\psi}$ is a linear interpolant, we have the following formula.

$$P\vec{\psi}(x) = \sum_{i=1}^{d-1} \left(|x_{J(i)}| - |x_{J(i+1)}| \right) \cdot \vec{\psi}(P_i) + |x_{J(d)}| \cdot \vec{\psi}(P_d) + \left(1 - |x_{J(1)}|\right) \cdot \vec{\psi}(P_0)$$

6.3 Weak Barycentric Interpolation on a Dyadic Grid

By the complete barycentric subdivision in Section 3.1, a cube may be subdivided by any neighborhood sharing a common face of dimension 0 to $d-1$. Hence the subdivision of a cube is affected by 2^d neighbors, and the barycentric interpolation in Section 3.2 should consider 2^d neighbors, which is very expensive when d is large. Since the most influential one of the neighboring cubes is the cube sharing a $(d-1)$ dimensional face, we weaken the barycentric interpolation in Section 3.2 into a barycentric interpolation just taking care of one neighbor.

Let $\vec{\psi} : D \to \mathbb{R}^c$ with $D = \cup_i trivert\,(C_i)$ be a function sampled by the Algorithm 1 in Section 2. Since the interpolation of $\vec{\psi}$ at $x \in \mathbb{R}^d$ is defined as the interpolation of $\vec{\psi}$ at $x^* \in G^0$, we may assume $x \in G^0$. Let C be the smallest cube in $\{C_i\}$ containing x as in Figure 5. In the Figure, A and B denote the vertices of C such that $C = \{A_i \le x_i \le B_i\}$, D is the center of C, and $E \in \partial C$ is the point connecting D and x satisfying the extrapolation relation between D and x; $E = (1-\lambda)D + \lambda x$ with $\lambda \ge 1$. The λ is given by

$$\lambda = \min_{1 \le i \le d} \frac{B_i - D_i}{|x_i - D_i|}$$

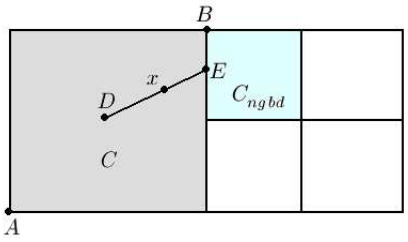

Figure 5: Weak Barycentric Interpolation on a Dyadic Grid

Let C_{ngbd} be the smallest cube in $\{C_i\}$ containing E. Then the weak barycentric interpolation of $\vec{\psi}$ at x is given by

$$\mathcal{P}\vec{\psi}(x) = \begin{cases} \mathcal{P}\vec{\psi}\,|_C\,(x^*) & if \quad C = C_{ngbd} \\ \frac{1}{\lambda} \cdot \mathcal{P}\vec{\psi}\,|_C\,(D) + \left(1 - \frac{1}{\lambda}\right) \cdot \mathcal{P}\vec{\psi}\,|_{C_{ngbd}}\,(E) & else \quad C \neq C_{ngbd} \end{cases}$$

Barycentric interpolations on a cube, $\mathcal{P}\vec{\psi}\mid_C$ and $\mathcal{P}\vec{\psi}\mid_{C_{ngbd}}$ were given in Section 6.2. By taking care of neighboring cubes, the interpolant $\mathcal{P}\vec{\psi}: G^0 \to \mathbb{R}^m$ is continuous between two cubes of different size, when they share a $(d-1)$ dimensional face.

7 Numerical Examples

Every example was implemented in C++ and run on a PC with 2.2GHz CPU and 2GB memory. The sampling algorithm in Section 2 was used to construct a discrete initial condition from a continuous initial data. The level set function was updated by the evolution algorithm in Section 4. The interpolation in the evolution algorithm employed the weak barycentric interpolation in Section 6.3.

7.1 Simulation of Reflections in Geometric Optics in $\mathbb{R}^2$

Geometric Optics is an asymptotic approximation for high frequency wave equation. It is represented as a partial differential equation by the eikonal equation, $u_t + c(x) \cdot \|\nabla u\| = 0$. The classical viscosity solution of the eikonal equation does not permit linear superposition which is naturally permitted in the wave equation. To overcome this deficiency, the wave fronts are embedded in higher dimension to enable multi-valued solutions and superpositions [15]. In this example, wavefronts are initially a circle of center at $(0,0)$ and of radius 0.5, and reflect at the boundary of $[-1,1]^2$. The initial condition $\vec{\phi}_0$ and the velocity field $\vec{V}$ are given by

$$\vec{\phi}_0(x,y,\theta) = \begin{pmatrix} x - 0.5 \cdot \cos(\pi\theta) \\ y - 0.5 \cdot \sin(\pi\theta) \end{pmatrix}$$

$$\vec{V}(x,y,\theta) = \begin{pmatrix} \cos(\pi\theta) \\ \sin(\pi\theta) \\ 0 \end{pmatrix}$$

respectively. The level set function $\vec{\phi}(x,y,\theta,t)$ is updated by

$$\vec{\phi}_t + (\vec{V} \cdot \nabla)\vec{\phi} = 0$$

Time step Δt was chosen as $\Delta t = \frac{\sqrt{\Delta x^2 + \Delta y^2 + \Delta\theta^2}}{2}$. The Lipschitz constants of $\vec{\phi}(x,y,\theta,t^n)$ was increased by $L^{n+1} = L^n \cdot \left(1 + \Delta t \cdot \frac{\pi}{8}\right)$ with $L^0 = \sqrt{1 + 0.25 \cdot \pi^2}$. The evolution was simulated until $T = 1.5$.

	Local Level Set Method				Level Set Method	
grid	space(MB)	rate	time(sec)	rate	space(MB)	rate
128^3	61.4	-	250	-	33.6	-
256^3	138	2.2	1160	4.6	268	8.0
512^3	290	2.1	5045	4.3	2147	8.0
1024^3	594	2.0	21068	4.2	17179	8.0

Table 1: Reflections in $\mathbb{R}^2$

7.2 Simulation of Reflections in Geometric Optics in $\mathbb{R}^3$

In this example, wavefronts are initially a circle of center at $(0,0,0)$ and of radius 0.8, and reflect at the boundary of $[-1,1]^3$. The initial condition $\vec{\phi}_0$ and the velocity field $\vec{V}$ are given by

$$\vec{\phi}_0(x,y,z,\theta_1,\theta_2) = \begin{pmatrix} x - 0.5\cdot\cos(\pi\theta_1)\cdot\sin(\pi\theta_2) \\ y - 0.5\cdot\sin(\pi\theta_1)\cdot\sin(\pi\theta_2) \\ z - \cos(\pi\theta_2) \end{pmatrix}$$

$$\vec{V}(x,y,z,\theta_1,\theta_2) = \begin{pmatrix} \cos(\pi\theta_1)\cdot\sin(\pi\theta_2) \\ \sin(\pi\theta_1)\cdot\sin(\pi\theta_2) \\ \cos(\pi\theta_2) \\ 0 \\ 0 \end{pmatrix}$$

respectively. The level set function $\vec{\phi}(x,y,z,\theta_1,\theta_2,t)$ is updated by

$$\vec{\phi}_t + (\vec{V}\cdot\nabla)\vec{\phi} = 0$$

Time step Δt was chosen as $\Delta t = \frac{\sqrt{\Delta x^2+\Delta y^2+\Delta\theta_1^2+\Delta\theta_2^2}}{2}$. The Lipschitz constants of $\vec{\phi}(x,y,z,\theta_1,\theta_2,t^n)$ was increased by $L^{n+1} = L^n\cdot\left(1+\Delta t\cdot\frac{\pi}{8}\right)$ with $L^0 = \sqrt{1+0.8\cdot 0.8\cdot\pi^2}$. The evolution was simulated until $T = 0.15$.

	Local Level Set Method				Level Set Method	
grid	space(MB)	rate	time(sec)	rate	space(MB)	rate
8^5	3.0	-	3	-	0.4	-
16^5	53	17.7	52	17.4	12.6	32.0
32^5	284	5.4	240	4.0	403	32.0
64^5	1092	4.2	4257	17.7	12885	32.0

Table 2: Reflections in $\mathbb{R}^3$

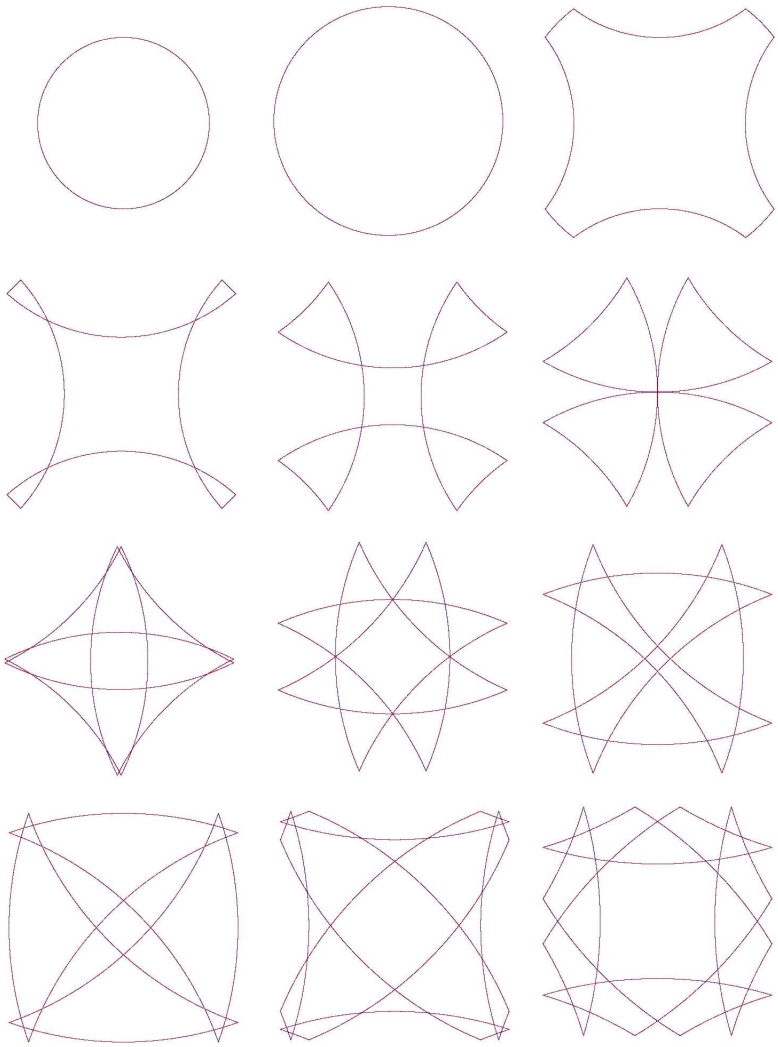

Figure 6: Reflections in $\mathbb{R}^2$ simulated on a 1024^3 grid

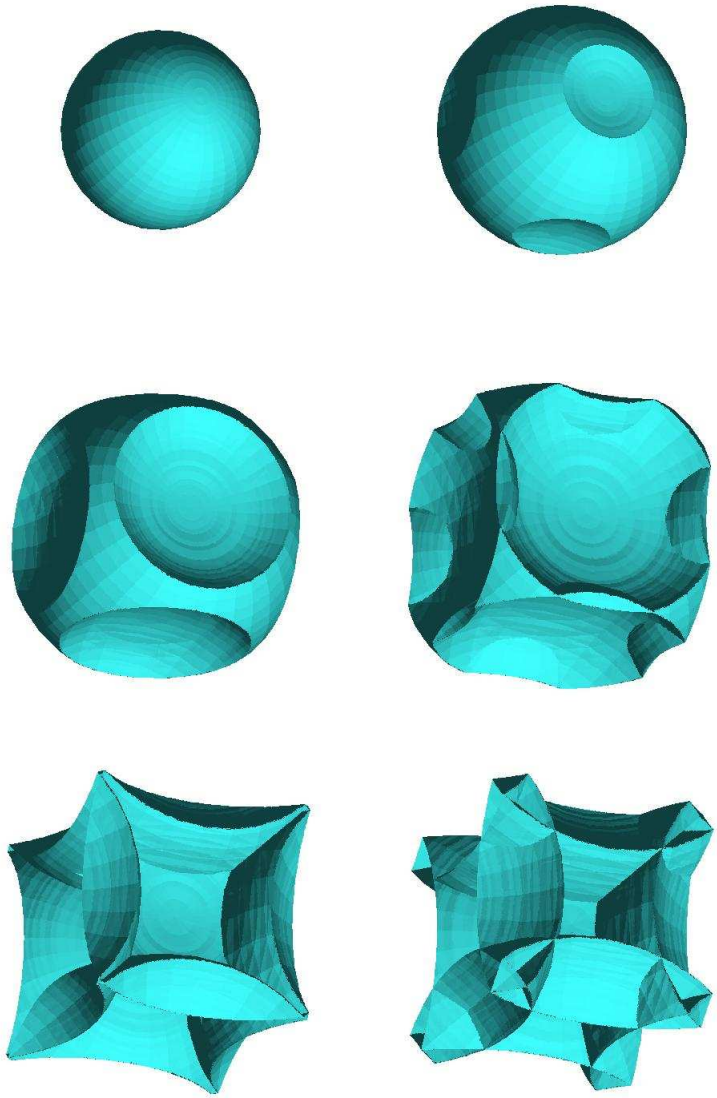

Figure 7: Reflections in $\mathbb{R}^3$ simulated on a 32^5 grid until 1.0 sec

8 Conclusion

We have introduced a new method that can track an interface in arbitrary dimension and codimension. By localizing the level set method near the interface, our new method significantly reduced the high computational expense of the level set method, while keeping the simplicity and efficiency of the level set method. Our method is stable under both the maximum norm and the Lipschitz semi-norm. Due to its formal Lipschitz stability, the method does not need any reinitialization of level set function. However, without reinitializations, the Lipschitz constant of a level set function might keep increasing by the estimate in Section 4, which makes the method a bit less efficient.

In the future, the author expects to employ the reinitialization algorithm in [15, 11] to the tree structure in our method.

References

[1] M. Droske, B. Meyer, M. Rumpf, and C. Schaller, *An adaptive level set method for medical image segmentation*, Lecture Notes in Computer Science IPMI 2001:416-422

[2] T. Chan and L. Vese, *Active contours without edges*, IEEE Trans. on Image Processing 10:266-277 (2001)

[3] L. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, Physica D 60:259-268 (1992)

[4] M. Sussman, P. Smereka, and S. Osher, *A level set approach to computing solutions to incompressible two-phase flow*, J. Compt. Phys. 114:146-159 (1994)

[5] H. K. Zhao, S. Osher, and R. Fedkiw, *Fast surface reconstruction using the level set method*, 1st IEEE Workshop on Variational and Level Set Methods, 8th ICCV, Vancouver 194-202 (2001)

[6] H. K. Zhao, S. Osher, B. Merriman, and M. Kang, *Implicit and nonparametric shape reconstruction from unorganized data using a variational level set method*, Comput. Vision and Image Understanding 80:295-314 (2000)

[7] R. Fedkiw, T. Aslam, and S. Xu, *The ghost fluid method for deflagration and detonation discontinuities*, J. Compt. Phys. 154:393-427 (1999)

[8] H. W. Kuhn, *Some topological lemmas in topology*, IBM J. Res. Develop. 4:518-524 (1960)

[9] L. Ambrosio and M. Soner, *Level set approach to mean curvature flow in arbitrary codimension*, 43:693-737 (1996)

[10] S. Osher and J. Sethian, *Fronts propagating with curvature dependent speed: algorithms based on hamilton-jacobi formulations*, J. Comput. Phys. 79:12-49 (1988)

[11] J. Strain, *Fast tree based redistancing for level set computations*, J. Comput. Phys. 152:648-666 (1999)

[12] J. Strain, *Tree methods for moving interfaces*, J. Comput. Phys. 151:616-648 (1998)

[13] D. Peng, B. Merriman, H. Zhao, S. Osher, and M. Kang, *A PDE Based Fast Local Level Set Method*, J. Comput. Phys. 155:410-438 (1999)

[14] L. Lorigo, O. Faugeras, W. Grimson, R. Keriven, R. Kikinis, A. Nabavi, and C. Westin, *Codimension-two geodesic active contours for the segmentation of tubular structures*, Conf. on Computer Vision and Pattern Recognition, IEEE 2000, 444-451

[15] S. Osher, L.-T. Cheng, M. Kang, H. Shim, and Y.-H. Tsai, *Geometric optics in a phase space based level set and eulerian framework*, J. Comput. Phys. 179:622-648 (2002)

[16] C. Min, *Simplicial isosurfacing in arbitrary dimension and codimension*, J. Comput. Phys. 190:295-310 (2003)

[17] H. W. Kuhn, *Some combinatorial lemmas in topology*, IBM. J. Res. Develop. 4:518-524, (1996)

[18] J. F. Sallee, *The middle cut triangulations of the n-cube*, SIAM J. Alg. Disc. Math. 5:407-419 (1984)

[19] M. M. Bayer, *Barycentric Subdivisions*, Pacific J. Math. 135:1-16 (1988)

[20] C. Weigle and D. C. Banks, *Complex-valued contour meshing*, IEEE Visualization 96 173-180 (1996)

[21] C. W. Lee, *Subdivisions and triangulations of polytopes*, Handbook of Discrete and Computational Geometry, CRC Press LLC, (1997)

[22] C. W. Lee, *Regular triangulations of convex polytopes*, Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift, (1991)

[23] H. Samet, *The design and analysis of spatial data structures*, Addison-Wesley (1990)

[24] H. Samet, *Applications of Spatial Data Structures: computer graphics, image processing and gis*, Addison-Wesley (1990)

[25] D. Adalsteinsson and J. Sethian, *A fast level set method for propagating interfaces*, J. Comput. Phys. 118:269-277 (1995) }

[26] P. Butchard, L.-T. Cheng, B. Merriman, and S. Osher, *Motion of curves in three spatial dimensions using a level set approach*, J. Comput. Phys. 170:720-741 (1999)