Algebraic Adaptive Signed Distance Field on GPU

Lixin Ren
ISCAS & UCAS
Beijing, China
renlxmail@163.com

Shusen Liu
ISCAS
Beijing, China
liuss@ios.ac.cn

Xiaowei He
ISCAS
Beijing, China
xiaowei@iscas.ac.cn

Yuzhong Guo
ISCAS
Beijing, China
guoyuzhong@iscas.ac.cn

Enhua Wu
ISCAS & UCAS
Beijing, China
FST, UM
Macao, China
ehwu@um.edu.mo

Abstract

Signed distance fields (SDFs) are commonly used in solid modeling and physically based animation. However, how to develop high-performance sparse data structures for signed distance field construction and boolean operations is challenging. Our motivation is to develop a representation for adaptive signed distance fields that allows fast construction and boolean operations between any two SDFs, named as the algebraic adaptive signed distance field (AASDF). To guarantee all AASDFs form an algebraic system, a novel hierarchical sparse octree is first presented. A bottom-up fast iterative method is then proposed to calculate the signed distance field based on the hierarchical sparse octree. Boolean operations of union, intersection and difference can also be taken with a similar hierarchical construction algorithm to obtain an adaptive signed distance field belonging to the complete set of AASDFs. Experiments show that our method not only shows performance comparable to state-of-the-art methods in constructing adaptive SDFs on GPU, but also can handle boolean operations between different models at an interactive speed.

Keywords: ***algebraic adaptive signed distance field (AASDF), boolean operations, GPU acceleration***

1. Introduction

A boolean operation, such as union, intersection, or difference, is one of the most important geometric tools in solid modeling and computational geometry. Since most geometric models are represented with surface meshes, a vast amount of work have been done to do boolean operations directly on surface meshes [21]. However, direct boolean operations on surface meshes have been proven to be a notoriously difficult job [37]. As an alternative solid modeling technique, signed distance fields (SDFs) has been commonly used to represent implicit geometric models for surface reconstruction [13], ray tracing [2] and collision detection [22], etc. Boolean operations taken on two SDFs that are approximated with a regular grid have the advantages of fast calculating speed and easiness in implementation. However, compared to the boundary representation using surface meshes, the volumetric representation using SDF typically requires to take a higher magnitude of storage space.

To save the memory cost, an vast amount of works have been proposed in recent years to provide sparse representations for the SDF, among which the most popular approaches are either based on octree [36] or N-tree [23]. The core idea of existing commonly used adaptive SDF construction algorithms are to create sampling points based on quadtrees or octrees, and then query the shortest distance to the meshes of model. To accelerate this process, various spatial structures are employed, including the rectangular grid, octree and bounding volume hierarchy (BVH) [32]. That is to say, state-of-the-art methods have already been able to construct adaptive SDFs at an interactive speed on GPU[20]. However, the dependence on the surface mesh makes the boolean operations of two adaptive SDFs rather difficult.

In this work, we propose algebraic adaptive signed distance field for GPU, enabling both parallel construction of adaptive SDFs and boolean operations. To form an algebraic system, a novel hierarchical sparse octree is first presented to facilitate the definition of AASDF. A bottom-up fast iterative method is then proposed to construct nodes

level by level as well as to update signed distance values across different levels. In constructing nodes at each level, we require to only generate nodes that are close to the domain boundary to satisfy the adaptivity requirement. Besides, we propose to generate two kinds of nodes for the hierarchical sparse octree, i.e., the internal nodes containing just 8 children and the leaf nodes that contain no children, therefore the completeness of boolean operations can be ensured. Finally, to maximally exploit GPU parallelism, we propose to refine the hierarchical sparse octree by embedding a uniform grid at the top-most level, thus the performance of all operations in the algebraic system of AASDFs including both the construction and boolean operations can be significantly improved.

In summary, the contributions of our work include:

- A hierarchical sparse octree that can be used to construct an algebraic system of AASDFs.
- A bottom-up fast iterative method to construct each element of AASDFs in parallel on GPU.
- A bottom-up algorithm to do boolean operations between two AASDFs in parallel on GPU.

2. Related Work

Considering that SDF has so wide range of applications, it is no surprise that there are plenty of works on algorithms of SDF which are difficult to cover fully here. The Fast Marching Method (FMM) [33], the Fast Sweeping Method (FSM) [44] and the Fast Iterative Method (FIM) [15] are extensively used numerical methods for solving the signed distance function as a special case of Eikonal equation. FMM is a label-setting method and resembles Dijkstra's method [6] designed to find the shortest path on graphs. It simulates a wavefront advancing from the boundary and updates a set of points in narrow band around wavefront, based on upwind difference schemes. The ordered data structure, for example a min-heap data structure or a simple priority queue [16], is used to manage the narrow band that is the most important aspect of the implementation of FMM. FMM has a time complexity of $O(n \log n)$, where n is the number of grid points. [42] presents a parallel implementation of FMM with a novel restarted narrow band approach based on the domain decomposition approach. FSM is a label-correcting method and has a time complexity of $O(n)$. This algorithm successively sweep the the whole grid following a pre-defined number of directions. Typically, forwards and backwards of the x and y dimensions in 2D (corresponding 8 directions in 3D) are used. [46] presents a domain decomposition based parallel algorithms of FSM. [5] sweeps though the domain in parallel with the Cuthill–McKee ordering. [4] presents the hybrid massively parallel fast sweeping method that uses distributed memory method on a coarse grained and shared memory method on a fine grained. [35] proposes the multi-level parallel domain decomposition strategy that uses the Cuthill–McKee ordering for both the coarse and fine grained. FIM is a label-correcting method that is inspired by FMM. The main idea of FIM is to update the points in narrow band affected by the wavefront without maintaining expensive data structures. It has a higher parallel potential because the points in narrow band are updated concurrently until they converge. The computational complexity of FIM is $O(n)$. [3] proposes multilevel parallel approach for FIM. [9] proposes lock free parallel implementation of FIM and group ordered FIM where the solution of the coarse level grid determines the updating order. [12] introduces improved FIM with modifications for updating values and for error correction.

In addition to the general algorithms for the Eikonal equation, there are some works on generating SDF on GPU or adaptive SDF. [26] introduces a preliminary attempt of cuda-based adaptive sdf. The adaptive grid topology is implemented on CPU. And then the distance to each triangular mesh is calculated on GPU. [43] uses the intersecting triangle lists of voxel to compute the distance on GPU, the construction of its octree data structure requires the explicit construction of the dense uniform grids. [20] proposes the hierarchial method to speed up the computation of distance, the topology based on octree is built in the up-bottom way, then a set of bounding volume hierarchy (multi-BVH) structure is employed to accelerate the distance queries of the adaptive sampling grids. [19] uses the DT algorithm [7] to compute the unsigned distance field on uniform grids on GPU, then a ray map is used to compute the winding number of points to determine the sign. [41] proposes a double layer algorithm to compute the approximate SDF from triangles. [18] uses polynomial degree of hexahedral grid (p) together with octree subdivision (h) to construct more accurate spatial topology.

The dense uniform grid is fast to construct and access, and have advantages in numerical discretization of differential operators, interpolation and other such algorithms. But the shortcomings are also obvious, the memory footprint of uniform grid increases rapidly with the expansion of the domain of interest, and even suffers memory bottlenecks. In addition the uniform nature dictates computing resources to be distributed uniform rather than concentrated in key region (e.g. boundary of models) [28]. To exploit the spatially sparse in 3D, multiple works have presented generalized sparse volumetric data structures. [23] introduces OpenVDB, a sparse volumetric data structure, that builds on a shallow tree with a high branching factor. OpenVDB is a standard for sparse volumes in the movie industry. [8] presents a GPU voxel database structure based on OpenVDB, which is a sparse hierarchy of grids for efficient GPU-based computation. [24] introduces NanoVDB, a mini ver-

sion of OpenVDB, which is applicable to both CPU and GPU but only for static topology structures. [10] proposes Taichi, a data-oriented programming language to effciently anthor and manage the sparse data structures.

Boolean operations on geometric objects have been of great interest and have given rise to many related works. [1, 17] demonstrate the exact algorithm for boolean operations directly on surface meshes. [39, 47, 38] present the approximate boolean operations of geometric objects with the help of Layered Depth Images(LDI). [25] performs boolean operations on uniform grids.

3. Background

In this section, we will present the basic theory that is required to derive the algebraic system of adaptive SDFs.

3.1. Eikonal equation

Given $\partial\Omega$ as the boundary of the a closed domain Ω, the signed distance field $\phi(\boldsymbol{x})$ defined on the whole metric space $\mathbf{X}$ is written as

$$\phi(\mathbf{x}) = \begin{cases} d(\mathbf{x}, \partial\Omega) & if \ \ \mathbf{x} \in \Omega \\ -d(\mathbf{x}, \partial\Omega) & if \ \ \mathbf{x} \in \mathbf{X} \backslash \Omega \end{cases}, \tag{1}$$

where d represents the distance to the boundary. If the boundary is smooth, the gradient of $\phi(\mathbf{x})$ satisfies the following Eikonal equation[40]

$$|\nabla \phi(\boldsymbol{x})| = 1 \tag{2}$$

where ∇ is the gradient operation and $|\cdot|$ is the Euclidean norm. Intuitively speaking, the Eikonal equation can be viewed as a wavefront propagating from the boundary $\partial\Omega$ outside in the direction of $\nabla\phi(\boldsymbol{x})$. The absolute value of $\phi(\boldsymbol{x})$ then represents the arrival time of the wavefront to the point $\boldsymbol{x}$.

Based on the above observation, three numerical methods are commonly used to solve the Eikonal equation, including the fast marching method (FMM) [34], the fast sweep method (FSM) [45] and the fast iterative method (FIM) [14]. Since the purpose of this work is to develop a GPU-friend, highly parallel algorithm to solve the Eikonal equation, we will propose a new method based on the fast iterative method that can work well on adaptive grids.

3.2. Fast iterative method

To make our paper self-contained, we present a brief overview of the standard FIM. The basic idea of FIM is to propagate values from a narrow band near the boundary to far regions. It can be summarized into the following two steps

- In the initialization step, FIM first initializes values for grid points that are located near the boundary and labels them as *source points*. The neighbors of the source points are added to the *active* list.
- In the update step, FIM updates the values of points in the active list from their neighboring source points. Once the value of an active point converges to a threshold, it will be added to the source points. All its neighboring points whose value is not converged will be added into the active list. This process will continue until there are no points in the active list any more.

Note both the initialization and update steps in FIM can be done in parallel. Besides, no complex data structures are required to store intermediate variables. As a result, if we use a uniform grid to store signed distance field, the FIM can be easily implemented on GPU. The difficulty lies in if we would like to exploit the sparsity of the signed distance field, how can we retain the good features in FIM?

4. Algebraic Adaptive Signed Distance Field

Study of parallel construction of adaptive SDFs on GPU is not new in computer graphics. For example, Liu et al. [20] has proposed a multi-BVH structure to accelerate building an exact adaptive distance field. However, none of those methods support efficient boolean operations. To form an algebraic system, the following requirements are imposed on each adaptive SDF within this study:

- **Adaptivity.** The signed distance field maintains a sparse data structure that can dynamically refine the grid resolution to the regions of interest. In other words, the refinement criterion should maintain adequate resolution of grid to resolve thin boundary features. For regions deep inside the volume, coarser grids are required to save the total memory cost.
- **Completeness.** Boolean operation between any two SDFs produces a new SDF that maintains a sparse data structure fulfilling the adaptivity requirement.
- **Parallelizable.** All steps in the construction of SDFs and boolean operations have a high degree of parallelization.

To fulfill above requirements, we develop a hierarchical sparse octree to store the signed distance field, as demonstrated in Fig. 1. Compared to a standard hierarchical octree, the main features of the hierarchical sparse octree used here include:

1. Each hierarchical sparse octree shares the same origin of the Cartesian coordinate system $\boldsymbol{C}_o$. During the construction of an AASDF, $\boldsymbol{C}_o$ would be used to align the boundary of the domain of interest as follows:

$$\boldsymbol{x}_o \leftarrow \boldsymbol{x}_o - frac\left(\frac{\boldsymbol{x}_o - \boldsymbol{C}_o}{\delta x}\right) \tag{3}$$

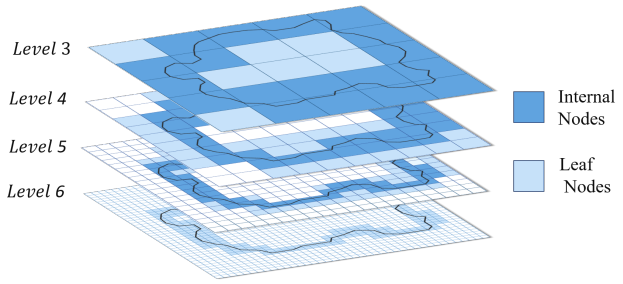

Figure 1: Schematic illustration of the hierarchical sparse octree used to construct AASDFs. It contains two kinds of node, including the internal node that has just 8 children and the leaf node.

where $\boldsymbol{x}_o$ is the origin of the domain of interest, δx is the grid spacing of the top-most level, $frac(\cdot)$ is a function used to calculate the fraction of the floating-point number. As discussed later in Sec. 5, alignment with Cartesian coordinate system facilitates boolean operations between two AASDFs.

2. At each level, the sparsity of the hierarchical octree is maintained by only creating nodes that are located near the boundary. Besides, Fig. 1 shows that the hierarchical octree contains only two kinds of nodes, i.e., the internal node and the leaf node. The internal node should contain just 8 children while the leaf node contains no children.

3. To fully exploit the parallelism of GPU, the depth of the hierarchical sparse octree is set to be adaptive as well. If the depth is too large, both the construction of the hierarchical sparse octree and node query can become much less efficient. Therefore, in our current implementation, only a small number of the most bottom levels of nodes are constructed. Levels that are higher than a user defined level number is simply combined into a uniform grid, which is quite similar to the strategy used in Taichi [11].

In the following context, we will demonstrate how to construct an AASDF level by level.

4.1. Construction

As shown in the Fig. 2, the whole construction pipeline can be divided into three stages: First, leaf nodes in the finest level of grid is constructed. Then, internal and leaf nodes in the intermediate levels are constructed. Finally, internal and leaf nodes in the top-most level of grid are constructed. To facilitate the construction, we define a structure that contains the following properties for each node:

- l: level number of the node, assuming the root node is at depth 0.
- m: morton code representing the node index in memory using a Z-order curve.
- ϕ: Signed distance value defined at the node center.
- $\mathbf{c}$: position center of the node.
- $\mathbf{p}$: the projection of the node center to the boundary.
- $\mathbf{n}$: the boundary normal defined on $\mathbf{p}$.
- $\mathcal{N}_c$: indices of all children nodes.
- $\mathcal{N}_e$: indices of all neighboring nodes.

4.1.1 Constructing nodes in the finest level

Without loss of generality, we assume boundary of the domain of interest is represented with a closed triangular mesh. By choosing an appropriate grid spacing Δx for the finest level, the objective in this stage is to construct nodes for grids that have overlap with any of the axis-aligned bounding box (AABB) of the boundary triangles. To identify all those grids, our idea is to first allocate one GPU thread for each triangle to find overlapping grids with the corresponding triangle independently. Given a triangle whose AABB is denoted as $[\mathbf{x}_1, \mathbf{x}_2]$, the index of the overlapping grids can be trivially calculated according to the following formula:

$$\boldsymbol{F}(\boldsymbol{x_1}) = f(\frac{\mathbf{x}_1 - \mathbf{x}_o}{\Delta x}), \;\; \boldsymbol{G}(\boldsymbol{x_2}) = g(\frac{\mathbf{x}_2 - \mathbf{x}_o}{\Delta x}) \tag{4}$$

with

$$\begin{aligned} f(x) &= \lfloor x \rfloor - (\lfloor x \rfloor \% 2) \\ g(x) &= \lceil x \rceil + (\lceil x + 1 \rceil \% 2) \end{aligned} \tag{5}$$

where $\lfloor \cdot \rfloor$ is the floor function, $\lceil \cdot \rceil$ the ceil function and $\%$ is the modulus operator. Therefore, all grid cells with indices in the region of $\boldsymbol{F}(\boldsymbol{x_1}) \times \boldsymbol{G}(\boldsymbol{x_2})$ are used to construct leaf nodes. Note the functions in Eq. 5 are used to guarantee all sibling nodes can also be created, thus their parent nodes can just have 8 children. Given the identified grid cells, all attributes required for a node can be easily calculated using basic mathematics.

However, since each triangle is processed independently, an additional step should be taken to remove duplicated nodes. We remove all duplicates by first sorting all nodes based on their Morton codes in parallel. For nodes having the same Morton code, we only keep the one with the smallest unsigned distance value. Since the distance calculating in the finest level is exact, all leaf nodes will be labeled as *source points*, and can later be used to initialize signed distance values for nodes in the next level.

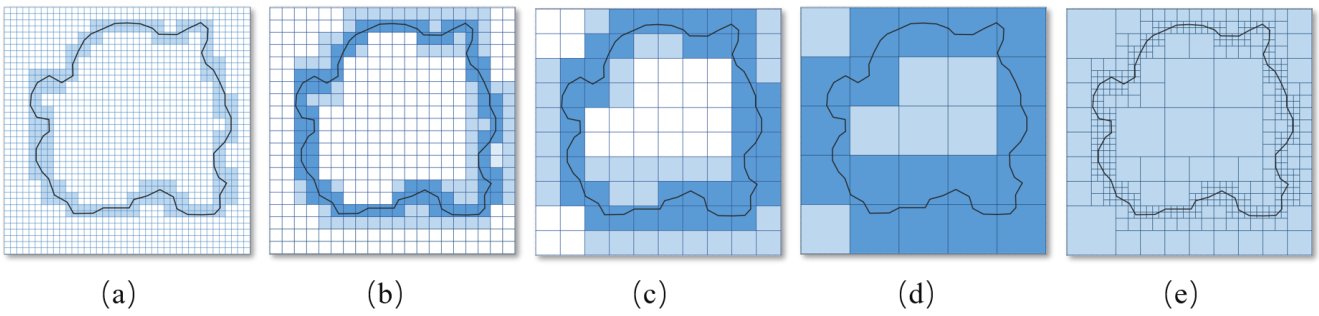

Figure 2: A sample of AASDF construction in 2D. (a) Leaf nodes in the finest level. (b) Nodes in the intermediate level, where the dark blue ones are the internal nodes, they are also the parents of (a), and the light blue ones are leaf nodes of that level. (c) Nodes in the intermediate level like (b). (d) Nodes in the top-most level with the internal (dark blue) and leaf (light blue) nodes. (e) The sparse topology of AASDF consisting of all leaf nodes.

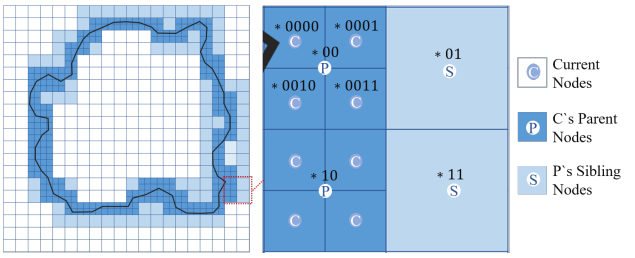

Figure 3: Based on the current nodes with morton code $\{[*0000], [*0001], [*0010], [*0011]\}$, their parent node and the sibling nodes of the parent node $\{[*00], [*01], [*10], [*11]\}$ are generated.

4.1.2 Constructing nodes in the intermediate levels

In contrast to a standard octree construction algorithm, the intermediate levels in our work are constructed in a bottom-up order level by level. Assuming we are currently constructing nodes for level l, our idea is to allocate one GPU thread to each node in level $l+1$ to construct its parent node N_l as well as all sibling nodes to N_l. Given the Morton code m_{l+1} for N_{l+1}, the morton code of its parent or the sibling node of its parent can be uniformly calculated as:

$$m_l = (m_{l+1} \gg 6) \ll 3 + m_{l+1}\&7 \tag{6}$$

where $\gg$ is the right shift operator, $\ll$ is the left shift operator and $\&$ is the bitwise AND operator. To facilitate understanding, Fig. 3 demonstrates a 2D case to show how the parent nodes and their siblings are created (Note in 2D space, Eq. 6 should be formulated as $m_{l+1} = (m_l \gg 4) \ll 2 + m_l\&3$). Consider a node who has a Morton code of $m = [*0000]$, its parent node with a Morton code $m = [*00]$ will be generated. Otherwise, if the node with a Morton code of $m = [*0001]$ is considered, the sibling node of the its parent who has a Morton code of $m = [*01]$ will be generated. However, nodes generated this way can still be duplicative, we therefore should take an additional step to remove duplicates, just as the one taken in constructing the finest level.

The question is how to calculate other properties for the newly created nodes, such as the node center $\mathbf{c}$, signed distance value ϕ, etc. Our idea is to first initialize node properties for internal nodes by calculating all required node properties from their children. For example, the value of ϕ_l can be calculated by taking the minimum of distances measured from the node center $\mathbf{c}_l$ to the projection point $\mathbf{p}_{l+1}$ stored in the child nodes. Note the calculated signed distance values are exact, therefore, all newly created internal nodes will be labeled as the *source points* for the FIM. Other leaf nodes are then added into the *active* list and their properties will be later updated with the FIM.

Now let us consider how to take the FIM to update the properties for nodes in the *active* list. Given a node N_l whose index is denoted as (i, j, k), our first job is to find all neighboring *source* points. We propose an axis-wise method to find all neighbors for each node. Taking the X-axis for example, we first calculate the following index for each node:

$$I_x = k * n_i * n_j + j * n_i + i, \tag{7}$$

where $(n_i \times n_j \times n_k)$ represents the grid resolution. After sorting all nodes by I_x in parallel, neighbors of N_l along the X-axis can easily be found by checking neighbors of I_x in the sorted array of nodes. Likewise, the neighbors along other axis can also be found by sorting I_y and I_z. Afterward, a standard FIM can be taken to update properties for nodes in the *active* list. Since there only exists a narrow band of nodes at each level, a small number of iterations are required for the FIM. After doing the FIM, all internal and leaf nodes will be labels as *source points*, thus providing initial values for node generation in the next level.

4.1.3 Constructing nodes in the top-most level

Compared to an intermediate level, construction of the top-most level differs in one major point that we use a uniform grid to store the signed distance field rather than just a narrow band near the boundary. The resolution of the uniform grid can be influenced by a lot of factors, including the size of domain of interest, GPU thread number and user-defined threshold, etc. After an appropriate resolution for the top-most level is chosen, the construction of the signed distance field follow the same procedure as that for the intermediate level. First, all internal nodes are initialized from their children and labeled as the *source* points. Then, the standard FIM is taken to iteratively update the signed distance values for other leaf nodes until they converge. Since the resolution of top-most level is usually low, a small number of iterations will be sufficient for the convergence.

5. Boolean operations

SDF is the function representation (F-rep) of a geometric object [27]. The analytic definitions of boolean operations of F-rep have been developed by Rvachev ([30, 31]). One of the analytical descriptions can be simple formulated as follows [29]

$$\begin{aligned} \phi_3 = \phi_1|\phi_2 &= min(\phi_1, \phi_2), && \text{union} \\ \phi_3 = \phi_1\&\phi_2 &= max(\phi_1, \phi_2), && \text{intersection} \\ \phi_3 = \phi_1\backslash\phi_2 &= \phi_1\&(-\phi_2), && \text{difference} \end{aligned} \tag{8}$$

which have C^0 continuity [27]. However, boolean operations between two AASDFs are not so straightforward due to the adaptive representation using hierarchical sparse octrees. In this section, we will present how to do boolean operations for two AASDFs and guarantee the newly generated SDF satisfies all properties defined in Sec. 4.

Given two AASDFs, a similar bottom-up procedure will be taken to construct the new AASDF. First the boolean operations are performed on the nodes in the finest level of two AASDFs according to Eq.8. Then the nodes in the intermediate levels and top-most level are constructed in the same way as described in Sec. 4.

The boolean operations of two AASDFs should be performed with the support of acquiring the signed distance values of arbitrary points. Therefore, for any point, how to get the smooth and accurate signed distance value of that point from the AASDFs becomes a priority problem to be solved. A general approach is to interpolate SDF values from their neighboring nodes. Due to the hierarchical octree based adaptive sampling, the construction of neighbor relationships is challenging. For that, we start from the neighbors in the same level. As shown in Fig.4, for leaf node N_l, $N_l^{'}$ is its neighbor at the same level. If $N_l^{'}$ has children nodes, we visit the nodes that is adjacent to N_l, and visits recursively down until reaching the leaf nodes. According to this rule, N_{l+1}^1 and N_{l+1}^2 are the final leaf nodes that are visited. Then we push the N_{l+1}^1 and N_{l+1}^2 into the indices list of neighboring nodes of N_l, while pushing N_l into the indices list of N_{l+1}^1 and N_{l+1}^2. For each leaf node, we assign a GPU thread to execute as described above. In this way, all neighboring nodes of the leaf nodes would be found.

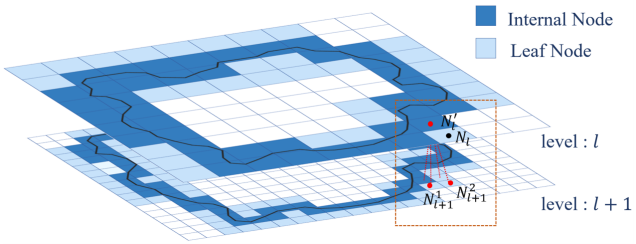

Figure 4: Nodes of two levels demonstrate the finding process of the neighboring nodes.

Due to the different sizes of the adaptive grid, linear interpolation introduces a large error, we choose to interpolate using the moving least squares method. For a point $\boldsymbol{x} = (x, y, z)$, the signed distance value of the point is formulated as:

$$\phi(\boldsymbol{x}) = a + b * x + c * y + d * z \tag{9}$$

where (a, b, c, d) is the coefficient which can be obtained by solving the following objective function:

$$min(J) = min(\sum_{i=1}^{n} \omega_i(\boldsymbol{x})(\phi(\boldsymbol{x_i}) - \phi_i)^2) \tag{10}$$

where $\boldsymbol{x_i}$ is the neighboring node, ϕ_i is the signed distance value of $\boldsymbol{x_i}$, and $\omega_i(\boldsymbol{x})$ is the weight function (here we use the cubic spline function).

Considering two AASDFs denoted as $\phi_1(\boldsymbol{x}_{o1}, \Delta x_1)$ and $\phi_2(\boldsymbol{x}_{o2}, \Delta x_2)$ that satisfy the properties defined in Sec. 4. The new AASDF $\phi^{'}(\boldsymbol{x}_o^{'}, \Delta x^{'})$ generated from the boolean operation of two AASDFs has an origin as:

$$\boldsymbol{x}_o^{'} = min(\boldsymbol{x}_{o1},\ \boldsymbol{x}_{o2}) \tag{11}$$

which naturally aligns with the origin of the Cartesian coordinate system. Furthermore, two grid spacings Δx_1 and Δx_2 have two possible distinct cases:

1. $\Delta x_1 = \Delta x_2$: the new AASDF can be acquired from the boolean operation of two AASDFs according to Eq.8 directly since the nodes in the finest level are naturally aligned.

2. $\Delta x_1 \neq \Delta x_2$: the grid spacing of the new AASDF can be set as one randomly chosen from Δx_1 and Δx_2,

and for the AASDF whose grid spacing is not selected, its nodes in the finest level would be reconstructed. The process of reconstructing is generally consistent to constructing, where the mere difference lies in the primitives of AABBs changing from triangles to cells.

6. Experiments and Results

Experiments are performed on a PC equipped with an Intel Xeon W 2245 CPU, 64 GB of RAM, and an NVIDIA GeForce RTX 3090 GPU with 24 GB memory. Our algorithm is implemented in C++ with the all expensive parts parallelized in CUDA.

6.1. Accuracy and efficiency of the construction process

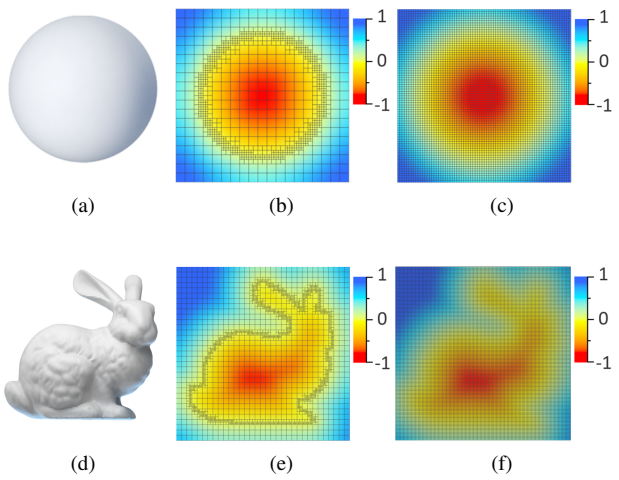

Figure 5: Comparison of sparse topology with uniform grids, the color indicates the signed distance values and the wireframe indicates the topology. (a) Model of the sphere. (b) The cross section of the sphere's AASDF. (c) The cross section of the sphere's SDF constructed on the uniform grids. (d) Model of bunny. (e) The cross section of the bunny's AASDF. (f)The cross section of the bunny's SDF constructed on the uniform grids.

To verify the accuracy of our method, the AASDFs are compared to the SDFs on the uniform grids. The SDFs on the uniform grids are implemented on GPU with FIM. The sparse topology and uniform grids share the same grid spacing on the finest level. As shown in Fig.5, the AASDFs have an accuracy comparable to the SDFs on uniform grids. However, our method significantly outperforms the uniform grids in terms of both construction efficiency and memory overhead, as shown in Tab.1. Moreover, as the model becomes more complex, e.g., with an increasing number of triangles, the advantage of using our AASDF is further enlarged.

Model	Name	Sphere	Bunny	Kitten
	Triangles	18,000	69,664	274,196
Uniform	Time(ms)	33	149	980
	Voxels	970,299	4,184,178	23,541,210
Ours	Time(ms)	25	37	86
	Voxels	112,664	324,488	1,139,606

Table 1: Comparison of construction time and memory overhead for AASDFs and uniform grids. The grid spacing on the finest level of sphere, bunny, and kitten are $0.035mm, 0.02mm, 0.0135mm$ respectively. The level number of AASDFs for the sphere, bunny, and kitten are set to 3, 3, 4 respectively.

In order to verify the efficiency of our algorithm, comparison experiments with multi-BVH [20] are performed on five different models. For a valid comparison, we maintain the number of sampling points to be comparable by adjusting the grid spacing for the finest level Δx. As shown in Tab.2, the construction time of our method is also comparable to that of multi-BVH for a comparable number of sampling points. Considering that multi-BVH dose not enable the construction of neighbor relationships for accessing adaptive SDFs, the statistical time of our method does not include this process either. In our method, the level number of the five models is 4, and the padding of the AABBs of triangular meshes is 0. For multi-BVH, the resolution of bunny and kitten is from 16^3 to 1024^3 and the resolution of david head, eros and ramesses is from 16^3 to 2048^3.

The grid spacing for the finest level is a parameter that controls the resolution of the adaptive topology and can be set according to the requirements. In addition the level number and the padding of AABBs of triangular meshes are also important parameters. The level number of Fig.6(a)(c) is 3, and of Fig.6(b)(d) is 4. It is obvious that more levels means a sparser top-most level grid with fewer sampling points, and naturally the distance field away from the boundary would be coarser. The padding of AABBs of triangular meshes in Fig.6(a)(b) is 0, and in Fig.6(c)(d) is 1. Obviously, more padding of AABBs of triangular meshes would generate more nodes in the finest level along the boundary of model. More padding may be required to obtain the correct global distance field when the model is more complex. All these parameters should be adjusted according to the models and requirements.

6.2. Accuracy and efficiency of boolean operations

When the grid spacing for the finest level of two models are not the same, both the larger or smaller grid spacing can be specified for boolean operations. Fig.7 demonstrates the union of two spheres in the case of consistent and inconsistent grid spacing. Fig.7(c) is the union with the consistent

Models	Name	Bunny	Kitten	David head	Eros	Ramesses
	Triangles	69,664	274,196	583,032	949,920	1,652,528
Multi-BVH[20]	Time(ms)	35	80	150	180	279
	Voxels	320,240	1,114,080	2,587,752	3,927,664	4,748,768
Ours	Time(ms)	25	62	134	215	294
	Voxels	312,380	1,136,974	2,583,774	3,974,160	4,805,552

Table 2: Comparison of the construction time of our method with multi-BVH[20].

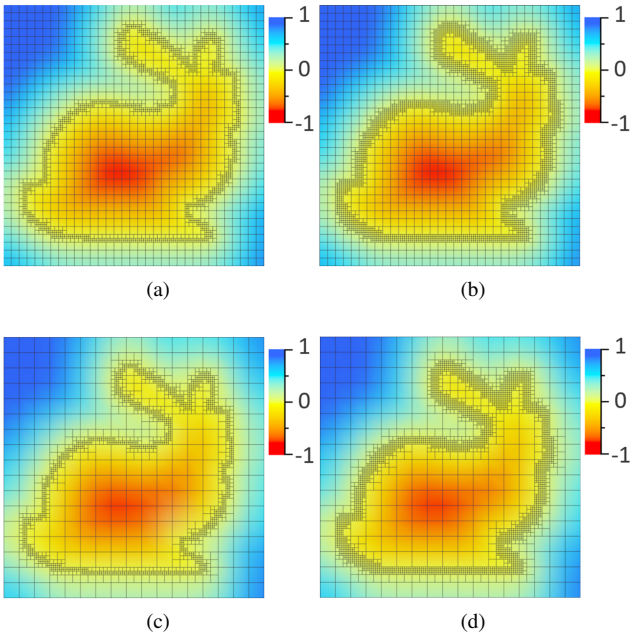

Figure 6: The cross sections of the bunny's AASDFs. The level number of (a), (b), (c), (d) is 3, 4, 3, 4, respectively, and their padding of AABBs of triangular meshes is 0, 1, 0, 1, respectively.

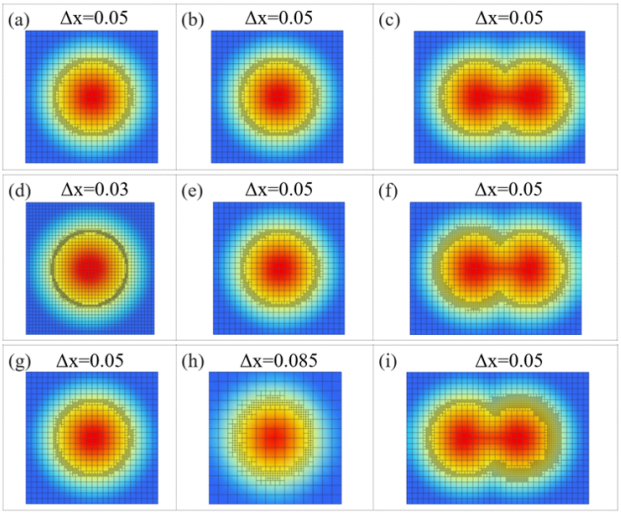

Figure 7: Union of two spheres with different grid spacing for the finest level. (a), (b) The cross sections of sphere with grid spacing $\Delta x = 0.05$. (c) The cross section of the union of spheres with cross sections of (a) and (b). (d), (e) The cross sections of sphere with grid spacing $\Delta x = 0.03$ and $\Delta x = 0.05$, respectively. (f) The cross section of the union of spheres with cross sections of (d) and (e). (g), (h) The cross sections of sphere with grid spacing $\Delta x = 0.05$ and $\Delta x = 0.085$, respectively. (i) The cross section of the union of spheres with cross sections of (g) and (h).

grid spacing, Fig.7(f) is the union with the larger one of the inconsistent grid spacing, and Fig.7(i) is the union with the smaller one. It can be seen that when the grid spacing of two models does not match, our method is still able to perform boolean operations and get results with comparable accuracy.

To verify the efficiency of our method for boolean operations, we perform union, intersection and difference operations of two AASDFs representing a bunny and kitten, respectively. Tab.3 provides statistics on the efficiency of performing boolean operations under different conditions. When the grid spacing of the two models are identical, the boolean operations have a time consumption comparable to that of the construction of AASDFs. The number of sampling points of the intersection operation in this condition is less than either of the two models, so less time is required. The number of sampling points for the union and difference operations are larger, so it takes more time. When the grid spacing of the two models are not identical, the boolean operations in that case take longer because the grids in the finest level need to be reconstructed for the model that does not satisfy the conditions.

The comparison between AASDFs and uniform grids is also performed for Boolean operations. The grid spacing of the uniform grids is same as the grid spacing of the finest level if AASDFs. As shown in Tab.4, in terms of both time and memory overhead of Boolean operations, the AASDFs outperforms the uniform grids. And as the models of Boolean operations become more and more complex, the

Δx of bunny		0.015	0.018	0.013
Δx of kitten		0.015	0.015	0.015
Bunny	Time(ms)	93	75	119
	Voxels	1,296,744	851,254	1,605,127
Kitten	Time(ms)	93	93	93
	Voxels	903,752	903,752	903,752
Union	Time(ms)	112	247	285
	Voxels	1,664,874	2,723,322	2,544,201
Inter-*	Time(ms)	66	154	213
	Voxels	588,938	705,613	681,095
Diff-*	Time(ms)	109	252	271
	Voxels	1,591,538	2,646,309	2,467,623

Table 3: Construction time and storage overhead for the construction process and Boolean operations. Δx is the grid spacing for the finest level. Inter-* and Diff-* represent the intersection and difference operations of bunny and kitten.

			Ours	Uniform
Two spheres	Union	Time(ms)	18	28
		Voxels	125,024	1,123,668
	Inter-*	Time(ms)	14	50
		Voxels	44,864	1,123,668
	Diff-*	Time(ms)	19	37
		Voxels	122,064	1,123,668
Bunny & Kitten	Union	Time(ms)	48	328
		Voxels	563,915	8,414,540
	Inter-*	Time(ms)	32	333
		Voxels	205,843	8,414,540
	Diff-*	Time(ms)	49	348
		Voxels	545,995	8,414,540

Table 4: Construction time and storage overhead for the Boolean operations of AASDFs and uniform grids. The grid spacing on the finest level of two spheres and bunny & kitten are $0.035mm, 0.02mm$ respectively. Inter-* and Diff-* represent the intersection and difference operations.

advantages become greater.

Fig.8 shows the comparison of different interpolation methods. Since it is a sparse topology, calculating the distance value of a point requires interpolation between grids of different sizes. Linear interpolation would cause jaggedness in the distance field, resulting in inaccurate calculated distance values. The moving least squares interpolation method can obtain smoother and more accurate distance fields.

Fig.9 shows the difference, union, and intersection operations of complex models. The first stage is the difference operation of a box and a box with ripples. In that process the grid spacing of the finest level is 0.03, the level number is 4,

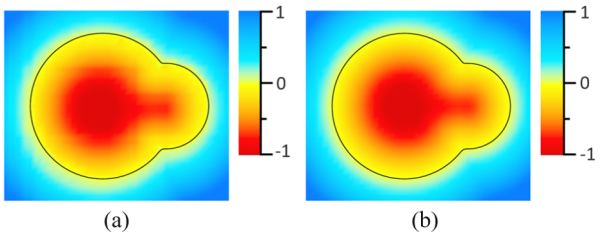

Figure 8: The cross sections of the union of two spheres with different interpolation method. (a) Result of linear interpolation. (b) Results of Moving Least Squares.

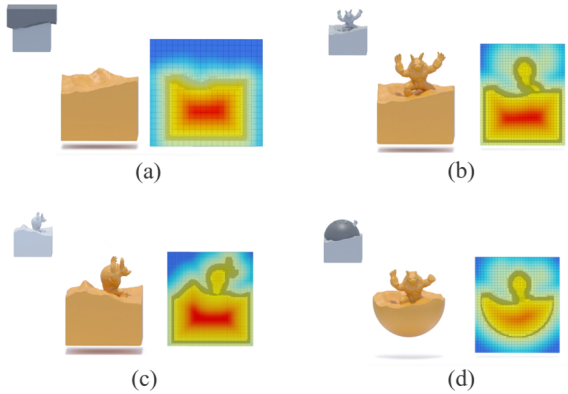

Figure 9: (a) Difference operations of box and ripples. (b) The front of the union operations of armadillo and box. (c) The left of the union operations of armadillo and box. (d) Intersection operations of sphere and armadillo in box.

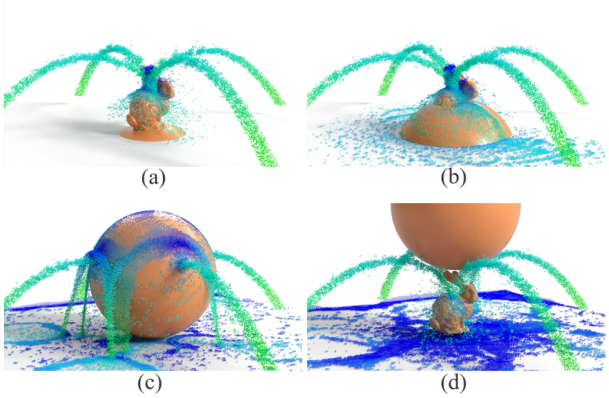

Figure 10: The union of bunny and sphere flushed by fluid.

the padding of the box with ripples is 3, and the padding of the box is 2. The second stage is the union operation of the difference model of the first stage and the armadillo. The grid spacing of that stage is 0.015, the level number is 4, the padding of the difference model and the armadillo are 3. The third stage is the intersection operation of the union

model of the second stage and the sphere. The grid spacing is 0.018, the level number is 4, the padding of the union model is 3, and the padding of the sphere is 0. This example demonstrates the ability of our method to perform continuous Boolean operations on different models. In the Fig.10, the AASDF obtained after Boolean operation of bunny and sphere is used as the boundary of the fluid simulation. The grid spacing of that example is 0.005, the level number is 4, the padding of bunny is 2, and the padding of sphere is 1. That sample demonstrates that the AASDF constructed by our method can be used as a stable boundary for fluid simulation.

7. Conclusion

We presented a complete set of algebraic adaptive signed distance fields for GPU. Compared to state-of-the-art methods, our method not only shows comparable construction performance and storage costs, but also can efficiently do boolean operations between different models at an interactive speed. As a result, our method can be integrated into a real-time fluid simulator to provide interactive collision detection between the fluid and dynamic boundaries.

While our method theoretically supports boolean operations between any two AASDFs, the performance and accuracy can be influenced by the grid spacing defined for the finest level of each AASDF. Therefore, we suggest do boolean operations between two AASDFs that have close values of the finest grid spacing. Besides, due to the alignment issue, the boundary of the top-most level jitters unnaturally. However, this jittering issue will not affect the signed distance value stored inside. We will consider to use other alignment strategies to fix the jittering issue. Finally, we plan to apply our method to more physical based simulations to handle collision detection in real-time.

Acknowledgement

We would like to thank anonymous reviewers for their valuable comments. This work was supported by the National Key R&D Program of China (No.2021YFB1715800), the National Natural Science Foundation of China (No.61872345, No.62072449), Youth Innovation Promotion Association, CAS (No.2019109).

References

[1] R. P. Banerjee and J. R. Rossignac. Topologically exact evaluation of polyhedra defined in csg with loose primitives. *Computer Graphics Forum*, 15(4):205–217, 1996. 3

[2] A. Bhojan. Rtsdf: Generating signed distance fields in real time for soft shadow rendering. In *Pacific Graphics (2020)*, 2020. 1

[3] F. Dang and N. Emad. Fast iterative method in solving eikonal equations: A multi-level parallel approach. *Procedia Computer Science*, 29:1859–1869, 2014. 2

[4] M. Detrixhe and F. Gibou. Hybrid massively parallel fast sweeping method for static hamilton–jacobi equations. *Journal of Computational Physics*, 322:199–223, 2016. 2

[5] M. Detrixhe, F. Gibou, and C. Min. A parallel fast sweeping method for the eikonal equation. *Journal of Computational Physics*, 237:46–55, 2013. 2

[6] E.W.Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1 1959. 2

[7] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. *Theory of computing*, 8(1):415–428, 2012. 2

[8] R. K. Hoetzlein. Gvdb: Raytracing sparse voxel database structures on the gpu. In *Proceedings of High Performance Graphics*, HPG '16, page 109–117, Goslar, DEU, 2016. Eurographics Association. 2

[9] S. Hong and W.-K. Jeong. A group-ordered fast iterative method for eikonal equations. *IEEE Transactions on Parallel and Distributed Systems*, 28(2):318–331, 2016. 2

[10] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019. 3

[11] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand. Taichi: A language for high-performance computation on spatially sparse data structures. *ACM Trans. Graph.*, 38(6), nov 2019. 4

[12] Y. Huang. Improved fast iterative algorithm for eikonal equation for gpu computing. *arXiv preprint arXiv:2106.15869*, 2021. 2

[13] M. Imre, J. Tao, and C. Wang. Efficient gpu-accelerated computation of isosurface similarity maps. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*, 2017. 1

[14] W. Jeong and R. Whitaker. A fast iterative method eikonal equations. *SIAM Journal on Scientific Computing*, (5):30, 2009. 3

[15] W.-K. Jeong and R. T. Whitaker. A fast iterative method for eikonal equations. *SIAM Journal on Scientific Computing*, 30(5):2512–2534, 2008. 2

[16] M. Jones, J. Baerentzen, and M. Sramek. 3d distance fields: a survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599, 2006. 2

[17] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha. Esolid—a system for exact boundary evaluation. *Computer-Aided Design*, 36(2):175–193, 2004. Solid Modeling and Applications. 3

[18] D. Koschier, C. Deul, and J. Bender. Hierarchical hp-adaptive signed distance fields. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '16, page 189–198, Goslar, DEU, 2016. Eurographics Association. 2

[19] B. Krayer and S. Müller. Generating signed distance fields on the gpu with ray maps. *The Visual Computer*, 35, 06 2019. 2

[20] F. Liu and Y. J. Kim. Exact and adaptive signed distance fields computation for rigid and deformablemodels on gpus. *IEEE transactions on visualization and computer graphics*, 20(5):714–725, 2013. 1, 2, 3, 7, 8

[21] D. McLaurin, D. Marcum, M. Remotigue, and E. Blades. Repairing unstructured triangular mesh intersections. *International Journal for Numerical Methods in Engineering*, 93(3):266–275, 2013. 1

[22] N. Mitchell, M. Aanjaneya, R. Setaluri, and E. Sifakis. Non-manifold level sets:a multivalued implicit surface representation with applications to self-collision processing. *Acm Transactions on Graphics*, 34(6):1–9, 2015. 1

[23] K. Museth. Vdb: High-resolution sparse volumes with dynamic topology. *ACM transactions on graphics (TOG)*, 32(3):1–22, 2013. 1, 2

[24] K. Museth. Nanovdb: A gpu-friendly and portable vdb data structure for real-time rendering and simulation. In *ACM SIGGRAPH 2021 Talks*, SIGGRAPH '21, New York, NY, USA, 2021. Association for Computing Machinery. 3

[25] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr. Level set surface editing operators. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 330–338, 2002. 3

[26] T. Park, S.-H. Lee, J.-H. Kim, and C.-H. Kim. Cuda-based signed distance field calculation for adaptive grids. In *2010 10th IEEE International Conference on Computer and Information Technology*, pages 1202–1206. IEEE, 2010. 2

[27] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The visual computer*, 11(8):429–446, 1995. 6

[28] W. Raateland, T. Hädrich, J. A. A. Herrera, D. T. Banuti, W. Pałubicki, S. Pirk, K. Hildebrandt, and D. L. Michels. Dcgrid: An adaptive grid structure for memory-constrained fluid simulation on the gpu. *Proc. ACM Comput. Graph. Interact. Tech.*, 5(1), may 2022. 2

[29] A. Ricci. A constructive geometry for computer graphics. *The Computer Journal*, 16(2):157–160, 01 1973. 6

[30] V. Rvachev. Methods of logic algebra in mathematical physics. *Kiev Izdatel Naukova Dumka*, 1974. 6

[31] V. Rvachev, L. Kurpa, K. Nasriddinov, and A. Shevchenko. The r-function method in problems of nonlinear deformation of plates. *International Applied Mechanics - INT APPL MECH-ENGL TR*, 23:861–866, 09 1987. 6

[32] M. Sanchez, O. Fryazinov, and A. Pasko. Efficient evaluation of continuous signed distance to a polygonal mesh. In *Proceedings of the 28th Spring Conference on Computer Graphics*, SCCG '12, page 101–108, New York, NY, USA, 2012. Association for Computing Machinery. 1

[33] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996. 2

[34] J. A. Sethian. Fast marching methods. *SIAM Review*, 41(2):199–235, 1999. 3

[35] A. Shrestha and I. Senocak. Multi-level domain-decomposition strategy for solving the eikonal equation with the fast-sweeping method. *IEEE Transactions on Parallel and Distributed Systems*, 29(10):2297–2303, 2018. 2

[36] J. Strain. Fast tree-based redistancing for level set computations. *Journal of Computational Physics*, 152(2):664–686, 1999. 1

[37] P. Trettner, J. Nehring-Wirxel, and L. Kobbelt. Ember: exact mesh booleans via efficient & robust local arrangements. *ACM Transactions on Graphics (TOG)*, 41(4):1–15, 2022. 1

[38] C. C. Wang. Approximate boolean operations on large polyhedral solids with partial mesh reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 17(6):836–849, 2011. 3

[39] C. C. Wang, Y.-S. Leung, and Y. Chen. Solid modeling of polyhedral objects by layered depth-normal images on the gpu. *Computer-Aided Design*, 42(6):535–544, 2010. 3

[40] O. Weber, Y. S. Devir, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph.*, 27(4), nov 2008. 3

[41] Y. Wu, J. Man, and Z. Xie. A double layer method for constructing signed distance fields from triangle meshes. *Graphical Models*, 76(4):214–223, 2014. 2

[42] J. Yang and F. Stern. A highly scalable massively parallel fast marching method for the eikonal equation. *Journal of Computational Physics*, 332:333–362, 2016. 2

[43] K. Yin, Y. Liu, and E. Wu. Fast Computing Adaptively Sampled Distance Field on GPU. In B.-Y. Chen, J. Kautz, T.-Y. Lee, and M. C. Lin, editors, *Pacific Graphics Short Papers*. The Eurographics Association, 2011. 2

[44] Zhao and Hongkai. A fast sweeping method for eikonal equations. *Mathematics of Computation*, 74(250):603–627, 2005. 2

[45] H. Zhao. A fast sweeping method for eikonal equations. *Mathematics of computation*, (250):74, 2005. 3

[46] H. Zhao. Parallel implementation of the fast sweeping method. *International Journal of Computer Mathematics - IJCM*, 25, 01 2006. 2

[47] H. Zhao, C. C. L. Wang, Y. Chen, and X. Jin. Parallel and efficient boolean on polygonal solids. *Vis. Comput.*, 27(6–8):507–517, jun 2011. 3