JCST
Vol.37 No.3 May 2022

ISSN 1000-9000(Print)
/1860-4749(Online)
CODEN JCTEEM

Journal of Computer Science & Technology

SPONSORED BY INSTITUTE OF COMPUTING TECHNOLOGY
THE CHINESE ACADEMY OF SCIENCES &
CHINA COMPUTER FEDERATION
SUPPORTED BY NSFC
CO-PUBLISHED BY SCIENCE PRESS &
SPRINGER

COMPUTER

Chen XR, Tang M, Li C *et al.* BADF: Bounding volume hierarchies centric adaptive distance field computation for deformable objects on GPUs. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37(3): 731–740 May 2022. DOI 10.1007/s11390-022-0331-x

BADF: Bounding Volume Hierarchies Centric Adaptive Distance Field Computation for Deformable Objects on GPUs

Xiao-Rui Chen[1] (陈潇瑞), Min Tang[1] (唐　敏), *Member*, *CCF*, *ACM*, Cheng Li[1] (李　澄)
Dinesh Manocha[2], *Fellow*, *ACM*, *IEEE*, and Ruo-Feng Tong[1] (童若锋), *Member*, *CCF*

[1] *Laboratory of Geometry, Image and Video Processing Enterprise Intelligent Computing, College of Computer Science and Technology, Zhejiang University, Hangzhou 310007, China*

[2] *Geometric Algorithms for Modeling, Motion, and Animation Laboratory, University of Maryland, Maryland 20742, U.S.A.*

E-mail: {chenxiaorui, tang_m, licharmy}@zju.edu.cn; dm@cs.unc.edu; trf@zju.edu.cn

Received March 31, 2020; accepted April 2, 2022.

Abstract We present a novel algorithm BADF (Bounding Volume Hierarchy Based Adaptive Distance Fields) for accelerating the construction of ADFs (adaptive distance fields) of rigid and deformable models on graphics processing units. Our approach is based on constructing a bounding volume hierarchy (BVH) and we use that hierarchy to generate an octree-based ADF. We exploit the coherence between successive frames and sort the grid points of the octree to accelerate the computation. Our approach is applicable to rigid and deformable models. Our GPU-based (graphics processing unit based) algorithm is about 20x–50x faster than current mainstream central processing unit based algorithms. Our BADF algorithm can construct the distance fields for deformable models with 60k triangles at interactive rates on an NVIDIA GTX GeForce 1060. Moreover, we observe 3x speedup over prior GPU-based ADF algorithms.

Keywords distance field, deformable object, graphics processing unit (GPU), octree, bounding volume hierarchy

1 Introduction

Distance fields are scalar fields that represent the minimum distance from any point in space to shapes or objects in the scene. They are typically computed for a finite number of points in 3D based on uniform or adaptive sampling. Adaptive distance fields (ADFs) are sampled according to the local detail and stored in a spatial hierarchy for efficient processing. They are used in many applications including motion planning, proximity queries, surface reconstruction, physics-based simulation, etc. [1]. Compared with uniform distance fields (UDFs), ADFs offer many advantages in terms of quicker query processing and lower storage overhead.

Distance fields are typically computed by repeatedly performing distance queries between given points and the boundary of an object. Each query can be accelerated using hierarchic data structures, e.g., spatial hashing or bounding volume hierarchies (BVHs). For rigid models, distance fields can be computed once during a pre-process stage. However, for deformable models, the distance fields need to be updated or recomputed on-the-fly, which may be challenging for real-time applications.

Many techniques have been proposed for faster distance field computation of rigid and deformable models that can exploit multiple cores on CPUs and GPUs (graphics processing units). In particular, the GPU parallelism can be used to perform the distance queries for multiple points in parallel [2, 3]. While UDF computation techniques use simple, regular grid based representation, ADF algorithms typically use an octree structure to store the distance fields. The computation of multiple data structures in terms of bounding volume hierarchies and octrees slows down the algorithms. In practice, distance field computation can be expensive and may not be fast enough for interactive applications

Regular Paper

This work was supported by the National Key Research and Development Program of China under Grant No. 2018AAA0102703, and the National Natural Science Foundation of China under Grant Nos. 61972341, 61972342, and 61732015.

or for the manipulation of deformable models.

Main Result. We present a novel BVH-based algorithm (BADF), to accelerate the distance field computation on GPUs. Our approach is designed to achieve higher performance on commodity processors for interactive applications. We present new techniques to build integrated data structures and exploit the coherence between successive frames for faster computation. The major components of our approach include the followings.

• *BVH-Centric Streamlined Data Structure.* We use a fast BVH-centric algorithm to construct the BVH and later use the BVH to compute an octree-based ADF. We first sort the model triangles based on the Morton code [4] and second, compute a BVH tree using the spatial information of the Morton code. We use the node relationship within a BVH tree and the location information of the Morton code to generate the octree-based ADF. The BVH tree bounding box is refitted to reduce the range of distances.

• *Acceleration to Distance Queries with Spatial-Temporal Coherence.* Our BADF algorithm records the nearest triangle on the boundary for each query point after the distance query for the current frame. The distance queries are performed according to the octree representation. During the next frame, the distance to the nearest triangle is used as an upper bound and we perform efficient pruning for BVH-based culling. This reduces the BVH traversal overhead for distance queries.

• *Accurate Distance Field by Sorting Query Points.* We avoid redundant distance query calculations of the octree's grid points by sorting all the grid points based on the Morton code of the octree nodes and processing the sorted grid points for distance queries. Compared with prior techniques, our approach can be used to compute ADF at a finer resolution with a similar runtime performance.

Our BVH-centric algorithm first constructs a BVH with Morton codes. We use this BVH for parallel distance queries and construct an octree-based ADF. We utilize the coherence to accelerate the distance queries. The ADF can be computed at almost interactive rates for deformable objects.

An overview of our approach is given in Fig. 1. We implement BADF and highlight its performance on rigid and deformable models with up to 1M triangles (see Table 1). We highlight our results using a grid resolution of 128^3 on an NVIDIA GTX GeForce 1060. For models of up to 69k triangles, our approach can compute ADF at 46.6 ms per frame (i.e., 20 fps). Compared with a recent GPU-based algorithm [3], we obtain up to 3x speedup.

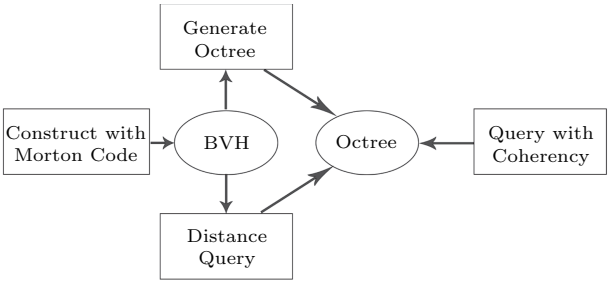


Fig.1. BADF algorithm pipeline.

Table 1. ADF Construction Time of Rigid Models on an NVIDIA GeForce GTX 1060

Benchmark	Number of Triangles ($\times 10^3$)	Construction Time (ms)
Andy	34	65.9
Bunny	70	46.6
Armadillo	213	53.6
Dragon	871	88.9
Buddha	1 100	101.3

2 Background

We give a brief overview of the existing distance field computation algorithm and describe our notation.

2.1 UDF and ADF

$$\Phi(\boldsymbol{\xi}) = s(\boldsymbol{\xi}) \inf_{\boldsymbol{\xi}^* \in \partial\mathcal{B}} \|\boldsymbol{\xi}^* - \boldsymbol{\xi}\|,$$

$$s(\boldsymbol{\xi}) = \begin{cases} -1, & \text{if } \boldsymbol{\xi} \in \mathcal{B}, \\ 1, & \text{otherwise.} \end{cases}$$

The distance function is defined as the Euclidian distance from a given point $\boldsymbol{\xi}$ to the nearest point $\boldsymbol{\xi}^*$ which lies on the domain's boundary $\partial\mathcal{B}$. Signed distance fields can be further classified as UDFs or ADFs based on whether they use uniform or adaptive sampling of a 3D space, respectively [5]. Krayer and Müller [6] used ray maps and distance transform to construct the uniform distance field, and their recent work has raised the distance field construction to about 200 ms for 70k triangles. However, using our algorithm, we can reduce the construction time to about 50 ms. ADF algorithms subdivide the space adaptively into an octree and only store the distance to the object scene at the octree's nodes. Compared with the uniform sampling of UDFs, ADF algorithms are superior in terms of computation time and memory overhead.

ADFs have been used for various applications, including ray tracing [7], collision detection [8], surface reconstruction [9], motion planning [10], visualization [11], and geometric modeling [12].

2.2 ADF Construction

The construction of ADFs is faster than that of UDFs. However, current ADF computation algorithms are unable to compute distance fields at interactive rates for complex deformable models. The main challenges are computing the octree and performing distance queries for each grid point of the octree.

Different techniques have been proposed to accelerate the ADF computation by exploiting the parallelism of the GPUs. Bastos and Celes [13] presented a method to compute ADFs on a GPU and store the octree nodes using a hash-based structure. Liu and Kim [3] described a method to compute distance fields on GPUs. They used the BVH as an acceleration structure for distance queries and constructed it in a top-down manner. All these methods work well on rigid models and do not offer interactive performance for deformable models.

2.3 Voronoi Diagrams and Distance Fields

Distance fields are closely related to the generalized Voronoi diagram (GVD) computation. A GVD divides the 3D space into generalized Voronoi cells based on the primitives closest to each point in the space. Hoff *et al.* [10] proposed a fast method to compute the approximate GVD using interpolation-based polygon rasterization hardware. With the continuous development of GPU architectures and general-purpose programmability, many faster techniques have been proposed for GVD computation [14–18]. The GVD can be regarded as a subset of the locus of distance field critical points. We can calculate the GVD based on the distance field or we can use an expansion algorithm to calculate the distance field in the 3D space based on the GVD.

2.4 Octree Generation

There is considerable work on octree computation, including bottom-up and top-down strategies.

Bottom-up Algorithms. Zhou *et al.* [19], Karras [20], and Bédorf *et al.* [21] presented techniques to compute octrees. Many prior methods are based on bottom-up approaches and Morton codes. Zhou *et al.* [19] used the resulting octree for point cloud reconstruction, thereby point cloud reconstruction needs to preserve the vertices, edges, and faces during octree computation. For a given level, Bédorf *et al.* [21] masked each particle and grouped the results with a parallel compaction algorithm. The masking and grouping procedures are repeated for every single level until all the particles are assigned to leaf nodes or the maximal depth of the tree is reached. Karras [20] constructed the BVH in parallel and detected all the edges and generated an octree based on the information of the edges. On the basis of the technique by Karras [20], Morrical and Edwards [22] further figured out the relationship between multiple objects. For those conflict cells containing multiple objects, Morrical and Edwards [22] further differentiated by their parallel quadtree construction algorithm. Our approach improves Karras' LBVH method [20] for octree computation.

Top-down Algorithms. Zhou *et al.* [23] used a top-down construction in another study about the KD-tree construction. However, using such parallel constructs on the GPU makes most of the computation cores idle, especially at the root of the tree. Liu and Kim [3] used a top-down approach to reduce the idle time of a processor core and instead compute multiple BVHs. In contrast to these methods, our approach is better suited to exploiting the multiple cores on the GPU.

2.5 Distance Queries

The time complexity of calculating the distance field algorithm is $O(m \times n)$, where m is the number of query points, and n is the time taken for each query point. Many techniques use BVHs to reduce the query time. Other methods tend to reduce the number of query points. The ADF formulation proposed by Frisken *et al.* [5] reduces a large number of sampling points by using an adaptive grid. Many other techniques have been proposed to accelerate the distance field computation.

• *Exact Distance Field Around an Object.* Sramek and Kaufmann [24] presented a method to compute distance field shells, which only calculates the distance field near the surface of the object. This method can reduce the sampling points by half, but it cannot be applied everywhere in the 3D space.

• *Level Sets for Propagating Accurate Distances.* Breen *et al.* [25] and Kimmel [26] proposed level sets for propagating accurate distances throughout the volume. Breen *et al.* [25] first calculated the closest points for the narrow band and zero set, and then used the fast marching method to compute the closest point.

• *Different Types of Query Points.* Yin *et al.* [27] divided the sampling points into three types: the points having triangles at the lowest level, the points on the

boundary of the lowest level nodes but without triangles, and the points at the center of the non-lowest-level nodes siblings. This method can reduce a large number of sampling points, but computes an approximate distance for some query points.

3 BADF: Adaptive Distance Field Generation

In this section, we present our novel algorithm for generating adaptive distance fields. The overall pipeline is highlighted in Fig.1.

3.1 BVH Generation

Our BVH generation algorithm (as shown in Fig.2) is based on the GPU algorithms proposed by Karras [20]. Karras's algorithm can construct a BVH fully in parallel according to the Morton code. The Morton code is a way of mapping quantized n-dimensional vectors into integer scalar values. We can sort the points in space according to Morton code. The Morton code for a given point contained within the 3D unit cube is defined by the bit string $X_0Y_0Z_0X_1Y_1Z_1...$, where the x coordinate of the point is represented as $0.X_0X_1X_2...$, and similarly for the y and z coordinates. The Morton code of a given point and the coordinates of this point can be freely converted. We extend Karras' method [20] so that it can be directly used to generate the octree. After we sort all the leaf nodes that correspond one-to-one with the triangle positions in the lexicographic order, the range occupied by leaf nodes covered by each internal node can be represented as a linear range $[i, j]$. We record $\delta(i, j)$ of each node, which denotes the length of the longest common prefix between Morton code keys k_i and k_j. For any $m, n \in [i, j]$, $\delta(m, n) \geqslant \delta(i, j)$. We can calculate δ of each node by comparing the Morton code keys of its leftmost and rightmost leaf nodes. δ is used for octree generation.

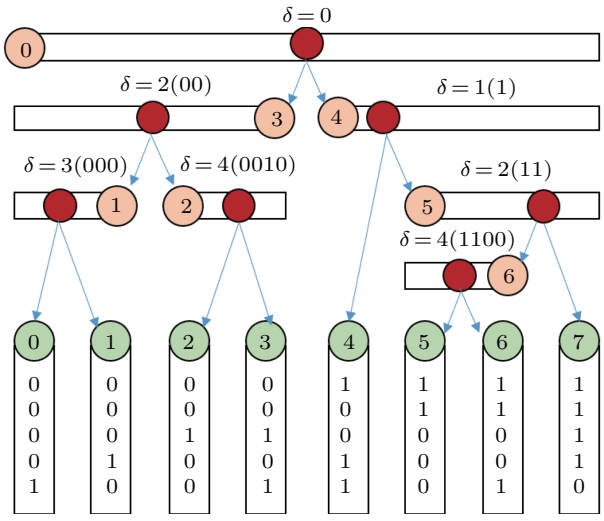


Fig.2. BVH generation. We highlight our parallel BVH computation algorithm. The range of keys covered by each node is indicated by the horizontal bar. The split position, corresponding to the first bit that differs between the keys, is indicated by a red circle. We also compute δ for each internal node, which is used for octree generation.

3.2 Octree Generation

After constructing the BVH, we use the parent-child node inheritance relationship and the spatial information from the relationship to generate the octree, as shown in Fig. 3. On the original BVH tree, if δ of the node can be divided by 3, it means that this node is an internal node of the octree, e.g., if $\delta_i = 3$, it implies that $node_i$ is the child of the root node in the octree. For a parent node with a prefix of length δ_{parent} and a child node with a prefix of length δ_{child}, if $\delta_{\text{child}}/3 - \delta_{\text{parent}}/3 > 0$, there is an octree node that can be generated between the child and parent nodes in the BVH tree. We first set up a maximum octree resolution controlled by the user as an input parameter. Based on the resolution, we determine the bit length of the Morton code of the

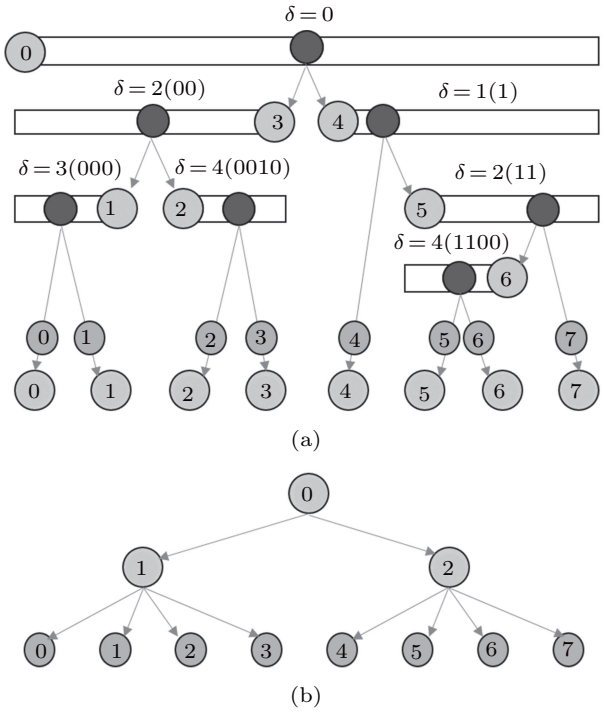


Fig.3. Octree generation. We traverse the BVH in a top-down manner, and check if we can insert the octree node on its left/-right children. Our combination of BVH and octree results in a faster computation. (a) Finding the leaf node of the octree. (b) Building the octree in parallel.

octree leaf node. We traverse the BVH in a top-down manner, and check if we can insert the traversed octree node on the left or right children of the node. If so, we can record the Morton code prefix for this node. The detailed algorithm is shown in Algorithm 1. We traverse the BVH in a top-down manner, and check if we can insert the octree node on its left/right children.

Algorithm 1. Octree Generation

```
1: procedure OCTREE GENERATION
2:                                          ▷ Traverse the BVH
3:     lvnum_i ← the number of nodes in level i
4:     d ← max depth of octree
5:     for N_i in each BVH node in parallel do
6:         δ_i ← δ of N_i
7:         δ_l ← δ of N_i.LeftChild
8:         δ_r ← δ of N_i.RightChild
9:         if δ_l/3 − δ_i/3 > 0 then
10:            if δ_l/3 = d then
11:                Add to octree leaf node queue Q
12:        if δ_r/3 − δ_i/3 > 0 then
13:            if δ_r/3 = d then
14:                Add to octree leaf node queue Q
15:                                      ▷ Now generate the octree
16:    for L_i in each queue Q in parallel do
17:        for k = d:1 do
18:            if the k-th ancestor of L_i has not been initialized
   then
19:                Initialize the k-th ancestor of L_i
```

3.3 BVH Refitting

To facilitate faster distance queries, we need to refit the constructed BVH. This is necessary because the BVH is obtained by the Morton code and the bounding volume of each node corresponds to the sub-space, which may not be the tightest bounding volume of all its triangles. In the process of refitting a BVH, we use a bottom-up approach and recalculate the bounding boxes of each node in parallel in the GPU. Specifically, we first refit all the leaf nodes in parallel, and then iteratively refit all the internal nodes with children nodes that have already been refitted. In practice, this parallel algorithm is quite efficient on GPU, and BVH refitting takes only approximately 0.3%–0.8% of the overall ADF construction time.

3.4 Query Reduction

After we compute an octree, we perform a distance query on each octree grid point to construct the final distance field. However, these grid points are repeated for adjacent octree nodes. For each octree node, we need to perform eight queries for its eight corners accordingly. Most of these queries are redundant since these corners are often shared by many octree nodes. Therefore, we assign each corner a Morton code, and use the code to ensure that there is only one query for each corner. In practice, we can reduce the number of queries by up to 2x–3x.

3.5 Distance Queries

We reuse BVH to speed up distance queries on the octree corners, which addresses a major efficiency bottleneck in our algorithm. The detailed algorithm is highlighted in Algorithm 2. We use BVH to speed up distance queries on the octree corners. As shown in Algorithm 2, we perform all the queries in parallel on the GPU (lines 25–45). Each GPU thread works on a query point C_i. In each thread, we traverse the BVH in a top-down manner, and store all the active BVH nodes into a stack S. We process all the BVH nodes in S until it becomes empty (lines 31–43). For each BVH node, if its left or right child is a leaf node, we compute the distance between the triangle in the child and C_i and update the minimum distance $minDist$. After all the BVH nodes in S are processed, we compute $minDist$ and its corresponding triangle $minItem$ (line 45) for C_i.

We also utilize spatial-temporal coherence to accelerate distance queries for deformable objects. Specifically, we record the triangle(s) with the minimum distance (the purple one in Fig.4) to the query point in the last frame and use its distance at the current frame as the initial minimum distance for the new query. Storing distances from the last frame and using them to compute upper bounds accelerate the computation of ADF by 1.5x for deformable models.

3.6 Sign Calculation

We use the angle weighted pseudonormal algorithm [28] to calculate the normal. This algorithm calculates the pseudo-normal by assigning the normal deflection by calculating the angular weights corresponding to the faces adjacent to the vertices. The sign of the query point is judged by the product of the pseudo-normal direction of the query point and the nearest point.

Algorithm 2. Distance Query

```
▷ Perform distance query between a triangle and a point
procedure DistQueryTP(Triangle(a, b, c), Point p)
                ▷ Calculate the distance from p to three corners
    dist_0 = distance(a, p)
    dist_1 = distance(b, p)
    dist_2 = distance(c, p)
                  ▷ Calculate the distance from p to three edges
    dist_3 = distance(ab, p)
    dist_4 = distance(bc, p)
    dist_5 = distance(ca, p)
      ▷ Calculate the distance from the point to the triangle plane
    dist_6 = distance(Δabc, p)
                                  ▷ Get the maximum distance
    return min(dist_i) for i ∈ [0, 6]

    ▷ Perform distance query between a bounding box and a point
procedure DQueryBP(Box(b_i, i ∈ [0, 7]), Point p)
          ▷ Calculate the distance from the point to the eight vertices
    dist_i = distance(b_i, p) for i ∈ [0, 7]
                                  ▷ Get the maximum distance
    return min(dist_i) for i ∈ [0, 7]

                ▷ Perform distance query for the octree corners
procedure Distance Query(Octree Corners C_i, BVH B)
    for query point C_i in parallel do
        Create local stack S
                          ▷ Use coherency for deformable objects
        minItem = T'_i
        minDist = DQueryTP(minItem, C_i)
        S.push(B.root())
        while do S.emty() == false
            curNode = S.pop()
            if curNode.rightChild().isleaf() then
                dist = DqueryTP(curNode.rightItem(), C_i)
                if dist < minDist then
                    minDist ← dist
                    minItem ← curNode.rightItem()
            else
                dist ← DQueryBT(curNode.rightBox(), C_i)
                if dist < minDist then
                    S.push(curNode.rightChild())
                                  ▷ Same to do with the left child
            Process curNode.leftChild()
                                  ▷ Get the maximum distance
        Record minDist and minItem for C_i
```

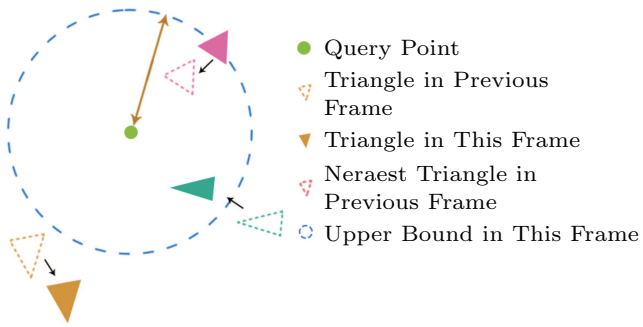


Fig.4. Utilizing spatial-temporal coherence for distance query. We record the triangle with the minimum distance (the purple one) to the query point during the last frame. The distance to this triangle is used as an upper bound during the next query and reduces the overhead of tree traversal.

4 Results and Comparison

In this section, we describe our implementation and compare the performance with the performance of prior methods.

4.1 Implementation

We implemented our ADF construction algorithm on an NVIDIA GeForce GTX 1060 (with 1 280 cores at 1.5 GHz and 6 GB memory). Our implementation used CUDA toolkit 9.1 and Visual Studio 2013 as the underlying development environment. We used a standard PC (Windows 7 Ultimate 64 bits/Intel I7 CPU@4G Hz/8 G RAM) to evaluate performance. We performed single-precision floating-point arithmetic for all the computations on the GPU. We also integrated our algorithm into a GPU-based cloth simulation system, I-Cloth [29], where the ADF computation is used for collision handling. We evaluated our BADF algorithm for different resolution distance fields for multiple rigid and deformable models.

4.2 Benchmarks

The construction time of our algorithm for different rigid models is shown in Table 1. As shown in the table, our algorithm is capable of constructing ADF for models with several hundreds of thousands of triangles within tens of milliseconds on a commodity GPU.

We used two benchmarks (andy and sphere) with deforming objects for evaluation, as shown in Fig.5. For the two benchmarks, Andy and Sphere, we observed significant speedup on ADF construction by using the temporal-spatial coherency between animation frames. We observed improved performance after the first few frames due to coherence. We replaced our distance field

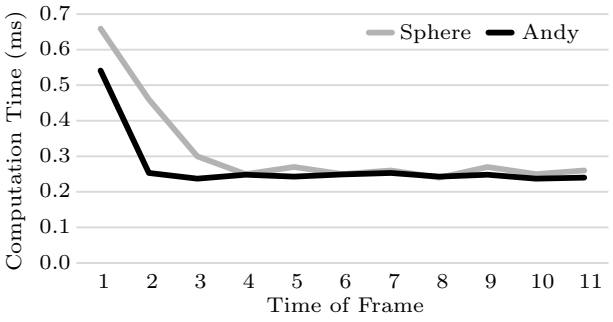


Fig.5. ADF computation of deformable models. The horizontal axis of the graph represents the current frame of the animation frame sequence, and the vertical axis represents the time required to build the ADF of the current frame.

algorithm with collision detection and collision response in the I-Cloth cloth simulation system [29]. We used ADF to compute the distance between the cloth and an object and used ADF to add penalty forces on the cloth nodes to compute the separation forces, which are a function of the distance.

The first deformable benchmark used for evaluation corresponds to Andy: a boy is performing Kungfu. Andy is a human body model with 33k triangles and the jacket of Andy has 22k triangles. In this scenario, the overall cloth simulation can run at approximately 7 FPS–10 FPS. Another benchmark used is the Sphere-cloth: a ball moving forward and backward is interacting with a cloth hanging with two corners. This benchmark has a piece of the rectangle cloth with 16k triangles and a sphere with 1k triangles. Our cloth simulation (with ADF computation) can run at approximately 10 FPS–13 FPS. As shown in Fig.5, our algorithm runs pretty fast for all the frames except the 1st frame. This highlights the benefit of our algorithm in terms of using the frame-to-frame coherence. We used the model Amadillo, Bunny, Buddha and Dragon to show the effect of our algorithm in Fig.6.

4.3 Comparison

Prior GPU-Based ADF Construction Algorithm. We compared our results with those of [3]. Since the source code implementation of Liu and Kim's algorithm [3] is not available, we estimated its performance on GTX 1060 based on the reported performance data and the number of GPU cores. We observed that BADF is up to 3x faster and provides an average speedup of 1.9x, as shown in Fig.7.

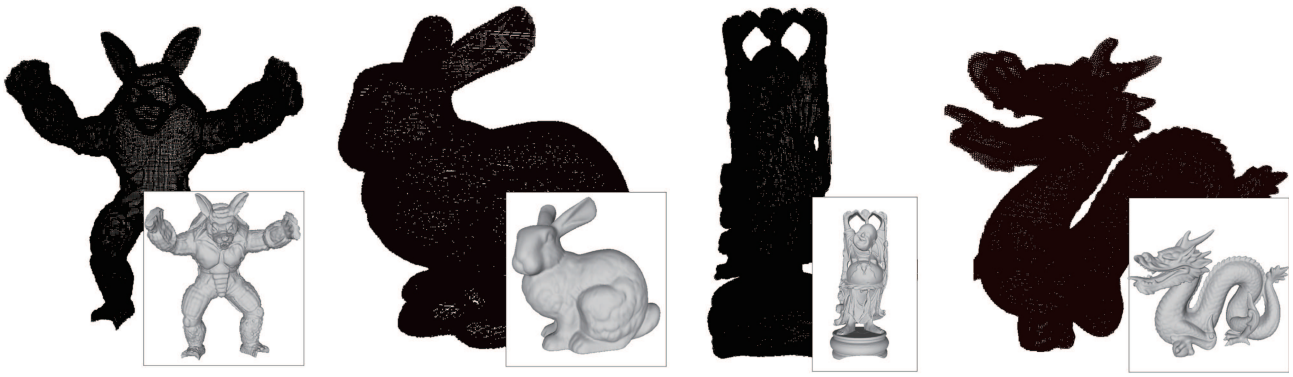
Our algorithm shows good performance improvement because we used a more parallel GPU-based algorithm to construct the octrees and BVH trees. Liu and Kim [3] constructed an octree and a BVH using a top-down construction method, thereby the time complexity of the algorithm is $O(n\log n)$. In contrast, we used the parallel construction method, thereby the time complexity of our ADF construction algorithm is $O(n)$. Therefore, as the number of triangles increases, we observed better performance improvement in these benchmarks using our algorithm.

Rigid Models vs Deformable Objects. Our algorithm is better suited to constructing dynamic or time-varying distance fields (e.g., for deforming objects). Our approach can exploit the frame-to-frame coherency between the frames. The use of coherence can provide 1.5x speedup, on average. We intercepted part of the animation frames to show the algorithm's construction effect on deformable objects in Fig.8.

Different ADF Resolutions. We also evaluated the performance of our ADF construction algorithm by varying specified resolutions. As shown in Fig. 9, as the resolutions increases, the ADF construction time slows down accordingly. As the resolution increases, the ADF computation time slows down accordingly. BADF can compute the distance fields at interactive rates for higher levels of the octree.

5 Conclusions

We presented a GPU-based ADF construction algorithm for rigid or deformable objects. We also integrated it into an interactive cloth simulation system to compute proximity constraints and collision response. The main data structure of our algorithm is a tight BVH that is constructed in parallel on GPU. BVH is



(a) (b) (c) (d)

Fig.6. ADF construction results of rigid models for (a) Armadillo, (b) Bunny, (c) Buddha and (d) Dragon.

used both in octree generation and in the process of speeding up distance queries on the octree nodes. We also utilized frame-to-frame coherence to accelerate the queries. Our algorithm can compute ADF for complex deformable models at interactive rates on commodity GPUs and offers up to 3x speedup over prior methods.

Our approach has some limitations. First, the quality and the resolution of ADF depend on a user-specified resolution and shape of the models. Second, the benefit of frame-to-frame coherence is observed after the first few frames.

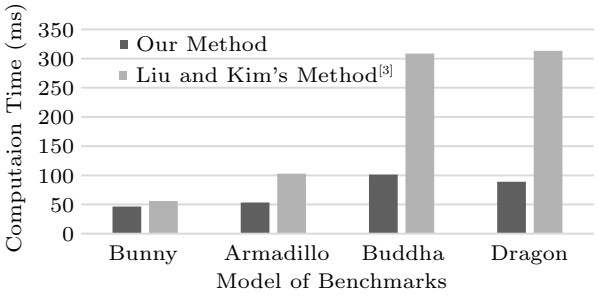
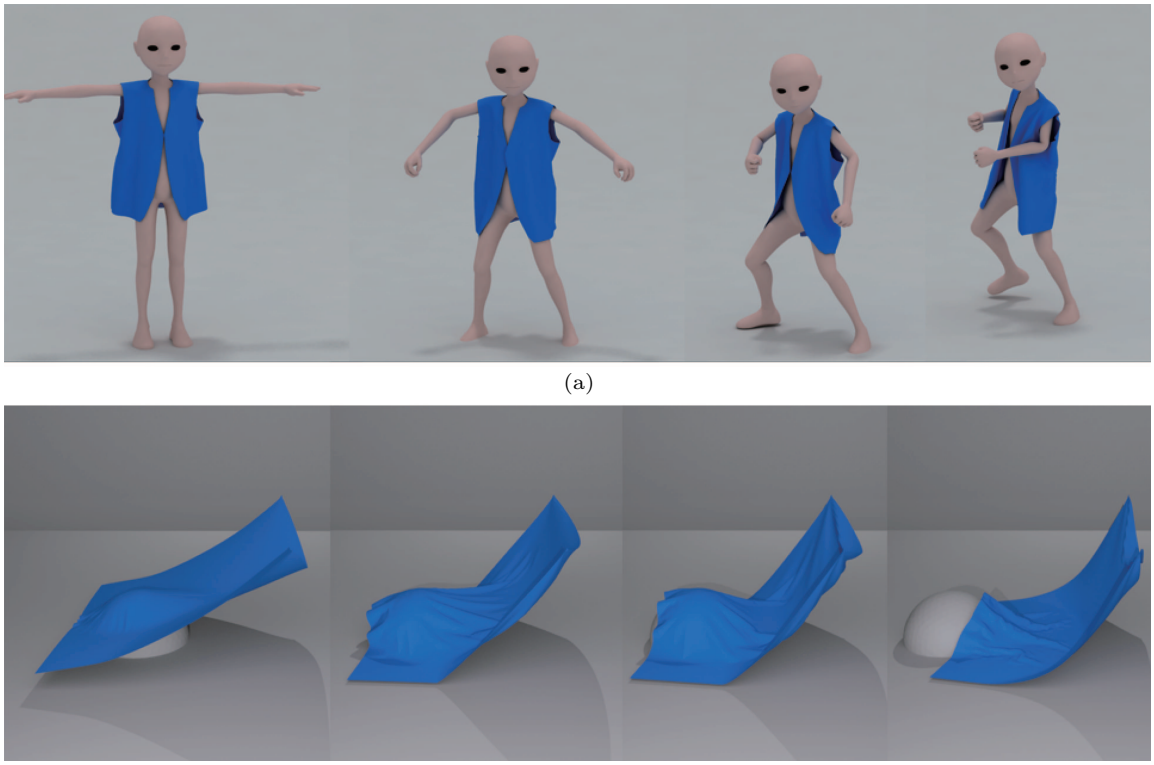


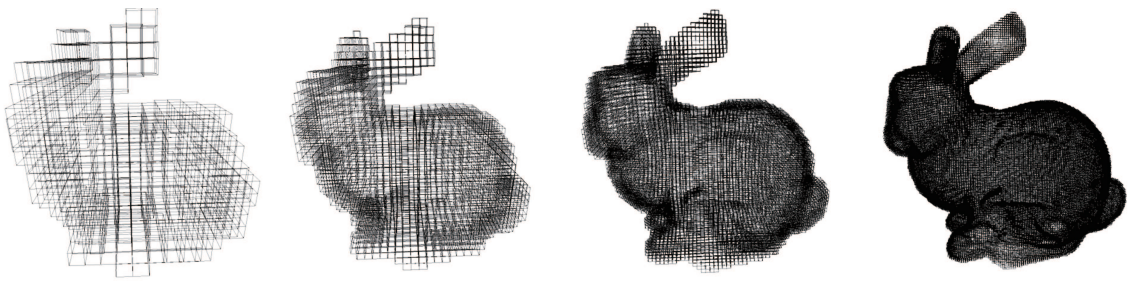
Fig.7. Performance comparison with [3]: we observe up to 3x speedups among all the benchmarks. These speedups are obtained due to better parallelism, reuse of BVH, frame-to-frame coherence, and culling of redundant queries.



(a)

(b)

Fig.8. Benchmarks for deformable models. We use two complex benchmarks, (a) Andy and (b) Sphere, and our ADF algorithm can compute distance fields at 15 FPS–22 FPS.



(a) (b) (c) (d)

Fig.9. Performance with different resolutions: depth represents the maximum degree of subdivision during the construction of the octree. The depth field resolutions of 5, 6, 7, and 8 are 32^3, 64^3, 128^3, and 256^3, respectively. (a) depth = 522.5 ms. (b) depth = 623.3 ms. (c) depth = 725.9 ms. (d) depth = 836.4 ms.

There are many avenues for future research. Aiming at overcoming the limitations, we would like to extend our algorithm from using a single GPU to multiple GPUs for interactive distance field computation on deformable models with millions of triangles.

References

[1] Jones M W, Bærentzen A, Srámek M. 3D distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 2006, 12(4): 581-599. DOI: 10.1109/TVCG.2006.56.

[2] Jones M W, Chen M. A new approach to the construction of surfaces from contour data. *Computer Graphics Forum*, 1994, 13(3): 75-84. DOI: 10.1111/1467-8659.1330075.

[3] Liu F, Kim Y J. Exact and adaptive signed distance fields computation for rigid and deformable models on GPUs. *IEEE Transactions on Visualization and Computer Graphics*, 2014, 20(5): 714-725. DOI: 10.1109/TVCG.2013.268.

[4] Morton G M. A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. IBM Ltd., 1966.

[5] Frisken S F, Perry R N, Rockwood A P, Jones T R. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. the 27th Annual Conference on Computer Graphics and Interactive Techniques*, July 2000, pp.249-254. DOI: 10.1145/344779.344899.

[6] Krayer B, Müller S. Generating signed distance fields on the GPU with ray maps. *The Visual Computer*, 2019, 35(6/7/8): 961-971. DOI: 10.1007/s00371-019-01683-w.

[7] Jamriška O. Interactive ray tracing of distance fields. In *Proc. the 14th Central European Seminar on Computer Graphics*, May 2010.

[8] Mitchell N, Aanjaneya M, Setaluri R, Sifakis E. Nonmanifold level sets: A multivalued implicit surface representation with applications to self-collision processing. *ACM Transactions on Graphics*, 2015, 34(6): Article No. 247. DOI: 10.1145/2816795.2818100.

[9] Calakli F, Taubin G. Ssd: Smooth signed distance surface reconstruction. *Computer Graphics Forum*, 2011, 30(7): 1993-2002. DOI: 10.1111/j.1467-8659.2011.02058.x.

[10] Hoff K E, Keyser J, Lin M, Manocha D, Culver T. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proc. the 26th Annual Conference on Computer Graphics and Interactive Techniques*, August 1999, pp.277-286. DOI: 10.1145/311535.311567.

[11] Kerwin T, Hittle B, Shen H W, Stredney D, Wiet G. Anatomical volume visualization with weighted distance fields. In *Proc. Eurographics Workshop on Visual Computing for Biomedicine*, July 2010, pp.117-124. DOI: 10.2312/VCBM/VCBM10/117-124.

[12] Frisken S F, Perry R N. Designing with distance fields. In *Proc. the 2006 ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques*, July 30-August 3, 2006, pp.60-66. DOI: 10.1145/1185657.1185675.

[13] Bastos T, Celes W. GPU-accelerated adaptively sampled distance fields. In *Proc. the 2008 IEEE International Conference on Shape Modeling and Applications*, June 2008, pp.171-178. DOI: 10.1109/SMI.2008.4547967.

[14] Cao T T, Tang K, Mohamed A, Tan T S. Parallel banding algorithm to compute exact distance transform with the GPU. In *Proc. the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Feb. 2010, pp.83-90. DOI: 10.1145/1730804.1730818.

[15] Fischer I, Gotsman C. Fast approximation of high-order Voronoi diagrams and distance transforms on the GPU. *Journal of Graphics Tools*, 2006, 11(4): 39-60. DOI: 10.1080/2151237X.2006.10129229.

[16] Hsieh H H, Tai W K. A simple GPU-based approach for 3D Voronoi diagram construction and visualization. *Simulation Modelling Practice and Theory*, 2005, 13(8): 681-692. DOI: 10.1016/j.simpat.2005.08.003.

[17] Rong G, Tan T S. Variants of jump flooding algorithm for computing discrete Voronoi diagrams. In *Proc. the 4th International Symposium on Voronoi Diagrams in Science and Engineering*, July 2007, pp.176-181. DOI: 10.1109/ISVD.2007.41.

[18] Wu X, Liang X, Xu Q, Zhao Q. GPU-based feature preserving distance field computation. In *Proc. the 2008 International Conference on Cyberworlds*, Sept. 2008, pp.203-208. DOI: 10.1109/CW.2008.62.

[19] Zhou K, Gong M, Huang X, Guo B. Data-parallel octrees for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 2011, 17(5): 669-681. DOI: 10.1109/TVCG.2010.75.

[20] Karras T. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In *Proc. the 4th Eurographics Conference on High-Performance Graphics*, June 2012, pp.33-37. DOI: 10.2312/EGGH/HPG12/033-037.

[21] Bédorf J, Gaburov E, Zwart S P. A sparse octree gravitational N-body code that runs entirely on the GPU processor. *Journal of Computational Physics*, 2012, 231(7): 2825-2839. DOI: 10.1016/j.jcp.2011.12.024.

[22] Morrical N, Edwards J. Parallel quadtree construction on collections of objects. *Computers & Graphics*, 2017, 66: 162-168. DOI: 10.1016/j.cag.2017.05.024.

[23] Zhou K, Hou Q, Wang R, Guo B. Real-time KD-tree construction on graphics hardware. *ACM Transactions on Graphics*, 2008, 27(5): Article No. 126. DOI: 10.1145/1409060.1409079.

[24] Sramek M, Kaufman A E. Alias-free voxelization of geometric objects. *IEEE Transactions on Visualization and Computer Graphics*, 1999, 5(3): 251-267. DOI: 10.1109/2945.795216.

[25] Breen D E, Mauch S, Whitaker R T. 3D scan conversion of CSG models into distance volumes. In *Proc. IEEE Symposium on Volume Visualization*, Oct. 1998, pp.7-14. DOI: 10.1109/SVV.1998.729579.

[26] Kimmel R. Fast marching methods for computing distance maps and shortest paths. Technical Report, Lawrence Berkeley National Laboratory, 1996. https://escholarship.org/uc/item/7kx079v5, Nov. 2021.

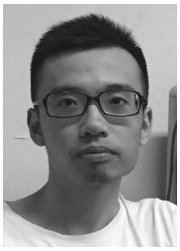
[27] Yin K, Liu Y, Wu E. Fast computing adaptively sampled distance field on GPU. In *Proc. the 19th Pacific Conference on Computer Graphics and Applications*, Sept. 2011. DOI: 10.2312/PE/PG/PG2011short/025-030.

[28] Ségonne F, Pacheco J, Fischl B. Geometrically accurate topology-correction of cortical surfaces using nonseparating loops. *IEEE Transactions on Medical Imaging*, 2007, 26(4): 518-529. DOI: 10.1109/TMI.2006.887364.

[29] Tang M, Wang T T, Liu Z Y, Tong R F, Manocha D. I-Cloth: Incremental collision handling for GPU-based interactive cloth simulation. *ACM Transaction on Graphics*, 2018, 37(6): Article No. 204. DOI: 10.1145/3272127.3275005.

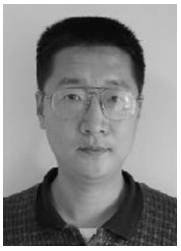
Xiao-Rui Chen received his B.S. degree in computer science from Shandong University, Jinan, in 2016. He is currently a Ph.D. candidate in Zhejiang University, Hangzhou. His current research interests include clothes simulation, GPU-algorithm, collision detection, and collision response.

Min Tang received his B.S., M.S., and Ph.D. degrees from Zhejiang University, Hangzhou, in 1994, 1996 and 1999, respectively. He is a professor of the College of Computer Science and Technology, Zhejiang University, Hangzhou. From June 2003 to May 2004, he was a visiting scholar at Wichita State University, Wichita. Between April 2007 and April 2008, and September 2016 and April 2017, he was a visiting scholar at the University of North Carolina at Chapel Hill. His research interests include collision detection/handling, cloth simulation, and GPU-based computation acceleration.



Cheng Li received his M.S. degree in computer science from Zhejiang University, Hangzhou, in 2016. He is currently working at Tencent. His current research interests include clothes simulation, GPU-algorithm, collision detection, and collision response.

Dinesh Manocha received his B.T. degree in computer science and engineering from the Indian Institute of Technology, Delhi, in 1987, and his M.S. and Ph.D. degrees in computer science at the University of California at Berkeley in 1990 and 1992, respectively. His research interests include geometric computing, interactive computer graphics, physics-based simulation and robotics. He has published more than 280 papers in these areas. He has received more than 11 Best Paper and Panel Awards at the ACM SuperComputing, ACM Multimedia, ACM Solid Modeling, Pacific Graphics, IEEE VR, IEEE Visualization, ACM SIGMOD, ACM VRST, CAD, I/ITSEC and Eurographics Conferences. He was selected as an ACM Fellow in 2009 "for contributions to geometric computing and applications to computer graphics, robotics and GPU computing", and is also an AAAS Fellow and an IEEE Fellow.



Ruo-Feng Tong received his B.S. degree from Fudan University, Shanghai, in 1991, and his Ph.D. degree from Zhejiang University, Hangzhou, in 1996. He is a professor of the College of Computer Science and Technology, Zhejiang University, Hangzhou. From June 1999 to May 2001, he was a visiting scholar at Hiroshima University, Hiroshima. His current research interests include virtual reality, computer vision, and artificial intelligence-based assisted medical diagnosis.

JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY

Volume 37, Number 3, May 2022

Content

JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY
《计算机科学技术学报》

Volume 37 Number 3 2022 (Bimonthly, Started in 1986)

Indexed in: **SCIE, Ei, INSPEC, JST, AJ, MR, CA, DBLP**

Edited by:
THE EDITORIAL BOARD OF JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY
Zhi-Wei Xu, Editor-in-Chief, P.O. Box 2704, Beijing 100190, P.R. China
Managing Editor: Feng-Di Shu E-mail: jcst@ict.ac.cn http://jcst.ict.ac.cn Tel.: 86-10-62610746

Sponsored by: Institute of Computing Technology, CAS & China Computer Federation
Supervised by: Chinese Academy of Sciences
Undertaken by: Institute of Computing Technology, CAS
Published by: Science Press, Beijing, China
Printed by: Beijing Baochang Color Printing Co. Ltd
Distributed by:
China: All Local Post Offices
Other Countries: Springer Nature Customer Service Center GmbH, Tiergartenstr. 15, 69121 Heidelberg, Germany
Available Online: https://link.springer.com/journal/11390

ISSN 1000-9000 05>

国内统一刊号: CN11-2296/TP 国内邮发代号: 2-578 RMB ¥160.00