Pacific Graphics (2011)
Bing-Yu Chen, Jan Kautz, Tong-Yee Lee, and Ming C. Lin (Editors)

Fast Computing Adaptively Sampled Distance Field on GPU

K. Yin[1], Y. Liu[1,2] and E. Wu[2,3]

[1]School of Information Engineering, Chang'an Univeristy, China
[2]Institute of Software, Chinese Academy of Sciences, China
[3]FST, University of Macau, Macao, China

Abstract

In this paper we present an efficient method to compute the signed distance field for a large triangle mesh, which can run interactively with GPU accelerated. Restricted by absence of flexible pointer addressing on GPU, we design a novel multi-layer hash table to organize the voxel/triangle overlap pairs as two-tuples, such strategy provides an efficient way to store and access. Based on the general octree structure idea, a GPU-based octree structure is given to generate the sample points which are used to calculate the shortest distance to the triangle mesh. Classifying sample points into three types provides a well tradeoff between performance and precision, and when implementing the algorithm on GPU, these samples are also organized into blocks to share the triangles among threads to save bandwidth. Finally we demonstrate efficient calculation of the global signed distance field for some typical large triangle meshes with pseudo-normal method. Compared to previous work, our algorithm is quite fast in performance.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Boundary representations;I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types

1. Introduction

Distance field is a volumetric dataset that defines the shortest distance between points and the boundary of an object. There are a lot of applications of signed or unsigned distance field, such as collision detection [KLCS04], visualization [KHS*10], and modeling tools [PF01, FP05]. However, the complexity of a naive method to generate distance field is O(nm) where n denotes the number of sample points and m denotes the number of triangles. So the computation of distance field is a slow procedure and many researchers work on the acceleration techniques.

To speed up the calculation of adaptively sampled distance field(ADF), we present several novel strategies. One is that we design a multi-layer hash table to store the intersecting triangle list for each voxel, so it is efficient to store the triangle indices without too much memory overhead. And like in most available methods, distance field is adaptively sampled through an octree-like structure . The novel part of sampling process is that samples are classified into three types so that different samples will take different calculation cost. As the result of sample classification, Only type-1(see sec.5.2) samples have the accurate distances computed, while type-2 and type-3 samples have approximated distances. Since type-2 and type-3 samples are relatively far from the mesh, we can gain only small improvement on relative error bound by computing an accurate value for them.

With these techniques, high resolution distance field computation for a large triangle mesh could be done interactively. For the buddha model with 1,087,716 triangles under a resolution of 512^3, the total time of ADF generation is still below 1 second(See Table 1).

2. Related Work

There are plenty of works about distance field construction, [JBS06] gives a good survey.

A typical method is based on Voronoi diagram. [SOM04, SGGM06] use GPU rasterization to calculate the distance field for each slice of a uniform spatial grid with Voronoi diagram bounds. [SPG03] uses GPU-based prism scan method with each primitive assigned a simple polyhedron enclosing its Voronoi cell. [ED08] presents a tetrahedra GPU scan method with bounded volumes for each primitive which

fixes the leaking artifacts sometimes produced by the prism scan.

Another popular solution is propagation method. [CK07] designs a fast hierarchical GPU-based algorithm for a voxel grid as stacks of 2D textures. [CCI08] constrains the signed distance field calculation within a narrow band around the surface of the polygonal model and uses sweeping method to compute the global field.

In addition, many ADFs works are proposed to allow efficient memory usage and adaptive accuracy. [PF01, FP05, F-PRJ00] use ADFs as a designing tool to create detailed shapes. [LH07] designs a primal tree structure which gives a high compression ratio with efficient random-access capability. [BC08] employs a 3D hashing scheme to store the adaptively sampled distance field for fast ADFs reconstruction on GPU.

With GPU programmability prosperity, distance field computation is often accelerated with rasterization and fragment programs [SPG03, SOM04, SGGM06, CK07, ED08]. However, with latest GPU progress, more flexible capability can be taken advantage of to alleviate the heavy computation burden. [PLKK10] uses GPU with CUDA to accelerate the signed distance field calculation. Since each thread processes only one triangle and iterates for each sample point, it is still less efficient.

3. Algorithm Overview

Figure 1 gives the data flow chart of our algorithm. Firstly we voxelize the triangle mesh and store intersecting triangle lists for each triangle into a well designed multi-layer hash table. In the meantime, the lowest level of the octree will also be built. Then from the bottom up the octree will be constructed. After this, samples could be determined by the constructed octree. Finally, we will compute the distance value for each sample and store the result into a simple hash table.

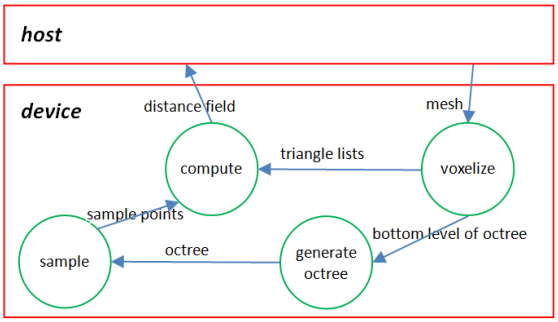

Figure 1: *The data flow chart of our algorithm.*

4. Intersecting Triangle Lists Building

This section gives how to determine whether a triangle overlaps with a voxel and how to store these intersecting triangle lists.

4.1. Determine Triangles That Overlap With Each Voxel

Firstly we calculate the axis-aligned bounding box (AABB) for each triangle, and then check all voxels overlapping with the AABB or inside the AABB to decide whether the triangle itself overlaps or partially overlaps these voxels or not. If overlapping happens, the triangle index t and voxel index c will be written into memory as a two tuple (t, c). As to the overlap test of one triangle with one voxel, [SS10] gives an efficient solution. Our voxelization part is based on their work:

- Judge whether the voxel overlaps with the plane containing the triangle;
- Judge whether the voxel overlaps with the 2D projections of the triangle on the three coordinate planes;
- If and only if the above two judgments are true, the voxel and the triangle overlap with each other.

4.2. Triangle Lists Storing

Since we cannot predict the number of triangles that overlap with a voxel, we could not allocate a fixed size memory space for the voxel to store its triangle list. Furthermore, it is difficult to allocate memory on GPU dynamically. To solve this problem, hash table can be used. However, the great difference of each voxel's list length will lead to overflow easily. If all overflow data are put into a single buffer, the buffer has to be designed extremely large. The cost to transverse the overflow buffer will be unacceptable as a result.

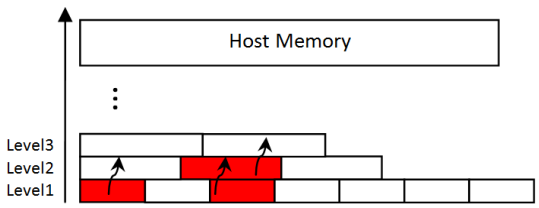

Figure 2: *Multi-layer hash table structure. Red color means the bucket full. Arrow indicates the bucket where the overflowing data will be put.*

To solve these above problems, we design a multi-layer hash table on GPU as shown in Figure 2. Our hashing method is a variation of cuckoo hashing [PR01]. Each layer is a list of buckets, and from bottom up the bucket number will become smaller and the volume higher. And a watchdog is used to count the available space for each bucket. When

the bucket in the bottom layer corresponding to one hash value of a two tuple (t, c) becomes full, another hash value will be calculated to make the two tuple stored in higher layer. If the top bucket is full, host memory will be used as overflow buffer. In fact, this rarely happens when hierarchy and granularity are designed well.

5. Octree and Samples

Without any pointers to memory, we design a very simple octree structure which provides good support for the distance field computation.

5.1. Octree Data Structure

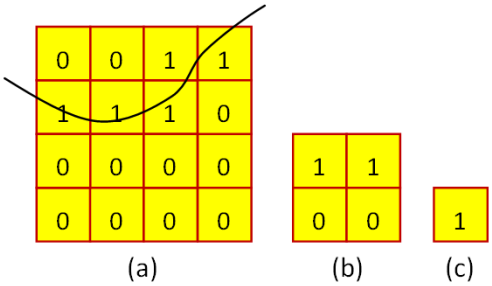

Figure 3: *A quadtree with depth 2 as illustration, here (a), (b), (c) stands for level-2, level-0, level-1 respectively.*

A full octree is used to divide the space into hierarchical nodes so that each node in the lowest level corresponds to a voxel. Each node is represented with 1 bit. If the bit equals 1, the node overlaps the mesh, and 0 means no overlap at all. Each level of octree are represented by a 3D array. Similar to mipmap, a 2D quadtree is used to illustrate the 3D array in Figure 3. Such kind of data structure provides random access capability, and easy traversing between parents and children. Obviously there is a limitation that many zeros are stored for many empty nodes. For a case of 256^3 resolution, 18.3MB memory will be needed. So this structure can only be used to accelerate the distance calculation. When the computation is done, this octree structure can be converted to other spatial efficient structure.

5.2. Sample Points Classification

Different from most existing works that treat all samples in the same way, we classify them into 3 types. For those points far from the surface, it is really unnecessary to compute an accurate distance value for them, since by it we can gain only small improvement on relative error bound. Moreover, accurate distance value of points far from mesh need much more triangles to be traversed. And these points are few in number, which is unsuitable for GPU computing. As shown in

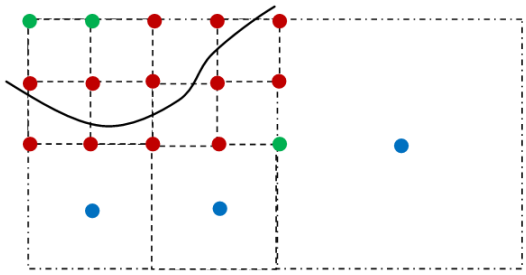

Figure 4: *Sample points are classified into three types. (Red means type-1, green means type-2, blue means type-3)*

Figure 4 in 2D, sample points are classified into three types: The type-1 sample points come from the corners of the lowest level nodes that marked as 1 (red points); the type-2 sample points are those points on the boundary of or in the 1-marked second lowest level node apart from those type-1 sample points (green points); all 1-marked non-lowest-level nodes' siblings are sampled at center as type-3 points (blue points).Hence type-1 sample points come directly from the corners of voxels, so they can represent the mesh exactly. We sample those type-3 points at the centers of the large nodes rather than the corners for fast computation. (see sec.7.2)

6. Distance Calculation of Type-1 Sample Points

In this section the method to calculate the distance value of type-1 samples is given. For type-1 samples, the calculation of diatance from a certain point to a certain triangle is based on the angle-weighted pseudo normal method [BA02, BA05].

6.1. Scope That the Nearest Triangle Locates

Given a point P, according to the above methods about voxelization and sampling, if P is a type-1 sample, the 8 neighbor voxels will contain at least one triangle. Assuming the edge length of the voxel is d, the maximum distance value from P to the mesh is $\sqrt{3}d$ (it's $\sqrt{2}d$ in 2D) as shown in Figure 5 with a 2D case. The closest triangle to P will overlap the sphere with center at P and radius $\sqrt{3}d$, it will also overlap the circum-cube (in blue color) of the sphere (in red color), so the closest triangle must overlap one or more voxels among the 4*4*4 voxels around P. In consequence, only the triangles contained in these 64 voxels will be visited to get the shortest distance. With the increase of the resolution, the triangles contained in the 64 voxels will be less. So this algorithm is efficient to process high resolution problems.

6.2. Parallel Computation in Blocks

However, the distribution of the sample points are discontinuous in space so that it is difficult to dispatch them to each

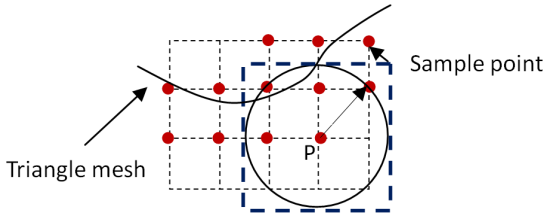

Figure 5: *Traverse range in 2D space for type-1 sample points.*

thread in parallel. Even if we dispatch them to each thread successfully, there is no enough on-chip memory for these overlapping triangles since each triangle has 3 vertices and 7 normal vectors including edge normal, vertices normal and surface normal. Those attributes require totally 120 bytes. Moreover, the neighbor points may visit the triangle list from the same voxels. How to take advantage of this property to save bandwidth should also be considered. To solve above problems, block strategy is used as shown in Figure 6.

Suppose the size of a block is n*n*n. The closest triangles to one sample point are contained in the 4*4*4 voxels centered around that point, so the closest triangles to the sample points in the block are contained in the (n+4)*(n+4)*(n+4) voxels centered around the center of the block. During the distance calculation for each point inside the block, all closest triangles will be visited to find the shortest distance.

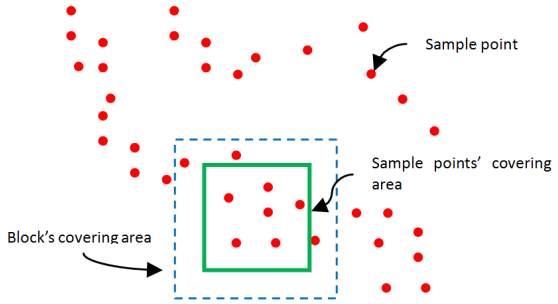

Figure 6: *Block strategy enables samples in the same block to share memory loading.*

With the block strategy, the triangle lists shared by many sample points only need to be loaded once. To overcome the disadvantage caused by the discontinuity of sample point distribution, these points in the same block can be compressed into a linear table, which can be achieved through atomic operation on on-chip memory.

7. Approximate Distances of Other Samples

As mentioned above, the distance of type-1 sample points can be calculated accurately. In this section we will discuss the approximate distance value of type-2 and type-3 samples.

7.1. Approximate Distances of Type-2 Samples

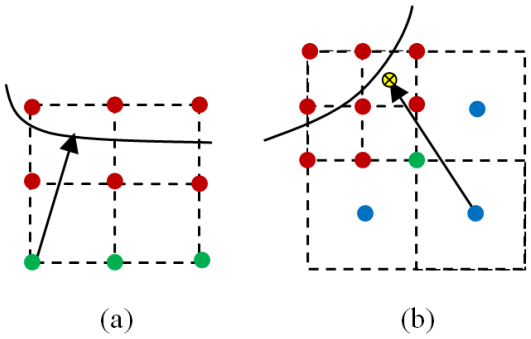

Figure 7: *Computing approximate distances of type-2 (green points) and type-3 samples (blue points).*

The distances of type-1 sample points to triangle mesh are accurate, but for type-2 points, just approximate distance values are given. As shown in Figure 7 (a), for a type-2 point, we will first determine which second lowest level node the point intersects, then read in the overlapping triangles of its sub-nodes to calculate the shortest distance. That is to say, when calculating the distances of the type-2 sample points, only eight overlapping triangle lists are considered.

7.2. Approximate Distances of Type-3 Samples

If one node is marked as 0 and not from the lowest level, and its parent is marked as 1, then the center of this node is a type-3 sample point. Such kind of points are far from the triangle mesh so it is unnecessary to calculate the distance directly, which is time-consuming. Hence we provide an alternative efficient method to approximate the exact distance. First, find the 1-marked sibling nodes of the node which contains the sample points, and then traverse these siblings' lowest level descendant nodes and calculate the distances of the sample point to the centers of these nodes. At last the shortest distance is the distance of the sample point to the triangle mesh.

Then we will determine the sign of the distance which is deduced directly from type-1 distances. Suppose the type-3 sample point's position is $\mathrm{S}(s_0,s_1,s_2)$, and the closest lowest level node's minimum corner's position is $\mathrm{V}(v_0,v_1,v_2)$, then the type-1 point $\mathrm{W}(w_0,w_1,w_2)$ shares the same sign with S as follows:

$$w_i = \begin{cases} v_i, & s_i \le v_i \\ v_{i+1}, & s_i > v_i \end{cases}$$

8. Experiments and Discussion

The described algorithm has been tested on a computer with an Intel Core i7 2.67GHz CPU and an NVIDIA Geforce GTX480 (1.5GB) graphics card. We test multiple models at three different resolutions, and Table 1 lists the detailed performance data. As shown in the table, these models have different amount of triangles. To illustrate the performance data clearer, Figure 8 shows that more triangles or higher resolution will take more time to calculate the global signed distance field.

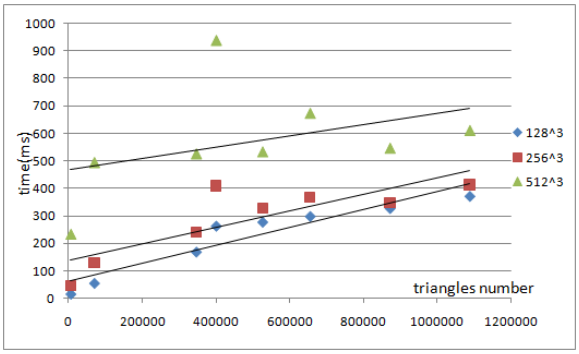

Figure 8: *The time taken for different triangle amounts and at different resolutions.*

Figure 9 shows the Stanford bunny model's distance field which is color-coded at a resolution of 256^3. Since different precision is used for different types of sample points, the image looks so blocky. However, in the overlap area of the right image, we can find it smoother with higher precision.

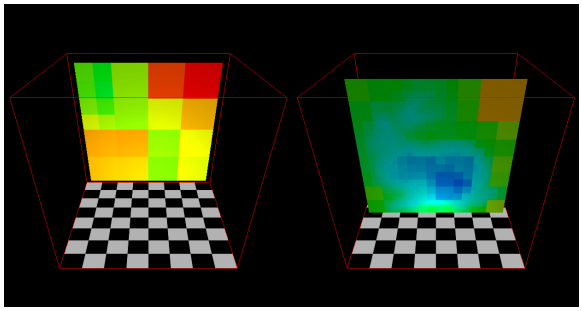

Figure 9: *Slices to display the bunny distance value which is color-coded at resolution* 256^3*, increasing from blue to green to red. The left one is the slice at minimum z, the right one is the slice at middle z which cuts the model.*

Our work can calculate signed ADF of large triangle models interactively. For reasonable resolutions, the total time is below 1 second. Here we also give some applications of the global signed distance field since the generation of ADFs is quite fast. Figure 10 gives a typical CSG operation based on ADFs [FP05]. We use the re-constructed geometry from the ADFs for the rendering.

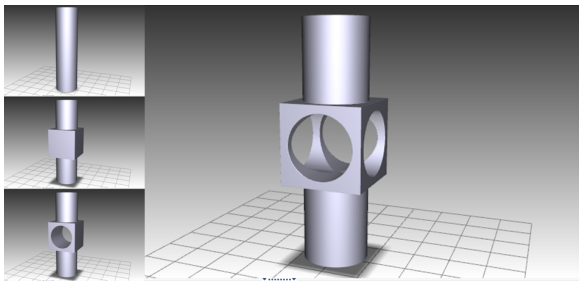

Figure 10: *CSG operation based on ADFs (the left column shows the procedure of CSG operations, the right is the final result).*

9. Conclusion and Future Work

We have presented a new algorithm to generate ADFs interactively for large triangle meshes on GPU. Unlike previous work, our method performs all computations on GPU. The multi-layer hash table provides a flexible method to store the triangle indices for each voxel. Based on the octree structure on GPU, these sample points are classified into 3 types, so some unnecessary computation burden is alleviated. In our implementation, the most time-consuming calculation part is improved with block trick. With these acceleration techniques used, the performance is much better than previous work.

In the future, we'd like to compress the data structure to process higher resolution problems. Another direction is to applay ADFs into fluid simulation, which may help the fluid-solid coupling by representing the solids as ADFs.

10. Acknowledgements

This research was supported by the National Grand Fundamental Research 973 Program of China under Grant No.2009CB320802, National Natural Science Foundation of China under Grant No. 60973066, 60833007, Open fund of the State Key LCS, Institute of Software, Chinese Academy of Sciences(SYSKF1004), Program for Changjiang Scholars and Innovative Research Team in University(IRT0951). Thank NVIDIA for providing us graphics cards. The Armadillo, Lucy, Buddha, Dragon and Bunny models were provided by the Stanford Computer Graphics Laboratory. The Heptoroid model was provided by UC Berkeley Rapid Prototyping Project.

References

[BA02] BAERENTZEN J., AANAES H.: Generating signed distance fields from triangle meshes. In *IMM-TECHNICAL REPORT-2002-21* (2002), vol. 1, pp. 1–34. 3

Table 1: *The performance of ADFs generation for different triangle meshes at different resolutions. (time unit: ms)*

Model	Resolution	Voxelization time	Octree building time	Type-1 sample points		Type-2 sample points		Type-3 sample points		Total time (ms)
				amount	time	amount	time	amount	time	
Cow	128^3	1.09	0.63	36920*s*	10.9	17926	3.1	6002	1.26	16.98
(5,804*tris*)	256^3	4.37	2.66	148227	25	73127	6.2	24357	8.74	46.97
	512^3	22.97	10.94	590792	98.4	297332	37.5	98761	65	234.81
Bunny	128^3	1.56	1.56	93989	43.8	46296	7.8	15650	1.56	56.28
(6,9666*tris*)	256^3	5	5.78	376533	86	188136	21.9	62686	11.09	129.77
	512^3	20.48	19.69	1508725	303.1	750938	76.5	251544	75.62	495.39
Armadillo	128^3	2.97	1.09	65518	148.5	31291	15.6	10665	1.57	169.73
(345,944*tris*)	256^3	3.91	4.22	266522	193.7	128695	29.7	43012	10.01	241.54
	512^3	8.13	15	1072189	348.4	530600	84.3	175020	70.79	526.62
Heptoroid	128^3	4.38	2.04	101787	239	35259	17.1	12517	1.4	263.92
(400,000*tris*)	256^3	7.03	6.25	476910	345.3	183330	40.6	54720	10.15	409.33
	512^3	15.78	19.37	1916187	673.4	950651	156.2	291894	74.21	938.96
Lucy	128^3	4.54	0.63	35980	254.7	14819	17.2	4793	1.24	278.31
(525,814*tris*)	256^3	5	2.81	153438	282.9	68426	28.1	21910	9.22	328.03
	512^3	7.81	11.25	629573	375	300372	73.5	97174	66.71	534.27
Hand	128^3	6.09	0.63	30920	273.4	12603	15.7	4503	3.29	299.11
(654,666*tris*)	256^3	7.03	2.65	132323	301.6	57851	29.7	18765	25.77	366.75
	512^3	9.69	10.32	542166	379.6	260806	70.3	82821	204.06	673.97
Dragon	128^3	7.03	0.94	66669	292.2	31396	26.5	10391	1.4	328.07
(871,414*tris*)	256^3	7.18	4.22	264951	292.2	133816	35.9	43821	9.54	349.04
	512^3	13.28	13.75	972866	371.9	556057	79.7	182630	68.74	547.37
Buddha	128^3	8.59	0.93	50747	331.3	21973	29.7	6910	1.56	372.08
(1,087,716*tris*)	256^3	8.28	3.28	209189	348.4	98047	43.8	31343	9.7	413.46
	512^3	13.75	12.19	802186	429.7	430346	87.5	137170	68.59	611.73

[BA05] BAERENTZEN J., AANAES H.: Signed distance computation using the angle weighted pseudonormal. *TVCG 11*, 3 (Sept. 2005), 243–253. 3

[BC08] BASTOS T., CELES W.: Gpu-accelerated adaptively sampled distance fields. In *Proc. IEEE International Conference on Shape Modeling and Applications 2008* (2008), vol. 31, pp. 171–178. 2

[CCI08] CHANG B., CHA D., IHM I.: Computing local signed distance fields for large polygonal models. *Computer Graphics Forum 27*, 3 (Sept. 2008), 799–806. 2

[CK07] CUNTZ N., KOLB A.: Fast hierarchical 3d distance transforms on the gpu. In *Proc. Eurographics 2007* (2007), vol. 31, pp. 93–96. 2

[ED08] ERLEBEN K., DOHLMANN H.: *GPU Gem3: Chapter 34. Signed Distance Fields Using Single-Pass GPU Scan Conversion of Tetrahedra*. Addison-Wesley, 2008. 1, 2

[FP05] FRISKEN S., PERRY R.: Designing with distance fields. In *ACM SIGGRAPH 2006 Courses* (2005), pp. 60–66. 1, 2, 5

[FPRJ00] FRISKEN S., PERRY R., ROCKWOOD A., JONES T.: Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. SIGGRAPH 2000* (2000), pp. 249–254. 2

[JBS06] JONES M., BAERENTZEN J., SRAMEK M.: 3d distance fields: A survey of techniques and applications. *TVCG 12*, 4 (Sept. 2006), 581–599. 1

[KHS*10] KERWIN T., HITTLE B., SHEN H., STREDNEY D., WIET G.: Anatomical volume visualization with weighted distance fields. In *Proc. Eurographics Workshop on Visual Computing for Biology and Medicine* (2010), pp. 117–124. 1

[KLCS04] KOLB A., LATTA L., C.REZK-SALAMA: Hardware-based simulation and collision detection for large particle systems. In *Proc. Graphics Hardware* (2004), pp. 123–131. 1

[LH07] LEFEBVRE S., HOPPE H.: Compressed random-access trees for spatially coherent data. In *Proc. Eurographics Symposium on Rendering* (2007), vol. 25, pp. 339–349. 2

[PF01] PERRY R., FRISKEN S.: Kizamu: A system for sculpting digital characters. In *Proc. SIGGRAPH 2001* (2001), vol. 20, pp. 47–56. 1, 2

[PLKK10] PARK T., LEE S., KIM J., KIM C.: Cuda-based signed distance field calculation for adaptive grids. In *Proc. IEEE International Conference on Computer and Information Technology (CIT 2010)* (2010), pp. 1202–1206. 2

[PR01] PAGH R., RODLER F. F.: Cuckoo hashing. In *Proc. Annual European Symposium on Algorithms* (2001), vol. 2161, p. 121ÍC133. 2

[SGGM06] SUD A., GOVINDARAJU N., GAYLE R., MANOCHA D.: Interactive 3d distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games* (2006), pp. 117–124. 1, 2

[SOM04] SUD A., OTADUY M., MANOCHA D.: Difi: Fast 3d distance field computation using graphics hardware. *Computer Graphics Forum 23*, 3 (Sept. 2004), 557–566. 1, 2

[SPG03] SIGG C., PEIKERT R., GROSS M.: Signed distance transform using graphics hardware. In *Proc. IEEE Visualization* (2003), pp. 83–90. 1, 2

[SS10] SCHWARZ M., SEIDEL H.: Fast parallel surface and solid voxelization on gpus. *ACM TOG 29*, 6 (Dec. 2010), 179:1–179:9. 2