Fast Hierarchical 3D Distance Transforms on the GPU

Nicolas Cuntz and Andreas Kolb

Computer Graphics Group, University of Siegen, Germany

Abstract

This paper describes a fast approximate approach for the GPU-based computation of 3D Euclidean distance transforms (DT), i.e. distance fields with associated vector information to the closest object point. Our hierarchical method works on discrete voxel grids and uses a propagation technique, both on a single hierarchy level and between the levels. Using our hierarchical approach, the effort to compute the DT is significantly reduced. It is well suited for applications that mainly rely on exact distance values close to the boundary.
Our technique is purely GPU-based. All hierarchical operations are performed on the GPU. A direct comparison with the Jump Flooding Algorithm (JFA) shows that our approach is faster and provides better scaling in speed and precision, while JFA should be preferred in applications that require a more precise DT.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Curve, surface, solid, and object representations

1. Introduction

Signed or unsigned distance fields have many applications in computer graphics, scientific visualization and related areas. Examples are implicit surface representation and collision detection [KLRS04], for skeletonization [ST04] or for accelerated volume raytracing [HSS*05].

Computing a 3D Euclidean DT is a well studied problem [Cui99]. Depending on the initial object representation, as voxel grid or as explicit geometric representation, different approaches have been proposed. We consider a voxel grid, where the voxels next to the boundary are classified as interior or exterior, assuming that the boundary is closed. Concerning the voxel grid approach, there are two major categories, propagation methods and methods based on Voronoi diagrams. Propagation methods propagate the distance information to the neighboring voxels, either by spacial sweeping or by contour propagation.

GPU-based approaches have been presented for DT computation in the 2D case for voxel grid input [ST04, RT06] and for 3D polygonal input data [SPG03, SGGM06].

Our contribution: The method presented in this paper works on 3D voxel grid input models and is based on the propagation approach. The approach uses a specific hierarchical technique consisting of *push-downs* and *pull-ups* in order to achieve a logarithmic behavior by exponentially reducing the number of propagation steps needed for the DT computation. The results show that errors can be reduced significantly with minor computational costs. We compare our approach to the Jump Flooding Algorithm (JFA) [RT06] in order to provide hints for the applicability of either algorithm to specific problem domains.

The remainder of this paper is structured as follows: Sec. 2 discusses related research results. Our hierarchical approach is described in Sec. 3. In Sec. 4, experimental results in direct comparison to the JFA are provided.

2. Prior Work

This section describes the main approaches to compute DTs on voxel grids utilizing programmable Graphics Processing Units (GPUs).

Considering a closed object boundary $\delta\Omega$, the Euclidean distance transform dt for a voxel $\mathbf{P}$ is defined as $\mathrm{dt}(\mathbf{P}) = (\mathrm{dt}_d(\mathbf{P}), \mathrm{dt}_\delta(\mathbf{P}))$, where

$$\mathrm{dt}_d(\mathbf{P}) = s_\mathbf{P} \min_{\mathbf{Q}\in\delta\Omega}\{\|\mathbf{P}-\mathbf{Q}\|\}, \mathrm{dt}_\delta(\mathbf{P}) = \arg\min_{\mathbf{Q}\in\delta\Omega}\{\|\mathbf{P}-\mathbf{Q}\|\}$$

where $s_\mathbf{P}$ denotes the sign w.r.t. $\delta\Omega$ (1: exterior, -1: interior), and $\arg\min_\mathbf{Q}$ is an operator returning a point $\mathbf{Q}$ constituting

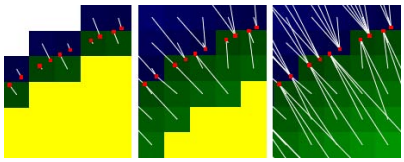

Figure 1: *Two propagation steps (white/yellow: $\pm M$, blue/green: +exterior/−interior) – note that* dt *is initialized with precise sub-pixel references in this example.*

the minimum. Thus, $\mathrm{dt}(\mathbf{P})$ stores the signed distance and the point $\mathbf{Q} \in \delta\Omega$ closest to $\mathbf{P}$.

2.1. The Voronoi Diagram Approach

A Voronoi diagram is a space partitioning into cells w.r.t. to a fixed set of points (also *sites* or *seeds*). Each cell contains all points closest to one seed. It is clear that the DT can be obtained by setting Voronoi sites onto the object's boundary.

2D Voronoi diagrams can easily be determined using rasterization techniques. Therefore, cones with a common opening angle are placed over each seed and the resulting scene is rendered from top-view using the OpenGL depth buffer function `GL_LESS`. Hoff et al. [HKL*99] extend this approach to other geometric objects and to 3D. Sigg et al. [SPG03] and Sud et al. [SGGM06] present GPU-based implementations of the Voronoi based approach.

2.2. The Propagation Approach

Consider a DT initialized close to $\delta\Omega$ in the following way:

$$\mathrm{dt}^0(\mathbf{P}) = \begin{cases} (0, \mathbf{P}) & \text{if } \mathbf{P} \in \delta\Omega \\ (\pm M, *) & \text{if } \mathbf{P} \in \Omega_{\pm} \text{ (exterior/interior)} \end{cases} \quad (1)$$

where M is greater than any distance between grid voxels. A propagation step for the distance works using a structure element $\mathcal{M}$, defining a local neighborhood:

$$\mathrm{dt}_d^{i+1}(\mathbf{P}) = s_{\mathbf{P}} \min_{\mathbf{Q} \in \mathcal{M}(\mathbf{P})} \{ \left\| \mathrm{dt}_\delta^i(\mathbf{Q}) - \mathbf{P} \right\| \}. \quad (2)$$

The sign $s_{\mathbf{P}}$ is taken from $\mathrm{dt}_d^i(\mathbf{P})$, and $\mathrm{dt}_\delta^{i+1}(\mathbf{P})$ is updated using the selected $\mathbf{Q}$. See Fig. 1 for a visualization of two sequential propagation steps.

A fast variant of this algorithm is given by Tsitsiklis [Tsi95]. This approach uses a priority queue approach to optimize the order of the DT updates. Strzodka and Telea [ST04] present a GPU based 2D approach using an arc length parametrization for $\delta\Omega$, which, in general, does not carry over to the 3D case.

Rong and Tan [RT06] present the Jump Flooding paradigm for general purpose computations on the GPU and apply it to the computation of Voronoi Diagrams and DTs. We will focus on its application to DT only.

JFA updates the whole voxel grid in each step, but varies $\mathcal{M}$ in each step in order to propagate references across longer distances. In the k-th step, $\mathcal{M}(p_x, p_y, p_z)$ is a $3 \times 3 \times 3$ voxel sub-grid defined as $\{(p_x \pm k, p_y \pm k, p_z \pm k)\}$. For a voxel grid with resolution n^3 the JFA can be summarized as follows:

1. initialize the voxel grid according to Eq. (1)
2. for $k = n/2, n/4, \ldots, 1$ do
   2.1. reference (and thus distance) propagation for all voxels $\mathbf{P} = (p_x, p_y, p_z)$ using $\mathcal{M}(\mathbf{P}) = \{(p_x \pm k, p_y \pm k, p_z \pm k)\}$

The JFA makes $\log(n)$ loops over the whole voxel grid. Rong and Tan [RT06] prove various properties of the JFA and present extensions to the algorithm in order to improve the accuracy. The two JFA variants are JFA+j, meaning that j additional propagation steps with step-size $2^{j-1}, \ldots, 1$ are performed, and JFA^2, where JFA is applied twice.

In comparison to JFA, the main benefit of our approach is an asymptotically faster run-time and better scalability between precision and speed. Moreover, caching is exploited by storing hierarchy levels in separate textures.

3. Fast Hierarchical Algorithm

This section describes the main steps involved in our hierarchical algorithm.

3.1. Algorithmic Overview

Assuming a voxel grid with initialized dt according to Eq. (1), our Fast Hierarchical Algorithm (FHA) consists of an iterative down-sampling phase (push-down, see Fig. 2) and an up-sampling phase (pull-up). During push-down, the voxel grid resolution is reduced by 2 in each step, defining $\mathcal{M}$ as the direct $2 \times 2 \times 2$ neighborhood w.r.t. the center of a cell on the finer level. For pull-up also $2 \times 2 \times 2$ neighborhoods are used, here w.r.t. the coarser grid. In the push-down phase, the DT gets coarser and more imprecise. A full reduction to a single voxel is not meaningful, thus the number of levels is limited to $N < \log(n)$. For pull-up, the DT from the finer and the coarser level are combined.

For a voxel grid with resolution n^3 the proposed FHA can be summarized as follows:

1. initialize the voxel grid according to Eq. (1)
2. for level $k = 1, \ldots, N$ (push-down)
   2.1. reference (and thus distance) propagation for all voxels on the coarse grid $\mathbf{P} = (p_x, p_y, p_z)$ using $2 \times 2 \times 2$ neighborhood
3. compute DT on the coarsest level by repeating propagation steps or by using JFA
4. for level $k = N-1, \ldots, 0$ (pull-up)
   4.1. combine DT on level $k+1$ with DT on level k using $2 \times 2 \times 2$ neighborhood

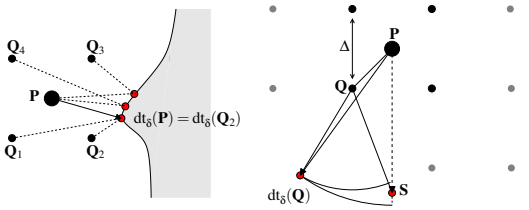

Figure 2: *Push-down (left): Computing* ${}^{k+1}\mathrm{dt}$ *for samples in a coarse grid. The closest ref. point w.r.t. level k triggers the push-down to level* $k+1$ *— error (right):* $\mathrm{dt}_\delta(\mathbf{Q})$ *is closer to* $\mathbf{Q}$ *than* $\mathbf{S}$ *while* $\mathbf{S}$ *is closer to* $\mathbf{P}$ *than* $\mathrm{dt}_\delta(\mathbf{Q})$.

The FHA usually incorporates j additional steps on each level during pull-up, yielding FHA+j, either using standard $3\times3\times3$ propagation steps (see Eq. (2)) or JFA steps for $k=2^{j-1},\ldots,2,1$. Clearly, the FHA is an approximate algorithm. Fig. 2, right, shows a situation where an error occurs.

3.2. Push-down

In the push-down pass (from fine to coarse resolution), distance information for voxels touching the object boundary are propagated to lower hierarchy levels.

This is done by super-sampling surrounding voxels, using a factor of 2 in each dimension. The DT ${}^{k}\mathrm{dt}$ is combined and propagated from level k to level $k+1$ using the following update rule (see Fig. 2, left):

$$ {}^{k+1}\mathrm{dt}_d({}^{k+1}\mathbf{P}) = s_{{}^{k+1}\mathbf{P}} \cdot \min_{{}^{k}\mathbf{Q}\in\mathcal{N}_2({}^{k+1}\mathbf{P})} \{ \left\| {}^{k}\mathrm{dt}_\delta({}^{k}\mathbf{Q}) - {}^{k+1}\mathbf{P} \right\| \} $$

with sign taken from ${}^{k}\mathrm{dt}_d({}^{k+1}\mathbf{P})$, and ${}^{k+1}\mathrm{dt}_\delta({}^{k+1}\mathbf{P})$ is updated using the selected ${}^{k}\mathbf{Q}$. Here, ${}^{k}\mathbf{P}$ and $\mathcal{N}_2$ denote a voxel on level k and the super-sampling neighborhood for the reduction factor 2, respectively. Note that the voxels initialized with $\pm M$ are properly handled in the next level, i.e. the ref. is set for ${}^{k+1}\mathbf{P}$ if at least one ${}^{k}\mathbf{Q}\in\mathcal{N}_2({}^{k+1}\mathbf{P})$ has a valid one.

We store all 3D voxel grids as stacks of 2D textures, yielding fast grid updates on the GPU.

3.3. Pull-up

For now, we assume that the correct DT ${}^{k}\mathrm{dt}$ for the coarse level is given. This is clearly the case for the coarsest level, because for this level, we calculate ${}^{k}\mathrm{dt}$ explicitly (Sec. 3.1). The pull-up pass (from coarse to fine resolution) works in much a similar way as push-down (Sec. 3.2). Eight surrounding samples in the coarse grid around a voxel $\mathbf{P}$ are checked and the minimal distance determines the reference point for $\mathbf{P}$. Since we have already written distance information near to the object boundary, this step is only performed for points which contain $\pm M$ as distance. Note that this can be done because each level is stored separately.

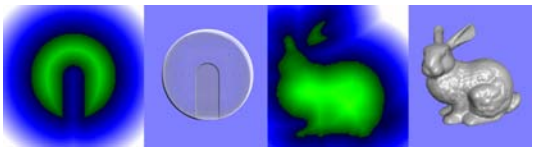

Figure 3: *One DT slice for a notched sphere (left) and the Stanford bunny (right) — FHA+2,* 128^3 *grid, push-down until* 8^3 *voxels; neg. distance are green, pos. blue; 28 FPS*

As mentioned before, the result after a pull-up pass is an approximation of ${}^{k-1}\mathrm{dt}$. To correct the error, we follow two strategies: First, FHA+j performs j additional propagation steps on each level. Second, the voxels surrounding the reference point $\mathbf{S}' = {}^{k-1}\mathrm{dt}_\delta({}^{k-1}\mathbf{P})$ resulting from the pull-up step in the same level are used for a refinement:

$$ {}^{k-1}\mathrm{dt}'_d({}^{k-1}\mathbf{P}) = $$
$$ s_{{}^{k-1}\mathbf{P}} \cdot \min_{{}^{k-1}\mathbf{Q}'\in\mathcal{N}_2(\mathbf{S}')} \{ \left\| {}^{k-1}\mathrm{dt}_\delta({}^{k-1}\mathbf{Q}') - {}^{k-1}\mathbf{P} \right\| \} $$

with sign taken from ${}^{k-1}\mathrm{dt}_d({}^{k-1}\mathbf{P})$, and ${}^{k-1}\mathrm{dt}'_\delta({}^{k-1}\mathbf{P})$ is updated using the selected ${}^{k-1}\mathbf{Q}'$.

Ideally, one would like a pull-up that generates ${}^{k-1}\mathrm{dt}$ from ${}^{k}\mathrm{dt}$ without any error, using a minimal number of propagation steps j on each level. Unfortunately, the optimal j is hard to determine, even though a constant error bound for ${}^{k-1}\mathrm{dt}$ can be given for the general situation.

4. Results and Conclusion

FHA has been tested using implicit and predefined geometries stored as voxel data (hardware: GeForce 8800 GTS). Fig. 3 shows the computed distance field for two examples.

4.1. Performance & Error and Comparison with JFA

To compare both algorithms, we use the following quality measures: rel. # wrong voxels, avg. distance error w.r.t the # wrong voxels and max. distance error (in voxel), i.e.

$$ e_{\mathrm{voxel}} = \frac{\#\text{ of wrong voxels}}{\#\text{ voxels}}, \quad e_{\mathrm{avg}} = \frac{\sum |\mathrm{dt}_d(\mathbf{P}) - \mathrm{dt}^*_d(\mathbf{P})|}{\#\text{ wrong voxels}} $$

$$ e_{\max} = \max\{ |\mathrm{dt}_d(\mathbf{P}) - \mathrm{dt}^*_d(\mathbf{P})| \} $$

where dt^* is the correct DT. The tests have been performed on a 32^3, 64^3 and 128^3 grid using the notched sphere. Comparing the performance, FHA is significantly faster than JFA (Fig. 4). A run-time analysis of the fragment programs involved in our algorithm shows that the push-down and the pull-up are texture-fetch-bounded, whereas the distance propagation is computation-bound.

The evaluation of accuracy (Fig. 5) when taking a large

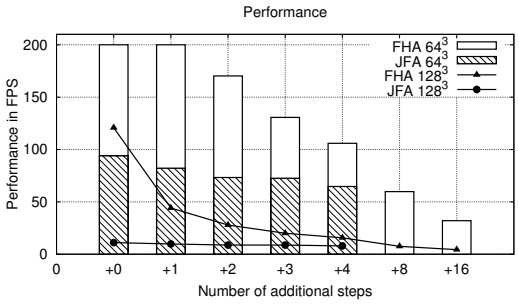

Figure 4: *Frame rates for JFA and FHA dep. on the number of additional propagation steps (object: notched sphere).*

Notched sphere, 64^3, 32457 seeds							
FHA	+0	+1	+2	+3	+4	+8	+16
e_{avg}	0.083	0.047	0.04	0.035	0.028	0.021	0.022
e_{max}	1.41	0.375	0.266	0.211	0.156	0.125	0.109
e_{voxel} %	35.312	7.607	5.019	3.686	2.618	1.314	0.529
JFA	+0	+1	+2	+3	+4	–	–
e_{avg}	0.032	0.022	0.022	0.022	0.023	–	–
e_{max}	0.181	0.059	0.059	0.059	0.059	–.	–
e_{voxel} %	0.309	0.102	0.097	0.097	0.119	–	–
Notched sphere, 32^3, 32457 seeds							
FHA	+0	+1	+2	+3	+4	+8	+16
e_{avg}	0.094	0.048	0.028	0.027	0.026	0.033	0.047
e_{max}	0.807	0.381	0.168	0.137	0.137	0.115	0.115
e_{voxel} %	29.569	4.852	2.637	1.910	1.511	0.369	0.201
JFA	+0	+1	+2	+3	–	–	–
e_{avg}	0.039	0.033	0.037	0.037	–	–	–
e_{max}	0.125	0.041	0.045	0.045	–	–	–
e_{voxel} %	0.128	0.012	0.024	0.024	–	–	–

Subdivided tetrahedron, 64^3				
	# seeds			
FHA+2	4	10	34	130
e_{avg}	–	–	–	–
e_{max}	0	0	0	0
e_{voxel} %	0	0	0	0
	# seeds			
JFA(+0)	4	10	34	130
e_{avg}	–	–	0.262	0.107
e_{max}	0	0	0.375	0.191
e_{voxel} %	0	0	0	0.01

Figure 5: *Error evaluation — e_{voxel} is given in %; note that for FHA+j, j must be in $\{0,\ldots,log(64)-1\}$ for res. 64^3.*

number of seeds (i.e. initialized voxels) yields that the number of wrong pixels as well as the maximum error is significantly larger for FHA compared to JFA. The quality of the error, however, is ambiguous. Taking the average error into account, we find that e_{avg} is, except for FHA+0, in the range of $2-4\%$ of a voxel. When taking a sparse geometry as input featuring a few hundreds of seeds, then the faster FHA+2 provides better results than JFA(+0). For this test, we used a simple tetrahedron mesh which is subdivided consequently.

Finally, replacing the j additional propagations for FHA using a $3\times3\times3$ local neighborhood with the last j JFA steps yields a slight improvement in accuracy, i.e. e_{voxel} is reduced up to 1% for the 64^3 grid.

4.2. Conclusion

FHA is faster (123 FPS for FHA vs. 11 FPS for JFA on a 128^3 grid) but less accurate than JFA for large number of seeds. For a small number of seeds, the accuracy of FHA+2 is comparable to JFA. Depending on the application, the resulting errors can be acceptable and the performance advantage of FHA is possibly more important in order to achieve an overall system with interactive frame rates. An example is given by the level set method. In particular, the hybrid particle level set technique can benefit from the reference part of the DT for particle reseeding without requiring full accuracy.

References

[Cui99] CUISENAIRE O.: Distance transformations: Fast algorithms and applications to medical image processing. *Ph.D. Thesis, UCL, Louvain-la-Neuve, Belgium* (October 1999).

[HKL*99] HOFF K., KEYSER J., LIN M., MANOCHA D., CULVER T.: Fast computation of generalized Voronoi diagrams using graphics hardware. *ACM Proceedings SIGGRAPH* (1999), 277–286.

[HSS*05] HADWIGER M., SIGG C., SCHARSACH H., BÜHLER K., GROSS M.: Real-time ray-casting and advanced shading of discrete isosurfaces. In *Proc. EUROGRAPHICS* (2005), pp. 303–312.

[KLRS04] KOLB A., LATTA L., REZK-SALAMA C.: Hardware-based simulation and collision detection for large particle systems. In *Proc. Graphics Hardware* (2004), pp. 123–131.

[RT06] RONG G., TAN T.-S.: Jump flooding in gpu with applications to vornoi diagram and distance transform. In *ACM Symposium on Interactive 3D Graphics and Games, 14–17 March, Redwood City* (2006), pp. 109–116.

[SGGM06] SUD A., GOVINDARAJU N., GAYLE R., MANOCHA D.: Interactive 3d distance field computation using linear factorization. In *Proc. Symp. on Interactive 3D graphics & games* (2006), pp. 117–124.

[SPG03] SIGG C., PEIKERT R., GROSS M.: Signed distance transform using graphics hardware. In *Proc. IEEE Conf. on Visualization* (2003).

[ST04] STRZODKA R., TELEA A.: Generalized distance transforms and skeletons in graphics hardware. In *VisSym* (2004), pp. 221–230.

[Tsi95] TSITSIKLIS J. N.: Efficient algorithms for globally optimal trajectories. In *IEEE Trans. on Automatic Control* (1995), pp. 1528–1538.