Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 245–254

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Using Temporary Explicit Meshes for Direct Flux Calculation on Implicit Surfaces

Paul Manstetten[1], Josef Weinbub[1], Andreas Hössinger[2], and Siegfried Selberherr[3]

[1] Christian Doppler Laboratory for High Performance TCAD, Institute for Microelectronics, TU Wien, Austria
[2] Silvaco Europe Ltd., United Kingdom
[3] Institute for Microelectronics, TU Wien, Austria
{manstetten,weinbub,selberherr}@iue.tuwien.ac.at
andreas.hoessinger@silvaco.com

Abstract

We focus on a surface evolution problem where the surface is represented as a narrow-band level-set and the local surface speed is defined by a relation to the direct visibility of a source plane above the surface. A level-set representation of the surface can handle complex evolutions robustly and is therefore a frequently encountered choice. Ray tracing is used to compute the visibility of the source plane for each surface point. Commonly, rays are traced directly through the level-set and the already available (hierarchical) volume data structure is used to efficiently perform intersection tests.

We present an approach that performs ray tracing on a temporarily generated explicit surface mesh utilizing modern hardware-tailored single precision ray tracing frameworks. We show that the overhead of mesh extraction and acceleration structure generation is compensated by the intersection performance for practical resolutions leading to an at least three times faster visibility calculation. We reveal the applicability of single precision ray tracing by attesting a sufficient angular resolution in conjunction with an integration method based on an up to twelve times subdivided icosahedron.

Peer-review under responsibility of the scientific committee of the International Conference on Computational Science

Keywords: Visibility, Explicit Mesh, Level-Set, Dynamic Surface

1 Introduction

Dynamic surfaces play a key role in a large number of areas including fluid simulations[1], computer graphics[2], and semiconductor fabrication[3]. Across all these application fields, the level-set method is widely used to represent the surface. The advantage of the method's inherent implicit representation is the robust handling of topological changes, for example, a splitting of an object into two parts or a merging of two objects[4]. In general, one time step in a dynamic

1877-0509
Peer-review under responsibility of the scientific committee of the International Conference on Computational Science
10.1016/j.procs.2017.05.067

surface simulation can be split into (a) the computation of the local surface velocities, (b) the advection of the level-set, and (c) the normalization of the level-set. When the model for the surface velocities depends on flux rates originating from a remote source, a computationally expensive three-dimensional visibility evaluation has to be performed for each surface point.

Our approach increases the performance of this computational bottleneck by integrating the flux on a temporarily generated explicit surface mesh. Utilizing modern hardware-tailored single precision ray tracing libraries for the visibility calculations on the explicit mesh promises a significant performance improvement compared to the established approaches which perform the ray tracing directly on the level-set data structure.

As a prerequisite for our approach, we investigate the sufficiency of single precision ray tracing in conjunction with the proposed numerical integration method using *pbrt-v3*[5][6] as a reference ray tracer with single and double precision routines. We use the *OpenVDB*[7][8] library to represent the surface as a narrow-band level-set. *OpenVDB* provides mesh-to-volume conversion, narrow-band tracking, advection schemes, ray tracing, and surface extraction. The intersection tests during the visibility evaluation are performed directly on the *OpenVDB* data structure with *OpenVDB*'s `LevelSetRayTracer`[9] and on the extracted mesh with the *Embree*[10][11] library.

All in all, this paper contributes the following findings relevant for accelerating direct flux calculations on implicit surfaces:

(1) A suitable flux integration method based on hemisphere sampling (Section 2)

(2) An analysis concerning the suitability of single precision arithmetics (Section 3)

(3) Performance results including an analysis of the introduced overhead (Section 4)

2 Flux Integration Method

To compute the flux of a planar source towards each surface point, the solid angle which renders the source visible has to be evaluated. We use a subdivided icosahedron to obtain a triangulation of the hemisphere defined by the local surface normal. Rays are traced towards each of the centroids of the triangulation. If the rays do not intersect with the geometry, the source is considered visible under the solid angle defined by the triangle. Once the visibility information is obtained, we conduct a numerical integration by evaluating the source contribution in the centroid's direction and applying a *centroid rule* analog to the *midpoint rule* in one-dimensional numerical integration[12].

2.1 Subdivided Icosahedron

The initial triangulation of the sphere is provided by an icosahedron (12 vertices, 20 faces). The initial triangles are congruent and equilateral. To increase the resolution, we subdivide the triangles according to [12]: For each triangle, (a) compute the midpoints of the edges, (b) project those midpoints onto the unit sphere, and (c) remove the original triangle and connect the 3 original vertices and the 3 new midpoints to form 4 new triangles. The resulting triangles will be almost congruent and almost of the same size.

Fig. 1 visualizes an icosahedron and its subdivisions up to n=4 together with the integration directions originating at the center. Table 1 provides detailed properties of the original icosahedron and the subdivision steps up to n=9. The number of triangles for subdivision step n is defined as

$$N_{tri}(n) = 20 \cdot 4^n. \tag{1}$$

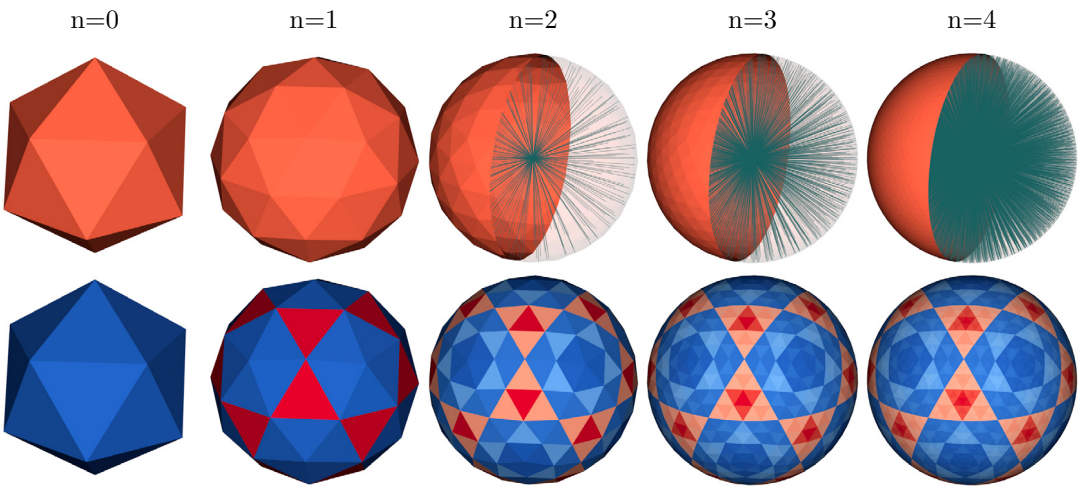

Figure 1: Initial icosahedron (n=0) and subdivisions up to n=4. Starting from n=2, the top row shows the integration directions originating at the center and pointing towards each centroid of the triangles of the spherical mesh. The bottom row visualizes the differences in size (blue = smaller, red = larger) of the triangles resulting from the subdivisions. Detailed numerical information about each subdivision step is listed in Table 1.

The corresponding number of vertices is defined by

$$N_{vert}(n) = \frac{20 \cdot 3}{5} + \frac{20 \cdot 3}{2} \cdot \sum_{i=0}^{n} 4^i \tag{2}$$

where it is already transparent that the resulting triangles are not congruent as the first summand originates from the initial vertices shared by 5 triangles. The subsequent summands originate from the subdivisions which generate vertices shared by 6 triangles. The angular resolution α_{res} in Table 1 is the angle corresponding to a spherical cap of the same area as a triangle. For subdivision n

$$\alpha_{res}(n) = \arccos(1 - \frac{4 \cdot \pi}{N_{tri}(n)} \cdot \frac{1}{2 \cdot \pi}) \tag{3}$$

where an equal distribution of the solid angle over all triangles is assumed. Further, Table 1 illustrates the effect of the subdivision on the spread of triangle areas $\Delta_{min}/\Delta_{max}$, revealing an asymptotic limit of about 75% for higher subdivisions.

2.2 Numerical Integration

The integration method permits infinite planar sources with an arbitrary angular distribution function $f_{src}(\mathbf{\Theta})$ with direction $\mathbf{\Theta}(\theta, \varphi)$. In the following, we utilize sources with a power cosine distribution and a downward mean direction. The angular distribution functions are defined as $f_{src}(\mathbf{\Theta}) = cos(\theta)^n$. For $n = 1$, the angular distribution function is reduced to $f_{src}(\mathbf{\Theta}) = cos(\theta)$ and therefore constitutes an ideal-diffuse source.

subdivisions	N_{tri}	N_{vert}	α_{res}	$\Delta_{min}/\Delta_{max}$	r_{min}
n=0	20	12	25.842	1.00000	0.9000000
n=1	80	42	12.839	0.84222	0.9750000
n=2	320	162	6.409	0.78820	0.9937500
n=3	1280	642	3.203	0.77377	0.9984375
n=4	5120	2562	1.602	0.77010	0.9996094
n=5	20480	10242	0.801	0.76918	0.9999023
n=6	81920	40962	0.400	0.76895	0.9999756
n=7	327680	163842	0.200	0.76889	0.9999939
n=8	1310720	655362	0.100	0.76888	0.9999985
n=9	5242880	2621442	0.050	0.76888	0.9999996

Table 1: Properties of an icosahedron and its subdivision up to n=9: N_{tri} = number of triangles, N_{vert} = number of vertices, α_{res} = angular resolution, $\Delta_{min}/\Delta_{max}$ = ratio of triangle areas, r_{min} = smallest radius occurring in triangulation.

We approximate the integral of the direct flux

$$F_i = \int_\Omega f_{src}(\mathbf{\Theta})(\mathbf{\Theta}\mathbf{n_i}) d\omega_{\mathbf{\Theta}}, \quad \text{with} \quad d\omega_{\mathbf{\Theta}} = sin\theta d\theta d\varphi \tag{4}$$

received at surface location i (surface normal = $\mathbf{n_i}$) by triangulating the hemisphere based on a subdivided icosahedron

$$F_i = \sum_{j=1}^{N_{tri}} \int_{\Delta_j} f_{src}(\mathbf{\Theta})(\mathbf{\Theta}\mathbf{n_i}) dA = \sum_{j=1}^{N_{tri}} f_{src}(\mathbf{\Theta_{c_j}})(\mathbf{\Theta_{c_j}}\mathbf{n_i}) \cdot \Delta_j \tag{5}$$

where $\mathbf{\Theta_{c_j}}$ is the direction towards the centroid of triangle j and Δ_j is the area of triangle j. A *centroid rule* is used in Eq. 5 to integrate over the area of a triangle.

Using a power cosine source and a visibility function $f_{vis}(\mathbf{\Theta})$ that evaluates to 0, if a surface is intersected in direction $\mathbf{\Theta}$, and 1 otherwise, the direct flux at surface location i is

$$F_i = \sum_{j=1}^{N_{tri}} \left[f_{vis}(\mathbf{\Theta_{c_j}}) \cdot cos(\mathbf{\Theta_{c_j}}\mathbf{n_{src}})^n (\mathbf{\Theta_{c_j}}\mathbf{n_i}) \cdot \Delta_j \right] \tag{6}$$

where $\mathbf{n_{src}}$ is the upward pointing normal of the source plane, and $\mathbf{n_i}$ is the normal direction of the surface at position i.

2.3 Validation

We use analytic solutions for the direct flux received on horizontal and vertical surfaces to validate our integration method. The direct flux originating from an unobstructed power cosine source on a horizontal surface is

$$F_{hori} = \int_0^{\pi/2} \int_0^{2\pi} \left[cos(\theta)^{n+1} \right] sin\theta d\theta d\varphi = \frac{2}{n+2}\pi \tag{7}$$

and for a vertical surface under the same source

$$F_{vert} = \int_0^{\pi/2} \int_0^{\pi} \left[cos(\theta)^n sin(\theta) sin(\varphi) \right] sin\theta d\theta d\varphi = 2 \int_0^{\pi/2} \left[cos(\theta)^n sin(\theta) \right] sin\theta d\theta \ . \tag{8}$$

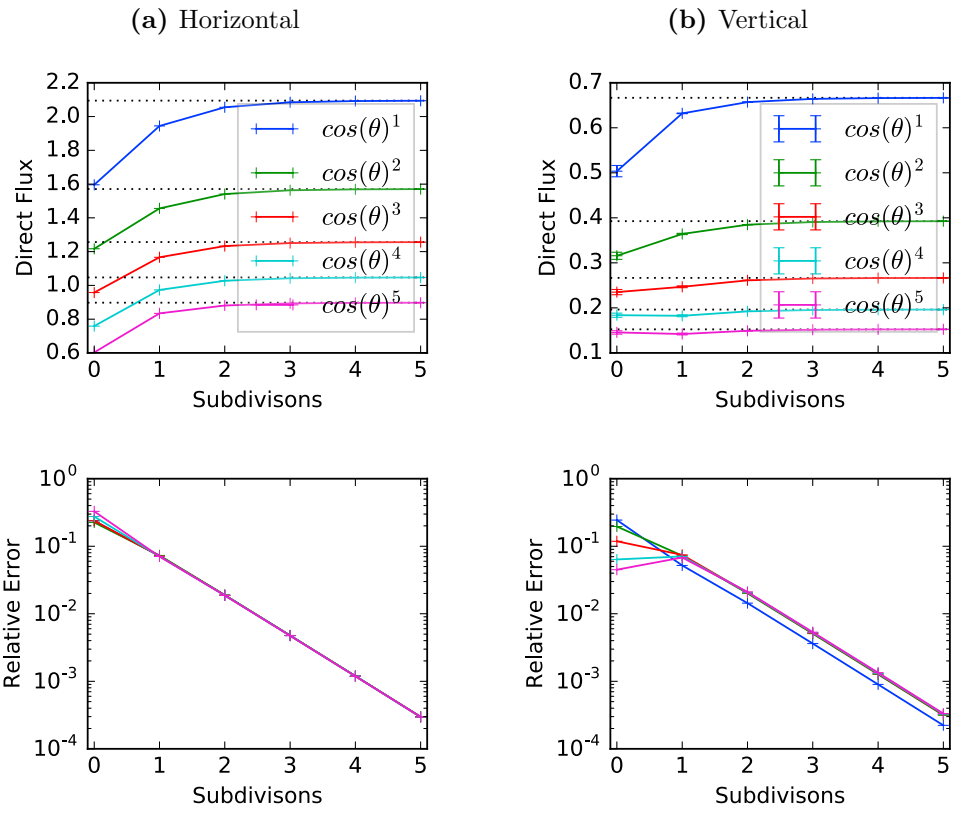

Figure 2: Direct flux received on a horizontal (a) and vertical (b) surface using our integration method. An unobstructed source with a power cosine distribution with exponents $n = [1, 5]$ is used. The resolution of the icosahedron is increased by up to 5 subdivisions, resulting in a maximal angular resolution $\alpha_{res} = 0.8$ degrees (see Table 1). The dotted lines are the analytic solutions obtained from Eq. 7 and 8. The solid lines in (b) show the average of the flux over all 4 vertical faces of a cube; the error bars indicate the maximum deviation amongst the 4 vertical faces.

In Fig. 2, we compare the numerical results with the analytic solutions for sources with power cosine exponents and subdivisions up to 5. The flux rates approach the analytic solutions with an relative error below 1% for 3 subdivisions and far below 0.1% for 5 subdivisions.

3 Single Precision Ray Tracing

Hardware-tailored ray tracing libraries like *Embree*[11](Intel, open source), *Optix Prime*[13] (nvidia, closed source), and *RadeonRays*[14](AMD, open source) provide highly-optimized ray tracing kernels for the corresponding platforms. The mentioned software packages support only single precision arithmetics; a change of the floating point type is not favored due to the internal optimizations which base on platform specific intrinsics.

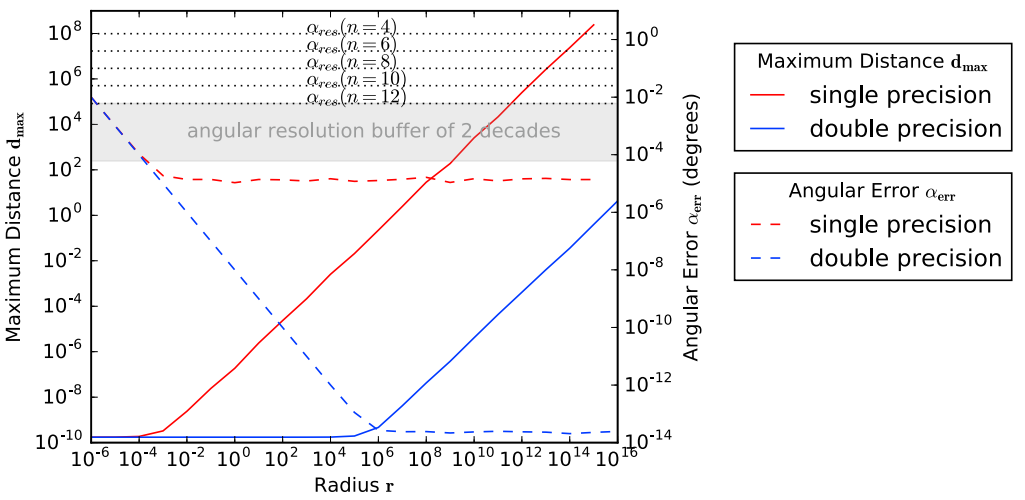

Figure 3: Evaluation of the angular resolution using single precision (red) and double precision (blue) ray tracing arithmetics. The maximum distance d_{max} between the intersection point (found by ray tracing) and the reference point is shown as solid line and the scale on the left. The resulting angular error α_{err} is shown as dashed line and the scale on the right. The angular resolutions of the subdivided icosahedron are shown as dotted lines for up to $n = 12$ subdivisions. Additionally for n=12, a *buffer* region (2 decades) for the angular resolution is shown as a gray box indicating the region, where the difference in the angular resolution drops below 1%.

The integration method described above (cf. Section 2.2) uses ray tracing to detect visibility of the source along a certain direction. The resulting information is binary and the influence of the arithmetic precision can be isolated. To evaluate the applicability of single precision ray tracing in our integration method, we compare the angular resolutions which are achieved for single and double precision ray tracing. A spherical mesh in double precision, generated by subdividing an icosahedron 6 times, serves as a reference mesh. The radius of the reference mesh is scaled from $r = 10^{-6}$ to $r = 10^{16}$ and rays are traced from the origin towards each vertex of the mesh. The distance between the intersection point (found by ray tracing) and the vertex on the reference mesh is computed for all vertices.

In Fig. 3 the maximum distance d_{max} and the corresponding angular error $\alpha_{err} = atan(d_{max}/r)$ is plotted over the radius r of the reference sphere. The maximum distance shows a constant value of 10^{-10} for $r < 10^{-3}$ (single precision) and $r < 10^{6}$ (double precision). The angular resolution of the subdivided icosahedrons are indicated with dotted lines. The *pbrt-v3* library [6] is used for single and double precision ray tracing. The results reveals that for radii of the reference sphere $r > 10^{-3}$, the achieved angular resolution is more than two decades smaller than the angular resolution of a 12 times subdivided icosahedron for both, single precision and double precision ray tracing. Assuming a minimum resolution ratio of 100/1 as sufficient, this justifies the use of single precision ray tracing in the numerical integration method described above as long as the number of subdivisions of the icosahedron does not exceed 12.

4 Performance Comparison

In the following performance comparison, we concentrate on the performance of the actual intersection test, i.e., the calculation of the intersection point with the mesh in the direction of the ray. The rays are traced against a narrow-band level-set representation of a sphere with radius $r_{mesh} = 1$ using *OpenVDB*'s `LevelSetRayTracer` and against a triangulated mesh (extracted from the level-set) using *Embree*. The implicit mesh is represented with *OpenVDB*'s default tree configuration `Tree4<float,5,4,3>` and a narrow-band half-width of 3 times the voxel size d_{vox}.

The origins of the rays are equally distributed on a sphere with radius $r_{org} = r_{mesh} + d$, leading to rays which start inside the mesh for $d < 0$ and rays which start outside the mesh for $d > 0$. At each origin, rays are traced towards the centroids of the triangles of a subdivided icosahedron (cf. Section 2.1). The `for` loop which iterates over the origins is OpenMP-parallelized. Table 2 summarizes the parameter range of the performance analysis. All performance measurements were performed on an Intel Core-i7-4790K system with 32GB of RAM. The CPU has 4 physical cores (8 threads with hyper-threading) with 8MB of shared L3 cache. The benchmarks were performed using *Embree* 2.12, *OpenVDB* 4.0, and compiled using gcc 6.1.1.

Fig 4 compares implicit and explicit ray tracing performance for different ray origins r_{org} and different surface resolutions. The resulting performance gain is between 3 and 6 for all possible combinations of the parameters in Table 2. For small meshes with less than 0.1 million triangles the performance gain is increasing. The limits of the achieved performance with 8 threads on the explicit mesh are about 100 MRays/s ($r_{org} = 1.5$, 15k triangles) and about 10 MRays/s ($r_{org} = 0.85$, 6.0m triangles).

Parameter	Values
Number of threads	1, 2, 4, 5, 6, 8
Subdivisions for search directions n_{trace}	1, 2, 3, 4, 5
Radius of origins r_{org}	1.5, 1.15, 0.85, 0.5
Voxel size d_{vox}	0.05, 0.01, 0.005, 0.0025
Dependent Parameter	
Number of active voxels $f(d_{vox})$	30k, 0.8m, 3.0m, 12.1m
Number of triangles $f(d_{vox})$	15k, 0.4m, 1.5m, 6.0m
Number of search directions $f(n_{trace})$	80, 320, 1280, 5120, 20480

Table 2: Parameter variations used in the performance comparison (k = kilo, m = million).

Fig 5 plots the achieved speedup for the parallelized `for` loop (which iterates over the origins) for explicit and implicit ray tracing. Both parallelizations show nearly an ideal speedup for 2 threads. The speedup spreads for 4 threads with a minimum speedup of 2.7 (*Embree*) and 2.0 (*OpenVDB*). The speedup for 8 threads is up to 6 for *Embree* and up to 5 for *OpenVDB*. The parameter combinations which show nearly no speedup between 2 to 4 threads (cf. Fig. 5b) were identified to have a low hit ratio (number of hits/number of traced rays < 0.13) and a large voxel size ($d_{vox} >= 0.01$). For 5 and more threads (entering the hyper-threading regime) this influence is compensated leading to an overall speedup between 3 and 4 for 6 threads.

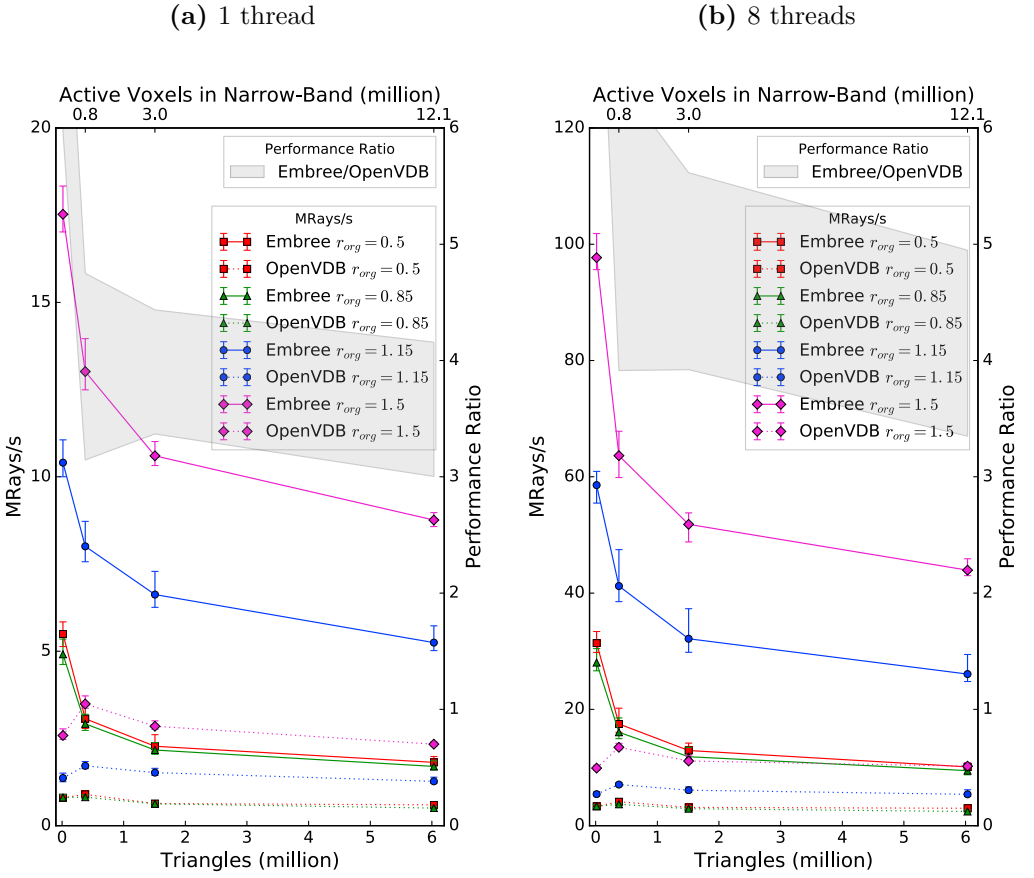

Figure 4: Performance comparison between ray tracing on the implicit surface (using *OpenVDB*) and on the explicit surface (using *Embree*). The ray tracing performance for different ray origins r_{org} is plotted over the surface resolution for 1 thread (a) and 8 threads (b). The top axis plots the number of active voxels in the narrow-band representation of the surface while the bottom axis labels the corresponding number of triangles in the extracted mesh. The error bars show the spread in performance when varying the number of search directions for each origin according to Table 2. The filled gray area is the range of the performance ratio of the explicit approach over the implicit approach.

The acceleration structure build-up of *Embree* is parallelized (internally) with 8 threads. Fig. 6a illustrates the explicit mesh extracted from the narrow-band level-set of a sphere for voxelsize $d_{vox} = 0.1$ and Fig. 6b plots the overhead introduced by the generation of the temporary explicit mesh and the construction of the acceleration structure. The maximum overhead is identified with a total of less than 1.4 seconds for a mesh with about 6 million triangles. The overhead per million triangles is about 0.2 seconds for meshes with more than 0.4 million triangles. An example configuration illustrates that the overhead is easily compensated by the ray tracing time: Assuming a ray tracing performance of 12 MRays/s on the implicit mesh (best case in our benchmarks) and 1280 traced rays per triangle ($n = 3$), the explicit ray tracing decreases the total simulation time at least by 35 seconds per million triangles.

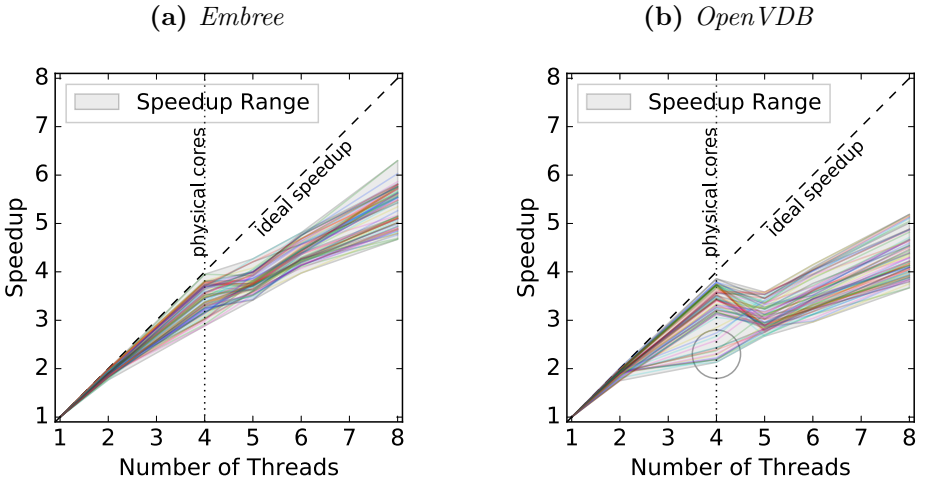

Figure 5: Speedup for the OpenMP-parallelized ray tracing with *Embree* (a) and *OpenVDB* (b). The speedup range represents limits resulting of the full parameter range (cf. Table 2). The dashed line indicates an ideal speedup and the dotted line marks the number of physical cores in the system. The circle in (b) marks a group of combinations with nearly no speedup between 2 and 4 threads.

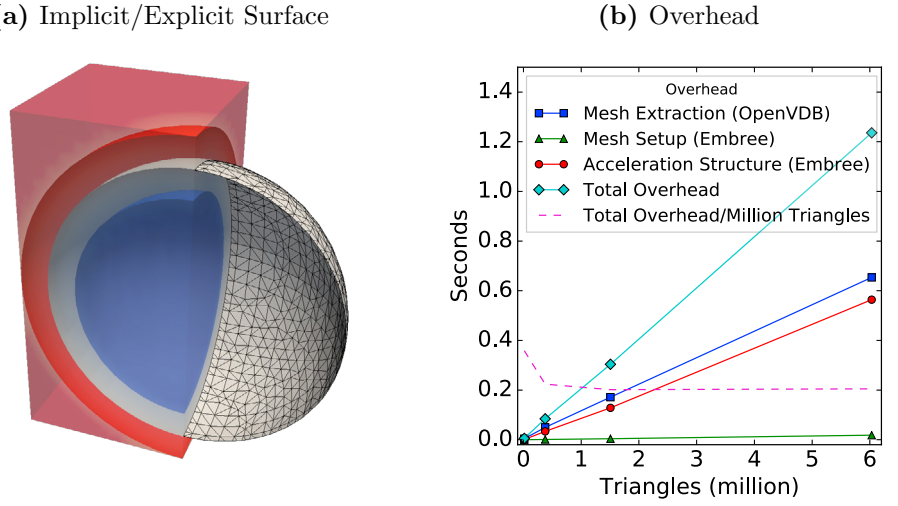

Figure 6: Implicit/explicit mesh of a sphere for voxel size $d_{vox} = 0.1$ (a); the red and blue iso-surfaces visualize the extends of the narrow-band. Overhead introduced by the mesh extraction and the setup of *Embree* for different surface resolutions (b).

5 Summary and Outlook

We investigated an approach to transfer the visibility calculation from a level-set based implicit representation of a surface to a temporary explicit mesh. The numerical integration of the direct flux originating from a planar source is performed by tracing rays against the explicit mesh. The obtained flux is then used to advect the implicit representation of the surface.

We show that single precision ray tracing has sufficient accuracy, when using the proposed integration method based on up to twelve times subdivided icosahedrons. The performance gain of explicit ray tracing (using *Embree*) over implicit ray tracing (using *OpenVDB*) is at minimum a factor of three for a wide range of scenarios. The overhead introduced by the mesh extraction and preparation is easily compensated for practical surface resolutions.

The proposed numerical integration method is ideally suited for additional optimizations: (a) adaptive sampling of the hemisphere using different levels of subdivisions, (b) reuse of adaptive sampling masks of neighboring surface points, and (c) parallelization based on coherent rays from a local group of surface points.

Acknowledgment

The financial support by the *Austrian Federal Ministry of Science, Research and Economy* and the *National Foundation for Research, Technology and Development* is gratefully acknowledged.

References

[1] C. Lee, J. Dolbow, and P. J. Mucha, "A Narrow-Band Gradient-Augmented Level Set Method for Multiphase Incompressible Flow," *Journal of Computational Physics*, vol. 273, pp. 12–37, 2014.

[2] R. K. Hoetzlein, "GVDB: Raytracing Sparse Voxel Database Structures on the GPU," *Proceedings of High Performance Graphics*, pp. 109–117, 2016.

[3] "Victory Process - 3d Process Simulator." [Online]. Available: http://www.silvaco.com/products/tcad/process_simulation/victory_process/victory_process.html

[4] J. A. Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University P ress, 1999, vol. 3, ISBN: 0521645573.

[5] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering, Third Edition: From Theory to Implementation*. Morgan Kaufmann, 2016, ISBN: 0128006455.

[6] "pbrt-v3." [Online]. Available: https://github.com/mmp/pbrt-v3

[7] "OpenVDB." [Online]. Available: http://www.openvdb.org/

[8] K. Museth, "VDB: High-Resolution Sparse Volumes with Dynamic Topology," *ACM Transactions on Graphics*, vol. 32, no. 3, p. 27, 2013.

[9] Museth, Ken, "Hierarchical Digital Differential Analyzer for Efficient Ray-Marching in OpenVDB," *Proceedings of SIGGRAPH*, p. 40, 2014.

[10] A. Áfra, I. Wald, C. Benthin, and S. Woop, "Embree Ray Tracing Kernels: Overview and New Features," *Proceedings of SIGGRAPH*, p. 52, 2016.

[11] "Embree." [Online]. Available: https://embree.github.io/

[12] K. Atkinson, "Numerical Integration on the Sphere," *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics*, vol. 23, no. 03, pp. 332–347, 1982.

[13] "OptiX Prime," Aug. 2016. [Online]. Available: https://developer.nvidia.com/optix

[14] "RadeonRays." [Online]. Available: https://github.com/GPUOpen-LibrariesAndSDKs/RadeonRays_SDK