Interactive Ray Tracing of Distance Fields

Ondřej Jamriška[*]
Supervised by: Vlastimil Havran[†]

Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
Czech Technical University in Prague

Abstract

Distance field is a flexible surface representation used in many applications. We study the algorithms for direct rendering of distance field surfaces representation via ray tracing in interactive frame rates. To accelerate the ray tracing we represent the distance fields by sparse block grid data structure. We compare the visual quality and the computation time for different methods of ray-surface intersection test and for surface normal estimation. We present a technique to accelerate the computation for primary rays from camera by projection. We also describe another technique for faster computation of shadow rays via volumetric occlusion. We show the results on four scenes of different complexities.

Keywords: Ray Tracing, Distance Field, Level Set

1 Introduction

Distance field is a versatile surface representation. Many applications need to represent dynamic surface that changes its shape over time in complex and unpredictable ways. For example, in solid modeling application, user may want to sculpt the shape of a surface by combination of adding material to existing model and carving holes into it. Another example is simulation of splashing water, where the interface between water and air needs to be tracked while it splits apart and merges together. In these circumstances, distance field representation is often used, due to its ability to handle operations that deform the surface and change its topology.

In addition to geometric modeling [2, 14] and simulation of liquids [8], applications utilizing distance fields include metamorphosis [3], geometric texturing [5], modeling of snow [10], and volume segmentation [16].

Image of a surface in distance field representation can be rendered using methods based on rasterization or ray tracing. Hardware acceleration makes rasterization an attractive option, however, due to the implicit nature of distance field representation, the surface must be first converted into set of rasterizable geometric primitives [12, 6].

[*]jamriond@fel.cvut.cz
[†]havran@fel.cvut.cz

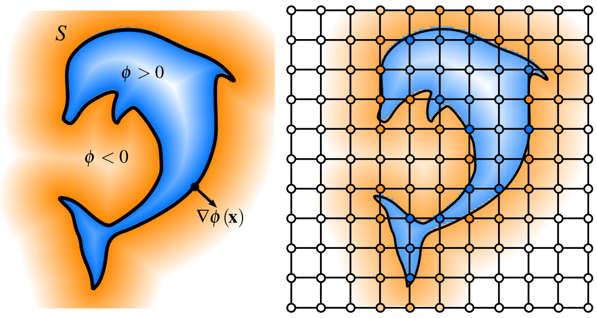

Figure 1: Distance field as a surface representation. Left: continuous distance field and a surface represented by its zero level set. Right: distance field sampled on grid.

Also, even though many sophisticated rendering techniques exist [9], the level of realism achievable with rasterization is limited.

Ray tracing is an image synthesis algorithm that is able to reproduce various optical phenomena. The algorithm traces individual paths of light as it propagates through scene. It relies on ability to compute the points where rays hit the surface. Since rays can be directly intersected with distance field, the costly conversion of surface into explicit representation is avoided.

In this paper, we will describe our raytracer that is capable of rendering surfaces in distance field representation at interactive frame rates. We designed it as a high quality rendering front-end for the kind of applications mentioned above. We thus assume that the surface is fully dynamic and its distance field can change each frame.

2 Distance Field

Distance field maps point in space to its shortest distance from nearest point on a surface. To measure the distance, Euclidean metric is often used. Distance field represents the surface in an implicit form, as a set of points that have zero distance to the surface. Example surface and its distance field is illustrated in Figure 1.

Surface S is defined as the zero level set of function ϕ,

$$S = \left\{ \mathbf{x} \in \mathbf{R}^3 \mid \phi(\mathbf{x}) = 0 \right\}.$$

Function $\phi : \mathbf{R}^3 \rightarrow \mathbf{R}$ is a signed distance function. At each point, $|\phi(\mathbf{x})|$ is the distance from $\mathbf{x}$ to nearest point $\mathbf{y}$ lying on the surface. By convention, the sign of $\phi(\mathbf{x})$ is negative when $\mathbf{x}$ is inside the surface, and positive when it is outside:

$$\phi(\mathbf{x}) = sign(\mathbf{x}) \cdot dist(\mathbf{x})$$

$$dist(\mathbf{x}) = \min_{\mathbf{y} \in S} ||\mathbf{x} - \mathbf{y}||$$

$$sign(\mathbf{x}) = \begin{cases} -1 & \text{if } \mathbf{x} \text{ is inside } S \\ +1 & \text{if } \mathbf{x} \text{ is outside } S, \end{cases}$$

where $||\cdot||$ is the Euclidean norm. Surface normal $\mathbf{n}$ corresponds to the gradient of ϕ,

$$\mathbf{n}(\mathbf{x}) = \nabla\phi(\mathbf{x}), \text{ and } ||\nabla\phi|| = 1.$$

For simple shapes, ϕ can be expressed in analytic form. However, for practical purposes ϕ is usually represented using set of samples located at discrete points in subregion of $\mathbf{R}^3$. Simple arrangement illustrated in Figure 1 puts samples on vertices of Cartesian grid. When value of ϕ is needed at point $\mathbf{x}$, it is reconstructed from nearby samples using interpolation. Trilinear interpolation calculates $\phi(\mathbf{x})$ as a linear combination of eight samples located at vertices of grid cell that contains $\mathbf{x}$.

3 Data Structure

Most applications need actual values of ϕ only at points that are close to surface, and just the sign of ϕ suffices away from surface. This allows for sparse sampling of ϕ, placing samples only inside narrow band around surface. Several data structures for sparse representation of distance field were proposed [4, 11, 13].

In our raytracer, we decided to use the sparse block grid data structure [4] for input distance field representation. Sparse block grid is easy to implement and allows random access in $O(1)$ time. Its memory consumption grows with $O\left(n^{2.25}\right)$, where n is the number of grid cells along one axis [4]. Due to these properties, we assume it is likely that practical applications may utilize sparse block grid as their internal distance field representation.

Instead of storing samples on whole grid, sparse block grid divides the grid into coarse blocks and stores only samples on subgrids corresponding to blocks that are near the surface. The data structure is illustrated in Figure 2. Each block of coarse grid contains a flag and a pointer. The flag indicates whether the block is inside, outside or near the surface. If block is near the surface, its pointer points to a fine subgrid that stores the samples of ϕ.

4 Ray-Surface Intersection

Given ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with origin $\mathbf{o}$ and direction $\mathbf{d}$, we are looking for its first intersection with the surface on given interval $[t_{min}, t_{max}]$. Since surface is defined by the zero level set of signed distance function ϕ, intersections occur at roots of $\phi(\mathbf{r}(t))$. Thus, the task of finding the intersection is a root-finding problem. Specifically, for $t \in [t_{min}, t_{max}]$, we want to detect if at least one root of $\phi(\mathbf{r}(t))$ exists, and find the smallest t corresponding to the first root.

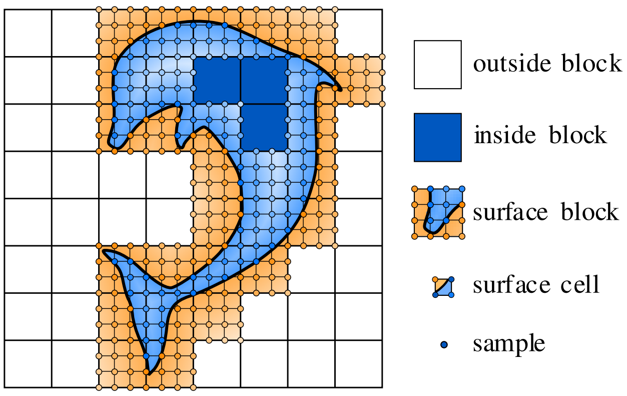

Figure 2: Compact representation of distance field using sparse block grid data structure.

4.1 Sparse Block Grid Traversal

The aim of traversal algorithm is to successively visit grid cells along a ray, identifying cells that contain the surface. For these cells, a test for an intersection needs to be performed. Traversal terminates as soon as the intersection is found or once the grid boundary is reached. The process is illustrated in Figure 3.

The traversal of sparse block grid consists of two nested loops. The outer loop iterates over blocks of coarse grid. When a visited block is entirely inside or outside the surface, the loop skips directly to the next block. Otherwise, the inner loop is invoked. Inner loop iterates over cells of a surface block and checks each cell if it contains the surface.

The check is based on comparison of signs of samples at cell vertices. Since the signed distance ϕ is negative inside the surface and positive outside, a cell contains the surface if the signs of samples differ. When such cell is visited, a test for intersection is performed. If ray intersects the surface inside the cell, the point of intersection is calculated.

In order to avoid fetching eight samples and checking their signs at each visited cell, we precompute a compact bit mask for every surface block. Each bit of the mask corresponds to one cell of the surface block. If cell contains the surface, its bit is set. Bit masks are recomputed every frame using single sequential pass over the cells of surface blocks. When the cell is visited during surface block traversal, we only test the corresponding bit in the block's bit mask to check if the cell contains the surface.

Grid traversal is initiated by intersecting ray with the grid's bounding box. We compute the points where ray

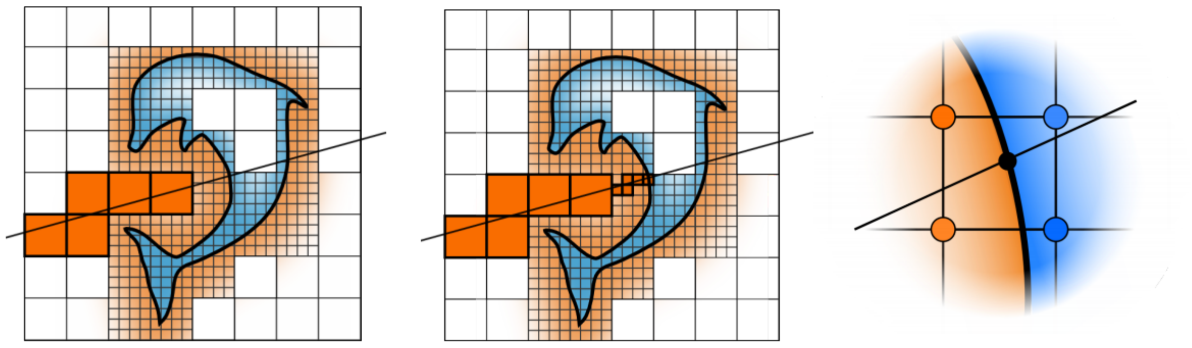

Figure 3: The process of finding the ray-surface intersection. The ray is first intersected with the grid's bounding box, and traverses blocks of the coarse grid (left). When the ray enters a surface block, it starts traversing the cells of the surface block's subgrid (middle). When a surface containing cell is encountered, the ray's intersection with the surface inside the cell is computed (right).

enters and exits the box using a ray–box intersection algorithm described in [17]. If ray has no intersection with the bounding box then it misses the grid and cannot intersect the surface. Otherwise, the grid traversal starts from the block that contains the point where ray enters the grid's bounding box. In case that rays origin is inside the grid, traversal starts from block containing the origin.

The algorithm for sparse block grid traversal is based on 3D–DDA algorithm [1]. At the beginning of sparse block grid traversal, we precompute several constants that can be reused during transitions between coarse and fine grid traversal in order to reduce their overhead.

4.2 Surface Cell Intersection

When ray enters a cell that contains the surface, its intersection with the surface inside the cell needs to be examined. The simplest method is to assume that the ray always hits the surface, and set the intersection point to the middle of points where ray enters and exits the cell. This method is fast, however, it produces artifacts that are visible especially on the contour of an object.

More accurate methods are based on finding the roots of $\phi(\mathbf{r}(t))$. Since ϕ is discretely sampled on grid, a continuous function needs to be reconstructed using interpolation. To reconstruct the value of ϕ at any point inside the cell, we use trilinear interpolation from eight samples at cell vertices.

Simple root-finding method first interpolates two values $\phi_{in} = \phi(\mathbf{r}(t_{in}))$ and $\phi_{out} = \phi(\mathbf{r}(t_{out}))$ at points where ray enters and exits the cell. If signs of ϕ_{in} and ϕ_{out} differ, the ray has hit the surface. The intersection point is calculated by approximating the position of a root as the point where line connecting values ϕ_{in} and ϕ_{out} crosses zero,

$$t_{hit} = t_{in} + (t_{out} - t_{in}) \frac{\phi_{in}}{\phi_{in} - \phi_{out}}.$$

Position of root can be further refined, as illustrated in Figure 4. First, the value of ϕ is interpolated at t_{hit}. Next, the interval $[t_1, t_2]$ is selected from two subintervals $[t_{in}, t_{hit}]$ and $[t_{hit}, t_{out}]$ such that $\phi(\mathbf{r}(t_1))$ and $\phi(\mathbf{r}(t_2))$ have different signs. A new root is then approximated as the zero crossing point of a line that connects values ϕ_{t_1} and ϕ_{t_2}. When repeated several times, this process is equivalent to a root finding technique called the false position method [15].

The convergence of false position method is only linear. This is not an issue, since we perform only small number of iterations anyway. We perform fixed number of 1-4 iterations because it is faster than using adaptive termination criterion, and visual results usually do not improve after more than 4 iterations.

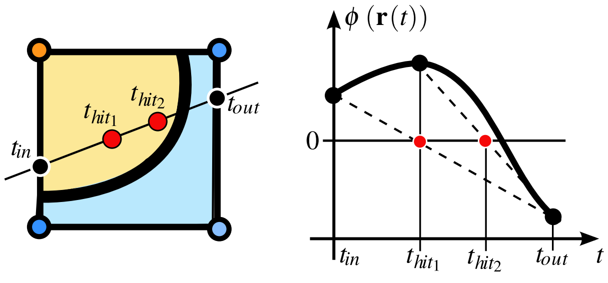

Figure 4: Locating the ray-surface intersection inside a cell. Left: ray passing through cell that contains the surface. Right: approximation of root position using first two steps of false position method.

5 Surface Normal Estimation

At the intersection point, a surface normal $\mathbf{n}$ needs to be evaluated for purposes of shading and spawning of secondary rays. In distance field representation, the surface normal corresponds to the gradient of the distance field. Since distance field is sampled on grid, the gradient has to be estimated using a combination of differentiation and interpolation.

We consider three gradient estimation schemes resulting from different ordering in which differentiation and interpolation is applied. The schemes vary in arithmetic complexity and in the number of samples they use. They also differ in the continuity of estimated normals, which affects the perceived smoothness of surface shape.

The first method is to find analytic derivatives of the interpolation filter. The filter is differentiated and analytical expressions of its partial derivatives are found. The components of $\nabla\phi$ are then obtained by convolving the samples of ϕ with filter's partial derivatives.

The second method is to first estimate partial derivatives of ϕ at sample locations using finite differencing. This yields values of $\nabla\phi$ at cell vertices. The $\nabla\phi(\mathbf{x})$ is then evaluated at $\mathbf{x}$ by interpolating gradients from cell vertices.

In the third method, the value of ϕ is first interpolated at number of points obtained by offsetting point $\mathbf{x}$ along coordinate axes. Next, $\nabla\phi(\mathbf{x})$ is computed by taking finite differences of interpolated values.

	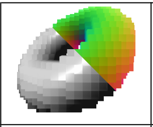	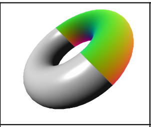	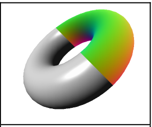
method	midpoint	1 iteration of false position	4 iterations of false position
#interpolations	2	2	5
time [ms]	8.0	8.4	11.2

Table 1: Surface cell intersection methods. Upper part of the torus shows normal in pseudo-color, lower part is shaded using Phong model.

	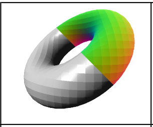	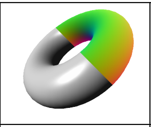	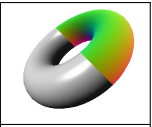
method	analytic derivative	difference of interpolations	interpolation of differences
#samples	8	16	24
#interpolations	0	3	1
time [ms]	7.1	9.5	8.4

Table 2: Surface normal estimation methods. Upper part of the torus shows normal in pseudo-color, lower part is shaded using Phong model.

6 Acceleration of Primary Rays

Casting of primary rays can be accelerated by skipping the traversal of empty blocks that are outside the surface. This is achieved by reducing the $[t_{min}, t_{max}]$ interval so that the ray starts and stops close to the points where it enters the first surface block and leaves the last surface block.

To find reduced t_{min} and t_{max} for each primary ray, we rasterize surface blocks into min/max-buffers. Instead of rasterizing six faces of each surface block, we only rasterize a conservative bounding square of the block's projection onto the image plane. The center of a bounding square is computed by projecting the block's center using the camera matrix, and the width is determined based on the blocks space diagonal and the depth of its center. Each square has assigned value t that is equal to the distance from block's center to the camera.

Bounding square is computed for each surface block. Squares that correspond to blocks that are behind the camera are culled. Remaining squares are first sorted by their t values. Next, they are rasterized in back-to-front and front-to-back order into min-buffer and max-buffer respectively.

Since values t_{min} and t_{max} need not to be precise, the resolution of min/max-buffers can be lower than the image resolution. This can result in improved performance when there is large overdraw of bounding squares.

7 Acceleration of Shadow Rays

Shadow ray query only has to detect if the ray intersects the surface, the actual point of intersection is not needed. This fact can be used together with the information already stored in sparse block grid to accelerate casting of shadow rays. Whenever ray passes through block whose flag indicates it is inside the surface, the ray must have also intersected the surface. We use a modified grid traversal to detect this situation and reduce the number surface cell intersection tests. This approach is inspired by the volumetric occluders technique described in [7].

The modified grid traversal first iterates only over blocks of the coarse grid. Traversal of a fine subgrid is not invoked when a surface block is visited. Instead, when a surface block is visited for the first time, the state of traversal variables is stored for a chance of later traversal restart. The traversal of coarse blocks then speculatively continues in the hope of encountering block that is inside the surface. The number of traversal steps elapsed from the first visited surface block is counted.

When the traversal visits a block that is inside the surface, the ray must have intersected the surface and traversal terminates. When the grid boundary was reached and no surface block was visited during traversal, the ray has no intersection with the surface and traversal also terminates. However, if a surface block was visited during the previous traversal and either the number of elapsed steps is greater than threshold n or the grid boundary is reached, the normal grid traversal is restarted from the first visited surface block using the stored state of traversal variables.

8 Results

Our raytracer is implemented in C++. The raytracer is parallelized using OpenMP. We employ a simple parallelization strategy: we divide the whole image into square tiles of $N \times N$ pixels and assign the rendering of individual tiles to different threads.

Performance was measured on system with two Intel Xeon E5440 2.83 GHz quad-core CPUs with 6 MB shared L2 cache and 4×32 kB L1 data caches. In all tests the image resolution was 512×512 pixels, and tile size was set to $N = 32$ pixels. Rendering was parallelized using 4 threads.

The test scenes were prepared by converting triangle meshes into distance field representation at several different resolutions. We used models of Bunny, Dragon and Happy Buddha from the Stanford 3D Scanning Repository. The subgrid resolution of surface blocks was always set to 4^3 samples.

Three methods for surface cell intersection are compared in Table 1, with the number of trilinear interpolations they use for intersection evaluation inside single surface cell, and the total time to render the whole image. To compare the visual output we used a low resolution distance field of a torus. Even though the midpoint method is the fastest, it produces severe visual artifacts. These, however, tend to be less visible as the size of a surface cell projection approaches size of the pixel. Results of single and four steps of the false position method are visually indistinguishable. Thus, in all subsequent test we used single step of false position method for surface cell intersections.

Surface normal estimation methods are compared in Table 2. Normals calculated using the analytic derivatives of trilinear filter are discontinuous across surface cell boundaries, which is pronounced at specular highlights. The interpolation of differences is faster than the difference of interpolations even though it uses more samples. In subsequent tests we estimated surface normals using interpolation of differences.

The effect of primary ray acceleration using min/max-buffers is illustrated in Figure 6. The number of sparse block grid traversal steps is visualized using pseudo-color. The average number of traversal steps per pixel is clearly reduced with min/max-buffers.

We compared the rendering performance without primary ray acceleration and with the acceleration using min/max-buffers at three different resolutions. The results are summarized in Table 3. In this test, single primary ray is cast per pixel, and simple shading model is evaluated at the intersection point. The use of min/max-buffer led to improved performance in all cases.

In final test we evaluated the performance of shadow ray acceleration using modified traversal. For this test, shadow

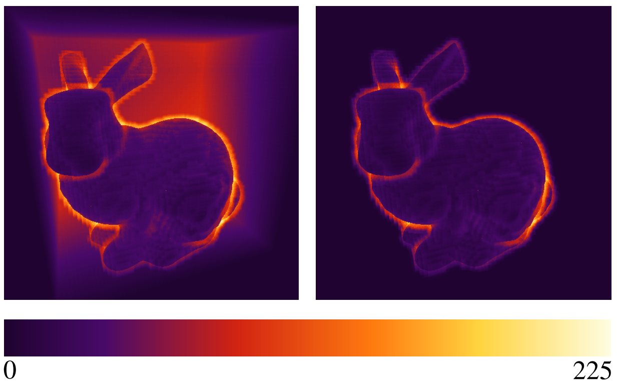

Figure 6: Comparison of the number of primary ray traversal steps per pixel with no acceleration (left), and with the acceleration using min/max-buffers (right).

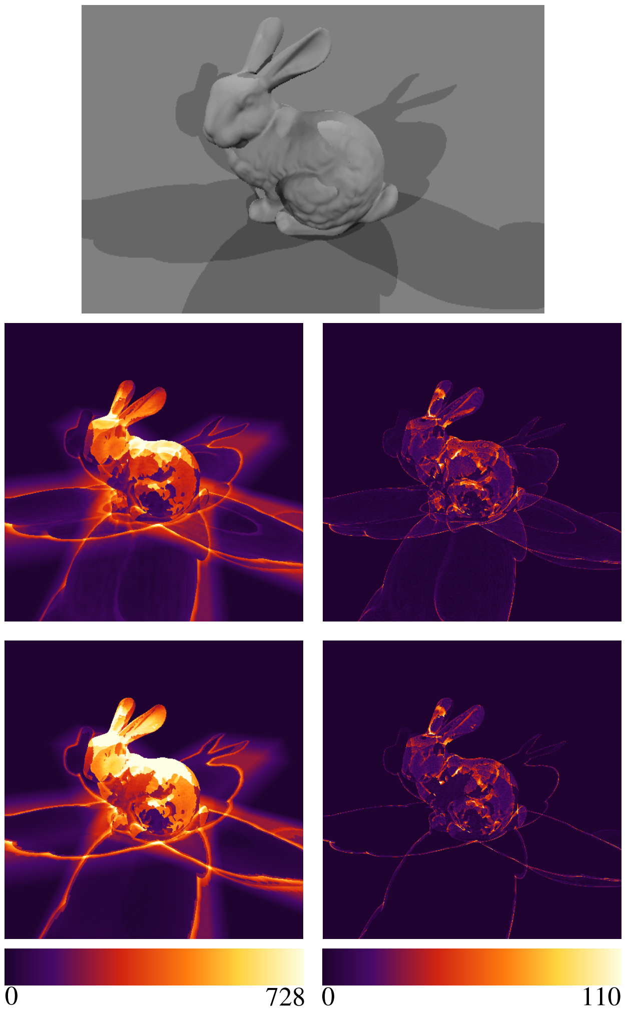

Figure 7: Comparison of the number of traversal steps and intersection tests for shadow rays using normal and modified sparse block grid traversal. Rendered image of the test scene is shown topmost. Top row: normal traversal. Bottom row: modified traversal. Left column: number of traversal steps per pixel. Right column: number of intersection tests per pixel.

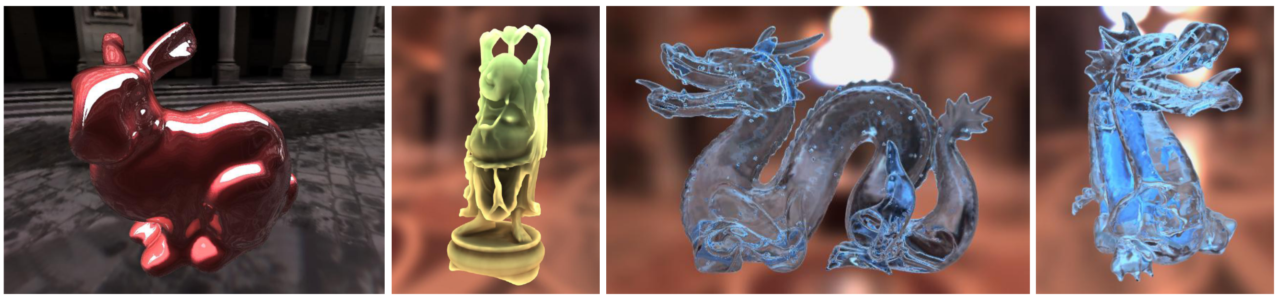

Figure 5: Three test scenes Bunny, Dragon, and Happy Buddha.

		without min/max-buffers		256x256 min/max-buffers			128x128 min/max-buffers			64x64 min/max-buffers		
model	resolution	#trav. steps [$\times 10^3$]	time [ms]	#trav. steps [$\times 10^3$]	time [ms]	speedup [%]	#trav. steps [$\times 10^3$]	time [ms]	speedup [%]	#trav. steps [$\times 10^3$]	time [ms]	speedup [%]
Bunny	$128 \times 128 \times 100$	5733	35.0	2885	28.6	18	3009	27.7	21	3247	28.3	19
Bunny	$256 \times 256 \times 200$	8915	45.0	2982	33.1	27	3186	32.3	28	3502	32.2	29
Bunny	$512 \times 508 \times 400$	15538	69.9	3056	51.1	27	3330	47.2	33	4287	50.0	28
Dragon	$128 \times 92 \times 60$	4264	28.8	2444	24.7	14	2538	23.9	17	2699	24.2	16
Dragon	$256 \times 184 \times 116$	6265	35.2	2536	28.1	20	2674	27.3	22	2902	27.4	22
Dragon	$512 \times 364 \times 232$	10435	50.4	2411	38.3	24	2620	35.7	29	3260	37.5	26
Buddha	$56 \times 128 \times 56$	3089	23.8	2225	22.1	7	2301	21.4	10	2409	21.6	9
Buddha	$108 \times 256 \times 108$	4146	28.0	2467	25.8	8	2568	25.9	8	2740	25.3	10
Buddha	$212 \times 512 \times 212$	6028	34.2	2360	33.9	1	2552	31.7	7	2973	32.5	5

Table 3: Comparison of primary ray casting performance without acceleration and with the acceleration using min/max-buffers.

rays are cast to five directional lights. Both the number of traversal steps and the number of surface cell intersection tests is recorded.

The effect of shadow ray acceleration is illustrated in Figure 7. The reduction of number of intersection tests is noticeable mainly inside large shadow areas on the ground plane. The number of traversal steps is locally lowered at some places and raised at other.

We compared the performance of the shadow ray casting with and without shadow ray acceleration. The results are summarized in Table 4. Tests were performed for two values of threshold n used in modified traversal. The threshold determines for how many steps the coarse grid traversal continues after the first surface cell is visited until it is restarted.

Although the number of intersection tests was always reduced using the modified traversal, the number of traversal steps increased in some cases due to traversal restart, leading to worse performance than that achieved with normal traversal.

9 Conclusions and Future Work

In this paper we have presented the techniques behind our interactive distance field raytracer. Basic methods for ray-surface intersection and surface normal estimation were described and compared. We have shown two simple acceleration techniques for casting of primary and shadow rays and analyzed their performance on three test scenes. In all tests we achieved reasonable frame rates ranging from $10 \sim 50$ fps.

In future work we would like to investigate means of accelerating the search for intersection either by augmenting the coarse grid with additional information encoding the empty space, or by building a hierarchical spatial subdivision data structure on top of the sparse block grid.

Acknowledgement

This work has been partially supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS 13139/10/800890 and by the Ministry of Education, Youth and Sports of the Czech Republic under the research program MSM 6840770014 and LC-06008 (Center for Computer Graphics), the Aktion Kontakt OE/CZ grant no. MEB 060906.

model	resolution	no acceleration			n=12				n=32			
		#trav. steps [$\times 10^3$]	#isect. tests [$\times 10^3$]	time [ms]	#trav. steps [$\times 10^3$]	#isect. tests [$\times 10^3$]	time [ms]	speedup [%]	#trav. steps [$\times 10^3$]	#isect. tests [$\times 10^3$]	time [ms]	speedup [%]
Bunny	$128 \times 128 \times 100$	7639	529	66.4	8259	348	59.4	10.5	8876	320	59.7	10.1
Bunny	$256 \times 256 \times 200$	11153	542	82.8	11663	345	72.8	12.2	12360	294	71.6	13.6
Bunny	$512 \times 508 \times 400$	18586	472	123.4	19137	292	108.9	11.7	20106	232	106.6	13.6
Dragon	$128 \times 92 \times 60$	6694	426	56.6	8145	365	56.7	-0.2	8548	342	56.7	-0.2
Dragon	$256 \times 184 \times 116$	9708	433	68.6	10791	333	64.7	5.7	11731	285	64.5	6.0
Dragon	$512 \times 364 \times 232$	15881	377	96.5	16693	264	87.0	9.9	17550	197	84.4	12.5
Buddha	$56 \times 128 \times 56$	734	104	16.6	1015	100	17.9	-7.9	1030	100	17.9	-8.3
Buddha	$108 \times 256 \times 108$	1110	116	19.7	1471	107	21.2	-7.6	1620	106	21.8	-10.4
Buddha	$212 \times 512 \times 212$	1581	96	24.6	1963	84	26.0	-5.9	2295	78	26.6	-8.2

Table 4: Comparison of shadow ray casting performance using normal and modified traversal.

References

[1] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Proc. Eurographics '87*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland, August 1987.

[2] J.A. Bærentzen and N.J. Christensen. Volume sculpting using the level-set method. In *International Conference on Shape Modeling and Applications*, pages 175–182, 2002.

[3] D.E. Breen and R.T. Whitaker. A Level-Set Approach for the Metamorphosis of Solid Models. *IEEE TVCG*, 7(2):173–192, 2001.

[4] R. Bridson. *Computational aspects of dynamic surfaces*. PhD thesis, Stanford University, 2003.

[5] A. Brodersen, K. Museth, S. Porumbescu, and B. Budge. Geometric texturing using level sets. *IEEE TVCG*, 14(2):277–288, 2008.

[6] C.S. Co, B. Hamann, and K.I. Joy. Iso-splatting: A Point-based Alternative to Isosurface Visualization. In *Proceedings of the 11th Pacific Conference on Computer Graphics*, pages 325–334, 2003.

[7] P. Djeu, S. Keely, and W. Hunt. Accelerating Shadow Rays Using Volumetric Occluders and Modified kd-Tree Traversal. In *Proceedings of Conference on High-Performance Graphics 2009*, pages 69–76, 2009.

[8] D. Enright, S. Marschner, and R. Fedkiw. Animation and Rendering of Complex Water Surfaces. *ACM Transactions on Graphics*, 21(3):736–744, 2002.

[9] R. Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[10] T. Hinks and K. Museth. Wind-driven snow buildup using a level set approach. In *Eurographics Ireland Workshop Series*, pages 19–26, Dublin, Ireland, 2009.

[11] B. Houston, M.B. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Transactions on Graphics*, 25(1):151–175, 2006.

[12] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM.

[13] M.B. Nielsen and K. Museth. Dynamic Tubular Grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, 26(3):261–299, 2006.

[14] R.N. Perry and S.F. Frisken. Kizamu: A system for sculpting digital characters. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, page 56. ACM, 2001.

[15] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

[16] R.T. Whitaker, D.E. Breen, K. Museth, and N. Soni. A framework for level set segmentation of volume datasets. In *Proceedings of ACM International Workshop on Volume Graphics*, pages 159–68, 2001.

[17] A. Williams, S. Barrus, R.K. Morley, and P. Shirley. An Efficient and Robust Ray–Box Intersection Algorithm. *Journal of Graphics Tools*, 10(1):49–54, 2005.