
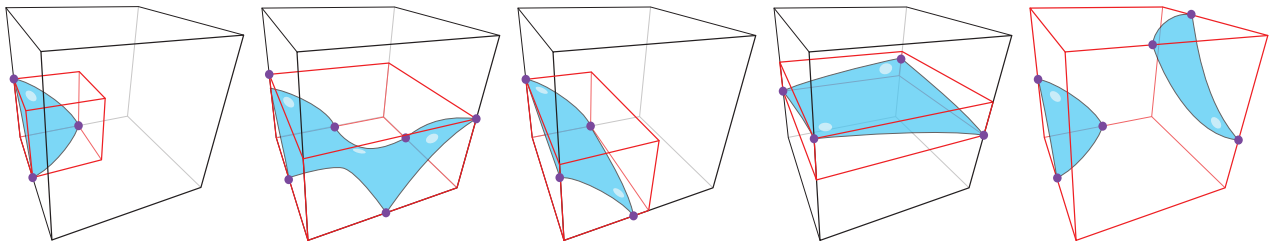


# Tight Bounding Boxes for Voxels and Bricks in a Signed Distance Field Ray Tracer

H. Hansson-Söderlund and T. Akenine-Möller 



**Figure 1:** The cubes in black outlines are the voxel extents and the blue regions are the trilinearly interpolated surfaces (Equation 1) inside the voxels. With our novel method for computing tight bounding boxes (shown in red), we have measured substantial performance improvements for ray tracing of signed distance field grids.

## Abstract

We present simple methods to compute tight axis-aligned bounding boxes for voxels and for bricks of voxels in a signed distance function renderer based on ray tracing. Our results show total frame time reductions of 20–31% in a real-time path tracer.

## CCS Concepts

• Computing methodologies → Ray tracing; Volumetric models;

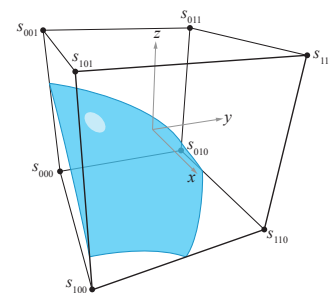
## 1. Introduction

Signed distance functions (SDFs) can be used to model impressive scenes with simple geometrical objects as building blocks and using operators, such as union, intersection, smooth subtraction, and smooth union [BBB\*97], on them. Several games have used SDFs extensively, e.g., Claybook [Aal18] and Dreams [Eval5], but SDFs are also used in game engines, such as Unreal Engine, to speed up rendering. In this short paper, we focus on accelerating ray tracing of the SDF grid, which consists of a collection of voxels with a trilinearly interpolated surface in each.

In a single voxel with  $2 \times 2 \times 2$  signed distance values,  $s_{ijk}$  with  $i, j, k \in \{0, 1\}$ , the standard equation for trilinear interpolation is

$$f(x, y, z) = (1-z) \left( (1-y) \left( (1-x)s_{000} + xs_{100} \right) + y \left( (1-x)s_{010} + xs_{110} \right) \right) + z \left( (1-y) \left( (1-x)s_{001} + xs_{101} \right) + y \left( (1-x)s_{011} + xs_{111} \right) \right), \quad (1)$$

where  $x, y, z \in [0, 1]$ . Note that the surface  $f(x, y, z) = 0$ , which is a third-order polynomial, inside a voxel is most often the one that is desired, as can be seen in Figure 2. In a recent method for ray tracing of SDF grids, the voxel extents were used as the bound-



**Figure 2:** A three-dimensional voxel with signed distances  $s_{ijk}$  at the  $2 \times 2 \times 2$  voxel corners. A possible surface formed by trilinear interpolation of the signed distances  $s_{ijk}$  is shown in blue.

ing box for a single voxel or brick of voxels in, e.g., the sparse voxel set (SVS) and sparse brick set (SBS) data structures respectively [HEAM22]. Reducing the size of a bounding volume (BV) is an important performance optimization since the number of rays likely to intersect it, is roughly proportional to the surface area of the BV [MB90]. We present two improvements over previous work.

The first is a novel technique to compute a tight axis-aligned bounding box (AABB) around just the surface extents inside a voxel. See Figure 1 for some examples. The second is to compute a tight box around the surface inside a brick, where a brick is a group of, e.g.,  $7^3$  voxels. Our results show significant performance boosts.

## 2. Voxel Box Computation

Since the surface in Equation 1 is located in  $[0, 1]^3$ , we seek to compute a tight bounding box defined by  $\mathbf{b}^{\min} \in [0, 1]^3$  and  $\mathbf{b}^{\max} \in [0, 1]^3$ , such that  $\mathbf{b}_i^{\min} \leq \mathbf{b}_i^{\max}$ , where  $i \in \{x, y, z\}$ . Scaling and translation can be applied to set the size and position afterward by e.g., transforming the intersecting ray. Note that there is a surface inside the voxel only if at least one  $s_{ijk} \geq 0$  and one  $s_{ijk} \leq 0$ .<sup>†</sup> We start by describing how to compute the intersection point between the trilinearly interpolated surface inside a voxel and a voxel edge. Since the surface reverts to linear interpolation in one variable along each edge (set  $x = 0$  and  $y = 0$  in Equation 1, for example), there can be at most one intersection along an edge. If the three-dimensional point at the start of a voxel edge is  $\mathbf{p}_0$  with signed distance  $s_0$  and the end of the edge is  $\mathbf{p}_1$  with signed distance  $s_1$ , then the surface (Equation 1) intersects the edge at

$$\mathbf{p} = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0), \text{ where } t = \frac{s_0}{s_0 - s_1} \in [0, 1], \quad (2)$$

if  $s_0 \cdot s_1 \leq 0$ , i.e., they have the different signs or at least one of them is zero. To avoid divisions by zero in Equation 2, i.e., when  $s_0$  is equal to  $s_1$ , we first perform careful initialization of  $\mathbf{b}^{\min}$  and  $\mathbf{b}^{\max}$ , and do not evaluate Equation 2 when  $s_0 = s_1$ . This includes the case where  $s_0 = 0$  and  $s_1 = 0$ , which indicates that there is surface along the entire edge, and as a consequence, that edge should be included in the box.

For box initialization, we note that if either of the four signed distances on a quadrilateral face, e.g.,  $x = 0$  or  $x = 1$ , on the unit cube is zero, then the corresponding bounding box coordinate should be 0 or 1 depending on which face it is. For the  $x$ -coordinate of the tight bounding box, we express this efficiently as

$$\begin{aligned} \mathbf{b}_x^{\min} &= (Z(s_{000}) | Z(s_{001}) | Z(s_{010}) | Z(s_{011}) == 1) ? 0 : 1, \\ \mathbf{b}_x^{\max} &= (Z(s_{100}) | Z(s_{101}) | Z(s_{110}) | Z(s_{111}) == 1) ? 1 : 0, \end{aligned} \quad (3)$$

where  $Z(t)$  is 1 if  $t$  is zero and otherwise it is 0, and  $|$  is binary OR. Note that if  $\mathbf{b}_x^{\min} = 1$  and  $\mathbf{b}_x^{\max} = 0$  then that indicates an invalid box. Similar computations are done for the  $y$ - and  $z$ -coordinates.

When the initialization of  $\mathbf{b}^{\min}$  and  $\mathbf{b}^{\max}$  has been taken care of, they can be used in conjunction with the edge intersection points to compute a tight axis-aligned bounding box (AABB) around only the surface inside the voxel. This box can be found as the smallest AABB around the initialized  $\mathbf{b}^{\min}$  &  $\mathbf{b}^{\max}$ , and all the existing edge intersection points (Equation 2) between the surface (Equation 1) and the 12 edges of the voxel. Next, follows a sketch of a proof of this statement.

<sup>†</sup> We include the 0 in this comparison, because  $f(x, y, z) = 0$  indicates that  $(x, y, z)$  lies on the surface.

## Proof Sketch

To start with, let us focus on a curve that is generated on a specific voxel face, e.g.,  $z = 0$ . The curve on that face is defined as

$$\begin{aligned} f(x, y) &= (1 - y)((1 - x)s_{000} + xs_{100}) + y((1 - x)s_{010} + xs_{110}) \\ &= s_{000} + x(s_{100} - s_{000}) + y(s_{010} - s_{000} + kx) \end{aligned} \quad (4)$$

where  $x, y \in [0, 1]$  and  $k = s_{110} - s_{010} - s_{100} + s_{000}$ . Some examples are shown in Figure 3. Note that one can rewrite Equation 4 for  $f(x, y) = 0$  as

$$y = \frac{-s_{000} - x(s_{100} - s_{000})}{s_{010} - s_{000} + kx}. \quad (5)$$

As can be seen in Figure 3, this curve is often split into two parts. However, when  $s_{110} - s_{010} - s_{100} + s_{000} = 0$ , it reverts to a single straight line.

Differentiating Equation 4 gives

$$\left( \frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right) = (s_{100} - s_{000} + ky, s_{010} - s_{000} + kx), \quad (6)$$

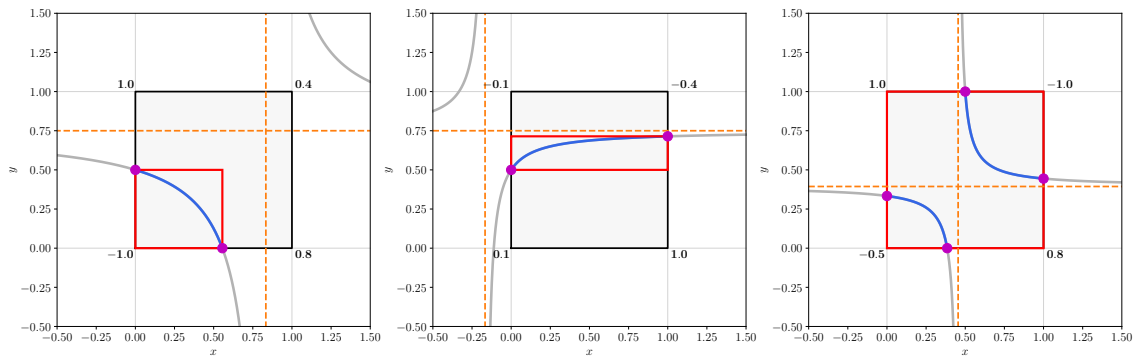
i.e., both  $\partial f/\partial x$  and  $\partial f/\partial y$  are lines in  $y$  and  $x$ , respectively, with the same coefficient  $k$  for  $y$  and  $x$ . This means that the curve  $f(x, y)$  is monotonic, but it can be discontinuous (unless it is a straight line) since the denominator in Equation 5 is a linear function in  $x$  and the denominator will be equal to 0 for  $x = (s_{000} - s_{010})/k$ .

We argue that a tight two-dimensional axis-aligned bounding box (AABB) of the curve on a voxel face is the smallest AABB around the intersection points between the curve and the four voxel edges. In the case of a straight line, it is definitely so. If only one curved segment is inside the unit square (left and middle curves in Figure 3), then that part of the curve must be bounded by the intersection points between the curve and the voxel edges, since the curve is monotonic (Equation 6). Two curved segments inside the unit square can only occur when two opposite corners have the same signs for their signed distances ( $s_{ijk}$ ), while the other two corners have opposite signs compared to these first distances. In this case, the tightest AABB on that voxel face is the entire voxel face, i.e., an AABB around the intersection points between the curve and voxel edges (right case in Figure 3).

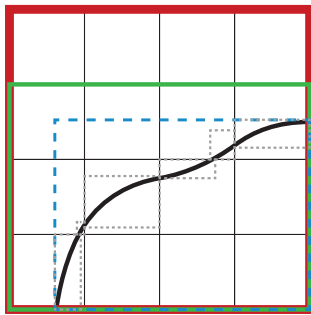
Since any two-dimensional, axis-aligned slice of the voxel has this behavior and since the curves are monotonic, it follows that a tight three-dimensional AABB around the surface inside the voxel can be found as the smallest AABB around the intersection points between the surface and the voxel edges. Imagine, for example, all the slices from  $z = 0$  to  $z = 1$ . For this set of slices, all the intersection points will occur on the faces of the voxel box. However, as we have seen, the extreme points on a voxel box occurs on the voxel edges. This concludes our sketch for a proof.

## 3. Brick Box Computation

In the sparse brick set (SBS) data structure [HEAM22], an AABB around each ‘‘brick’’ of voxels were used, and a bounding volume hierarchy (BVH) [MOB\*21] built around those AABBs. A brick of voxels could be, e.g.,  $7 \times 7 \times 7$  voxels. The bounding box around the entire brick was used, which was an oversight, since a simple improvement would be to compute the smallest AABB around



**Figure 3:** Examples of curves, generated using bilinear interpolation, on square voxel faces. The signed distances, which were used to generate the curves in blue and gray are shown at the corners of the gray square. Note that the red bounding boxes can be found as the smallest axis-aligned bounding box around the purple circles, which are intersection points between voxel face edges and the curve. The asymptotes are drawn as dashed horizontal and vertical lines.



**Figure 4:** Here we show a two-dimensional brick consisting of  $4 \times 4$  voxels containing a black curve. Previously, the box (red) around the entire brick was used when building the bounding volume hierarchy. One improvement is to instead compute a box (green) around the non-empty voxels. We propose an even tighter alternative shown as a dashed blue box. This is computed as the smallest AABB around the tight voxel boxes (gray, dashed) from Section 2.

the non-empty voxels. We improve upon this further by proposing to compute tight bounding boxes for each non-empty voxel in the brick using the method from Section 2. The final brick box is then the smallest AABB around all these tight voxel boxes. This is illustrated in Figure 4, and as can be seen, this has the potential to generate even smaller bounding boxes.

#### 4. Results

We have implemented our algorithm in the Falcor [KCK\*21] rendering framework and compared to the methods by Hansson Söderlund et al. [HEAM22]. In particular, we compare to the sparse voxel set (SVS) and the sparse brick set (SBS) data structures, where we added the method from Section 2 to the former and the method from Section 3 to the latter. In the SVS data structure, an AABB is computed around each voxel and a BVH is built around all such

AABBs. This BVH is then ray traced using the GPU. In contrast, the SBS data structure stores only one AABB around each brick of voxels. In our case, a brick is  $7^3$  voxels, which is the same as the SBS method by Hansson Söderlund et al. [HEAM22]. A BVH is then built around all the brick AABBs and ray traced using the GPU.

We have gathered results on both an NVIDIA RTX 3090 and RTX 4090. Before measuring, rendering was done at full speed for 180 seconds to warm up the GPU to avoid temporary high frequency clocking. The camera was animated and measurements done over 4,500 frames at  $1920 \times 1080$  pixels. All times are reported in milliseconds and include path tracing, accumulation of samples, and tone mapping.

The scenes that we used for evaluation are shown in Figure 5 where the grid resolution was  $512^3$  for each object. Our performance evaluation was done using only the root solver by Marmitt et al. [MKWF04] and using only the interpolated continuous normals and fast shadow ray testing [HEAM22]. The other results from Hansson Söderlund et al.'s work are expected to preserve the same relations, and hence omitted. The main timing results are summarized in Table 1.

On the RTX 3090, total frame time reductions were 20–31% for SVS and 20–28% for the RTX 4090. For SBS, the reductions were 20–29% for the RTX 3090 and 20–27% for the RTX 4090. For such a small modification, the performance improvement is rather substantial, in our opinion. The main results above for SBS were using the method with the smallest AABBs that we could compute, i.e., using the method depicted with a blue dashed line in Figure 4. We do not report full results for the method with the green box in Figure 4, and instead only note that it was about 3% slower than the proposed method using the smallest AABBs.

#### 5. Conclusions and Future Work

We have presented techniques to compute tight bounding boxes for the surface inside a voxel defined by trilinear interpolation of

RTX 3090 / RTX 4090	Goblin	Heads	Ladies
SVS [HEAM22]	12.7 / 5.8	25.0 / 9.5	23.0 / 8.9
SVS (ours)	10.1 / 4.6	17.2 / 6.8	16.2 / 6.5
Reduction	20.3% / 19.7%	31.2% / 28.0%	29.8% / 26.6%
SBS [HEAM22]	21.7 / 10.0	37.3 / 16.4	35.4 / 16.2
SBS (ours)	17.4 / 8.0	26.7 / 11.9	26.2 / 11.9
Reduction	19.9% / 20.0%	28.5% / 27.3%	26.0% / 26.2%

**Table 1:** Total frame times for the three scenes in Figure 5 rendered at  $1920 \times 1080$  pixels. Times are reported in milliseconds. As can be seen, our SVS variant with tighter bounding boxes reduces total frame time by 20–31% compared to the previous method, and our SBS variant with tighter boxes reduces total frame time by 20–29%. Measurements were done on NVIDIA RTX 3090 / 4090. Note that the percentages were computed using full precision and so do not always match if the percentage is computed directly from the numbers from the table.

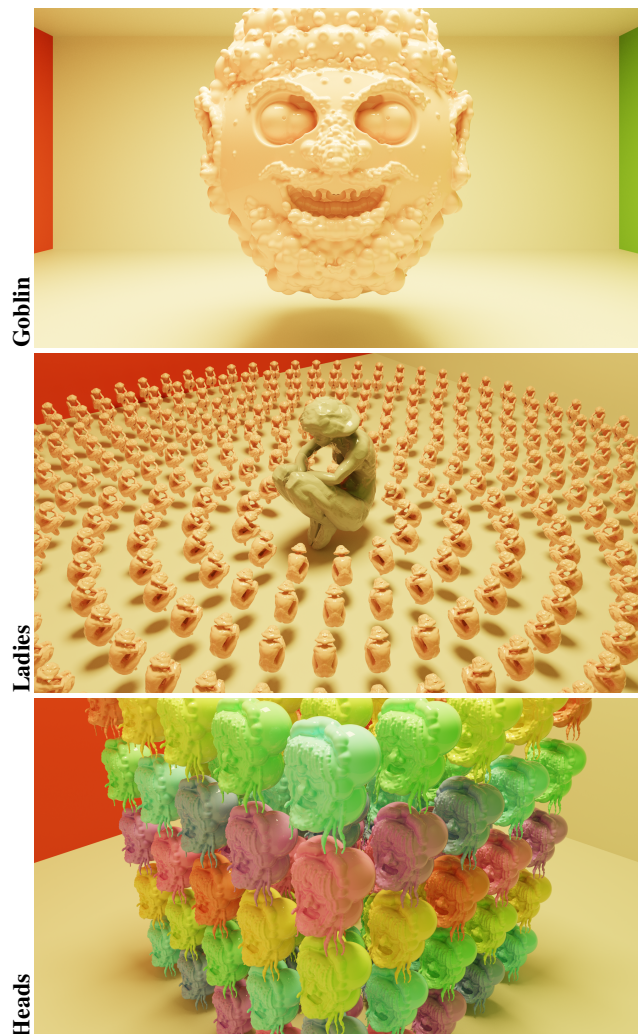
signed distances at the voxel corners. In addition, a similar technique was presented for bricks, i.e., small blocks of voxels. When averaging over two GPUs and three scenes, our SVS method gave 26% total frame time reduction. For SBS, the number was 25%.

For future work, adding an occupancy map [MDKH01] to SBS might provide additional decrease in rendering time. An occupancy map stores one bit indicating whether there is a surface in the corresponding voxel. This map is accessed first and additional voxel data is read only if the occupancy map bit was one.

**Acknowledgments** Thanks to Brian Karis for tipping us off about the first part of brick box computation.

## References

- [Aal18] AALTONEN S.: GPU-based Clay Simulation and Ray Tracing Tech in Claybook. In *Game Developers Conference* (2018). 1
- [BBB\*97] BLOOMENTAL J., BAJAJ C., BLINN J., WYVILL B., CANI M.-P., ROCKWOOD A., WYVILL G.: *Introduction to Implicit Surfaces*. 1997. 1
- [Eva15] EVANS A.: Learning from Failure: a Survey of Promising, Unconventional and Mostly Abandoned Renderers for Dreams PS4, a Geometrically Dense, Painterly UGC Game. In *Advances in Real-Time Rendering in Games, SIGGRAPH Courses* (2015). 1
- [HEAM22] HANSSON SÖDERLUND H., EVANS A., AKENINE-MÖLLER T.: Ray Tracing of Signed Distance Function Grids. *Journal of Computer Graphics Techniques* 11, 3 (September 2022), 94–113. URL: <http://jcgt.org/published/0011/03/06/>. 1, 2, 3, 4
- [KCK\*21] KALLWEIT S., CLARBERG P., KOLB C., YAO K.-H., FOLEY T., WU L., CHEN L., AKENINE-MÖLLER T., WYMAN C., CRASSIN C., BENTY N.: The Falcor Rendering Framework, August 2021. URL: <https://github.com/NVIDIAGameWorks/Falcor>. 3
- [MB90] MACDONALD J. D., BOOTH K. S.: Heuristics for Ray Tracing using Space Subdivision. *The Visual Computer* 6, 3 (1990), 153–166. 1
- [MDKH01] MEISSNER M., DOGGETT M., KAMUS U., HIRCHE J.: Accelerating Volume Rendering using an On-chip SRAM Occupancy Map. In *IEEE International Symposium on Circuits and Systems* (2001), pp. 757–760. 4
- [MKWF04] MARMITT G., KLEER A., WALD I., FRIEDRICH H.: Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing. In *Vision, Modeling, and Visualization* (2004), pp. 429–435. 3
- [MOB\*21] MEISTER D., OGAKI S., BENTHIN C., DOYLE M. J., GUTHE M., BITTNER J.: A Survey on Bounding Volume Hierarchies for Ray Tracing. *Computer Graphics Forum* 40, 2 (2021), 683–712. 2



**Figure 5:** The test scenes used for performance evaluation. The performance measurements were done using path tracing at one sample per pixel with three bounces and one square light source. However, the images here have converged by accumulating samples for each pixel over consecutive frames.