Transparent rendering and slicing of integral surfaces using per-primitive interval arithmetic

M. Aydinlilar[1], C. Zanni[1]

[1]Université de Lorraine, CNRS, Inria, LORIA

Abstract

We present a method for efficient incorporation of integral surfaces within existing robust processing methods such as interval arithmetic and segment-tracing. We based our approach on high-level knowledge of the field function of the primitives. We show application to slicing and transparent rendering of integral surfaces based on interval arithmetic.

1. Introduction

Implicit volumes are defined as the set of points where $f(\mathbf{p}) \geq c$ with f a scalar field and c a given iso-value. They have many valuable properties for modeling and fabrication with additive manufacturing technologies. They allow defining effortlessly smooth volumes with arbitrary topology and provide a simple way to query whether a given point in space is inside or outside the volume.

In this work, we consider skeleton-based implicit surfaces, namely convolution surfaces [BS91] and scale-invariant integral surfaces [ZBQC13] which are both defined by integrating a kernel along a skeleton. These definitions allow representing a wide variety of geometry ranging from artistic shapes to lattices and microstructures. Recent research [PRZ17, LLZ*21] have shown that skeleton-based implicit representations provide good properties both for modeling self-supporting structures (which is essential for fabrication) and decreasing the stress concentration due to the blended shape at joints between the trusses of a lattice.

One of the core operations to prepare a model for 3D printing is called slicing. It consists of computing cross-sections of the model within horizontal planes. This can be done straightforwardly with implicit volumes by sampling field values on a regular grid defined in a given slicing plane. The same algorithm can be used both for slicing and transparent rendering [Lef13]. Such a strategy can help to reduce the number of required field evaluations.

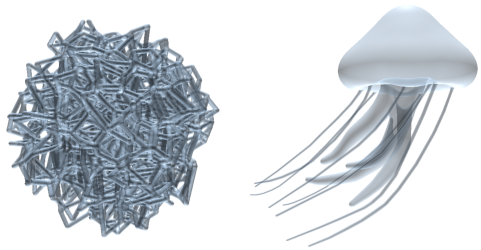

Figure 1: *Transparent rendering results for micro-structures (left) and an artistic model (right).*

One of the challenges with the usage of integral surfaces is the efficient incorporation in existing robust processing methods for slicing and rendering such as interval arithmetic [Mit90] and segment-tracing [GGPP20]. This is due to the complex closed-form expressions of the integrals defining the field.

We propose using both high-level knowledge on the integral surface's primitives and analysis of the field variation to incorporate them more efficiently in algorithms based on either segment-tracing or interval arithmetic. Namely, we present a way to compute precise inverval bounds for field values and directional derivatives (i.e. directional Lipschitz constant). We use these contributions inside a ray-tracing algorithm used for both rendering and slicing. In addition to robust processing, this would allow their use in a broader context (e.g., combining them with other implicit representations).

Scale-invariant integral surface In this work, we focus on scale-invariant integral surfaces [ZBQC13] which provide simple radius control, self-similarity of blending at all scales, and limited blurring of details with summation blending. Similarly to convolution surfaces [BS91], they are defined by integration of a kernel along a skeleton :

$$f(\mathbf{p}) = \int_{\mathbf{q}\in S} \frac{1}{\tau_S(\mathbf{q})} k\left(\frac{\|\mathbf{p}-\mathbf{q}\|}{\tau_S(\mathbf{q})}\right) \mathrm{d}\mathbf{q} \qquad (1)$$

where S is a line segment with associated linearly varying radius τ_S and k is a kernel function. In this study, we focus on compact support kernels that provide localized field evaluation :

$$k(d) = N \left(1 - \left(\frac{d}{\sigma}\right)^2\right)^{\frac{n}{2}} \; if\ d < \sigma\,, 0\ otherwise. \qquad (2)$$

where n is the degree of the kernel, σ is a constant impacting the visual smoothness of blends and N is a normalization factor. For this kernel, with n even, both the field f and its gradient have closed-form expressions (see [ZBQC13]).

© 2022 The Author(s)
Eurographics Proceedings © 2022 The Eurographics Association.

DOI: 10.2312/egs.20221027

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

2. Previous work

We will first discuss the general methods for ray-tracing of implicit surfaces then the slicing methods, most of them relying on the same methodology as ray-tracing approaches.

Ray-tracing is one of the ways to visualize implicit volumes. For each pixel of an image, intersections between a ray $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d}$ and the implicit surface need to be computed. This is equivalent to solving $f \circ \mathbf{r}(t) - c = 0$ where c is the iso-value of interest. For opaque rendering, only the first root has to be localized, while all roots need to be computed for transparent rendering.

This is a long-studied problem, for which two prominent families of techniques have been introduced : the approaches relying on self-validating numerical methods like interval arithmetic [Moo66] and the ones relying on Lipschitz bounds like sphere-tracing [Har96].

On the one hand, interval arithmetic-based methods for ray-tracing implicits [Mit90, CHMS00] replace arithmetic operations on floating points with operations on intervals of floating points. They then use bisection for root localization on these intervals. Extensions of these methods include the usage of revised-affine arithmetic [FPC10] and efficient evaluation on GPU [Kee20]. However, for integral surface representation, error accumulation due to the complexity of the expression makes these methods impractical (see supplemental material for evidence).

On the other hand, sphere-tracing [Har96] relies on the knowledge of a Lipschitz bound, e.g., an upper bound on the norm of the field gradient, in order to compute a sphere free of any iso-surface around a given point. This allows stepping along the ray without ever passing through the surface and converging toward the iso-surface. A notable extension of sphere-tracing is segment-tracing [GGPP20] which computes local directional Lipschitz bound on the ray to increase the step size. Computing transparency with this family of methods requires careful algorithm parametrization as roots are fixed points for the algorithm. Until now, no directional Lipschitz bounds have been derived for integral surfaces.

Note that a dedicated ray-tracing method for scale-invariant integral surfaces exists [AZ21], however, while providing fast rendering, it only supports opaque rendering, does not provide guarantees and do not allow the combination of this type of surface with other implicit representations.

Slicing A first strategy to slice implicit volumes consists in categorizing all voxels of a given slice by computing field value at voxel centers [LLZ*21]. As discussed in [Lef13], it is also possible to slice implicit volumes by relying on transparent rendering algorithms. Once all roots have been computed along rays associated to voxel lines, the voxels in/out status can easily be deduced by jointly iterating over the roots and the voxels. This is the approach we use in this paper.

Another recent approach relies on a variant of interval and revised affine arithmetic, such as relying on space subdivision using quadtrees [PMF*20].

Acceleration structure for field query An orthogonal problem of integral surfaces is the efficient selection of skeleton primitives that influence a given area in space. Lists of primitives sorted by support entry point (given an input direction) have been used for this purpose both for rendering (per-pixel linked-list [Bru19, AZ21]) and slicing (single global list of primitives based on the slicing direction [LLZ*21]). In the present paper, we combine both approaches.

3. Our method

Our objective is to efficiently use integral surface segment primitives with linearly varying radius inside existing robust methods such as interval arithmetic and segment-tracing. We first present a simple and efficient way to compute field bounds for use in interval arithmetic on a per-primitive basis. Then we study directional Lipschitz bounds that allow improvement of our bounds in specific interval configurations and provide a way to use integral surface within the segment-tracing algorithm. Finally, as an application of our method, we present a transparent rendering algorithm that can be used for rendering and slicing.

3.1. Per-primitive field bounds for interval arithmetic

In order to use integral surfaces inside interval arithmetics-based algorithms, we need to derive field value bounds given an interval $[t_0, t_1]$ along a ray, e.g., finding an interval $[f_-, f_+]$ such that:

$$f \circ \mathbf{r}([t_0, t_1]) \in [f_-, f_+]$$

For the line segment primitives with linearly varying radius, given an input ray direction, the field is always monotonously increasing and then decreasing.

Monotonous intervals can be detected by checking the sign of the derivatives at the interval end-points. With this observation, for monotonous intervals, the interval inclusion is trivial (see Figure 2(Left)). We can directly use the endpoint values as bounds:

$$[f_-, f_+] = [\min(f \circ \mathbf{r}(t_0), f \circ \mathbf{r}(t_1)), \max(f \circ \mathbf{r}(t_0), f \circ \mathbf{r}(t_1))]$$

For ambiguous intervals where we have both positive and negative derivatives (increasing and decreasing values), we can either use infinite values as upper bound :

$$[f_-, f_+] = [\min(f \circ \mathbf{r}(t_0), f \circ \mathbf{r}(t_1)), +\infty] \,,$$

or calculate a bound on the absolute value of directional derivative and extrapolate field values based on the maximal derivative and the field values at the interval endpoints. We can then compute the upper bound as the intersection between two lines (see Figure 2(Right)). We describe the calculation of Lipschitz bound in the next section. This calculation allows us to bound the field values robustly. The resulting per-primitive intervals can then be used in classical interval arithmetics evaluation.

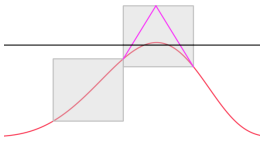

Figure 2: *Ranges on two intervals, monotonous (left) and ambiguous (right).*

© 2022 The Author(s)
Eurographics Proceedings © 2022 The Eurographics Association.

3.2. Directional Lipschitz bound calculation

In order to provide tighter bounds on the intervals containing a maximum, we derive a directional Lipschitz bound on a per primitive basis. To do so, we study the absolute value of the derivative of $f \circ \mathbf{r}$, which is defined by :

$$|\nabla f(\mathbf{p})^T \mathbf{d}| = \left| \int_{\mathbf{q} \in S} \frac{1}{\tau_S^2(\mathbf{q})} k' \left(\frac{\|\mathbf{p}-\mathbf{q}\|}{\tau_S(\mathbf{q})} \right) \mathbf{d}^T \frac{\mathbf{p}-\mathbf{q}}{\|\mathbf{p}-\mathbf{q}\|} \, \mathrm{d}\mathbf{q} \right| \quad (3)$$

This formula could be bounded by studying the integral of the absolute value. However, we can obtain a tighter bound by separating the integral into two parts, the one where the integrand is positive and the one where it is negative, then taking the sub-integral with the largest absolute value. The segmentation is defined by a linear inequality $(\mathbf{p}-\mathbf{q})^T \mathbf{d} > 0$ (see Figure 3).

As described in [AZ21], we can compute the range $[s_0, s_1]$ of skeleton points which support intersects the ray, as well as the argmin of $\|\mathbf{p}-\mathbf{q}\|/\tau_S(\mathbf{q})$. Using these two informations, we can bound the first two terms within the integral. As τ_S appears in the denominator of the integrand, we bound it by the minimum of $\tau_S(s_0)$ and $\tau_S(s_1)$. For the term using k', we adopt a similar strategy as in [GGPP20]. We use knowledge on the kernel to select either the global bound of k' if it is in range :

$$\frac{n}{\sqrt{n-1}} \left(1 - \frac{1}{n-1}\right)^{\frac{n}{2}-1} \frac{N}{\sigma},$$

otherwise, the maxima that is reached at one of the interval endpoints. Then, we only have to compute a bound on a simpler integral, e.g. for the positive domain:

$$\int_{\mathbf{q} \in S / (\mathbf{p}-\mathbf{q})^T \mathbf{d} > 0} \mathbf{d}^T \frac{\mathbf{p}-\mathbf{q}}{\|\mathbf{p}-\mathbf{q}\|} \, \mathrm{d}\mathbf{q}$$

By bounding the positive and negative domains based on both the range of interest $[t_0, t_1]$ and the range $[s_0, s_1] \cap [0,1]$(see Figure 3), it is possible to show that upper bound (positive domain) and lower bound (negative domain) can be computed by evaluating the integral with the newly defined larger domain at the end of the range (e.g., in t_0 and t_1 respectively).

We can then obtain the global bound by taking the maximum

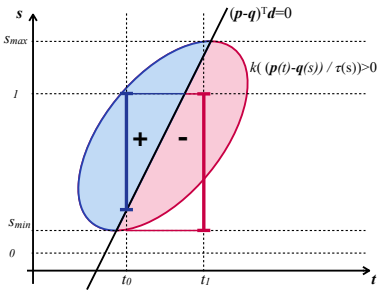

Figure 3: *Clipping-space for ray/primitive intersection. The primitive is parametrized with $s \in [0,1]$. Input range for Lipschitz bound query is $t \in [t_0, t_1]$. The support of a primitive can be divided into two halves based on the sign of $(\mathbf{p}-\mathbf{q})^T \mathbf{d}$. The blue half (resp. red) has a positive (resp. negative) contribution to the derivative.*

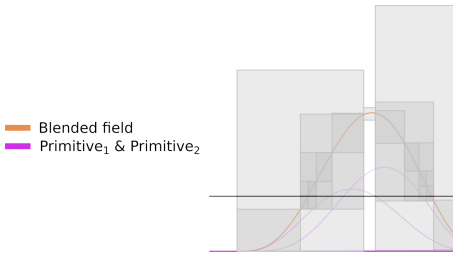

Figure 4: *Progressive range computation for two primitives (purple) along a ray. The blended field (summation) is displayed in orange. The ray is initially segmented in three (one cut per-primitive).*

of both absolute values. While the integral admit a closed-form expression, the formula is numerically instable when the minimal distance between the ray line and segment line tend toward zero. We instead bound the integrand on the domain of interest which also has the advantage of being less costly to evaluate (details are provided in supplemental material).

4. Slicing and rendering : ray processing

As an application, we present ray-processing for both transparent rendering and slicing of integral surfaces blended with summation blending, i.e. $f(\mathbf{p}) = \sum f_i(\mathbf{p})$. For slicing, we use the methodology described in [Lef13].

First, in order to process rays efficiently, we use both the same per-pixel linked-list of primitives as well as the ray segmentation procedure described in [AZ21]. The per-pixel linked list provides empty space skipping based on kernel supports without any field evaluation. A given ray is pre-segmented using the point where the minimal distance between a ray and line-segment is reached (e.g., where the kernel function is maximized). This segmentation allows us to have intervals close to the ideal behavior where they would be either monotonously increasing or decreasing. During ray-processing, per-primitive interval ranges are combined with regular interval arithmetic to obtain the interval range of the global field. In all our examples, we use summation blending only. This can be extended to other blending operations as long as an interval inclusion can be found for them.

After ray segmentation, the core of the processing loop is similar to other interval methods such as [FPC10]. We check whether an intersection is possible, then either discard the interval or bisect until the required precision is reached (see Figure 4). Resolution used for slicing is the half the size of a voxel.

For transparent rendering implementation, we also use polynomial interpolation as soon as we have small enough intervals, this reduces the number of field evaluations in comparison to bisection.

5. Results

We compare both our bound computation to regular interval arithmetic methods (e.g. using directional derivative computed from closed form gradient for the Lipschitz bound). We observe large improvements in bound quality (see supplemental material). In order to demonstrate the usefulness of our method, we present applications to both slicing and transparent rendering, both relying on the same ray-processing methodology.

© 2022 The Author(s)
Eurographics Proceedings © 2022 The Eurographics Association.

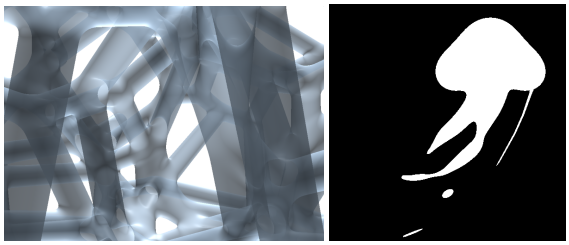

Figure 5: *Detail on micro-structure with smooth blending (left) and a slice produced for additive manufacturing (right).*

	[LLZ*21]	[LLZ*21] Bucket 64^2	Our Bucket 64^2
jellyfish	27.2*s* / 55.4*s*	11.1*s* / 21.9*s*	2.9*s* / 3.5*s*
microstructure	201*s* / 414*s*	5.1*s* / 10.3*s*	6.9*s* / 8.0*s*

Table 1: *Slicing models into respectively 512 and 1024 slices with an XY resolution of* 512×512 *voxels.*

Ray-tracing Our ray-tracing implementation was done in OpenGL. Our runtimes are measured on an NVIDIA GeForce GTX 1650 graphics card with a resolution of 700x700. The jellyfish model (Figure 1(right)), consisting of 154 line segment primitives, was rendered using an average of 431.4 ms per frame. The micro-structure model (Figure 1(left)), consisting of 2895 primitives was rendered with an average of 1037.1 ms per frame.

Note that the memory requirement for the stack used in recursive processing is limited thanks to the initial ray segmentation. Depending on the application, computational time could be reduced by stopping the ray processing loop based on the level of opacity reached instead of capturing all transparent layers.

Slicing When using the method for slicing, we use the same acceleration structure as in [LLZ*21] with the notable exception that we create one linked-list of primitives per bucket of pixels instead of a single linked-list for the full space. This can dramatically impact runtime for complex skeletons sliced at high resolution. We compare our slicing runtime with the method of [LLZ*21] in Table 1. Comparisons were run on an Intel Corei7-8850H (2.60GHz, single-core used). Due to the bucketing strategy's major positive impact, we applied it to both methods. We observe larger runtime improvements when the resolution required along the ray direction increases. Indeed adding more slices does not require additional field evaluations in our case (only the generation of the voxel image from the computed roots is increased). Note that the ray direction is always the printing direction in our implementation, but this could be modified based on input models.

Our directional Lipschitz bound also allows the use of integral surfaces within the segment-tracing algorithm. From our experiments, bisection-based approach tends to be more efficient when computing all the roots (see supplemental material).

5.1. Conclusion

We have presented a simple way to use scale-invariant integral surfaces inside both interval arithmetic framework and segment-tracing algorithm. For interval arithmetic, we only rely on simple observation of the field behavior. A similar approach could also apply to a broader range of primitives. For the derivation of directional Lipschitz bound, we rely on careful analysis of the field integral to provide efficient bound computation. We provide applications of our methodology to both slicing and transparent rendering. Contrary to previously dedicated algorithms, the proposed approach would allow incorporation of integral surfaces in more general frameworks and allow their use with other representations or more complex blending.

While applied to scale-invariant integral surfaces, our methodology can also be applied to other integral surfaces with linearly varying weights. Several improvements are worth investigating, such as the adaptation to interval arithmetic queries on volumes and tighter bound computation. The latter requires a careful trade-off between the tightness of the computed bound and the required amount of computation to obtain them.

Acknowledgments

This work was supported by the ANR IMPRIMA(ANR-18-CE46-0004). We thank Nathaniel Seyler for his help with bucketing.

References

[AZ21] AYDINLILAR M., ZANNI C.: Fast ray tracing of scale-invariant integral surfaces. *Computer Graphics Forum* (2021). 2, 3

[Bru19] BRUCKNER S.: Dynamic visibility-driven molecular surfaces. In *Computer Graphics Forum* (2019), vol. 38(2), pp. 317–329. 2

[BS91] BLOOMENTHAL J., SHOEMAKE K.: Convolution surfaces. In *Proceedings SIGGRAPH '91* (1991), ACM, pp. 251–256. 1

[CHMS00] CAPRANI O., HVIDEGAARD L., MORTENSEN M., SCHNEIDER T.: Robust and efficient ray intersection of implicit surfaces. *Reliable Computing 6*, 1 (2 2000), 9–21. 2

[FPC10] FRYAZINOV O., PASKO A., COMNINOS P.: Fast reliable interrogation of procedurally defined implicit surfaces using extended revised affine arithmetic. *Computers & Graphics 34*, 6 (2010), 708–718. 2, 3

[GGPP20] GALIN E., GUÉRIN E., PARIS A., PEYTAVIE A.: Segment Tracing Using Local Lipschitz Bounds. *Computer Graphics Forum* (2020). 1, 2, 3

[Har96] HART J. C.: Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10 (Dec 1996), 527–545. 2

[Kee20] KEETER M. J.: Massively parallel rendering of complex closed-form implicit surfaces. *ACM Trans. Graph. 39*, 4 (jul 2020). 2

[Lef13] LEFEBVRE S.: Icesl: A gpu accelerated csg modeler and slicer. In *18th European Forum on Additive Manufacturing (AEFA'13)* (2013). 1, 2, 3

[LLZ*21] LIU S., LIU T., ZOU Q., WANG W., DOUBROVSKI E. L., WANG C. C.: Memory-efficient modeling and slicing of large-scale adaptive lattice structures. *Journal of Computing and Information Science in Engineering 21*, 6 (2021). 1, 2, 4

[Mit90] MITCHELL D. P.: Robust ray intersection with interval arithmetic. In *Proceedings on Graphics Interface '90* (CAN, 1990), Canadian Information Processing Society, p. 68–74. 1, 2

[Moo66] MOORE R.: *Interval Analysis*. Prentice-Hall series in automatic computation. Prentice-Hall, 1966. 2

[PMF*20] POPOV D., MALTSEV E., FRYAZINOV O., PASKO A., AKHATOV I.: Efficient contouring of functionally represented objects for additive manufacturing. *Computer-Aided Design 129* (2020). 2

[PRZ17] PANETTA J., RAHIMIAN A., ZORIN D.: Worst-case stress relief for microstructures. *ACM Transactions on Graphics 36*, 4 (2017). 1

[ZBQC13] ZANNI C., BERNHARDT A., QUIBLIER M., CANI M.-P.: SCALe-invariant Integral Surfaces. *Computer Graphics Forum 32* (2013). 1

© 2022 The Author(s)
Eurographics Proceedings © 2022 The Eurographics Association.