Segment Tracing Using Local Lipschitz Bounds

Eric Galin[1] Eric Guérin[2] Axel Paris[1] Adrien Peytavie[1]

[1] Univ Lyon, Université Lyon 1, CNRS, LIRIS, France [2] Univ Lyon, INSA-Lyon, LIRIS, CNRS, France

Abstract

We introduce Segment Tracing, a new algorithm that accelerates the classical Sphere Tracing method for computing the intersection between a ray and an implicit surface. Our approach consists in computing the Lipschitz bound locally over a segment to improve the marching step computation and accelerate the overall process. We describe the computation of the Lipschitz bound for different operators and primitives. We demonstrate that our algorithm significantly reduces the number of field function queries compared to previous methods, without the need for additional accelerating data-structures. Our method can be applied to a vast variety of implicit models ranging from hierarchical procedural objects built from complex primitives, to simulation-generated implicit surfaces created from many particles.

1. Introduction

Implicit surfaces are a general tool for representing shapes by using iso-surfaces of so-called field functions. Hierarchical implicit surface models are a powerful representation for modeling and animating shapes of arbitrary topology. As compared to other surface representations, implicit surfaces offer many advantages in terms of performing geometric operations like Boolean operations, blending, warping and offsets. Implicit surfaces have demonstrated to be useful for a vast variety of applications including modeling of blob-shaped and organic objects, surface reconstruction, point-based modeling, and extraction of smooth surface from fluid simulations. Another interesting aspect of procedurally defined implicit surfaces is their ability to represent complex shapes in compact representation, *i.e.* with a relatively small memory footprint.

However, visualizing implicit surfaces has always been considered difficult because of the indirect characterization of the surface. Broadly, there are two major visualization methods for rendering implicit surfaces: they may be rendered either indirectly by first converting them into meshes using polygonization algorithms (we refer the reader to [DALJ*15] for a complete overview of existing techniques), or by directly ray-tracing them. Although polygonization converts implicit surfaces into a mesh representation that lends itself for graphics hardware rendering, they are typically not guaranteed: they may not accurately detect the correct topology or miss detailed parts of the implicit surface unless using a high resolution grid. This results in a memory intensive data structure to represent an otherwise compact model, as outlined in [SJNJ19]. In contrast, *Ray tracing* directly samples the implicit surface by computing the intersection point along a ray.

In this paper, we focus on ray-tracing implicit surfaces using the Sphere Tracing algorithm proposed in [Har96]. This conceptually simple and general algorithm adaptively marches along the ray to find the first ray-object intersection. It requires the function to be Lipschitz, *i.e.* have a bounded derivative. A good approximation of the Lipschitz bound is important as the adaptive stepping distance is inversely proportional to this bound. Its accurate computation remains an open problem though, and existing techniques rely on external spatial subdivision techniques, such as octrees or regular grids, which involve memory consuming data structures and pre-processing.

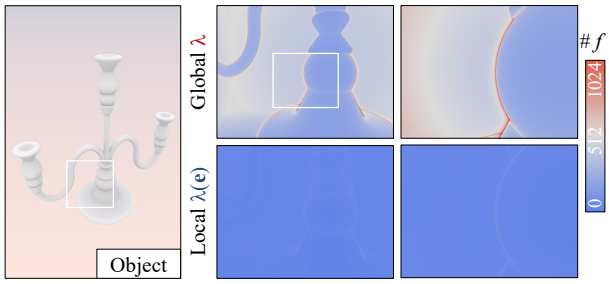


Figure 1: *We tackle the problem of computationally intensive Sphere Tracing by evaluating local Lipschitz bound along the ray during the ray marching process. Our Segment Tracing method reduces the number of field function queries #f and accelerates ray-object intersection.*

A key observation of our work is that hierarchical constructive implicit surface models such as field function representations [PASS95], the BlobTree [WGG99], or procedurally defined scalar fields [RMD11] implicitly provide a direct means for computing an accurate Lipschitz bound. In this paper, we introduce an optimized algorithm that takes advantage of the mathematical properties provided by the constructive definition of the field function.

Our algorithm adaptively marches towards the surface, and evaluates the local Lipschitz bound along the ray at every step so as to better adapt the stepping distance (Figure 1). Our method can be applied to a variety of hierarchical implicit surface models, accelerates ray tracing and does not necessitate additional pre-processing or accompanying memory intensive data-structure for storing the local properties of the implicit model. The algorithm is compact, computationally efficient and easy to implement, and lends itself for graphics hardware implementation.

More precisely, our contributions are two-fold: 1) we introduce Segment Tracing, a method that accelerates the Sphere Tracing algorithm by computing local Lipschitz bound; 2) we show how to practically compute those bounds for a variety of primitives and operators. Experiments demonstrate that our algorithm reduces the number of field function queries and therefore speeds-up intersection computations for different implicit surface models.

2. Related Work

In this section, we review existing techniques for computing the intersection between a ray and an implicit surface. A broad introduction to implicit surfaces can be found in [BW97].

Analytic techniques aim at computing the piecewise closed form expression of the field function along the ray. When the function along the ray is a polynomial equation, solutions and thus intersections can be solved analytically for low degree polynomials, *i.e.* quadrics, cubics and quartics, and by Descartes rule of signs [Han83], Sturm sequences [Wij85], and Laguerre's method [WT90] for higher degree polynomials. Nishita *et al.* [NN94] proposed to express the field function along the ray by Bézier functions and employed Bézier clipping for accelerating the computation of the intersection; this method was adapted to graphics hardware in [KSN08]. Loop *et al.* [LB06] implemented ray tracing of algebraic surfaces on graphics hardware by approximating implicit forms with piecewise Bernstein polynomials. Sherstyuk [She99] proposed to approximate the field function along the ray by low degree polynomials to speed up computation, yet at the expense of a less accurate intersection computations.

Interval analysis Guaranteed ray intersection requires extra information, which in most cases is produced by the derivative of the function. Interval analysis finds ray intersections by defining the function and its derivative on intervals instead of single values. Mitchell [Mit90] ray traced implicit models by combining recursive Interval Arithmetic to isolate monotonic ray intervals and standard bisection as a root refinement method. Gamito *et al.* [GM07] proposed reduced Affine Arithmetic for ray casting specific implicit displacement surfaces formulated with blended noise functions. Knoll *et al.* [KHK*09] addressed the fast implementation of interval and affine arithmetic for rendering arbitrary implicit surfaces. Performance was achieved through low-level hardware optimization and coherent traversal methods.

Ray marching is a general algorithm that progressively marches along the ray with constant steps and makes no assumptions about the mathematical properties of the field function along the ray. The robustness is achieved by setting a low incremental step, which makes the algorithm extremely computationally intensive. Perlin *et al.* [Per89] used ray-marching to render complex objects modeled with noisy functions. Singh *et al.* [SN10] proposed an adaptive marching point algorithm for real-time ray tracing of arbitrary implicit surfaces on graphics hardware. In contrast to Sphere Tracing methods, the accuracy and performance of their technique depend on a predefined surface dependent marching step size.

Lipschitz techniques were introduced by Kalra *et al.* [KB89] who proposed a robust method for ray tracing algebraic and some non-algebraic surfaces given Lipschitz bounds of the field function and first derivative. The Lipschitz condition allowed to create an efficient octree partitioning guaranteed to contain the implicit surface, and to find ray intersections within each cell.

The principles of adaptive ray-marching were first applied to the rendering of deterministic fractal geometry [HSK89]. Hart [Har96] introduced Sphere Tracing that marches along the ray toward the first intersection in adaptive steps guaranteed not to penetrate the implicit surface according to the Lipschitz criterion. Unlike the work of [KB89] it does not require that the evaluated function should be C^2 continuous but only a bound on the magnitude of the derivative. Over-stepping [KSK*14] is a Sphere Tracing acceleration heuristic that consists in stepping along the ray slightly farther than the safe stepping distance, checking that the spheres of two consecutive marching steps still overlap. A specific accelerated implementation of Sphere Tracing was proposed in [GGP*15] to compute the intersection of a procedural height field defined by a construction tree and a ray. The algorithm takes advantage of the hierarchical combination of compactly supported terrain primitives organized into a tree to compute local Lipschitz constants, which correspond to the local maximum slope of the terrain, and speed-up computations.

Recently, Seyb *et al.* [SJNJ19] proposed a variant of Sphere Tracing for directly rendering deformed signed distance field to avoid computationally demanding and complex global Lipschitz bound overestimates that would reduce performance. The method steps along a curved ray in the undeformed space according to the signed distance. They mention that another strategy to accelerate Sphere Tracing would consist in computing local Lipschitz bounds, which is the focus of our work.

Acceleration techniques aim at reducing the number of field function queries, which are the most computationally intensive part. Hart [Har96] prescribed the use of acceleration data structures such as grids or octrees for storing local Lipschitz bound, therefore allowing larger steps and reducing the number of field function evaluation. [GPP*10] proposed a specific fitted bounding volume hierarchy construction to reduce the number of field function queries for point-based implicit surfaces.

Compared to previous work, our framework relies on an accurate local Lipschitz bound computation. Contrary to other techniques, our method does not necessitate additional memory consuming acceleration structures. Instead we rely on the hierarchical structure of the implicit model to perform acceleration during queries. This allows for faster field function and gradient evaluations and larger and safer marching steps without the need for any pre-processing.

3. Algorithm

An implicit surface is mathematically defined as the set of points in space $\mathbf{p}$ that satisfy the equation $f(\mathbf{p}) = 0$, formally:

$$S = \{\mathbf{p} \in \mathbb{R}^3 \,|\, f(\mathbf{p}) = 0\}$$

Given a ray Δ and its parametric equation $\delta(t) = \mathbf{o} + t\,\mathbf{u}$ with $\mathbf{o}$ the origin of the ray, $\mathbf{u}$ the normalized direction and t the distance to the origin, computing the intersections between the ray and the surface $\Delta \cap S$ consists in finding the solutions of the equation $f \circ \delta(t) = 0$.

Recall that a function $f : \mathbb{R}^3 \to \mathbb{R}$ is Lipschitz over Ω if and only if there exists a positive constant λ such that:

$$\forall (\mathbf{p},\mathbf{q}) \in \Omega \times \Omega,\ |f(\mathbf{p}) - f(\mathbf{q})| \leqslant \lambda \|\mathbf{p} - \mathbf{q}\|$$

The Lipschitz constant λ is the minimum bound satisfying this equation. In practice, Lipschitz constants are often overestimated by a bound.

3.1. Sphere tracing

Sphere Tracing consists in marching along the ray Δ from the origin $\mathbf{o}$ towards the surface by an adaptive increment defined as $s(\mathbf{p}) = |f(\mathbf{p})|/\lambda$ that is sufficiently small to guarantee that it does not penetrate the surface (Figure 2). For simplicity, we denote the field function along the ray as $f(t) = f \circ \delta(t)$. We denote μ the threshold value from which we consider that we are so close to the surface that we intersect it. Let λ denote the global Lipschitz bound of f. The algorithm can be outlined as follows:

1. Compute the intersection between the ray and the domain $[t_-, t_+] = \Delta \cap \Omega$, and initialize $t = t_-$.
2. While $t < t_+$, compute $f(t)$.
   2.1 If $|f(t)| < \mu$, then an intersection occurs.
   2.2 Otherwise increment t with $s(t) = |f(t)|/\lambda$ and continue.

While marching adapts to the field function values $f(t)$ along the ray, the use of a global Lipschitz bound λ over the entire domain Ω limits the overall efficiency of this otherwise simple and elegant algorithm.

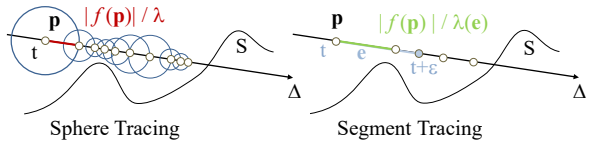


Figure 2: *Sphere Tracing marches along the ray with adaptive steps computed from a global Lipschitz bound λ, which results in small steps and a high number of field function queries. In contrast, Segment Tracing evaluates the local Lipschitz bound $\lambda(\mathbf{e})$ on candidate segments $\mathbf{e}$ which results in fewer, larger yet safe steps.*

3.2. Segment tracing

Our approach leverages this limitation by computing local Lipschitz bound during the marching process (see accompanying video). Locally, *i.e.* in small domains, the gradient of f can be evaluated more precisely, which leads to a smaller Lipschitz bound. This is combined to the fact that the direction of the gradient ∇f can be different from the direction of the ray, which again yields more accurate Lipschitz bound values.

Rather than partitioning the domain into a grid and evaluating the Lipschitz bound for every voxel, we propose an accelerated Segment Tracing algorithm adapted to our model and based on the evaluation of two queries: the field function evaluation at a given point $f(\mathbf{p})$ local and directional Lipschitz bound $\lambda(\mathbf{e})$ on a segment $\mathbf{e}$ (Figure 2).

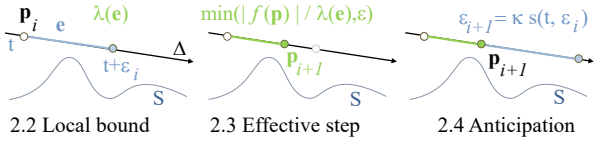


Figure 3: *Overview of the main steps of the algorithm.*

At every step i, we try to move forward by a candidate distance ε_i and compute the effective stepping distance denoted as $s(t,\varepsilon_i)$. The candidate distance defines an interval $[t, t+\varepsilon_i]$ along the ray corresponding to the segment $\mathbf{e}(t,\varepsilon_i) = [\delta(t), \delta(t+\varepsilon_i)]$. We compute the Lipschitz bound $\lambda(\mathbf{e}(t,\varepsilon_i))$ of the field function over the segment $\mathbf{e}(t,\varepsilon_i)$ to define the adaptive and accurate stepping increment as:

$$s(t,\varepsilon_i) = \min\left(|f(t)|/\lambda(\mathbf{e}(t,\varepsilon_i)), \varepsilon_i\right)$$

The algorithm resembles Sphere Tracing in spirit with the following changes (see Figure 3). First, we need the definition of an initial step ε_0. Then, step 2.2 is modified to take into account the computation of the local Lipschitz bound. Step 2.3 computes the effective stepping distance, taking care to ensure that the step is smaller than the candidate distance ε_i. Finally, step 2.4 adapts the next candidate stepping distance ε_{i+1} at every step i. The algorithm modification may be outlined as follows:

2.2 Compute $\lambda(\mathbf{e}(t,\varepsilon_i))$.
2.3 Evaluate $f(\mathbf{p}_i)$ and compute the position $\mathbf{p}_{i+1}$ with the safe stepping increment $s(t,\varepsilon_i)$.
2.4 Compute the next candidate stepping distance $\varepsilon_{i+1} = \kappa s(t,\varepsilon_i)$.

Step 2.4 of the algorithm repeatedly increases the candidate marching step by multiplying the previous safe stepping distance by a geometric amplification factor κ to try to move forward along the ray by larger distances. The computation of $s(t,\varepsilon_i)$ guarantees that the step length is intersection-safe.

The initial step ε_0 does not have a major impact over the overall performance of the algorithm. If the value ε_0 is too large, then the Lipschitz bound $\lambda(\mathbf{e}(t_0,\varepsilon_0))$ is evaluated over a long segment, thus is likely to be large as well, and consequently the next candidate distance ε_1 will be probably a small value computed as $\varepsilon_1 = |f(t_0)|/\lambda_0$. At the limit, when $\varepsilon_0 \to \infty$, the segment $\mathbf{e}$ becomes the entire ray Δ and $\lambda(\mathbf{e}(t_0,\varepsilon_0))$ is computed as the Lipschitz bound of f along the ray Δ, which is an upper bound. Conversely, if ε_0 is small, then the Lipschitz bound is evaluated on a reduced domain, and the next steps are likely to increase geometrically by a factor κ.

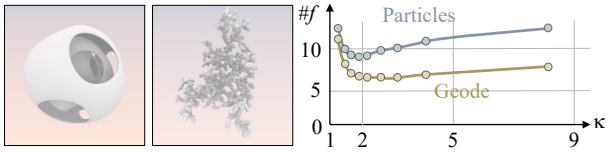


Figure 4: *Number of field function evaluations for Segment Tracing two different implicit models.*

The amplification factor $\kappa > 1$ defines the next candidate marching step ε_{i+1} according to the previous stepping distance $s(t_i, \varepsilon_i)$. Small values tend to preserve the locality of the computation of the Lipschitz bound during the overall process, whereas larger values increase the candidate marching step when marching away from the surface, thus resulting in fewer steps. Note that when $\kappa \to \infty$, the computed $\lambda(\mathbf{e})$ is actually the Lipschitz bound of the entire ray Δ. Our experiments show that in practice, the acceleration coefficient should be $\kappa \approx 2$ as illustrated in Figure 4.

Because the evaluation of λ is local to a segment, the underlying implicit model should provide a query way to evaluate it. An efficient implementation should include a simultaneous computation of both $f(t)$ and $\lambda(\mathbf{e}(t,\varepsilon))$ so that common calculations should be factored and performed only once (Section 4). The overhead introduced by these computations is compensated by the larger steps obtained with more accurate Lipschitz bound (Section 5).

4. Local Lipschitz bound computation

Here we consider an implicit model defined as a hierarchical construction tree [WGG99]; note that our work adapts to other hierarchical function representations such as procedurally defined scalar fields [RMD11] or function representations [PASS95]. The leaves of the tree are compactly supported skeletal primitives, whereas the nodes are operators that combine and aggregate their sub-trees (Figure 5). The field function computation $f(\mathbf{p})$ requires the traversal of the tree and the result depends on the evaluation of the combination of the primitives. Similarly, the computation of $\lambda(\mathbf{e})$ is obtained by traversing the tree. The structure implicitly implements a bounding volume hierarchy that allows for efficient pruning during the evaluation.

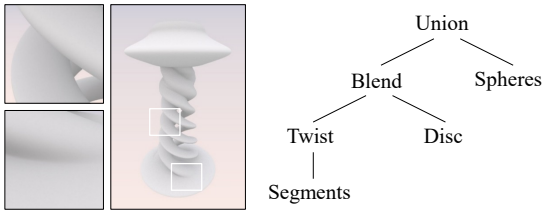


Figure 5: *Example of a hierarchical implicit surface model and its construction tree.*

Evaluating $\lambda(\mathbf{e})$ may be very complex for some primitives or deformation operators. In those cases, our strategy consists in using coarser bounds that are simpler or more efficient to compute. Let $\mathbf{e}$ a segment, $\mathbf{S} \supset \mathbf{e}$ its bounding sphere and $\Delta \supset \mathbf{e}$ its supporting ray. We denote $\lambda(\Delta)$ and $\lambda(\mathbf{S})$ the Lipschitz bound of f over the entire ray Δ and the sphere $\mathbf{S}$ respectively, we have $\lambda(\mathbf{e}) \leq \lambda(\Delta)$ and $\lambda(\mathbf{e}) \leq \lambda(\mathbf{S})$. Since f is built from a hierarchical combination of compactly supported primitives, the local Lipschitz bound are significantly smaller than the global bound λ, which reduces the number of queries, denoted as $\#f$, and accelerates Segment Tracing. In the next sections we describe the computation of λ for different primitives (Section 4.1), binary operators such as Boolean and blending (Section 4.2), affine transformations (Section 4.3) and deformations (Section 4.4).

4.1. Primitives

Skeletal primitives are defined as a combination of a compactly supported falloff filter [SM09] function $g : \mathbb{R} \to \mathbb{R}$ with the Euclidean distance to a skeleton d: $f = g \circ d$. Different types of skeletons can be used: points, line segments, discs, circles [WGG99], or even volumetric shapes such as spheres, cylinders or cones [BG04]. The field function along the ray may be written as:

$$f(t) = g \circ d \circ \delta(t)$$

The derivative of the field function along the ray is defined as:

$$f'(t) = g' \circ d \circ \delta(t) \, \nabla d \circ \delta(t) \cdot \delta'(t)$$

Recall that $\delta'(t) = \mathbf{u}$ where $\mathbf{u}$ is the unit vector of the direction of the ray. We need to evaluate the bound of the derivative of $|f'|$ over the segment $\mathbf{e}$, which is bounded by the product of two terms:

$$\lambda(\mathbf{e}) \leq |g' \circ d \circ \delta(\mathbf{e})| \, \|\nabla d \circ \delta(\mathbf{e}) \cdot \mathbf{u}\|$$

The term $|g' \circ d \circ \delta(\mathbf{e})|$ represents the Lipschitz bound of g applied to the image of the distance of the segment $d \circ \delta(\mathbf{e})$, whereas $\|\nabla d \circ \delta(\mathbf{e}) \cdot \mathbf{u}\|$ represents the bound of the dot product of gradient of the field function times with the unit direction $\mathbf{u}$.

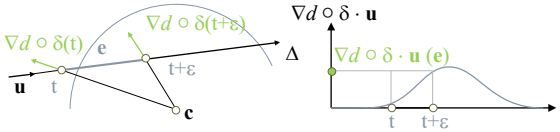


Figure 6: *Exact computation of the image of* $|\nabla d \circ \delta \cdot \mathbf{u}(\mathbf{e})|$ *for a point primitive: for this segment it is equal to the interval* $[|\nabla d \circ \delta(t) \cdot \mathbf{u}|, |\nabla d \circ \delta(t+\varepsilon) \cdot \mathbf{u}|]$ *where* ∇d *is the gradient of the Euclidean distance to the point* $\mathbf{c}$.

Gradient bound In the general case, a simple yet not accurate bound can be obtained as follows. Recall that $|\nabla d \circ \delta \cdot \mathbf{u}| \leq \|\nabla d\| \, \|\mathbf{u}\|$, where $\mathbf{u}$ is a unit vector and d represents the Euclidean distance to a skeleton. In fact $\|\nabla d_{\partial\Omega}\| = 1$ holds for all open domains, and indeed $d_{\partial\Omega}$ is 1-Lipschitz; therefore:

$$|\nabla d \circ \delta \cdot \mathbf{u}| \leq 1$$

We propose to improve this bound for some primitives such as points, spheres, or line segments, for which it is possible to compute the closed form expression of $\nabla d \circ \delta \cdot \mathbf{u}$ and obtain a tighter bound over $\mathbf{e}$. A typical example is the point primitive: let $\mathbf{c}$ the center, $\nabla d(\mathbf{p}) = (\mathbf{p} - \mathbf{c})/\|\mathbf{p} - \mathbf{c}\|$ and it is possible to compute the image of $\nabla d \circ \delta \cdot \mathbf{u}$ for any segment $\mathbf{e}$ (Figure 6).

Computing such a tight bound is effective for the parts of the ray that tend to be orthogonal to the gradient $\nabla f(\mathbf{p})$; in those cases $|\nabla d \circ \delta \cdot \mathbf{u}(\mathbf{e})| \ll 1$. During the adaptive marching process, those cases occur in particular whenever the ray Δ gets close to the implicit surface, especially at grazing angles. Those cases are computationally intensive for Sphere Tracing as the radius of the spheres, defined as $f(t)/\lambda$ becomes smaller as $f(t)$ drops to 0 near the surface. In contrast, our method computes a better Lipschitz bound for those segments and allows to step over longer distances $f(t)/\lambda(\mathbf{e})$ near the surface (Figure 7).

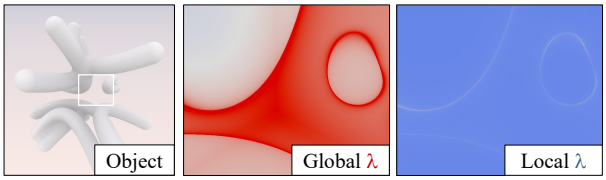


Figure 7: *Our method accelerates ray tracing for rays passing close to the surface by increasing the marching steps, which reduces the number of iterations compared to Sphere Tracing that necessitates many smaller steps.*

Falloff bound For simplicity, we note $\delta(\mathbf{e}) = \delta([t, t+\varepsilon])$. The term $|g' \circ d \circ \delta(\mathbf{e})|$ can be bounded by evaluating the distance interval to the skeleton $d \circ \delta(\mathbf{e})$ and then computing the bound of g' by Interval Analysis. This involves the computation of the derivative of the falloff filter function (Figure 8), which is detailed in Appendix A.

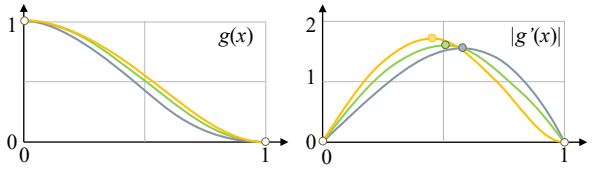


Figure 8: *Example of some falloff filter functions: Wyvill's C^1 sextic in green [WMW86], Wyvill's C^2 sextic in orange [WGG99], and a C^1 quartic in blue.*

Computing the interval image of the Euclidean distance to a skeleton $d \circ \delta(\mathbf{e})$ may be difficult for some complex primitives such as curves or volumetric skeletons including cones and cylinders. Our solution consists in approximating it by finding an enclosing interval. We have the following inclusion property: since the distance is 1-Lipschitz, then $d \circ \delta(\mathbf{e}) \subset [d(\mathbf{c}) - r, d(\mathbf{c}) + r]$ where $\mathbf{c}$ is the center of $\mathbf{e}$, and $r = \|\mathbf{e}\|/2$ its half length. This provides us with a means to bound $|g' \circ d \circ \delta|$ over the segment $\mathbf{e}$.

4.2. Binary operators

Binary operators include Boolean, *i.e.* union, intersection and difference, and blending operators. Recall that blending is defined as $f = f_\mathcal{A} + f_\mathcal{B}$, and that union and intersection can be defined as $f_\mathcal{U} = \max(f_\mathcal{A}, f_\mathcal{B})$ and $f_\mathcal{I} = \min(f_\mathcal{A}, f_\mathcal{B})$ respectively. Thus, for every type of operator, the global Lipschitz bound can be computed according to the bounds $\lambda_\mathcal{A}$ and $\lambda_\mathcal{B}$ of its sub-trees $\mathcal{A}$ and $\mathcal{B}$ as prescribed in [Har96]. Recall that the global Lipschitz bound λ of a blending node is defined as $\lambda = \lambda_\mathcal{A} + \lambda_\mathcal{B}$, and the bounds for Boolean operators are defined as: $\lambda = \max(\lambda_\mathcal{A}, \lambda_\mathcal{B})$.

Local Lipschitz bounds are computed similarly by recursively querying the sub-trees according to the segment parameter $\mathbf{e}$:

$$\lambda(\mathbf{e}) = \lambda_\mathcal{A}(\mathbf{e}) + \lambda_\mathcal{B}(\mathbf{e}) \qquad \lambda(\mathbf{e}) = \max(\lambda_\mathcal{A}(\mathbf{e}), \lambda_\mathcal{B}(\mathbf{e}))$$

Similar results may be obtained for other binary operators. Pasko *et al.* [PASS95] introduced a class of functions for computing Boolean operations, for instance union may be computed as:

$$f_\mathcal{U} = f_\mathcal{A} + f_\mathcal{B} + \sqrt{f_\mathcal{A}^2 + f_\mathcal{B}^2}$$

By computing the gradient of f, we obtain the bound:

$$\lambda(\mathbf{e}) = 2(\lambda_\mathcal{A}(\mathbf{e}) + \lambda_\mathcal{B}(\mathbf{e}))$$

4.3. Affine transformations

Let $\mathcal{D}$ denote an affine transformation node, $a : \mathbb{R}^3 \to \mathbb{R}^3$ the corresponding transformation, and $\mathcal{N}$ its sub-tree. The corresponding field function is defined as $f_\mathcal{D} = f_\mathcal{N} \circ a$. Computing the Lipschitz bound for affine transformations, *i.e.* rotation, translation and scaling, is easy to handle since they transform a segment $\mathbf{e}$ into another segment $\widetilde{\mathbf{e}} = a^{-1}(\mathbf{e})$. Translations and rotations are 1-Lipschitz, *i.e.*, $\lambda_\mathcal{D}(\mathbf{e}) = \lambda_\mathcal{N}(\widetilde{\mathbf{e}})$, and $\lambda_\mathcal{D}(\mathbf{e}) = 1/\alpha\, \lambda_\mathcal{N}(\widetilde{\mathbf{e}})$ for a scaling of factor α.

4.4. Deformations

Traditionally in implicit surface modeling, non-linear deformations operators are unary nodes, denoted as $\mathcal{W}$, characterized by a warping function $\omega : \mathbb{R}^3 \to \mathbb{R}^3$ that deforms space or a region of space and thus the scalar field of the underlying sub-tree $\mathcal{N}$. Their corresponding field function is defined as $f_\mathcal{W} = f_\mathcal{N} \circ \omega^{-1}$. The gradient of the deformation $\mathcal{W}$ is defined as:

$$\nabla f_\mathcal{W} = \nabla(f_\mathcal{N} \circ \omega^{-1}) = \nabla f_\mathcal{N} \circ \omega^{-1} \cdot \mathbf{J}_{\omega^{-1}}$$

We evaluate the Lipschitz bound by bounding the gradient of the field function and the Euclidean norm of the Jacobian matrix: $\|\nabla(f_\mathcal{N} \circ \omega^{-1})\| \leq \|\nabla f_\mathcal{N} \circ \omega^{-1}\| \, \|\mathbf{J}_{\omega^{-1}}\|$. Thus, for a given input segment $\mathbf{e}$, two local bounds need to be computed: the bound of the norm of the gradient $\|\nabla(f_\mathcal{N} \circ \omega^{-1})(\mathbf{e})\|$ of the transformed segment, and the bound of the norm of the Jacobian $\|\mathbf{J}_{\omega^{-1}}(\mathbf{e})\|$.

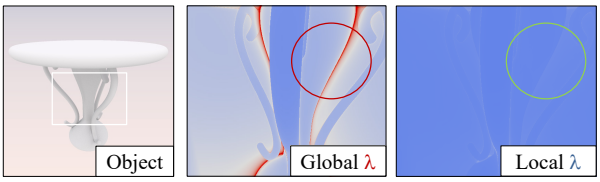


Figure 9: *Our Segment Tracing algorithm reduces the number of field function queries for non-linear deformations. In this example, the stem was created by combining two successive tapering operators in the construction tree.*

The complexity stems from the fact that we need to evaluate the bound of the gradient of the transformed segment $\widetilde{\mathbf{e}} = \omega^{-1}(\mathbf{e})$,

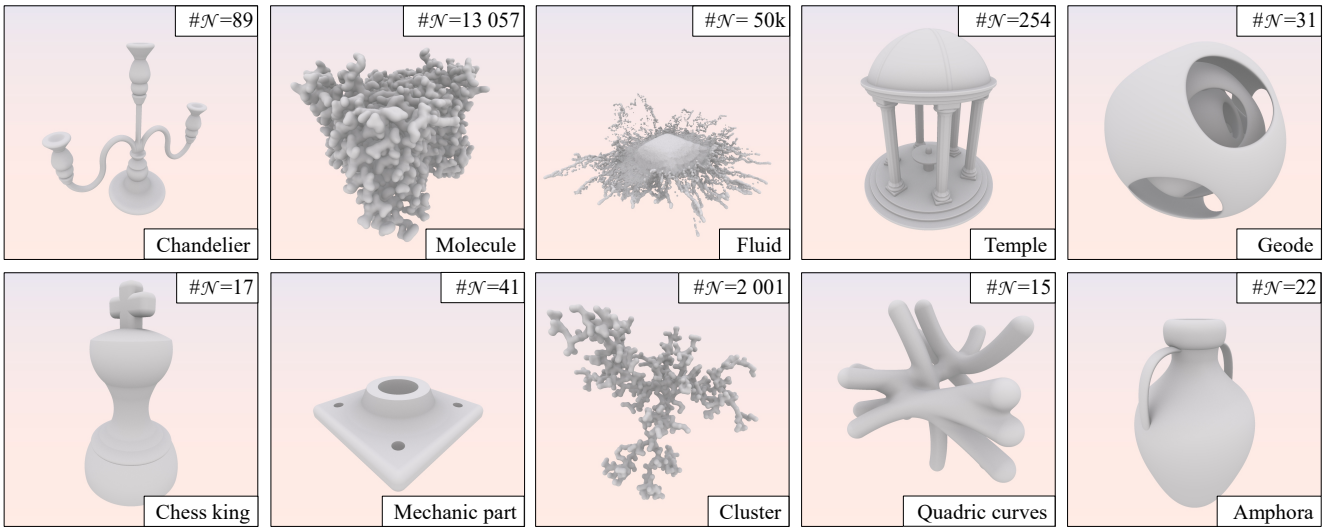


Figure 10: *Different models used for gathering ray tracing statistics.*

which is a curve, and the bound of the norm of the Jacobian over the segment $\|\mathbf{J}_{\omega^{-1}}(\mathbf{e})\|$. Seyb *et al.* [SJNJ19] recently addressed the construction of the transformed ray $\omega(\Delta)$ into curve for a restricted class of deformations. This strategy cannot be applied in our context however, since we need to compute the inverse transformed segment $\omega^{-1}(\mathbf{e})$ at every step of the Sphere Tracing algorithm, recursively traversing the construction tree, and possibly traversing other deformation operators. Instead, we proceed as follows.

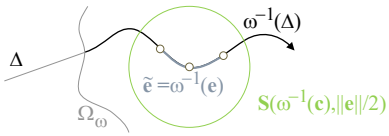


Figure 11: *The gradient along the deformed segment* $\widetilde{\mathbf{e}}$ *is bounded by the gradient of the bounding sphere* $\mathbf{S}$.

Gradient For every deformation, we compute a bounding sphere $\mathbf{S}$ such that $\mathbf{S} \supset \omega^{-1}(\mathbf{e})$. This allows us to bound $\|\nabla f_{\mathcal{N}} \circ \omega^{-1}(\mathbf{e})\|$ by recursively querying the sub-tree of the implicit surface model to evaluate $\|\nabla f_{\mathcal{N}}(\mathbf{S})\|$, simpler to compute although less accurate bound (Figure 11). Let λ denote a global Lipschitz bound of ω^{-1}, and let $\mathbf{c}$ denote the center of the segment, we have the following inclusion property:

$$\omega^{-1}(\mathbf{e}) \subset \mathbf{S}(\omega^{-1}(\mathbf{c}), \lambda\|\mathbf{e}\|/2)$$

Thus, we can compute an upper bound by traversing the warping node and bounding the norm of the gradient over the sphere:

$$\sup\|\nabla f_{\mathcal{N}} \circ \omega^{-1}(\mathbf{e})\| \leq \sup\|\nabla f_{\mathcal{N}}(\mathbf{S})\|$$

Jacobian Recall that the Euclidean norm of a matrix $\mathbf{A}$ can be defined as the square root of the spectral radius of $\mathbf{A}\mathbf{A}^{\star}$, where $\mathbf{A}^{\star}$ denotes the transpose of the matrix. Thus, we compute the norm of the Jacobian:

$$\|\mathbf{J}_{\omega^{-1}}\| = \sqrt{\rho(\mathbf{J}_{\omega^{-1}}\mathbf{J}^{\star}_{\omega^{-1}})}$$

Closed form expressions can be computed for some kinds of deformations, in particular tapering [Bar84] (see Figure 9 and Appendix for details). In the general case however, the spectral norm may be difficult to compute. For vector spaces of finite dimension, all norms are equivalent, and in particular $\|\mathbf{J}_{\omega^{-1}}\| \leq \|\mathbf{J}_{\omega^{-1}}\|_F$ where $\|\mathbf{J}_{\omega^{-1}}\|_F$ denotes the Frobenius norm, defined as the squared root of all squared matrix elements. Therefore, in those cases, we rely on a less accurate but easier to compute Frobenius norm instead.

5. Results and discussion

We implemented the implicit surface model and our algorithm in C++. All examples in this paper were created on a desktop computer equipped with Intel® Core *i*7, clocked at 3 GHz with 16 GB of RAM. A standalone application using graphics hardware was coded using OpenGL 4.3 and timings clocked on an nVidia® GeForce 2070.

5.1. Performance

We compared our Segment Tracing algorithm to Sphere Tracing on different types of models (Figure 10). Table 1 and Table 2 report the corresponding number of field function queries $\#f$ and the rendering time respectively for primary ray-object intersection computation. 512×512 images were rendered on the CPU without any parallel implementation.

In each table, we report the statistics for the standard Sphere Tracing algorithm (global λ) and three variations of our method. *Segment* is the full Segment Tracing algorithm with the most accurate Lipschitz bound $\lambda(\mathbf{e})$ evaluation. *Sphere* computes a less accurate bound $\lambda(\mathbf{S})$ associated to the embedding sphere of segment $\mathbf{e}$ and evaluates this bound for every increment. This algorithm can

Object	Sphere Tracing	Ours		
		Segment	Sphere	Ray
Chandelier	22.87	0.09	0.22	0.67
Molecule	463.06	0.26	1.61	18.52
Fluid	704.13	0.16	1.25	48.11
Temple	19.74	0.33	2.88	3.25
Geode	11.44	0.31	1.68	1.39
King	1.30	0.17	0.26	0.30
Mechanic	7.73	0.27	0.28	0.50
Quadric	9.34	0.23	1.17	3.28
Cluster	461.31	0.12	0.74	4.06
Amphora	0.83	0.09	0.14	0.26

Table 1: *Number of field function queries $\#f$ in millions for different objects (primary rays).*

Object	Sphere Tracing	Ours		
		Segment	Sphere	Ray
Chandelier	0.85	0.05	0.05	0.06
Molecule	35.42	1.43	1.14	2.20
Fluid	32.93	5.47	5.82	5.08
Temple	2.04	0.39	0.86	0.49
Geode	0.89	0.19	0.67	0.21
King	0.09	0.07	0.05	0.04
Mechanic	0.61	0.10	0.08	0.07
Cluster	23.18	0.25	0.29	0.44
Quadric	1.35	0.14	0.40	0.65
Amphora	0.25	0.09	0.08	0.11

Table 2: *Rendering time (in seconds) for ray tracing different models (primary rays).*

lead to better computation times because the intersection tests are easier to compute, while keeping a local Lipschitz bound. Finally, *Ray* evaluates the Lipschitz bound along the entire ray $\lambda(\Delta)$ and then performs standard Sphere Tracing tracing using this bound (Figure 12). In our framework, this is equivalent to computing $\lambda(\mathbf{e})$ with an infinite segment (see Section 3) and in general $\lambda(\mathbf{e}) \ll \lambda$ where λ is the Lipschitz bound of the entire object. This simplified version of our method can be efficient for specific configurations.

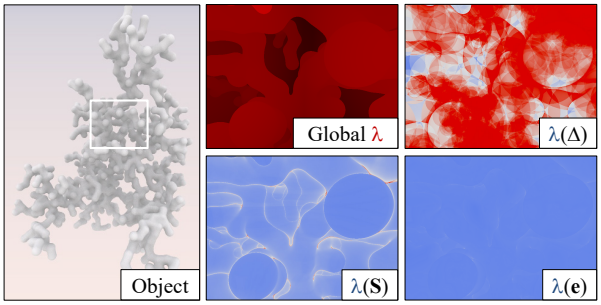


Figure 12: *Number of field function evaluation for Sphere Tracing and our method using different local Lipschitz bound computation.*

Number of field function queries The first observation is that Segment Tracing always yields the smallest number of field function queries for primary ray-object intersections. This is the direct consequence of the local Lipschitz bound computation, which optimizes the marching step, and thus reduces the number of field function queries $\#f$ by a factor depending on the number, complexity and spatial distribution of primitives. Table 1 shows that $\#f$ can be reduced by one to two order of magnitudes for models composed of a few dozens of complex primitives, such as the *Chandelier* or the *Temple*. The number of field function queries is even further reduced for large models featuring thousands of simple primitives, as exemplified by the *Molecule* or *Cluster* models.

Secondary ray-object intersections used for global illumination, shadow computation or effects such as reflection or refraction, are also accelerated by our method. We observed similar acceleration factors for secondary rays which demonstrates the effectiveness of the local Lipschitz bound computation combined with the acceleration coefficient κ used for updating the candidate stepping length.

Timings Any variation of our method, *i.e. Segment*, *Ray* or *Sphere*, improves rendering time compared to Sphere Tracing. The largest gains are observed either for complex objects composed of thousands of primitives (*Cluster*, *Molecule*, *Fluid*) or created with complex primitives with computationally demanding field functions, for instance cubic or quadric curve primitives, or volumetric primitives such as cylinders or cones.

Reducing the number of field function queries does not necessarily improve computation time. The accurate Lipschitz bound computation comes at the price of more computationally intensive queries, in particular the overhead introduced by the evaluation of $\lambda(\mathbf{e})$. The *Ray* algorithm requires one Lipschitz bound $\lambda(\Delta)$ computation for the entire ray, and multiple field function evaluations (at every step), whereas the *Sphere* and *Segment* algorithms require multiple queries to evaluate $f(\mathbf{p})$ and $\lambda(\mathbf{e})$ at every step. Experiments show that when the implicit objects have an almost uniform distribution of primitives and a uniform Lipschitz bound over their support Ω, the benefit is limited or negative in terms of speed (*Chess king*, *Amphora*, or *Mechanic*). In those cases, it is preferable to employ the *Ray* or *Sphere* versions of our algorithms.

Graphics hardware implementation We implemented and compared our algorithm to the classical Sphere Tracing on the GPU (see accompanying video). The acceleration factor, close to an order of magnitude, is similar to the CPU version: about 6 Hz for the standard Sphere Tracing versus 67 Hz for our Segment Tracing method. We did not focus on the optimization of the shader which was beyond the scope of this study: more primitives and operators could be adapted and optimized, allowing for interactive rendering of more complex objects.

5.2. Comparison with other methods

The computation of the ray-implicit surface intersections is penalized by the regions of space where the gradient magnitude is the greatest. Accelerating data structures, such as regular voxel grids or adaptive octrees, can speed-up intersections computation, at the expense of a pre processing step and an increased storage cost.

Octrees We compared our Segment Tracing algorithm to Sphere Tracing combined with an octree as described by [Har96]. The cells $\mathbf{C}$ store a local Lipschitz bound which is computed using the query $\lambda(\mathbf{S})$ where $\mathbf{S} \supset \mathbf{C}$ denotes the bounding sphere of the cubic cell. Cells that do not straddle the implicit surface, *i.e.* empty or completely inside the object, are detected by using the Lipschitz criterion, which further accelerates the ray-surface intersection computation. Figure 13 reports the relative performance between our method and an accelerating octree structure for several implicit models.

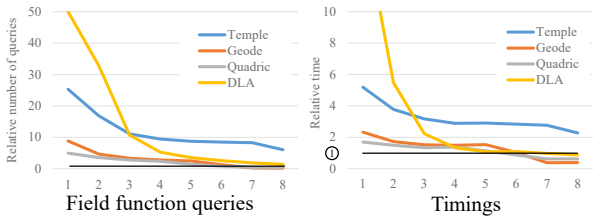


Figure 13: *Relative comparison between our method and Sphere Tracing optimized with an octree storing the Lipschitz bounds in its cells; statistics are reported as a function of the octree depth, and the black line refers to our method.*

Experiments show that octrees perform better than Segment Tracing if their depth δ is larger than 7, which corresponds to an average extra memory cost of 1 to 4 Mb depending on the geometry of the implicit model. For dense models with complex primitives combined together such as the *Temple*, the octree becomes competitive only for depths $\delta \geq 9$. This can be explained as the $\lambda(\mathbf{e})$ takes into account the direction of the segment and is more accurate than the bounds $\lambda(\mathbf{S})$ stored in the cells. For sparse models such as the *Chandelier*, the octree prunes empty space and generally perform slightly faster than our method, even at octree depths $\delta \approx 5$. Moreover, field function queries are performed for all the intersections between the ray Δ and the cells, whereas the Segment Tracing algorithm is not affected by this phenomenon. In contrast, our method preserves the compact format of hierarchical implicit surface models and compares favorably in terms of memory as the accelerating data structure becomes memory demanding.

Binary search methods recursively subdivide an interval, *i.e.*, a segment, into two sub-intervals until either an intersection is detected, or a rejection criterion is satisfied. A complete comparison between Lipschitz techniques and interval analysis using Interval Affine [Mit90] or Affine Arithmetic [GM07] is beyond the scope of this paper. Still, binary search methods clearly benefit from our local Lipschitz bound to exclude intervals earlier in the recursive bisection process. Let $\mathbf{e} = [\mathbf{a}, \mathbf{b}]$ a segment and $\mathbf{c}$ its center, $\mathbf{e}$ does not intersect the surface if $|f(\mathbf{c})|/\lambda(\mathbf{e}) > \|\mathbf{e}\|$.

Moreover, the computation of $\lambda(\mathbf{e})$ can improve bisection by discarding some parts of the interval where the Lipschitz criterion guarantees that no intersection should occur. Instead of cutting the segment $\mathbf{e}$ in two equal parts, we shrink the interval $\mathbf{e}$ into $[\widetilde{\mathbf{a}}, \widetilde{\mathbf{b}}]$ where $\widetilde{\mathbf{a}} = \mathbf{a} + |f(\mathbf{a})|/\lambda(\mathbf{e})$ and $\widetilde{\mathbf{b}} = \mathbf{b} - |f(\mathbf{b})|/\lambda(\mathbf{e})$ (Figure 14). We implemented this optimized binary search using local Lipschitz bounds with improved bisection. We observed similar field function query reduction as for Segment Tracing, and the corresponding timing speed-ups, when compared to standard binary search, which demonstrates the effectiveness of our accurate Lipschitz bound computation.

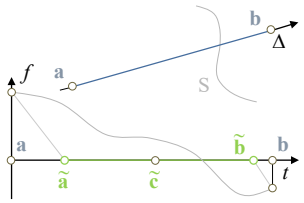


Figure 14: *Optimized bisection with refined sub-intervals using the local Lipschitz bound* $\lambda(\mathbf{e})$.

Enhanced Sphere Tracing method [KSK*14] consists in over stepping along the ray slightly farther than the safe stepping distance $o(\mathbf{p}) = (1+\sigma)|f(\mathbf{p})|/\lambda$, where $\sigma \in [0,1]$ represents the over-stepping factor (Figure 15), checking that the spheres of two consecutive marching steps still overlap, and with possible failure cases defaulting to conventional Sphere Tracing.

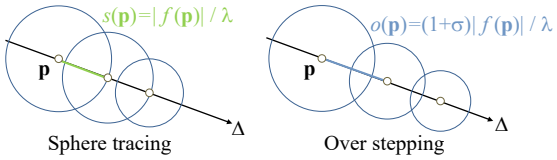


Figure 15: *Over stepping is a heuristic that increases the safe stepping distance by a fixed factor* σ.

In practice, $\sigma \approx 1.2$ which consequently reduces the number of field function queries by $\times 1.2$, and directly speeds-up computation time by the same factor. In contrast, our method performs an accurate evaluation of the Lipschitz bound along the candidate segment, which reduces the number of field functions queries by up to three order or magnitude, which in turn result in accelerations of up to $\times 10$ because of the extra computational cost of Lipschitz bound queries. Over stepping is not directly compatible with our approach as it is strongly based on the definition of a global bound λ.

5.3. Discussion

Computing the local Lipschitz bound allows to adapt the marching distance $s(t, \varepsilon)$ according to the field value $f(\mathbf{p})$ and the bound of the norm of the local gradient. It is no surprise that the number of field function queries $\#f$ should decrease using our method: we take advantage of some important information about the local behavior of f and, more importantly, its variation in the direction of the ray. For complex field functions or models composed of many primitives, timings demonstrate that the extra effort needed to evaluate the local Lipschitz bound is negligible compared to the acceleration provided by larger steps.

Those improvements come at the expense of a prior analysis of the primitives and operators however. While Sphere Tracing only necessitates the implementation of the field function f and a global Lipschitz bound (which is often guessed and prescribed by the user), our method requires that every node should implement several complex queries, in particular the computation of the Lipschitz bound over a segment $\lambda(\mathbf{e})$, and over a sphere $\lambda(\mathbf{S})$.

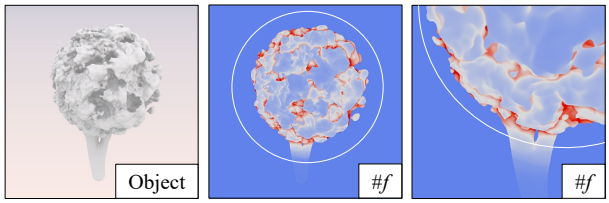


Figure 16: *Example of an object composed of primitives implementing local Lipschitz bound queries, and non optimized primitives. Our method can accelerate ray tracing wherever possible, and defaults to slower marching for the parts of the object where only a global Lipschitz bound is known (outlined as a disc).*

Still, our algorithm is general and easy to extend with more primitives and operators. Moreover, our method automatically reverts to the standard Sphere Tracing framework locally should the computation of local bounds be too complex or computationally intensive for a given operator or primitive.

Figure 16 illustrates this property by showing an object composed of some curve primitives implementing an optimized Lipschitz bound query, and a high frequency noise-based primitive for which the local queries $\lambda(\mathbf{e})$ and $\lambda(\mathbf{S})$ were only approximated using a coarse global bound over its region of influence.

6. Conclusion

We have presented Segment Tracing, an efficient algorithm for ray tracing hierarchical skeletal implicit surfaces. By computing the local Lipschitz bound along the ray, we adapt the marching distance to the scalar field and its local gradient. Our algorithm improves Sphere Tracing, significantly reduces the number of field function queries, speeds up computations, without the need for any accelerating data structure such as voxel grids or octrees for storing the local Lipschitz bound. We also demonstrated that our method can be implemented on graphics hardware, at the expense of the implementation of extra functions, i.e. the computation the Lipschitz bound over a segment, a ray or a spherical domain, that increase the overall complexity of the shader.

Accelerating the direct rendering of compact procedurally defined implicit surfaces opens several interesting research directions for the future. Better algorithms for performing all the fundamental queries and improving the performance of the different kinds of primitives, blending and warping operators, would be worth investigating. Another avenue for future work consists in enriching the implicit surface model with new primitives and deformations operators that would lend themselves for efficient local field function and gradient computations. Finally, a more complete GPU implementation of the method would allow for interactive modeling of complex scenes featuring many primitives of different types.

References

[Bar84] BARR A. H.: Global and local deformations of solid primitives. *SIGGRAPH Computer Graphics 18*, 3 (1984), 21–30. 6

[BG04] BARBIER A., GALIN E.: Fast distance computation between a point and cylinders, cones, line swept spheres and cone-spheres. *Journal of Graphic Tools 9*, 2 (2004), 31–39. 4

[BW97] BLOOMENTHAL J., WYVILL B. (Eds.): *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. 2

[DALJ*15] DE ARAÚJO B. R., LOPES D. S., JEPP P., JORGE J. A., WYVILL B.: A survey on implicit surface polygonization. *ACM Computing Surveys 47*, 4 (2015), 60:1–60:39. 1

[GGP*15] GÉNEVAUX J.-D., GALIN É., PEYTAVIE A., GUÉRIN É., BRIQUET C., GROSBELLET F., BENES B.: Terrain modeling from feature primitives. *Computer Graphics Forum 34*, 6 (2015), 198–210. 2

[GM07] GAMITO M. N., MADDOCK S. C.: Ray casting implicit fractal surfaces with reduced affine arithmetic. *The Visual Computer 23*, 3 (2007), 155–165. 2, 8

[GPP*10] GOURMEL O., PAJOT A., PAULIN M., BARTHE L., POULIN P.: Fitted BVH for Fast Raytracing of Metaballs. *Computer Graphics Forum 29*, 2 (2010), 7–288. 2

[Han83] HANRAHAN P.: Ray tracing algebraic surfaces. *SIGGRAPH Computer Graphics 17*, 3 (1983), 83–90. 2

[Har96] HART J. C.: Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer 12*, 10 (1996), 527–545. 1, 2, 5, 8

[HSK89] HART J. C., SANDIN D. J., KAUFFMAN L. H.: Ray Tracing Deterministic 3D Fractals. *SIGGRAPH Computer Graphics 23*, 3 (1989), 289–296. 2

[KB89] KALRA D., BARR A. H.: Guaranteed Ray Intersections with Implicit Surfaces. *SIGGRAPH Computer Graphics 23*, 3 (1989), 297–306. 2

[KHK*09] KNOLL A., HIJAZI Y., KENSLER A., SCHOTT M., HANSEN C., HAGEN H.: Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. *Computer Graphics Forum 28*, 1 (2009), 26–40. 2

[KSK*14] KEINERT B., SCHÄFER H., KORNDÖRFER J., GANSE U., STAMMINGER M.: Enhanced sphere tracing. In *Proceedings of Smart Tools & Apps for Graphics* (Cagliari, Italy, 2014), Eurographics Association. 2, 8

[KSN08] KANAMORI Y., SZEGO Z., NISHITA T.: GPU-based fast ray casting for a large number of metaballs. *Computer Graphics Forum 27*, 2 (2008), 351–360. 2

[LB06] LOOP C., BLINN J.: Real-time GPU rendering of piecewise algebraic surfaces. *ACM Transactions on Graphics 25*, 3 (2006), 664–670. 2

[Mit90] MITCHELL D. P.: Robust ray intersection with interval arithmetic. In *Proceedings on Graphics Interface* (Toronto, Canada, 1990), Canadian Information Processing Society, pp. 68–74. 2, 8

[NN94] NISHITA T., NAKAMAE E.: A method for displaying metaballs by using Bézier clipping. *Computer Graphics Forum 13*, 3 (1994), 271–280. 2

[PASS95] PASKO A., ADZHIEV V., SOURIN A., SAVCHENKO V.: Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8 (1995), 429–446. 1, 4, 5

[Per89] PERLIN K.: Hypertexture. *SIGGRAPH Computer Graphics 23*, 3 (1989), 253–262. 2

[RMD11] REINER T., MÜCKL G., DACHSBACHER C.: Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions. *Computer & Graphics, Proceedings of Shape Modeling International 35*, 3 (2011), 596–603. 1, 4

[She99] SHERSTYUK A.: Fast ray tracing of implicit surfaces. *Computer Graphics Forum 18*, 2 (1999), 139–147. 2

[SJNJ19] SEYB D., JACOBSON A., NOWROUZEZAHRAI D., JAROSZ W.: Non-linear sphere tracing for rendering deformed signed distance fields. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia) 38*, 6 (2019). 1, 2, 6

[SM09] SHIRLEY P., MARSCHNER S.: *Fundamentals of Computer Graphics*, 3rd ed. A. K. Peters, Ltd., USA, 2009. 4

[SN10] SINGH J. M., NARAYANAN P. J.: Real-time ray tracing of implicit surfaces on the GPU. *IEEE Transactions on Visualization and Computer Graphics 16*, 2 (2010), 261–272. 2

[WGG99] WYVILL B., GUY A., GALIN E.: Extending the CSG tree – warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (1999), 149–158. 1, 4, 5

[Wij85] WIJK J. V.: Ray tracing objects defined by sweeping a sphere. *Computers and Graphics 9*, 3 (1985), 283–290. 2

[WMW86] WYVILL G., MCPHEETERS C., WYVILL B.: Data structure for soft objects. *The Visual Computer 2*, 4 (1986), 227–234. 5, 10

[WT90] WYVILL G., TROTMAN A.: Ray-tracing soft objects. In *Computer Graphics International* (1990), Springer Japan, pp. 469–476. 2

Appendix A: Falloff filter function analysis

The evaluation of the Lipschitz bound for skeletal primitives requires the evaluation of the maximum of the absolute value of the derivative of the function $g'(x)$ on an interval $[a,b]$. This involves finding the zeroes of the second-order derivative $g''(x) = 0$. In their normalized expression, falloff filter functions are smoothly decreasing (monotonic) functions $g : [0,1] \to [0,1]$ such that $g''(x) = 0$ has only one root, denoted as x_0. The bound of $g'([a,b])$ is computed as follows:

$$\begin{cases} \lambda = 0 & \text{if } a \geq 1 \\ \lambda = |g'(x_0)| & \text{if } x_0 \in [a,b] \\ \lambda = \max(|g'(a)|, |g'(b)|) & \text{otherwise} \end{cases}$$

Wyvill function. The C^2 falloff filter function is defined as:

$$g(x) = (1-x^2)^3 \text{ if } x < 1 \qquad 0 \text{ otherwise}$$

Derivatives are:

$$g'(x) = -6x(1-x^2)^2 \qquad g''(x) = -6(1-x^2)(1-5x^2)$$

The second-order derivative has one positive vanishing value $x_0 = 1/\sqrt{5}$ in unit interval $[0,1]$. The maximum absolute value of $g'(x)$ is obtained by evaluating $g'\left(1/\sqrt{5}\right)$, thus $\lambda_0 = 96\sqrt{5}/125 \leq 1.72$.

Quadric function. The quadric C^1 function is defined as:

$$g(x) = (1-x^2)^2 \text{ if } x < 1 \qquad 0 \text{ otherwise}$$

Derivatives are:

$$g'(x) = -4x(1-x^2) \qquad g''(x) = -4(1-3x^2)$$

The second-order derivative has one vanishing value $x_0 = 1/\sqrt{3}$ in $[0,1]$, and the maximum absolute value is $\lambda_0 = 8\sqrt{3}/9 \leq 1.54$.

The falloff filter function for Blobs described in [WMW86] is built from the previous quadric and modified to satisfy the boundary conditions $g'(0) = g'(1) = 0$ and $g(1/2) = 1/2$:

$$g(x) = -4/9x^6 + 17/9x^4 - 22/9x^2 + 1 = (1-x^2)^2(9-4x^2)/9$$

Derivatives are:

$$g'(x) = -4/9x(6x^4 - 17x^2 + 11) = -4/9x(x^2-1)(6x^2-11)$$

$$g''(x) = -4/9(30x^4 - 51x^2 + 11)$$

The second-order derivative has one root in $[0,1]$: $x_0 = \sqrt{51 - \sqrt{1281}}/2\sqrt{15}$, and the maximum derivative is:

$$\lambda_0 = (17\sqrt{1281} + 453)\sqrt{17/2278125 - \sqrt{1281}/6834375} \leq 1.59$$

Appendix B: Bound of the norm of the Jacobian

Tapering can be defined as the transformation $\omega^{-1}(\mathbf{p}) = (\alpha(z)x, \alpha(z)y, z)$ where $\alpha(z)$ defines the tapering coefficient along the vertical axis. Let $\beta = \alpha'$, the Jacobian matrix is symmetric and defined as:

$$\mathbf{J}_{\omega^{-1}} = \begin{pmatrix} \alpha & 0 & \beta \\ 0 & \alpha & 0 \\ \beta & 0 & 1 \end{pmatrix}$$

The roots of the characteristic polynomial are α^2 and $1/2(\alpha^2 + 2\beta^2 + 1 \pm (1+\alpha)\sqrt{\alpha^2 - 2\alpha + 4\beta^2 + 1})$. Since $\beta^2 \geq 0$, the largest root, thus the Euclidean norm, is:

$$\|\mathbf{J}_{\omega^{-1}}\| = 1/\sqrt{2}\sqrt{\alpha^2 + 2\beta^2 + 1 + (1+\alpha)\sqrt{\alpha^2 - 2\alpha + 4\beta^2 + 1}}$$

Thus for a given input interval $\mathbf{e}$, $\|\mathbf{J}_{\omega^{-1}}(\mathbf{e})\|$ can be obtained by developing the equation or by interval analysis.