Volume xx (200y), Number z, pp. 1–19

Fast ray tracing of scale-invariant integral surfaces

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

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

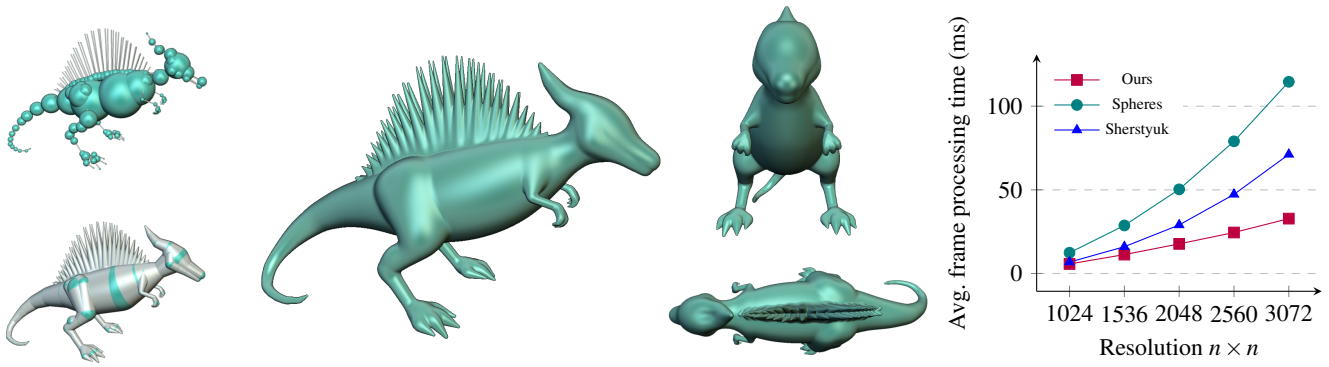


Figure 1: *Scale-invariant integral surfaces provide a way to define smooth surfaces from skeletons with prescribed radii defined at their vertices (with linearly interpolated radii along the skeleton edges). The generated surface can be seen as a smoothed version of an infinite union of spheres centered on the skeleton edges. We propose a new rendering pipeline allowing to visualize such surfaces in real-time. We provide comparison to revisited state of the art techniques on a large range of skeleton types for a variety of GPUs. Our method provides improvements for various resolutions on all combination of tested models and GPU hardware (see right graph. More comparisons in the results section).*

Abstract

Scale-invariant integral surfaces, which are implicit representations of surfaces, provide a way to define smooth surfaces from skeletons with prescribed radii defined at their vertices. We introduce a new rendering pipeline allowing to visualize such surfaces in real-time. We rely on the distance to skeleton to define a sampling strategy along the camera rays, dividing each ray into sub-intervals. The proposed strategy is chosen to capture main field variations. Resulting intervals are processed iteratively, relying on two main ingredients; quadratic interpolation and field mapping, to an approximate squared homothetic distance. The first provides efficient root finding while the second increases the precision of the interpolation, and the combination of both results in an efficient processing routine. Finally, we present a GPU implementation that relies on a dynamic data-structure in order to efficiently generate the intervals along the ray. This data-structure also serves as an acceleration structure that allows constant time access to the primitives of interest during the processing of a given ray.

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing; Volumetric models;***

1. Introduction

Implicit surfaces are well known for their capacity to represent smooth shapes with arbitrary topology [Blo97] and provide a well-defined volume for applications such as additive manufacturing. Convolution surfaces [BS91] combine both implicit surfaces and skeletal representations. Skeleton-based representation combine well with implicit surfaces and provide several advantages for manipulating shapes during modeling and processing tasks [TDS*16]. Such representations allow reasoning on the shape structure and volume, for instance, to define volumes with minimal wall or feature thickness for robust fabrications or volume preservation during deformation [LYHG17; ARM*19]. Skeletons also provide direct control over the volume through the skeleton's vertices yielding rapid volume sketching capabilities [PIX10; JLW10]. The Scale-Invariant Integral Surface representation (SCALIS) [ZBQC13] extends convolution surface formulation to provide features such as precise radius and blending control at different scales.

One of the main difficulties when working with implicit surfaces is to provide efficient visualization. During modeling and final rendering (i.e., processing), visualization may have different requirements in terms of time and approximation. For interactive modeling it is best to have a method requiring no pre-computations and no transformations of the native representation. Provided that short enough computational times can be reached to achieve an acceptable frame rate, ray tracing provides several advantages over meshing. In addition to being trivially parallelizable, ray tracing computations are done in the image space therefore limiting the computations to the parts of the object that are actually visible. This also opens the opportunity to develop output sensitive methods while offering high quality rendering.

When ray tracing implicit surfaces the first challenge is to obtain a robust computation of the closest ray/isosurface intersection along the ray. If we target integral surfaces, such as SCALIS, two additional problems have to be tackled. First, in order to perform field evaluation, one needs to determine the primitives that influence a given location in space. For large skeletons with many primitives, searching through the primitives to find the ones that influence a given location is the main bottleneck. Second, since the SCALIS field evaluations for individual primitives are relatively expensive to compute, a brute force approach requiring a large number of field evaluations is not an option for any interactive application.

In order to overcome these challenges and provide a fast and precise ray-tracing-based integral surface visualization, our strategy combines four main components; 1) We introduce, during rendering, a segmentation of rays in intervals that captures main field variations while generating only a small number of sub-intervals to be processed. 2) We transform field values to a space that allows good quadratic polynomial approximation of the SCALIS field on each of the sub-intervals defined in the previous step along the ray. 3) Fast quadratic roots computation allows to iteratively build polynomial interpolations that rapidly converge toward the real field function. And 4) in order to limit computational overhead, selecting primitives whose supports overlap a given interval along the ray is performed using a dynamic view-dependent acceleration structure. The last component is built on the fly relying on fast GPU construction of A–buffers which have been previously employed for metaballs rendering [Bru19] and transparency [Thi11; MCTB11; MCTB12].

This results in a method that goes beyond Sherstyuk's fast ray-tracing algorithm for convolution surfaces [She99], notably allowing a better support of large radius variation while being less sensitive to skeleton tessellation.

1.1. Background: Implicit Modeling

In this section, we present required background on skeleton-based implicit modeling and put forth the notations that will be used throughout the paper (for a more detailed introduction to implicit modeling, see [Blo97]).

An implicit surface is defined by a scalar field f and an isovalue c as the set of points $\mathbf{p}$ in space satisfying the equation $f(\mathbf{p}) = c$.

Figure 2: Left: *a skeleton with prescribed radius on its vertices,* Middle: *infinite union of spheres defined by linear interpolation of prescribed radius along skeleton edges (this is also a union of sphere-cone),* Right: *resulting scale-invariant integral surface which can be seen as a smoothing of the sphere-cone union.*

Different surfaces can be blended by combining their fields' contributions. Such combinations are often performed hierarchically in a Blobtree data structure [WGG99], where each node of the tree represents a composition operator and each leaf is an implicit primitive.

Among implicit surface definitions there exist several conventions on the meaning of the field values. On one hand we have the functional representation F-Rep [PASS95] usually close to a *signed-distance field* where a volume is defined by $f(\mathbf{p}) \leq 0$ and where the field f tends toward infinity when moving away from the surface. On the other hand we have *density field* such as metaball-like surfaces [Bli82; WMW86; NHK*85] defining a volume by $f(\mathbf{p}) \geq c$ (usually with $c = 1/2$ or $c = 1$) and where field value tend toward 0 away from the object. Given a decreasing function k called a *kernel*, the simplest density fields can be defined by applying k to the distance to a point primitive. Such fields can then be combined with composition operators. The most simple smooth blending operator is the *sum*, used to blend an arbitrary number of input primitives:

$$f(\mathbf{p}) = \sum_i f_i(\mathbf{p}) \tag{1}$$

where f_i is the field function defined by the i-th primitive. Among kernel families used in practice, we can note the *Compact Polynomial* kernels that are defined as:

$$k_{i,\sigma}(d) = \begin{cases} \left(1 - \left(\frac{d}{\sigma}\right)^2\right)^{\frac{i}{2}} & \text{if } d < \sigma\,, \\ 0 \text{ otherwise.} \end{cases} \tag{2}$$

and the *Cauchy* kernels which can be defined as:

$$k_{i,\sigma}(d) = \frac{1}{\left(1 + \left(\frac{d}{\sigma}\right)^2\right)^{\frac{i}{2}}} \tag{3}$$

They form families of kernels parametrized by a degree parameter i (defining the smoothness of the surface for the compact polynomial kernel) and a scale parameter σ (defining the extent of blends during composition).

Skeleton-based implicit modeling In order to model more complex shapes other primitives beyond points can be used, for instance

segments, triangles or Bézier curves. Such primitives can by organized in a skeleton structure in order to facilitate their edition. In this work, we focus on segment primitives.

We define a skeleton as a set of line segment primitives S_i with prescribed radius information at the endpoints (or vertices - see Figure 2, left). For each point $\mathbf{q}$ on the line segment, a radius $\tau_{S_i}(\mathbf{q})$ can be defined as the linear interpolation of the radii of the two endpoints. Hence, for each such line segment primitive S_i, a *sphere-cone* can be defined as the infinite union of balls defined by the skeleton points $\mathbf{q}$ belonging to the segment with associated radius $\tau_{S_i}(\mathbf{q})$ (see Figure 2, middle).

We present here all the implicit primitives that will be discussed in the remainder of the paper; all these primitives are defined in terms of distance between a point $\mathbf{p}$ in space to a point $\mathbf{q}$ on the line segment:

$$d(\mathbf{p},\mathbf{q}) = \|\mathbf{p}-\mathbf{q}\| \tag{4}$$

which, divided with the radius $\tau_{S_i}(\mathbf{q})$ at the skeleton point $\mathbf{q}$, defines the *homothetic distance* to a single skeleton point:

$$h(\mathbf{p},\mathbf{q}) = \frac{d(\mathbf{p},\mathbf{q})}{\tau_{S_i}(\mathbf{q})} \tag{5}$$

or to line segment primitives :

$$h_{S_i}(\mathbf{p}) = \min_{\mathbf{q}\in S_i} h(\mathbf{p},\mathbf{q}) \tag{6}$$

Given a kernel k, this allows to define a density function for the line segment S_i:

$$f_i(\mathbf{p}) = \max_{\mathbf{q}\in S_i} k \circ h_{S_i}(\mathbf{p},\mathbf{q}) \tag{7}$$

with $\circ$, the composition operator.

When using such segment primitives with summation blend, bulging appears at skeleton junctions. This problem can be resolved by using integral surfaces, such as a convolution surface [BS91] or SCALIS primitive [ZBQC13]. Both formulations provide an independence from skeleton subdivision when using blending by summation thanks to the additivity property of the integral. Note that both representations are equivalent for constant unit radius. The SCALIS field, main focus of this paper, is defined as:

$$f_i(\mathbf{p}) = \frac{1}{N_{k,c}} \int_{S_i} \frac{k \circ h(\mathbf{p},\mathbf{q})}{\tau_{S_i}(\mathbf{q})}\, \mathrm{d}\mathbf{q} \tag{8}$$

where the normalization factor $N_{k,c}$, which depends on the chosen isovalue c and kernel k, is used in order to achieve the prescribed radius around the skeleton. Visually, the resulting isosurface appears as a smoothing of the sphere-cones associated to the prescribed radii (see Figure 2, right).

In addition to a direct radius control, this formulation simplifies modeling by providing scale-invariance properties: scaling both the skeleton geometry and the associated radii with a factor s results in a scaling of the isosurface of interest by the same factor - e.g. blending behavior is independent from the scale. This scale property also reduces blurring of details when blended in larger shapes.

The method developed in this paper holds for both Equation (7) and (8) with a summation blend.

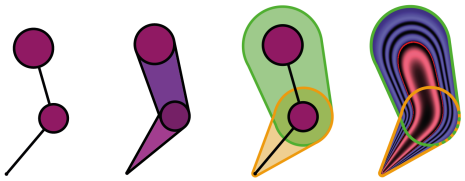
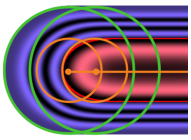


Figure 3: Leftmost: *line segment primitives with prescribed radius at vertices.* Left: *associated sphere-cones.* Right: *Primitive supports when using a compact kernel. The supports are also sphere-cones and their radii are proportional to the radii τ_{S_i} prescribed on the skeleton.* Rightmost: *a slice of the SCALIS scalar field, the points that are inside of the volume defined by $f(\mathbf{p}) > c$ are depicted in red, the points outside are depicted in blue. The points outside of the supports of all primitives have a null field value and are depicted in white.*

Kernels The main kernel families used to define skeleton-based implicit surfaces are Cauchy, Inverse and Compact Polynomial kernels, all parametrized by a degree parameter i and a scale parameter σ (see Equations (2,3)). We mostly focus on the *Compact Polynomial* kernel due to its desirable properties: local support - required for efficient field evaluation on large skeletons - and efficient closed form evaluation of Equation (8) for line segment primitives (with linearly interpolated prescribed thickness τ). In this case, the support of an individual primitive (e.g. the volume for which the field value is not zero) is also a sphere-cone that is defined by scaling the two primitive endpoints' radius by a factor σ (see Figure 3 which also describes the convention for the display of the fields). The Inverse kernel can be seen as a special case of the Cauchy Kernel when the scale parameter σ tends toward 0.

End-point corrector Due to the definition of the field as an integral, see Equation (8), the prescribed radius is not achieved in the tangential direction near skeleton end-points. In [ZBQC13], a simple correction scheme is presented to overcome this drawback. It consists in adding a segment primitive with constant radius, the length of which is proportional to the prescribed radius at the end vertices of the skeleton as well as on a constant depending on the kernel used (see Figure inset). In the remainder of the paper, we will discuss the implication of those *end-point correctors* or their absence. In some cases, such as a truss structure without end-points, all the assumptions related to the presence of end-point correctors can be used.



2. Previous work: Raytracing implicit surfaces

Rendering implicit surfaces is a subtle trade-off between efficiency and accuracy and has triggered a lot of research. In order to visualize an implicit surface with the ray tracing approach, one has to find the first parameter along a given ray:

$$\mathbf{r}(x) = \mathbf{o} + x\mathbf{d} \tag{9}$$

for which the field value evaluates to the isovalue of interest c, e.g. finding x such that :

$$f \circ \mathbf{r}(x) = c$$

or equivalently :

$$f \circ \mathbf{r}(x) - c = 0 \tag{10}$$

Henceforth, we only consider Equation (10) and discuss the detection of zero crossings along the ray.

A first solution to do so it to apply the ray marching approach to isolate the root by advancing with a constant step size until a sign change is detected in $f \circ \mathbf{r} - c$. Then, applying another method, such as a constrained Newton method, to localize the root more precisely. The behavior of such method is highly correlated to the choosen stepsize: for small steps the method becomes computationally expensive, for larger step size, some isosurface crossing can be missed.

Available techniques tend to differ depending on the properties of the implicit surface to be ray-traced, for instance density field versus distance field.

Polynomial definitions/approximations Several ray tracing algorithms rely on polynomial root finding algorithms to locate the isosurface along the ray, either because the field itself is defined by polynomials or because a polynomial approximation of the field is built along the ray.

For point primitives defined with the compact polynomial kernel (Equation (2)), the field function along the ray is a piecewise polynomial. For small degree (up to degree 4), this property can be used to compute a closed-form expression of the root [GPP*10]. For higher degrees, the Bézier clipping algorithm can be used [NN94; KSN08] to iteratively converge toward the first root of the polynomial (or to rapidly reject root-free intervals).

For convolution surfaces, polynomial approximations of the field value can be built along the ray [She99]. During rendering, each primitive can be approximated by one or several polynomials on the interval defined by the intersection between the ray and the primitive support. The interval is uniformly subdivided and polynomial approximations are calculated from Hermite data (field value and derivative) sampled at subinterval boundaries (see Figure 4). The roots of resulting cubic polynomial approximations can be computed analytically. When primitives are combined with the summation operator, new polynomials are defined by summation of polynomials on the intersection of primitive subintervals.

Choosing an adequate level of subdivision is problematic: on one hand, small number of intervals can result in poor approximation depending on both the viewpoint and the skeleton tessellation even more so in the context of SCALIS as it becomes more difficult to capture the maximal field contribution of a given primitive. On the other hand higher number of intervals results in higher computational time due to additional field evaluation. Furthermore, when performing computation on GPU, the incoherence of the generated subintervals require additional data management to avoid the computation of additional field values.

An alternative to interval subdivision is to rely on a higher degree approximation defined from additional sampling positions [She99; JTZ09]. However, such an approach prevents the use of closed form expressions and can also be subject to increased fitting error due to introduction of oscillations in a higher degree polynomial approximation due to Runge's phenomenon.

For arbitrary field definition, Taylor polynomial approximations were also used to perform root isolation [SWR18]. Adaptive intervals are defined along the ray using a gradient based heuristics, each interval is analyzed using two Taylor approximations of degree two defined from both end-points of the interval, intervals are subdivided until the Taylor approximations find a single root. A bounded Newton method is then used on the resulting interval. Such an approach requires the computation of a higher order derivative and a well chosen initial step-size (either user defined or based on global property of the Hessian) in order for the root isolation to be guaranteed. In our context the initial step size would be based on the smallest radius used in the scene, hence inefficient.

Lipschitz constant The principle of Lipschitz continuity was first used by [KB89] for guaranteed localization of the intersections between a ray and an implicit surface. The method uses Lipschitz bounds of both the field function and its gradient along the ray direction in order to build an octree space partitioning used to prune empty areas. The roots are then isolated by investigating change in the gradient and the sign of the field function. Such approach requires the existence and calculability of the first and the second derivatives of the field function.

The existence of a global Lipschitz constant for signed distance field has been used in [Har96] to define a robust ray-tracing algorithm : the sphere tracing. This algorithm belongs to the ray-marching family. Each step size is computed as a function of the current field value and global Lipschitz constant guaranteeing that the isosurface is never missed. This approach is widely used for direct content creation in shaders [QJ13]. A large number of extensions to this approach have been proposed, such as safe overrelax-

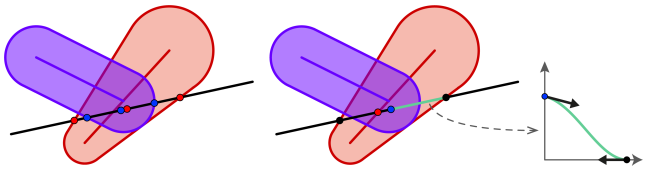


Figure 4: Left: *In [She99], boundary of primitive supports are used to subdivide the ray in order to compute local polynomial approximations of field values. The approximations are then used to compute an approximation of the isosurface position. To increase the approximation precision, supports are uniformly subdivided before generating the final segmentation which can result in intervals of highly incoherent size.* Middle: *our ray segmentation only relies on estimated maximal influence of individual primitives to define an initial segmentation with a small number of subintervals. We then rely on dynamic subdivision in order to increase precision of approximation wherever required.* Right: *Hermite data defined at interval's end-points and associated polynomial interpolation.*

ation [KSK*14], locally defined Lipschitz constant [GGP*15], optimal overrelaxation for planar surfaces [BV18] and visualization of implicit surfaces under derformation [SJNJ19]. Signed distance fields were also used to render geometry in games, either through sphere-tracing on mip-mapped voxelized data [Ord18] or generation of point cloud [Mol20].

The main limitation of this family of algorithms is the arbitrarily large number of steps required for grazing rays. When applied to a density field instead of a distance field, the algorithm also suffers from the shape of the kernel function (null gradient near kernel support boundary) and the difficulty to compute a tight Lipschitz bound for an N-ary summation operator. For instance if there are large differences of radius for SCALIS primitives, the Lipschitz constant per primitive is inversely proportional to the prescribed radius therefore creating unnecessarily small steps globally.

Such limitations have been tackled recently in the segment tracing algorithm [GGPP20], drastically decreasing the number of steps required for compactly supported primitives combined in a Blobtree [WGG99]. The key idea is to compute a local directional Lipschitz bound by using the fact that primitives are defined as a composition of a distance function and a kernel function. We discuss and compare our technique later in Section 8.

Another approach was used in [Bru19] to ray trace Blinn soft objects with constant radius, the sphere-tracing algorithm is applied after performing a mapping that allows to compute a signed distance approximation to the density field. Our method relies on the same type of field normalization which is discussed in more detail in the next section.

Interval/Affine arithmetics For arbitrary function-based field definition, a robust approach to ray tracing is to rely on iterative interval refinement and interval arithmetics [Mit90; CHMS00] to compute a bound on field variation on a given interval. Each time the estimated bound contains the isosurface c the interval is subdivided into two. Affine arithmetics [FPC10] can be used to obtain tighter bounds.

In [Kee20], one of the main drawback of this category of methods is tackled (albeit on an alternative direct rendering approach): efficient computation on large expressions. It relies on a representation of the expression well adapted to GPU evaluation as well as on-the-fly simplification of the expression. This method targets general implicit surfaces, the author hints at reduced efficiency of the interval evaluations in presence of shapes with many smooth blends. Our focus, skeletons with large number of primitives, falls in this category.

Acceleration data structures For large skeletons, computational times are also highly correlated to the time required to iterate over all the primitives that influence a given point along the ray (e.g. during field evaluation).

In [GPP*10] a bounding volume hierarchy (adapted for the management of the blending operation) was used for efficient evaluation. More recently, [Bru19] has relied on a dynamic data structure that is efficiently rebuilt at each frame by using the GPU rendering pipeline: for each ray, a linked list of intersection points with supports of each primitive is computed relying on a GPU based quadric visualization algorithm [SWBG06]. We extend this approach for better management of segment primitive with varying radius (primitives whose support are also sphere cones).

3. Our approach

Our main shape to be ray-traced is the SCALIS field (see Equation (8)) defined by a skeleton consisting of line segment primitives. Since SCALIS field evaluations are expensive to compute, brute force ray-marching approaches are not feasible. We therefore propose to rely on *local* field interpolations in order to reduce the number of field evaluations required for calculating the ray-surface intersections. To achieve high fidelity interpolations with lower degree polynomials, we propose to subdivide the ray into intervals that capture the main field variations along the ray. A different interpolation is used and refined in each interval, locally producing an accurate approximation. The overall idea and the processing of an example ray can be seen on Figure 5.

Our approach relies on a bijective mapping of the SCALIS field to an approximate smooth homothetic distance field (squared) which has the exact same isosurface of interest as the SCALIS field. All processing is done on this *normalized* field. This new field exhibits similar variations as the homothetic distance fields (squared) to individual primitives ${h_{S_i}}^2$ (see Equation (6)).

We use the correspondence between the normalized field and the ${h_{S_i}}^2$ fields in order to define our ray-subdivision strategy: the intervals are defined by cutting the ray at the minima of ${h_{S_i}}^2$ for each individual primitive S_i (purple curves in Figure 5 (a)). We expect such cuts to limit the number of oscillations of the normalized field in a given subinterval, a property required to perform field analysis through low degree polynomial interpolation. As can be observed in Figure 5 (a), the normalized field has this desirable property on each sub-intervals. By cutting the ray at the minima of ${h_{S_i}}^2$, the likelihood of finding a point within the implicit surface is increased, such points allows to isolate a root by detecting a sign change. While the local minima of the homothetic distance field along the ray provides only an approximation to the local minima of the SCALIS field, this approach behaves well in practice (see section 8).

Our ray-tracing algorithm generates and processes intervals on the fly in depth-order until an isovalue crossing is detected. This corresponds to the main loop (A) in the diagram in Figure 5. We process a subinterval by iteratively refining a local polynomial interpolation. The interpolation is first initialized using the Hermite data sampled at interval end-points. Then, it is refined by reducing the interval extent, cutting it at the root of the polynomial which interpolates the field on the reduced interval (inner loop (B) in the diagram and graph of Figure 5). Provided that the polynomial interpolation has a root within the sub-interval, the root of the successive interpolations will converge toward the real isosurface as the interval gets iteratively smaller. We use smooth piecewise quadratic polynomials to obtain fast and stable root computations.

We first discuss the mapping from density field to homothetic squared distance field in section 4. Then, in section 5 we describe our ray subdivision strategy and in section 6 we present the approach used to perform the processing of a ray's subinterval. Fi-

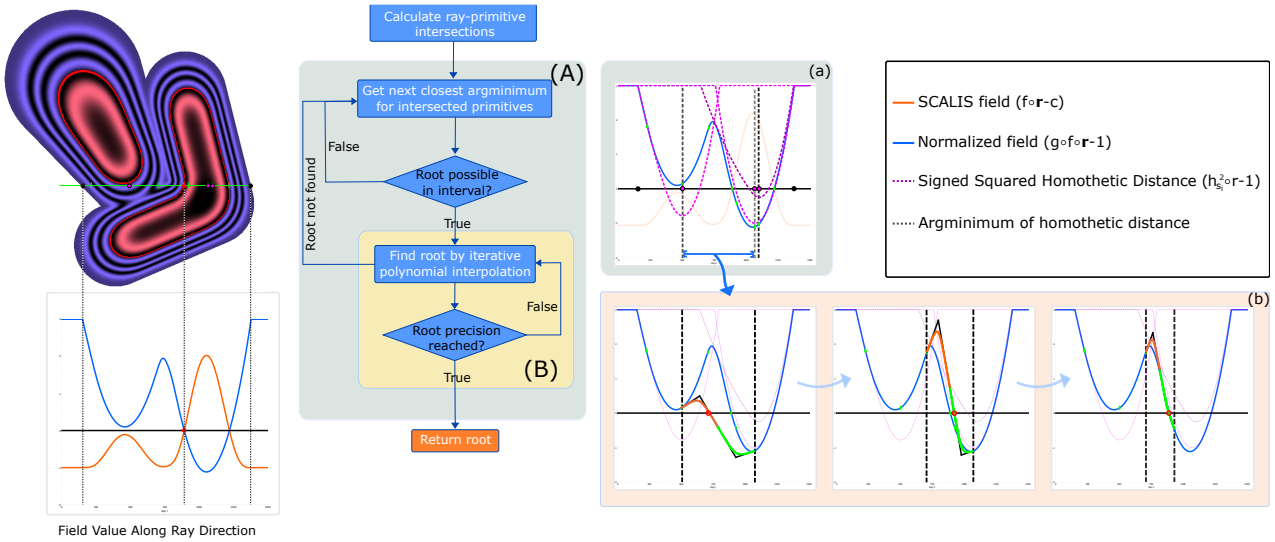


Figure 5: *General pipeline of our method.* Top Left: *A slice of the SCALIS field consisting of three primitives and a ray (green) to be processed. Below it, the SCALIS field variation along the ray $f \circ \mathbf{r}$ (orange) and the normalized field values (in blue) are given. We are interested in locating the zero-crossing (root of the Equation (10)) marked with the red disk. In the middle the two main loops of the algorithm are denoted as (A) and (B) and on the right these are shown respectively for the given ray: dividing the ray into intervals (a) and iterative root refinement by polynomial interpolation (b). The normalized field exhibits limited variations on sub-intervals defined by cutting the ray at local minima of the homothetic (squared) distance to individual primitives (purple curves in (a)).*

nally, we describe in section 7 our GPU implementation relying on a dynamic data structure.

4. Approximated squared homothetic distance

Scale-invariant integral surfaces are defined from the homothetic distance to skeleton points as described in section 1.1. For most of the kernels used in practice, it is actually defined from a *squared* homothetic distance. We use this property to introduce a mapping that allows to compute more precise quadratic interpolations. We first present the specific configurations where a quadratic polynomial can be fitted exactly to the mapped values, then discuss the general case.

4.1. Remapping SCALIS field values

In previous works on the integral surfaces, the study of infinite primitives has proven its usefulness for thickness control around skeletons [ZBQC13] and blending control [ZGC15]. We use it to analyze the homothetic distance to the isosurface and provide better polynomial interpolation. We first introduce a field mapping for infinite line primitives, then generalize it for arbitrary skeletons.

Infinite line primitive Let us consider an infinite line primitive with a constant prescribed radius τ_0. For the main kernel families used to define skeleton-based implicit surfaces, the SCALIS field for such primitive in isolation can be defined as a function of distance d to the line by using a kernel $\tilde{k}$ of the same family with a different degree. Hence, the field can be defined as :

$$f_{line,\tau_0}(d) = f_{line,1}\left(\frac{d}{\tau_0}\right) = \lambda\tilde{k}\left(\frac{d}{\tau_0}\right) \tag{11}$$

where $\lambda = c/\tilde{k}(1)$. For the *Compact Polynomial* kernel of order i, we have $\tilde{k} = k_{i+1,\sigma}$ and for the *Cauchy* kernel, we have $\tilde{k} = k_{i-1,\sigma}$.

We generalize the approach of [Bru19] from metaballs to full skeletons with varying radius. The *homothetic squared* distance (instead of a Euclidian distance) to the *line* primitive (instead of point primitive) can be computed at any given location in space from the field value by applying the inverse of the function defined in Equation (11) followed by squaring the result :

$$g(f) = \left({f_{line,1}}^{-1}(f)\right)^2 = \left(\tilde{k}^{-1}\left(\frac{f}{\lambda}\right)\right)^2 = \left(\frac{d}{\tau_0}\right)^2 \tag{12}$$

For the *Compact Polynomial* kernel, this develops into:

$$g(f) = \left(\frac{d}{\tau_0}\right)^2 = \sigma^2 \begin{cases} ((1-(1-\frac{1}{\sigma^2})(\frac{f}{c})^{\frac{2}{i+1}})) \text{ if } f > 0\,, \\ 1 \text{ otherwise.} \end{cases} \tag{13}$$

while for the *Cauchy* kernel, this develops into:

$$g(f) = (1+\sigma^2)\left(\frac{c}{f}\right)^{\frac{2}{i-1}} - \sigma^2 \tag{14}$$

with the scale parameter σ and the order i of the kernel as defined in Equation (2,3). Note that in the case of radius $\tau_0 = 1$, this is also the Euclidean squared distance to the line.

Similarly, as $g(c) = 1$, a *signed* squared homothetic distance to

the isosurface of interest can be computed:

$$g(f) - 1 \tag{15}$$

In the remainder of the text, we call the field resulting from the application of Equation (15) to the SCALIS field the *normalized* field.

For a line with constant radius in isolation (or a long enough segment for the compact polynomial kernel), it is important to note that for a given ray, the squared distance from the ray to the line is a degree two polynomial. Hence, for such a configuration, the squared homothetic signed distance can be *exactly* interpolated by a quadratic polynomial defined by Hermite data sampled at an arbitrary position along the ray (i.e. by evaluating $g \circ f \circ \mathbf{r} - 1$ and its derivative at the ray parameter t).

Other special configurations For a few other configurations, the function $g \circ f \circ \mathbf{r} - 1$ is also a quadratic polynomial, hence it can be exactly interpolated by a quadratic polynomial defined by any Hermite data sampled along the ray. Such configurations are: any rays for two equal line primitives, all rays belonging to the bisecting plane between two parallel line primitives, the three rays that correspond to the equidistant lines for two crossing line primitives and the two rays corresponding to equidistant lines for two arbitrary line primitives. Indeed in all those cases, the global field is defined by $2f_{line,\tau_0}(d)$, hence a constant can be factored out from Equation (13).

Generalization Interestingly, Equation (13) can be used directly on the SCALIS field generated from a more general skeleton in order to compute a smooth approximation of the homothetic squared distance to the surface. Indeed, since g is a strictly decreasing function on $\mathbb{R}^+$, hence injective, the *normalized* field leaves all the level set unchanged (in terms of geometry), including the one corresponding to the isosurface of interest (which now corresponds to an isovalue of 0 instead of c). And, outside of the blending area (e.g. branching) and the neighborhood of skeleton end-points, the field behavior will tend toward one of an infinite line before the application of g, hence toward a squared homothetic distance after application of g.

We designed the normalized field to have good properties for our algorithm: it does not create new oscillations in the field and is less dependent on the kernel used. The mapping has the beneficial property of removing the inflexion points of $f \circ \mathbf{r} - c$ that are due to the Cauchy and Compact polynomial kernel shape (see Figure 6).

5. Ray subdivision

In order to simplify the processing of a given ray, we want to define intervals with limited number of oscillations (typically either one or two local minima of $g \circ f \circ \mathbf{r} - 1$ exists in the defined intervals). This provides better interpolations in configurations where the normalized field along the ray is not exactly a quadratic function.

We further use the fact that the normalized field can be seen as a smooth approximation of the homothetic distance (squared) to the surface $\min_i(h^2_{S_i}) - 1$ in order to simplify the processing of a given ray.

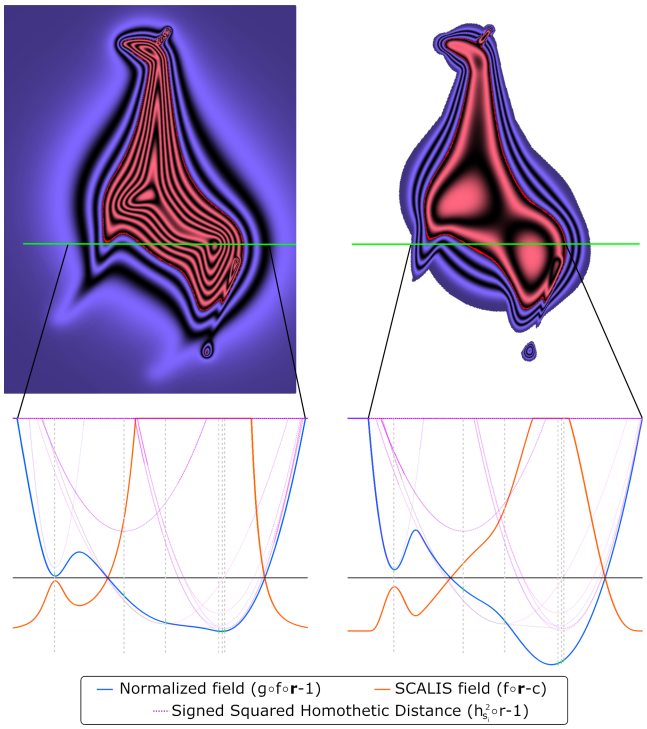


Figure 6: *A sample ray on the skeleton defined in Figure 2.* Left: *Cauchy Kernel.* Right: *Compact Polynomial Kernel. The field variation along the ray $f \circ \mathbf{r}$ (orange curve) exhibits limited variations on sub-intervals defined by cutting the ray at local minima of homothetic (squared) distance to individual primitives (purple curves) . A high correlation between these values can also be observed, and is even more noticeable after mapping $f \circ \mathbf{r}$ to approximate homothetic squared distances (blue curve).*

5.1. Correlation with $h^2_{S_i} - 1$

In order to ease the analysis of the field, we take inspiration from the Linderberg principle of absence of local extrema creation by convolution with a specific family of kernel [Lin91]. While we are not exactly in this context (Cauchy and Gaussian verify exactly the condition, compact kernel is only close to it, SCALIS is not a convolution when using varying radii), we experimentally observe a *similar* behavior. Furthermore, the mapping g is decreasing, it cannot create new local extrema, which allows to do a similar observation on the normalized field (see Figure 5 and 6). Similarly, we expect the local normalized field behavior to directly correlate to homothetic squared distance $h^2_{S_i}$ to individual primitives including in presence of varying radius. Indeed, it is defined by blending (i.e. smoothing) contributions of all skeleton-points based on their proximity (i.e. with sharper kernels the correlation between $g \circ f \circ \mathbf{r} - 1$ and $h^2_{S_i} \circ \mathbf{r} - 1$ increases).

The aforementioned evidence hints that a good sampling strategy is achievable by computing the arguments of the minima of the homothetic distance to the individual primitives $h_{S_i} \circ \mathbf{r}$.

5.2. Argminimum of homothetic distance along a ray

In order to generate our ray subdivision on the fly, we need to find efficiently the argument of the minimum of the homothetic distance from a segment primitive S_i with linearly varying radius (Equation (6)) to the ray $\mathbf{r}$, i.e. we are trying to solve:

$$\underset{t}{\operatorname{argmin}}\, h_{S_i} \circ \mathbf{r}(t) \tag{16}$$

Let us define the segment primitive S_i, parametrized by:

$$\mathbf{p}(s) = \mathbf{p}_0 + s\,\mathbf{u} \;\text{ and }\; \tau(s) = \tau_0 + s\,\Delta\tau \tag{17}$$

with $\mathbf{u}$ a unit vector and the parameter s in the range $[0, L]$, L being the length of the segment. For a skeleton point $(\mathbf{p}(s), \tau(s))$ in isolation, the Euclidean distance and the homothetic distance have the same argument of the minimum along the ray, the minima only differing by a factor $1/\tau(s)$. For the Euclidian distance, the minima (squared) have a closed form expression :

$$d_{\mathbf{r}}{}^2(\mathbf{p}(s)) = \|(\mathbf{m} + s\mathbf{u}) - (\mathbf{m} + s\mathbf{u})^T \mathbf{d}\,\mathbf{d}\|^2 \tag{18}$$

with $\mathbf{m} = \mathbf{p}_0 - \mathbf{o}$, where o is the ray origin and $\mathbf{d}$ is the ray direction.

Therefore, in order to solve Equation (16), we can invert the order in which minima are computed and we can instead study:

$$\underset{s\in[0,L]}{\operatorname{argmin}} \frac{d_{\mathbf{r}}^2(\mathbf{p}(s))}{\tau(s)^2} \tag{19}$$

This gives the skeleton parameter s_{min} for which the minimum is reached. The ray parameter t_{min} can easily be derived from it (e.g. by computing the orthogonal projection of the skeleton point $\mathbf{p}(s_{min})$ onto the ray).

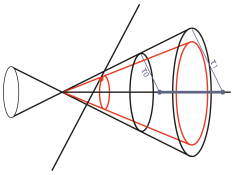
We therefore have to study a rational function whose poles are not in the range of interest (i.e. in the range $[0, L]$), thus the minimum of the function is either reached at the boundary of the interval $[0, L]$ or where the derivative cancels. By the cancellation of the derivative, we get a linear equation in the form $bx + c = 0$ with:

$$b = \tau_0 L^2 (1 - (\mathbf{u}^T\mathbf{d})^2) - \Delta\tau L \mathbf{u}^T(\mathbf{m} - (\mathbf{m}^T\mathbf{d})\mathbf{d})$$

and

$$c = \Delta\tau \|\mathbf{m} - (\mathbf{m}^T\mathbf{d})\mathbf{d}\|^2 + \tau_0 L \mathbf{u}^T(\mathbf{m} - (\mathbf{m}^T\mathbf{d})\mathbf{d})$$

Note that Equation (16) could be solved directly. Recall that the sphere-union associated to a given primitive includes a cone section. Depending on the ray/cone configuration, it amounts to finding the smallest scaling factor to apply to the segment's radii such that there exists a single intersection between the ray and the scaled sphere-cone, see Figure inset. We find that our approach requires less computations and is easier to compute in a numerically stable way.



End-point correctors The minima of $h_{S_i} \circ \mathbf{r}$ are highly correlated to the maximal field contributions for a given primitive. However, due to the integral nature of the SCALIS field definition, field variations diverge from this behavior near primitive end-points. While this is not problematic in general (due to blending by summation of adjacent primitives), this could become a problem near skeleton end-points. This problem is related to the problem of radius shrinkage at skeleton end-point observed in [ZBQC13]. When applying the *end-point correctors*, we force an additional sampling position near skeleton end-points, increasing the correlation between the argument of the minima of $h_{S_i} \circ \mathbf{r}$ for individual primitives and the ones of $g \circ f \circ \mathbf{r} - 1$. Note that in the absence of correctors, a heuristic presented in section 6.2 can compensate this shortcoming.

6. Ray processing

The ray subdivision strategy presented in the previous section not only provides intervals with limited number of oscillations, it also limits the initial number of intervals along a given ray. Smaller number of intervals result in smaller number of compulsory field evaluations while processing a given ray.

We also subdivide the ray when switching from an area not overlapped by any primitive support to an area overlapped by at least one primitive support (and vice-versa). This way, we can avoid processing large empty areas. This is easily achieved with the data structure presented in section 7.

We first present the quadratic polynomial interpolation used to define local field interpolation within a given subinterval, then we present the processing of a given subinterval.

6.1. Quadratic polynomial interpolation

Working with quadratic polynomial interpolation has the advantage of much more efficient, simpler and numerically stable root computation. As discussed in section 4.1, thanks to the field normalization, it also provides exact interpolation, in specific cases, independently of the kernel used. However, some configurations of Hermite data at interval's end-points cannot be realized by a quadratic interpolation (four constraints for only three degrees of freedom, see Figure 7). In such incompatible configuration, a possible approach would be to rely on Hermite-Birkhoff data, ignoring one of the two derivatives at interval end-points (see green curve in Figure 7). Instead, we propose an alternative solution that allows to respect the Hermite data at both end-points by relying on two piecewise quadratic polynomials, and without computing any additional field values. For simplicity we cut the initial interval in two sub-parts of equal size. Our interpolant is defined as a one dimensional Bézier

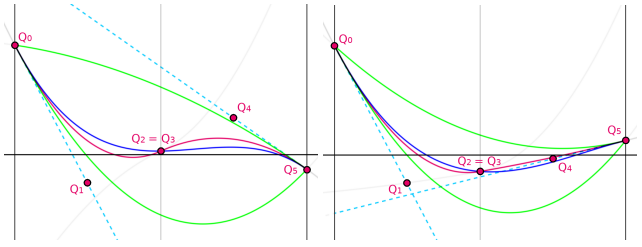


Figure 7: *Two Hermite data configurations and associated polynomial interpolations: Hermite cubic interpolation in blue, quadratic interpolation by part in red and quadratic Hermite-Birkhoff interpolation in green.*

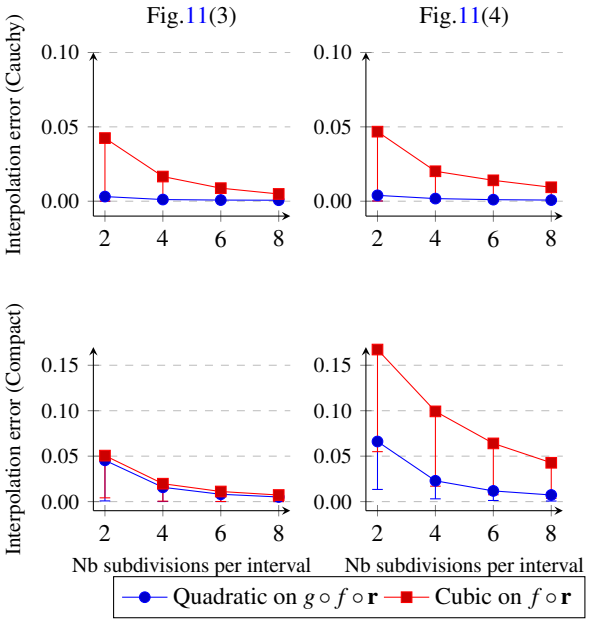


Figure 8: *Comparison of convergence between the quadratic interpolation of $g \circ f \circ \mathbf{r} - 1$ and the cubic interpolation of $f \circ \mathbf{r} - c$. Both averages (main curve) and medians are computed over the intervals defined from our segmentation strategy with different level of subdivision of those initial intervals. Note that for the Cauchy kernel we ignore intervals for which all field values are below* 0.015. *This allows to minimize the impact of the kernel clipping required due to the infinite support of the Cauchy kernel. For each graph, 5k rays are launched in the scene of Figure 11 (middle and far right respectively).*

curve on each part. For such curves, the control polygons are constrained such that the control points are equally spaced in abscissa. The ordinates of control points are chosen to verify Hermite data and interpolation smoothness. From the six control points $Q_{i=0..5}$ (three for each of the Bezier curves), the two first control points $Q_{0,1}$ of the first interval (respectively, last points $Q_{4,5}$ for the second intervals) are constrained by Hermite data. Remaining control points on both sub-intervals should have the same value $Q_2 = Q_3$ to ensure continuity and they should lie on the line joining the two adjacent control points Q_1 and Q_4 to ensure smoothness, which leads to the following ordinates:

$$\frac{1}{2}(h_1 + h_2) + \frac{L}{8}(h_1^{'} - h_2^{'}), \tag{20}$$

where (h_i, h_i') is the Hermite data at each end of the interval. Note that this value is also equal to the value returned by the cubic polynomial interpolation at the middle of the range (see Figure 7) as well as the average of the two Hermite-Birkhoff interpolations and still provide perfect interpolation when Hermite data are compatible.

In the case of the compact polynomial kernel, for general configurations, we experimentally observe a similar convergence rate (function of the interval size) for the quadratic interpolation on the normalized field $g \circ f \circ \mathbf{r} - 1$ and cubic interpolation on the original field $f \circ \mathbf{r} - c$ (see Figure 8, right). As expected, better results are obtained for primitives with limited radius variation and blending. For the Cauchy kernel, we obtain much better interpolations with the normalized field for small numbers of interval's subdivision, as such kernel is badly suited for low degree polynomial interpolation on larger intervals.

6.2. Processing of an interval

We generate intervals iteratively and process them on the fly. For each interval we apply a routine in order to check if the isosurface is crossed. If so, we return the position of the isosurface along the ray, else we move to the next interval (see section 7).

Let us define $[t_{begin}, t_{end}]$ as a sub-interval to process. We use the quadratic interpolation of Hermite data presented in Section 6.1 in order to estimate the isosurface position if it exists. The estimated position t_{root} is defined as the first root of the interpolation defined from Hermite data evaluated at t_{begin} and t_{end}.

The routine iteratively reduces the size of the interval in order to define Hermite interpolations of increasing precision (see Figure 5). By doing so, the estimate t_{root} converges toward the exact isosurface location.

The interval size is reduced by moving one of its endpoints to the estimate t_{root} depending on the Hermite data evaluated in t_{root}. If no clear choice is possible (e.g. no sign change is detected in normalized field value and neither of the sub-intervals $[t_{begin}, t_{t_{root}}]$ and $[t_{root}, t_{end}]$ has a constant sign for their directional derivative, the first sub-interval is chosen and the second interval is stored to allow one step of backtracking.

The iteration continues until the presence of the isosurface can be rejected or the isosurface has been located within a given error threshold. The structure of the interval processing loop is given in Algorithm 1.

Heuristic/Oracle On intervals presenting large radius variations (or in absence of end-point correctors) the polynomial interpolation might overestimate normalized field values $g \circ f \circ \mathbf{r} - 1$, opening the possibility of missing the isosurface. One solution to this problem would be to uniformly subdivide intervals in order to obtain better initial polynomial interpolations. However this comes at a large increase in the number of intervals to be processed. In order to avoid

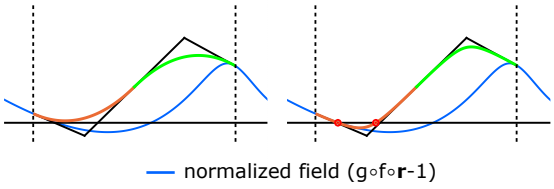


Figure 9: *A polynomial interpolation can present smaller variations than the actual field function, resulting in possible missed isovalues. Until a root is isolated, we rely on rational Bézier curves to address this problem.*

Algorithm 1 Processing interval $(t_{begin}, h_b, t_{end}, h_e)$.

```
// h_b, h_e are tuples containing Hermite data
n ← 0, saved ← None
while ++n < 32 do
  t_root ← Oracle(t_begin, t_end, i)
  if t_root == ∞ and !saved then
    return false, ∞
  else if t_root < ∞ then
    h_r ← HermiteData(t_root)
    if |h_r.val| < ε then
      return true, t_root
    end if
    b_prim ← h_r.prim > 0 and h_b.prim < 0
    if !saved and h_r.val ≥ 0 and h_e.val < 0 and b_prim then
      saved ← [(t_root, h_r), (t_end, h_e)]
    end if
    if h_r.val < 0 or b_prim then
      t_end, h_e ← t_root, h_r
    else
      t_begin, h_b ← t_root, h_r
    end if
  else if saved then
    (t_begin, h_b), (t_end, h_e) ← saved
    saved ← None
  end if
end while
return false, ∞
```

this, our algorithm uses a heuristic to limit the need for extra subdivision. We modify the oracle estimating the existence of a root (and its position if it exists) until an isosurface crossing has been isolated (i.e. until a sign change exist between interval end-points).

We rely on rational Bézier curves until a root is guaranteed. Such curves allow to have an interpolation that is arbitrarily closer to the control polygon hence increasing the chance to detect an isovalue crossing (see Figure 9). Note that with rational functions, root computation remains the same as long as the pole is not in the range of interest. In our implementation, a weight equal to 3 on the middle control point of the interpolation curve was sufficient on all tested examples.

With the modified oracle, even without any interval subdivision, failure cases are rare (see section 8), especially compared with the behavior of the sphere-tracing algorithm that requires arbitrarily large number of steps therefore creating holes in the surface, when working with limited number of steps for rapid rendering (see again section 8).

7. Dynamic data structure : efficient GPU implementation

When evaluating the SCALIS field on large skeletons with compact kernels, the computation time is mostly driven by the selection of primitives whose supports overlap the evaluation point. Similarly to [Bru19], our GPU implementation works in two steps, first, building a dynamic data structure where entry and exit points in primitives supports are stored per ray, then processing the ray. We rely on the stored data to serve as an acceleration data structure to efficiently retrieve the primitives whose supports overlap the given interval of the ray. When using the Cauchy kernel, we clip the support of a given primitive by a sphere-cone whose scaling $\sigma_{clipping}$ is computed to guarantee a maximal error $\varepsilon_{clipping}$. This allows us to use the same approach for both kernel families.

The first step relies on a fast GPU construction of A–buffers [Thi11; MCTB11; MCTB12]. A geometry shader is used to generate a quad per skeleton segment such that the quad covers the projected segment primitive support. As the support of a given primitive is defined by intersecting some quadrics (a sphere-cone), we compute the quad using [SWBG06]. We adapt the proposed approach to compute segment-aligned quads to limit the number of generated fragments for which ray/sphere-cone intersections are computed (in particular for primitives whose maximal radii are comparatively small to the segment length). The ray-supports intersections are computed in a fragment shader. Entry and exit points in the sphere-cones, as well as the associated primitive id, are inserted in the A–buffer linked-list (see Figure 10) and sorted on the fly according to the depth of the entry [LHL14]. Note that contrary to [Bru19], the use of a compact support kernel avoids the need to cut-off primitives influences in this first step.

Of course, any variant of A–buffer creation could be used to accelerate this first step, for instance postponing the sorting of fragments until the ray processing step [Bru19] or using advanced memory management [SF14].

Once the first step of the processing is done, the linked-lists of the A–Buffer define intervals with a fixed set of segment primitives. We now face a choice. Subdividing the ray based on those intervals would minimize the maximum number of segment primitive to be stored per fragment shader call. However, it would also increase the number of field computations required. Increasing the interval size too much might require more memory to store all segment primitives whose supports overlap each intervals. On the opposite, subdividing the ray only when switching from an area not

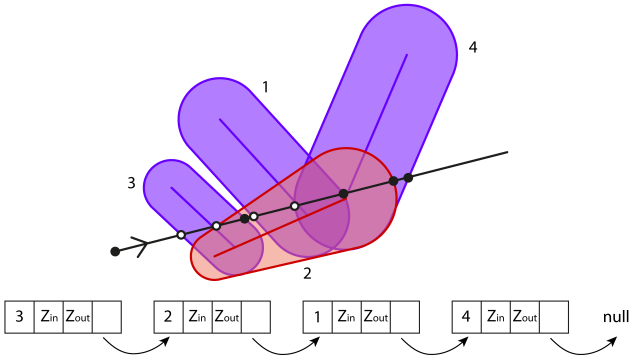


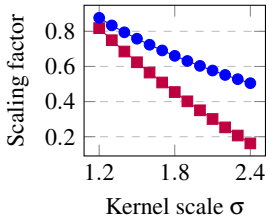
Figure 10: *A-buffer linked-list associated to a given pixel after rasterization of all skeleton's primitives. Note that entry depth z_{in} and exit depth z_{out} in the support of a given primitive are stored in the same linked-list node. This linked-list is the main entry to the ray processing algorithm and allows to retrieve efficiently the primitives whose supports overlap a given ray's subinterval.*

overlapped by any primitive support to an area overlapped by at least one primitive support (and vice-versa) would require storage of a large number of segment primitives at once depending on the ray/skeleton configuration, hence increasing memory usage of each individual thread.

We choose to combine this second subdivision strategy to our contribution presented in section 3. This presents a nice trade-off: it limits the number of segment primitives to be stored at once while generating long enough intervals, hence limiting the number of compulsory field evaluation. The fragment shader that renders the view generates and processes the intervals on-the-fly using algorithm 1 until an intersection with the isosurface is found. The resulting linked-list processing loop is given in Algorithm 2.

Due to possible numerical instability in the normalized field derivative computation (the latter being non continuous when the field switch from null to non null), we apply one step of sphere tracing to avoid those areas, note that this require no field evaluation as the field is null at the support limit (see comment in Algorithm 2).

Toward output sensitivity Similarly to Bruckner's approach to achieve output sensitivity [Bru19], it is possible to render an occluding depth buffer in order to limit the number of primitives that will be registered in the linked-lists. The depth buffer is initialized with depth values corresponding to positions that are guaranteed to be in the volume. Then all primitives whose support entries are beyond the occluding depth can be safely ignored. For efficiency those depth values need to be computed in a first render path on a per primitive basis. Due to the integral formulation of the field, the minimal radius around primitives does not necessarily correspond to the prescribed radius (e.g. typically for bent skeletons). In order to overcome this difficulty, we rely on two assumptions: usage of the skeleton *end-point correctors* and bounded radius variations: none of the two spheres at a primitive end-point completely encompass the other - e.g. each vertex should have an influence on the geometry of the union of sphere. Note that we do not render occluders for primitives corresponding to end-point correctors. In this context, thanks to the scale invariant property of the SCALIS representation, it is possible to analyze the worst skeleton configuration to derive a scaling factor for the sphere cones' radii such that the implicit surface is guaranteed to enclose the modified primitives. Computation of this scaling factor is discussed in Appendix A. The quality of the minimal volume is related to the kernel scale parameter, the larger the scale the smaller the guaranteed volume is. For instance, for typical Compact polynomial kernel scale σ ranging from 1.5 to 2, the scaling factor range from ≈ 0.62 to ≈ 0.35 (see Figure inset - the purple curve correspond to our bounded radius variation). Similarly, for more constrained radius variations, better bounds can be computed (see Figure inset, the blue curve corresponds to constant radius). In section 8, we discuss rendering times with and without this optimization. In absence of end-point correctors, an alternative strategy would be to rely on the field behavior of primitives in isolation. Indeed, for long enough primitives with constant radius, there is a portion of the segment



Algorithm 2 Processing linked list.

```
GenerateRay(i,j)
primitives ← emptyHeap()
argmins ← emptyHeap()
frags ← InitFragList(i, j)
t_argmin ← ∞
t_entry ← frags.currZIn()
while !(frags.empty() and primitives.empty()) do
  t_b, h_b ← t_e, h_e
  t_exit ← primitives.back().val
  if primitives.empty() or t_argmin < t_exit or t_entry < t_exit then
    if primitives.empty() then
      t_b, h_b ← t_entry, (g(0), ···)                    {see below}
    end if
    while t_argmin > t_entry and !frags.empty() do
      t_prim ← Argmin(σ, frags.currSegId())
      t_out ← frags.currZOut()
      primitives.insert({t_out, frags.currSegId()} )
      argmins.insert(t_prim)
      t_argmin ← argmins.front()
      frags.next()
      t_entry ← (!frags.empty())?frags.currZIn() : ∞
    end while
    t_e, h_e = t_argmin, HermiteData(t_argmin)
  else
    t_e, h_e ← t_exit, (g(0), ···)                       {see below}
  end if
  // see text for management of entry/exit of non-null field area
  if t_b < t_e then
    f, t ← processInterval(t_b, h_b, t_e, h_e)
    if f then
      GenerateIsoSurface(t)
      return
    end if
  end if
  primitives.popElements([](x){x ≤ t_e})
  argmins.popElements([](x){x ≤ t_e})
  t_argmin ← (!argmins.empty())?argmins.front() : ∞
end while
```

primitives for which the prescribed radius is guaranteed. While we do not use this property in our implementation, it could be used to improve the occluders for some specific skeleton configurations.

8. Results

Our method was implemented in C++ using the OpenGL library and shaders were programmed in GLSL. We have tested our implementation on a large range of objects including artistic shapes (see Figures 1, 2, 13, 15), truss structures (see Figure 16), branching structures (see Figure 11), procedural foam structures (see Figure 12) and random skeletons (see Figure 11) using both low-end and high-end graphic cards (namely an Intel UHD Graphics 630, an nVidia Quadro P1000 and an nVidia GeForce 2080 RTX). Examples have been chosen to present a large variety of skeletons in terms of number of segments, local density of primitives and whether or not the radii change. In order to assess the efficiency

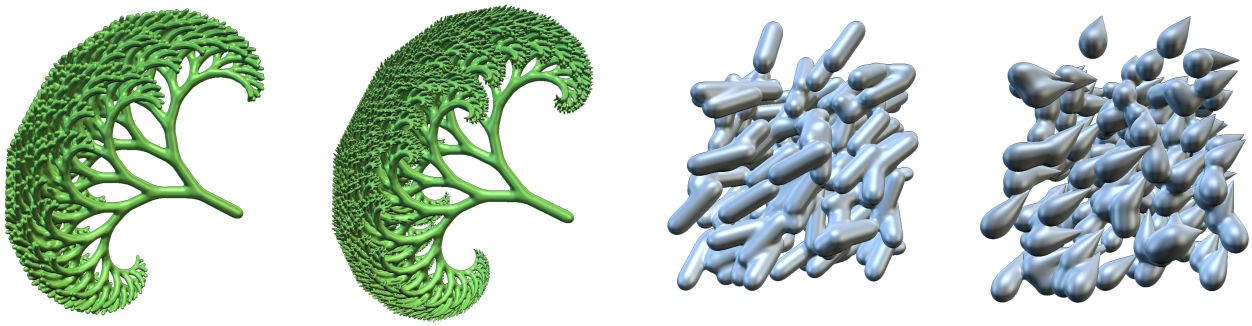


Figure 11: Left: *Procedural trees.* Right: *Random skeletons with constant and varying radii used to test the resilience of our algorithm.*

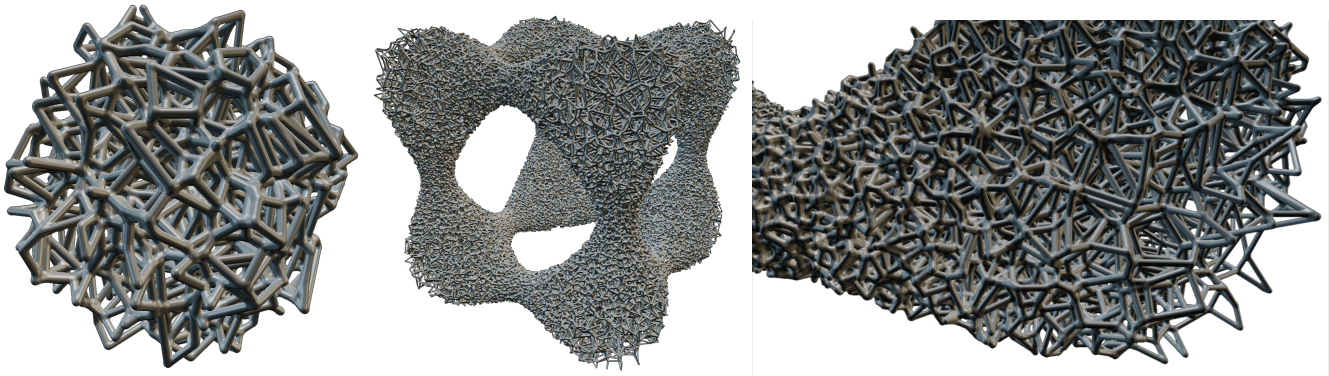


Figure 12: *Procedural foams whose skeletons are defined as edges of a voronoi diagram of points.*

Figure 13: *Real-time rendering allows users to efficiently explore kernel scale parameters for a given skeleton.*

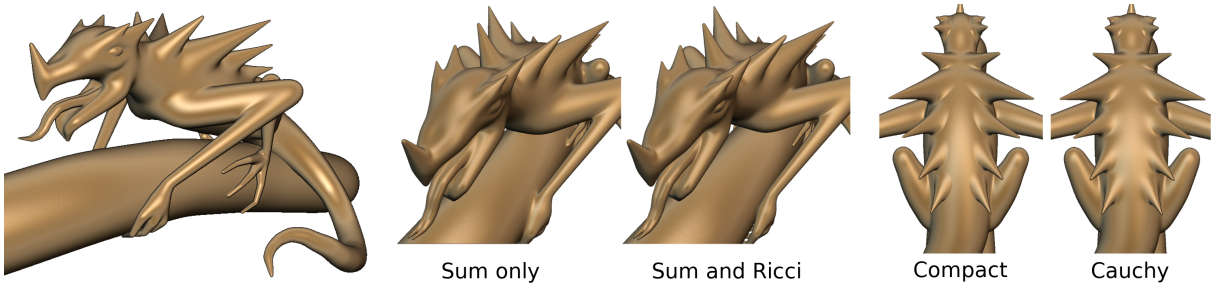


Figure 14: Left: *Model generated using a Cauchy kernel and Ricci blendings of subcomponents (main body, eyes, tongue, legs and branch),* Middle: *Comparison with summation-only blending,* Right: *Comparison with compact polynomial kernel.* .

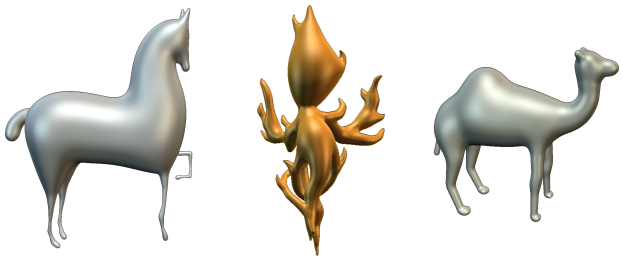


Figure 15: *Skeleton-based free form modeling.*

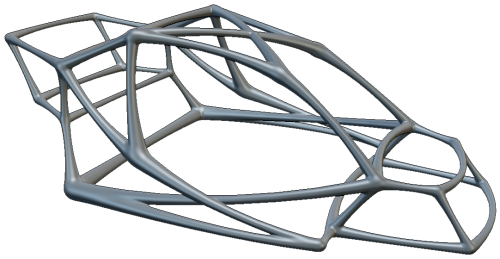


Figure 16: *Truss structure.*

of our method, we compare it to several state of the art techniques: sphere-tracing on the field $g^{\frac{1}{2}} \circ f \circ \mathbf{r} - 1$ (similar in spirit to [Bru19], see Appendix B for the computation of the Lipschitz constant), Shertyuk fast ray-tracing [She99] and segment-tracing [GGPP20] on $f \circ \mathbf{r} - c$. Comparison includes both GPU rendering time and statistics per ray (computed on a CPU implementation). For Sherstyuk fast ray-tracing, we only compare rendering time as field evaluations are not performed in the same way. Comparisons with the segment-tracing algorithm are done in the context of homothetic distance primitives (e.g. Equation (7)) instead of SCALIS primitives (e.g. Equation (8)) as we do not have a computation routine for the local directional Lipschitz bounds on the latter equation. It only includes ray statistics. Our study mostly focuses on the compact polynomial kernel, we explicitly specify in the text when experiments are run with the Cauchy kernel.

Note that in practice, it is is also possible to blend different skeletons with a blending sharper than the summation blending, e.g. by relying on Ricci's blending [Ric73] (see Figure 14). Indeed, such blending keeps the correlation of the field with the homothetic squared distance to individual primitives by defining a blending behavior parametrized between a summation blend and a maximum blend.

GPU rendering time We record the average rendering time for a set of viewpoints around our test objects. In order to provide fair comparisons, the GPU implementation of the sphere-tracing and the one of the Shertyuk ray tracing both rely on the A-buffer. In our implementation of Sherstyuk's method, we use four subdivisions per primitive, which correspond to the coarsest subdivision allowing to obtain visually acceptable results on primitives with constant radius. Our implementation of the sphere-tracing routine also uses a ray subdivision strategy in order to discard space outside of any primitive support as well as to limit the maximal number of segments to be stored and processed at the same time during the ray processing loop. On each subinterval the number of sphere-tracing steps is bounded (typically 256 in our implementation).

For the compact polynomial kernel, all renderings are done at a resolution of 2048×2048 and rendering times are provided in Table 1. On all tested examples, we observe improvements of the global rendering time with our method in comparison to the most competitive one (which is dependent on the model). Rendering time reduction ranges from -34.2% to -48.7% on the Intel card, from -51.5% to -78.0% on the nVidia Quadro card and from -35.3% to -72.1% on the nVidia RTX card. In addition, both sphere-tracing and Sherstyuk's fast ray tracing can present visual artifacts (see Figure 17, note that those artifacts can be reduced at the expense of additional computation time). It is also important to note that in presence of a large amount of blending - for instance if a large number of segments connect in one vertex (such as in Figure 12) - the sphere-tracing algorithms would require usage of under-relaxation to properly find the surface.

We also have tested our method at higher resolutions where we observe similar improvements (see Figure 20). In order to perform a stress case on the nVidia RTX, we rendered a foam structure consisting of 526k segments using the compact polynomial kernel, running at 15 frames per second (see Figure 12, right). Finally, we provide individual timing for the A-buffer creation step (with and without occluders) and the ray processing steps. As expected, usage of occluders can have a large impact in terms of rendering time (as well as memory consumption) depending on the nature of the skeleton.

For the Cauchy kernel, we mainly tested our method on the nVidia RTX card with a lower resolution of 1536×1536 in order to account for the higher number of primitive overlaps generated by the larger clipping sphere-cone (e.g. $\sigma_{clipping} = 4$ for $\sigma = 0.3$ and $\varepsilon_{clipping} = 0.005$), otherwise this would lead to an A-buffer overflow. The use of a kernel with a larger footprint results in an increase of computational time. With our method, timings range from 7 to 54 milliseconds (excluding Fig.12(Middle)). With respect to sphere tracing, this corresponds to a rendering time reduction ranging from -46.2% to -81.3%. We observed similar improvements on the two other GPUs when run at a resolution of 1024×1024.

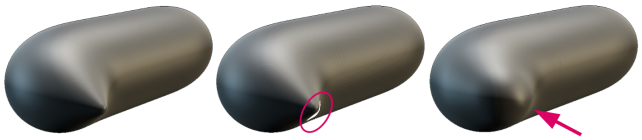


Figure 17: Left: *our method presents minimal errors in isosurface intersections,* Middle: *when using sphere tracing with a number of steps limited to a few hundreds (in order to limit runtime cost) holes can appear in the surface,*Right: *Sherstyuk fast ray tracing can produce large deformation of the surface depending on the viewpoint.*

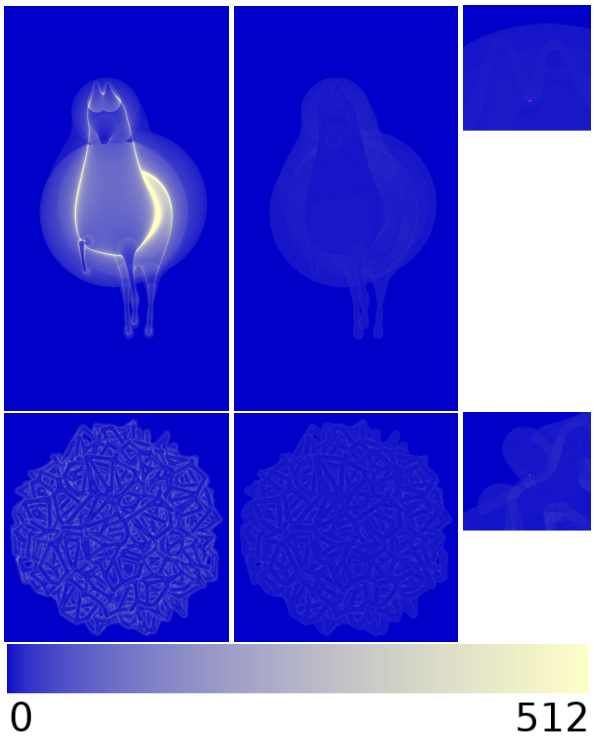


Figure 18: *Number of field value computations used during the processing of a ray. The red pixels correspond to isosurface crossing missed by our method.* Left: *Sphere-tracing* Right: *Our method.*

Note that we do not compare runtime with the Sherstyuk method as it produces numerous artifacts due to poor polynomial interpolation.

Interactive modeling The rendering time achieved by our method allows interactive manipulation of a large range of skeletons. Both, random and procedural skeletons (see Figure 11), can be generated interactively. In our implementation skeletons are generated on the CPU at each parameter change. Fast rendering also allows real-time exploration of kernel parameter as depicted in Figure 13.

Statistics In order to compute per ray statistics, we have launched sets of rays in six directions (three main axis with both positive and negative orientations). For each model, we have computed the axis-aligned bounding-box from sphere-cone supports. In each direction, we sample the ray origins uniformly in the associated bounding-box's face such that a total of 10 millions rays are launched per model. For each ray, we measure the average, median and maximal number of steps required to process the ray. Note that the maximal number of rays are more important for SIMD architectures. Statitics are provided in Table 2 and 3. It is important to recall that our method relies on gradient computation for the evaluation of directional derivative. As described in [ZBQC13], both field and gradient can be computed for less than twice the evaluation cost of a single field value. Furthermore, as both values are computed at once, segment parameters are only fetched once from memory.

We observe a reduction for all statistics when using our method (see Table 2), including when primitives present small radius variations (e.g. the radius is either approximately constant over the full skeleton or is approximately constant by part on the skeleton). Larger reductions are observed in presence of varying radii.

However, the main benefit is achieved along grazing rays that present the maximal number of steps (a well-known problem of the sphere-tracing algorithm - see Figures 21 and 18).

When using distance primitives instead of SCALIS primitives, this limitation of the sphere-tracing algorithm can be alleviated by the use of directional Lipschitz bounds (e.g. the segment-tracing algorithms). In this context, our method still provides reduction of the maximal number of steps required along grazing rays (see Table 3).

In order to validate the robustness of our method, we have also measured errors along rays. In practice, less than $10^{-3}\%$ of the rays are missing an isosurface crossing, all of them are grazing rays (see Figure 18 and 21). Those few errors are due to Hermite data configurations for which control polygons of the interpolant do not cross the isovalue 0. This limits the efficiency of our rational interpolation heuristic. In order to mitigate this shortcoming, it could be interesting to investigate 2D quadric Bezier curves which would allow to increase the amplitude of the control polygon by moving the abscissa of the control points.

Limitation Our algorithm and its implementation present a few limitations. First, the heap used to store segment primitives during the ray processing loop should be large enough to accommodate all primitives whose supports overlap the initial intervals defined by the argument of the minimum of homothetic distance. Observe that this limitation could be mitigated by adding an additional break condition in Algorithm 2 based on the current heap size. Secondly, output invariance could be improved by combining our current strategy with occluders generated from primitives considered in isolation. Finally, the A-buffer can become expensive to build for finely tessellated curves (due to overlaps between primtive supports). Two main directions could be investigated : curve primitives [FHZ19] and curved simplification based on local radius and kernel scale. Similarly, finely tessellated skeleton also increase the number of argument of the minimum of homothetic distance along the ray.

9. Conclusion and Future work

We have introduced a new rendering algorithm for scale-invariant integral surfaces. Our approach presents no visible artifacts while decreasing computation time by up to 70% in comparison to revisited previous work. Our key contributions are 1) the usage of a new initial sampling strategy of the rays based on homothetic distance to segment primitives, 2) the introduction of a new field mapping combined to quadratic polynomial interpolation, and 3) an efficient GPU implementation relying on a A-buffer data structure. We believe this technique could help in further study of a larger range of integral surfaces.

Among future directions of research, we can mention management of triangle primitives (these share the main properties used in our method) and transparent rendering (relying on depth slabing to

Object	Intel UHD Graphics 630			nVidia Quadro P1000			nVidia GeForce 2080 RTX		
	Ours	Sphere	Sherstyuk	Ours	Sphere	Sherstyuk	Ours	Sphere	Sherstyuk
Fig.11(4)	290.4	551.9	565.9	84.2	300.4	262.5	7.9	28.2	18.4
Fig.11(3)	284.4	374.4	526.2	80.0	169.0	230.3	7.9	16.1	17.0
Fig.11(2)	243.0	388.0	384.1	127.8	302.0	268.8	12.0	27.2	22.5
Fig.11(1)	169.5	244.6	289.6	74.6	169.0	182.1	7.3	15.6	15.3
Fig.13(Middle)	121.3	204.9	178.4	31.8	120.0	79.9	6.2	12.9	7.7
Fig.2	116.5	169.8	177.1	23.8	81.9	70.8	6.3	12.9	7.7
Fig.15(Middle)	159.3	272.0	265.4	39.9	151.0	115.4	6.6	15.1	10.0
Fig.1	129.7	247.8	202.4	33.6	152.7	93.3	6.7	22.0	13.2
Fig.15(Left)	207.7	366.2	383.0	49.2	182.0	151.1	6.9	18.9	11.3
Fig.15(Right)	117.5	187.1	181.1	29.1	106.8	76.3	5.9	10.4	7.0
Fig.16	90.5	120.6	150.7	22.5	69.1	76.0	5.5	8.4	7.5
Fig.12(Left)	294.5	390.0	447.8	119.4	236.0	246.0	11.3	25.6	22.7
Fig.12(Middle)	-	-	-	-	-	-	62.5	259.9	97.3

Table 1: *Rendering Times (in milliseconds) averaged over full rotations around the object, at 2048x2048.*

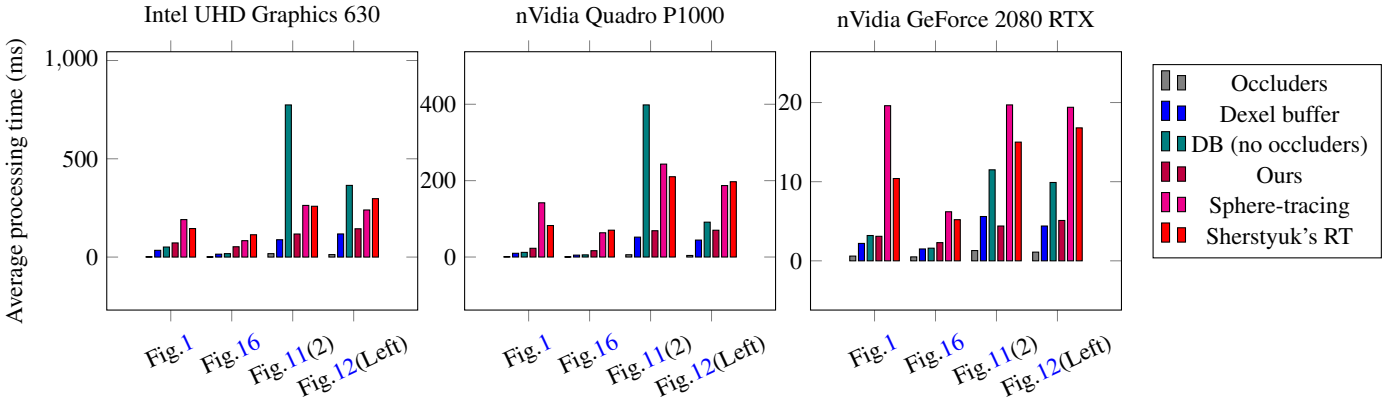


Figure 19: *Individual times for each substep of the methods (occluding depth buffer, dexel buffer creation with and without occluders and ray processing). For models presenting a large number of depth layers, occluders provide non negligible improvement of rendering time.*

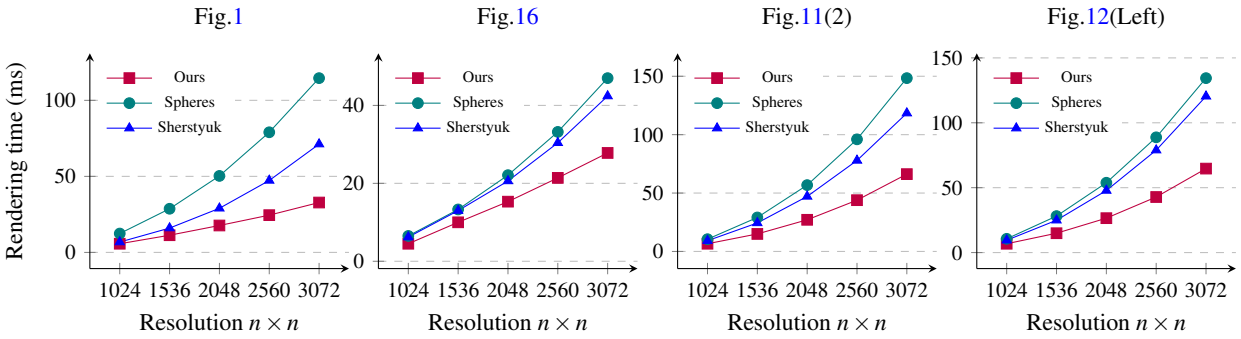


Figure 20: *Average frame processing time versus rendering resolution on a nVidia GeForce 2080 RTX.*

Object	# of Skeletons	Our Method			Sphere Tracing		
		Avg	Median	Max	Avg	Median	Max
Fig.12(Left)	4722	5	4	37	16	11	24580
Fig.11(1)	255	5	4	80	18	13	2389
Fig.11(3)	201	6	6	84	19	14	2124
Fig.15(Right)	127	5	5	27	36	24	2980
Fig.1	213	5	5	132	68	22	42214
Fig.15(Left)	55	5	5	23	73	48	8298
Fig.11(4)	201	8	7	84	413	330	21770
Fig.13(Middle)	95	5	4	23	31	23	4866
Fig.15(Middle)	55	5	5	21	37	20	7559
Fig.2	36	4	4	34	28	18	4372
Fig.16	608	5	4	46	16	10	8345

Table 2: *Statistics calculated for our method and sphere tracing for Compact Polynomial. Similar results are calculated for the Cauchy kernel with decrease in median number of steps in the range between 36.3% and 97.9%*

Object	# of Skeletons	Our Method			Segment Tracing		
		Avg	Median	Max	Avg	Median	Max
Fig.12(Left)	4722	5	4	53	9	7	383
Fig.11(1)	255	5	5	55	9	8	549
Fig.11(3)	201	5	4	24	9	7	340
Fig.15(Right)	127	4	5	20	16	14	413
Fig.1	213	5	4	65	15	11	736
Fig.15(Left)	55	5	5	16	23	23	302
Fig.11(4)	201	6	4	25	177	197	1680
Fig.13(Middle)	95	4	4	18	15	14	277
Fig.15(Middle)	55	4	6	18	13	16	531
Fig.2	36	4	5	16	12	12	524
Fig.16	608	5	6	74	8	7	348

Table 3: *Statistics calculated for segment tracing[PGMG09] and our method on blob primitives.*

limit A-buffer creation cost and memory usage). Finally, we believe that a combination of an A-buffer acceleration structure and/or a new sampling strategy to existing rendering techniques for more general implicit surfaces such as [GGPP20; Kee20] could provide interesting optimizations.

Acknowledgements

This work was supported by the ANR IMPRIMA(ANR-18-CE46-0004). We thank Sylvain Lefebvre and Salim Perchy for their proofreading. We thank Pierre-Alexandre Hugeron and Jonas Martinez for their help in the creation of the skeletons. All skeletons of free-form shapes have been modeled using the online skeleton-based modeler capsulesketch.org. We thanks Maxime Quiblier for early access to this modeler.

References

[ARM*19] ANGLES, BAPTISTE, REBAIN, DANIEL, MACKLIN, MILES, et al. "VIPER: Volume invariant position-based elastic rods". *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 2.2 (2019), 1–26 1.

[Bli82] BLINN, JAMES F. "A Generalization of Algebraic Surface Drawing". *ACM Trans. Graph.* 1.3 (July 1982), 235–256 2.

[Blo97] BLOOMENTHAL, JULES, ed. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., 1997. ISBN: 155860233X 1, 2.

[Bru19] BRUCKNER, STEFAN. "Dynamic Visibility-Driven Molecular Surfaces". *Computer Graphics Forum*. Vol. 38(2). Wiley Online Library. 2019, 317–329 2, 5, 6, 10, 11, 13.

[BS91] BLOOMENTHAL, JULES and SHOEMAKE, KEN. "Convolution surfaces". *Proceedings SIGGRAPH '91*. ACM, 1991, 251–256 1, 3.

[BV18] BÁLINT, CSABA and VALASEK, GÁBOR. "Accelerating Sphere Tracing". *Proceedings of the 39th Annual European Association for Computer Graphics Conference: Short Papers*. EG. Delft, The Netherlands: Eurographics Association, 2018, 29–32. URL: http://dl.acm.org/citation.cfm?id=3308470.3308480 5.

[CHMS00] CAPRANI, OLE, HVIDEGAARD, LARS, MORTENSEN, MIKKEL, and SCHNEIDER, THOMAS. "Robust and Efficient Ray Intersection of Implicit Surfaces". *Reliable Computing* 6.1 (Feb. 2000), 9–21. DOI: 10.1023/A:1009921806032. URL: https://doi.org/10.1023/A:1009921806032 5.

[FHZ19] FUENTES SUÁREZ, ALVARO JAVIER, HUBERT, EVELYNE, and ZANNI, CÉDRIC. "Anisotropic convolution surfaces". *Computers and Graphics* 82 (2019), 106–116. DOI: 10.1016/j.cag.2019.05.018. URL: https://hal.inria.fr/hal-02137325 14.

[FPC10] FRYAZINOV, OLEG, PASKO, ALEXANDER A., and COMNINOS, PETER. "Fast reliable interrogation of procedurally defined implicit surfaces using extended revised affine arithmetic". *Computers & Graphics* 34 (2010), 708–718 5.

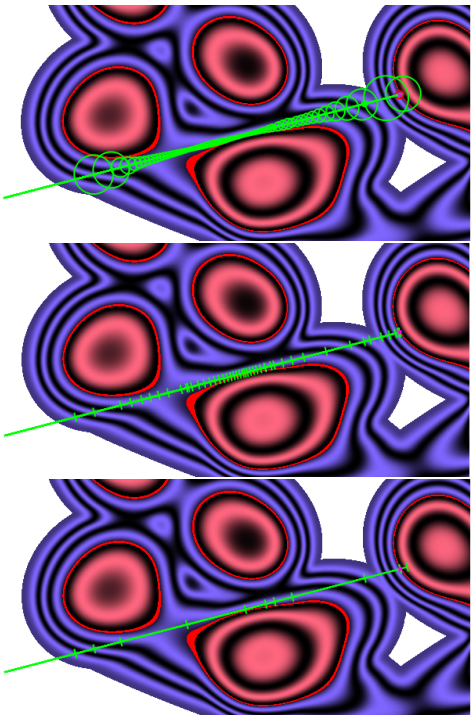


Figure 21: *Comparison of grazing rays in a slice of the scene of Figure 11 (far left) for sphere-tracing, segment-tracing and our method. Note that homothetic distance primitives (Equation (7)) are used for this figure in order to allow direct comparison of the three methods.*

[GGP*15] GÉNEVAUX, JEAN-DAVID, GALIN, ERIC, PEYTAVIE, ADRIEN, et al. "Terrain Modelling from Feature Primitives". *Computer Graphics Forum* 34.6 (May 2015), 198–210. DOI: 10.1111/cgf.12530. URL: https://hal.archives-ouvertes.fr/hal-01257198 5.

[GGPP20] GALIN, ERIC, GUÉRIN, ERIC, PARIS, AXEL, and PEYTAVIE, ADRIEN. "Segment Tracing Using Local Lipschitz Bounds". *Computer Graphics Forum* (2020). URL: https://hal.archives-ouvertes.fr/hal-02507361 5, 13, 16.

[GPP*10] GOURMEL, OLIVIER, PAJOT, ANTHONY, PAULIN, MATHIAS, et al. "Fitted BVH for Fast Raytracing of Metaballs". *Computer Graphics Forum* 29.2 (May 2010), xxx–xxx 4, 5.

[Har96] HART, JOHN C. "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces". *The Visual Computer* 12.10 (Dec. 1996), 527–545. ISSN: 1432-2315. DOI: 10.1007/s003710050084. URL: https://doi.org/10.1007/s003710050084 4.

[JLW10] JI, ZHONGPING, LIU, LIGANG, and WANG, YIGANG. "B-Mesh: A Modeling System for Base Meshes of 3D Articulated Shapes". *Computer Graphics Forum* 29.7 (2010), 2169–2177. DOI: 10.1111/j.1467-8659.2010.01805.x 1.

[JTZ09] JIN, XIAOGANG, TAI, CHIEW-LAN, and ZHANG, HAILIN. "Implicit modeling from polygon soup using convolution". *Vis. Comput.* 25.3 (Feb. 2009), 279–288. ISSN: 0178-2789. DOI: 10.1007/s00371-008-0267-3 4.

[KB89] KALRA, D. and BARR, A. H. "Guaranteed Ray Intersections with Implicit Surfaces". *SIGGRAPH Comput. Graph.* 23.3 (July 1989), 297–306. ISSN: 0097-8930. DOI: 10.1145/74334.74364. URL: http://doi.acm.org/10.1145/74334.74364 4.

[Kee20] KEETER, MATTHEW J. "Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces". *Proceedings of SIGGRAPH*. 2020 5, 16.

[KSK*14] KEINERT, BENJAMIN, SCHÄFER, HENRY, KORNDÖRFER, JOHANN, et al. "Enhanced Sphere Tracing". *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. Ed. by GIACHETTI, ANDREA. The Eurographics Association, 2014. ISBN: 978-3-905674-72-9. DOI: 10.2312/stag.20141233 5.

[KSN08] KANAMORI, YOSHIHIRO, SZEGO, ZOLTAN, and NISHITA, TOMOYUKI. "GPU-based Fast Ray Casting for a Large Number of Metaballs". *Comput. Graph. Forum* 27 (Apr. 2008), 351–360. DOI: 10.1111/j.1467-8659.2008.01132.x 4.

[LHL14] LEFEBVRE, SYLVAIN, HORNUS, SAMUEL, and LASRAM, ANASS. "Per-Pixel Lists for Single Pass A-Buffer". *GPU Pro 5: Advanced Rendering Techniques*. Ed. by ENGEL, WOLFGANG. A K Peter / CRC Press, Mar. 2014. URL: https://hal.inria.fr/hal-01093158 10.

[Lin91] LINDEBERG, TONY. "Discrete Scale-Space Theory and the Scale-Space Primal Sketch". PhD thesis. 1991 7.

[LYHG17] LAN, LEI, YAO, JUNFENG, HUANG, PING, and GUO, XIAOHU. "Medial-axis-driven shape deformation with volume preservation". *Visual Computer* 33 (May 2017), 1–12. DOI: 10.1007/s00371-017-1401-x 1.

[MCTB11] MAULE, MARILENA, COMBA, JOÃO LUIZ DIHL, TORCHELSEN, RAFAEL P., and BASTOS, RUI. "A survey of raster-based transparency techniques". *Computers & Graphics* 35.6 (2011), 1023–1034 2, 10.

[MCTB12] MAULE, M., COMBA, J.L.D., TORCHELSEN, R., and BASTOS, R. "Memory-Efficient Order-Independent Transparency with Dynamic Fragment Buffer". *Proc. 25th Conference on Graphics, Patterns and Images (SIBGRAPI)*. IEEE. 2012, 134–141 2, 10.

[Mit90] MITCHELL, DON P. "Robust ray intersection with interval arithmetic". 1990 5.

[Mol20] MOLECULE, MEDIA. *Dreams*. 2020. URL: http://dreams.mediamolecule.com/ 5.

[NHK*85] NISHIMURA, H., HIRAI, M., KAWAI, T., et al. "Object modeling by distribution function and a method of image generation". *Transactions IECE Japan* 4 (1985), 718–725 2.

[NN94] NISHITA, TOMOYUKI and NAKAMAE, EIHACHIRO. "A Method for Displaying Metaballs by using Bezier Clipping". *Comput. Graph. Forum* 13 (Aug. 1994), 271–280. DOI: 10.1111/1467-8659.1330271 4.

[Ord18] ORDER, SECOND. *Claybook*. 2018. URL: https://www.claybookgame.com/ 5.

[PASS95] PASKO, A., ADZHIEV, V., SOURIN, A., and SAVCHENKO, V. "Function representation in geometric modeling: concepts, implementation and applications". *The Visual Computer* 11.8 (1995), 429–446 2.

[PGMG09] PEYTAVIE, ADRIEN, GALIN, ERIC, MERILLOU, STEPHANE, and GROSJEAN, JEROME. "Arches: a Framework for Modeling Complex Terrains". en. *Computer Graphics Forum (Proceedings of Eurographics)* 28.2 (2009), 457–467 16.

[PIX10] PIXOLOGIC. *Zbrush*. 2010. URL: http://www.zbrush.com 1.

[QJ13] QUILEZ, INIGO and JEREMIAS, POL. *Shadertoy*. 2013. URL: https://www.shadertoy.com/ 4.

[Ric73] RICCI, A. "Constructive geometry for computer graphics". *Computer Journal* 16.2 (1973), 157–60 13.

[SF14] SCHOLLMEYER, A. and FROEHLICH, B. "Direct Isosurface Ray Casting of NURBS-Based Isogeometric Analysis". *IEEE Transactions on Visualization and Computer Graphics* 20.9 (2014), 1227–1240 10.

[She99] SHERSTYUK, ANDREI. "Fast Ray Tracing of Implicit Surfaces". *Comput. Graph. Forum* 18 (1999), 139–147 2, 4, 13.

[SJNJ19] SEYB, DARIO, JACOBSON, ALEC, NOWROUZEZAHRAI, DEREK, and JAROSZ, WOJCIECH. "Non-linear sphere tracing for rendering deformed signed distance fields". *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38.6 (Nov. 2019). DOI: 10.1145/3355089.3356502. in press 5.

[SWBG06] SIGG, CHRISTIAN, WEYRICH, TIM, BOTSCH, MARIO, and GROSS, MARKUS. "GPU-based Ray-casting of Quadratic Surfaces". *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*. SPBG'06. Boston, Massachusetts: Eurographics Association, 2006, 59–65. ISBN: 3-905673-32-0. DOI: 10.2312/SPBG/SPBG06/059-065. URL: http://dx.doi.org/10.2312/SPBG/SPBG06/059-065 5, 10.

[SWR18] SINGH, JAG MOHAN, WASNIK, PANKAJ, and RAMACHANDRA, RAGHAVENDRA. "Hessian-based Robust Ray-tracing of Implicit Surfaces on GPU". *SIGGRAPH Asia 2018 Technical Briefs*. SA '18. Tokyo, Japan: ACM, 2018, 16:1–16:4. ISBN: 978-1-4503-6062-3. DOI: 10.1145/3283254.3283287. URL: http://doi.acm.org/10.1145/3283254.3283287 4.

[TDS*16] TAGLIASACCHI, ANDREA, DELAME, THOMAS, SPAGNUOLO, MICHELA, et al. "3D Skeletons: A State-of-the-Art Report". *Computer Graphics Forum* (2016). ISSN: 1467-8659. DOI: 10.1111/cgf.12865 1.

[Thi11] THIBIEROZ, NICOLAS. "Order-Independent Transparency using Per-Pixel Linked Lists". *GPU Pro 2*. Ed. by ENGEL, WOLFGANG. A K Peters, 2011, 409–431 2, 10.

[WGG99] WYVILL, BRIAN, GUY, ANDREW, and GALIN, ERIC. "Extending the CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System". *Computer Graphics Forum* 18.2 (1999), 149–158. ISSN: 1467-8659. DOI: 10.1111/1467-8659.00365 2, 5.

[WMW86] WYVILL, GEOFF, MCPHEETERS, CRAIG, and WYVILL, BRIAN. "Data structure for soft objects". *The Visual Computer* 2.4 (Aug. 1986), 227–234 2.

[ZBQC13] ZANNI, CÉDRIC, BERNHARDT, ADRIEN, QUIBLIER, MAXIME, and CANI, MARIE-PAULE. "SCALe-invariant Integral Surfaces". *Computer Graphics Forum* 32 (8 2013) 1, 3, 6, 8, 14.

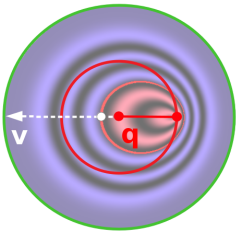
[ZGC15] ZANNI, CÉDRIC, GLEICHER, MICHAEL, and CANI, M-P. "N-ary implicit blends with topology control". *Computers & Graphics* 46 (2015), 1–13 6.

Appendix A: Radii scaling for sphere-cones enclosure

We present here the worst case skeleton configuration used to compute the scaling factor of the sphere-cones' radii that is used for computing the occluding depth buffer.

Usage of the end-point correctors guarantees the existence of a minimal length of skeleton around any skeleton point $\mathbf{q}$ with radius τ (i.e. the length of the correctors on both sides of $\mathbf{q}$) given that $\mathbf{q}$ is not part of an end-point corrector.

We want to find the minimal distance between the isosurface and the point $\mathbf{q}$. Hence, we should study the field variation along arbitrary directions starting from $\mathbf{q}$. For an arbitrary direction $\mathbf{v}$, the minimal distance will be reached for the fastest decreasing field. Such field is obtained if the guaranteed length of skeleton is leaving $\mathbf{q}$ in the direction $-\mathbf{v}$ (kernels are decreasing functions). Similarly the influence of a given skeleton point is decreasing more rapidly for smaller prescribed radii.



Hence, the worst case scenario is reached for two segment primitives in the direction $-\mathbf{v}$ with radius τ in $\mathbf{q}$ and the fastest authorized radius variation (see Figure inset). Thanks to the scale invariance of the surface, we only need to compute the scaling factor for a unit radius which we perform numerically (which corresponds to the distance between $\mathbf{q}$ and the isosurface in the direction $\mathbf{v}$ - see Figure inset). Note that this worst case configuration is actually independent from the actual skeleton configuration, no additional skeleton processing is required.

Appendix B: Lipschitz constant computation

When applying sphere-tracing, computing the Lipschitz constant per primitive (e.g. ignoring the ray/primitive configuration) can have a large impact in presence of varying radius along primitives. Indeed, for the studied implicit surfaces the Lipschitz constant is inversely proportional to the smallest radius. In order to alleviate this problem in our sphere-tracing implementation, we compute the Lipschitz constant both per primitive and per ray. This can be done by computing the primitive point with the smallest radius whose kernel support is intersected by the ray. Based on Equation (18), the kernel support associated to a skeleton point is intersected by the ray if and only if:

$$\|(\mathbf{m}-s\mathbf{u})-(\mathbf{m}-s\mathbf{u})^T\mathbf{d}\,\mathbf{d}\|^2-\sigma^2(\tau_0+s\Delta\tau)^2\leq 0$$

where σ is the kernel scale. The smallest radius can then be deduced from these polynomial roots.