Acta Cybernetica **25** (2021) 555–573.

Footvector Representation of Curves and Surfaces[*]

Gábor Valasek[ab], Csaba Bálint[ac], and András Leitereg[ad]

Abstract

This paper proposes a foot mapping-based representation of curves and surfaces which is a geometric generalization of signed distance functions. We present a first-order characterization of the footvector mapping in terms of the differential geometric invariants of the represented shape and quantify the dependence of the spatial partial derivatives of the footvector mapping with respect to the principal curvatures at the footpoint. The practical applicability of foot mapping representations is highlighted by several fast iterative methods to compute the exact footvector mapping of the offset surface of constructive solid geometry (CSG) trees. The set operations for footpoint mappings are higher-order functions that map a tuple of functions to a single function, which poses a challenge for GPU implementations. We propose a code generation framework to overcome this that transforms CSG trees to the GLSL shader code.

Keywords: computer graphics, constructive solid geometry, signed distance function

1 Introduction

This paper introduces a foot mapping based representation of shapes. The footpoint is the closest point of the shape boundary to an arbitrary query position and the footvector is the displacement from the latter to the footpoint. The proposed representation leverages the footvector mapping of the encoded shape and it is a geometric generalization of signed distance functions.

We present the theoretical background of this approach in Section 3 and show how the differential geometric invariants at the footpoint govern the development of the footvector mapping in space. More specifically, we highlight the connection

[*]EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies — The Project is supported by the Hungarian Government and co-financed by the European Social Fund. Supported by the ÚNKP-20-3 New National Excellence Program of the Ministry for Innovation and Technology.

[a]Eötvös Loránd University, Budapest, Hungary
[b]E-mail: `valasek@inf.elte.hu`, ORCID: 0000-0002-0007-8647
[c]E-mail: `csabix@inf.elte.hu`, ORCID: 0000-0002-5609-6449
[d]E-mail: `leanil@inf.elte.hu`, ORCID: 0000-0003-0705-0214

DOI: `10.14232/actacyb.290145`

between the first-order behaviour of the footvector mapping and the offsets of the shape and, as such, the principal curvatures. These results are presented for curves in the plane and surfaces in space.

Following the theoretical discussion, we introduce a method to create CSG models with precise offset operations using footvector mappings. In particular, we present two iterative algorithms on foot mappings in Section 4 and Section 5 to compute the footvector function of the intersection of two shapes. We discuss the robustness/speed trade-off between these in Section 1 where we apply these methods to the offset of intersections.

In general, implicit surface representations ($g : \mathbb{R}^3 \to \mathbb{R}$) usually describe 'inside' region as the set of negative values ($\{g < 0\} := \{\boldsymbol{x} \in \mathbb{R}^3 : g(\boldsymbol{x}) < 0)$, and the 'outside' as the positive $\{g\rangle 0\} \subseteq \mathbb{R}^3$ region of space. Set operations can be defined with minimum and maximum operations on the argument implicit functions.

A special case of implicit representations are the signed distance functions (SDFs) that map the signed distance from the boundary to every point in space. They offer efficient real-time visualization with sphere tracing [7]. Unfortunately, they are not closed under the min/max representation of set operations, that is the minimum and maximum of SDF arguments may not be a precise SDF. Even though the shape of the surface and the convergence of sphere tracing are invariant under these operations, the loss of the exact SDF property is disadvantageous for the offset operation.

We define the offset surface as the set of points that are r distance away from the original surface. To take an offset of a surface given by a signed distance function, one must only subtract the offset radius [2]. However, if g_1 and $g_2 : \mathbb{R}^3 \to \mathbb{R}$ are SDFs, then the $r > 0$ offset of their intersection is

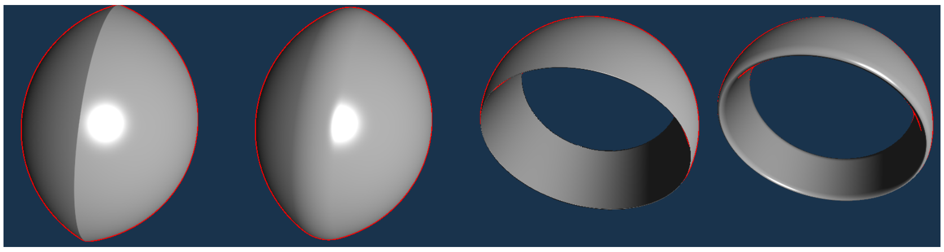
$$\boldsymbol{x} \mapsto \max(g_1(\boldsymbol{x}), g_2(\boldsymbol{x})) - r \iff \boldsymbol{x} \mapsto \max(g_1(\boldsymbol{x}) - r, g_2(\boldsymbol{x}) - r)$$

which is the same as taking their offsets and then their intersection. However, these operations are not supposed to be interchangeable.

For example, the positive offsets of spheres are larger spheres, so their intersection is an intersection of larger spheres with a sharp edge, yet, the offset of the original intersection set supposed to be a pill-shaped oval surface. This is a wrong result as these operations should not be interchangeable. The reason is that the intersection operation is imprecise near-surface edges and corners and the function values do not increase correctly. Our algorithm solves this issue allowing real offset operations on set operation results. Figure 1 demonstrates this key difference between signed distance functions and footvector mappings.

We extend the precision of analytic distance functions for foot mappings by defining the primitives, set operations, and offset on them. The set operations, however, do not operate on a single implicit function value, but operate on whole functions. This means that set operations on footvector mappings are higher order functions that also produce a function from the input functions:

$$\cap, \cup \in (\mathbb{R}^3 \to \mathbb{R}^4) \times (\mathbb{R}^3 \to \mathbb{R}^4) \to (\mathbb{R}^3 \to \mathbb{R}^4) .$$



(a) SDF (b) Foot mapping (c) SDF (d) Foot mapping

Figure 1: The offset of the intersection of two spheres and the offset of a cylinder subtracted from a sphere. The offset operation should leave the surface smooth for which the distance representation is inadequate.

Our representation maps to $\mathbb{R}^4$ because besides the footvector we also have to encode whether the point is inside or outside the geometry. The footvector is assigned to the first three coordinates, and the signed distance value is in the fourth.

There are many other ways to compute the offset surface. Most of these operate on parametric representations, which may not be available in our case. For implicit representations, fast distance transforms are commonly used. These are efficient to execute, but the visualization is not direct because a grid of values must be computed and stored. Thus, these algorithms are limited in accuracy by the available memory. Yet, the resulting distance representation is fast, and offset surfaces can be created quickly making this a viable choice for e.g. text rendering in 2D. Three dimensional distance fields are even more expensive to store which motivated our grid-free approach.

Moreover, methods based on distance transform rely on a discretization of the distance function and marching the distance values by approximating the distance to the surface from the neighbouring values [5, 13] or use simplified proxy geometries to infer a more precise distance value [14]. The local computations introduce errors that can be amplified by set operations, making them worse than the min and max SDF operations. To increase precision and reduce memory usage, we devised iterative algorithms that compute the distance to the intersection set from any point $\boldsymbol{x} \in \mathbb{R}^3$ to the intersection object. In exchange, the presented algorithms require a different representation of the surfaces.

To achieve real-time direct visualization of such offset surfaces, the GPU is required. However, our set operations operate on functions, and we cannot simply pass functions as arguments in shader code, because GPUs only support inlineable functions. For this reason, we implemented a code generator that created the implementation for the footvector mapping from a given CSG tree. The higher-order set operations had to be implemented into the code generator to run our iteration on various argument footvector mappings. These results are presented in Section 6.

2 Preliminaries

From any sample point $\boldsymbol{x} \in \mathbb{R}^n$ a signed distance function (SDF) $g : \mathbb{R}^n \to \mathbb{R}$ is a continuous function that evte aluates to the signed Euclidean distance measured from the surface. That is

$$\big|g(\boldsymbol{x})\big| = d(\boldsymbol{x}, \{g = 0\}) \qquad (\forall\, \boldsymbol{x} \in \mathbb{R}^3),$$

where $d(\boldsymbol{x}, A)$ is the point-to-set distance, and $\{g = 0\}$ is the zero level-set. The dimension is $n = 2$ for the plane and $n = 3$ for 3D space. The sign encodes whether $\boldsymbol{x}$ is inside $\{g \leq 0\}$ or outside $\{g > 0\}$ which allows set-operations to be defined on SDFs. Let us take g_1 and g_2 signed distance functions, then according to [7],

$$\begin{aligned} d\big(\boldsymbol{x}, \{g_1 \leq 0\} \cup \{g_2 \leq 0\}\big) &\geq \big|\min\{g_1(\boldsymbol{x}), g_2(\boldsymbol{x})\}\big| \qquad (\forall\, \boldsymbol{x} \in \mathbb{R}^n)\ , \\ d\big(\boldsymbol{x}, \{g_1 \leq 0\} \cap \{g_2 \leq 0\}\big) &\geq \big|\max\{g_1(\boldsymbol{x}), g_2(\boldsymbol{x})\}\big| \qquad (\forall\, \boldsymbol{x} \in \mathbb{R}^n)\ . \end{aligned} \tag{1}$$

The $\min(g_1, g_2) = \boldsymbol{x} \mapsto \min\{g_1(\boldsymbol{x}), g_2(\boldsymbol{x})\}$ function is an implicit function of the union of $\{g_1 \leq 0\}$ and $\{g_2 \leq 0\}$ objects. The resulting function is a good lower-estimate of the distance on the inside of the union, whereas it is exact on the outside of the union. For this reason, many SDF representations use min and max operations to combine primitive geometries into complex scenes [6, 8]. However, the approximation is imprecise on the union for the min operation, and only exact within the intersection for the max intersection SDF approximation.

The precision can be quantified for any $g : \mathbb{R}^n \to \mathbb{R}$ SDF by comparing the real distance-to-surface value to that of the function:

$$q_g(\boldsymbol{x}) := \frac{\big|g(\boldsymbol{x})\big|}{d(\boldsymbol{x}, \{g = 0\})} \qquad (\forall\, \boldsymbol{x} \in \mathbb{R}^n) \tag{2}$$

Signed distance function estimates (SDFEs) are defined using the above local precision. If there exists a $c > 0$ global precision such that $0 < c \leq q_g(\boldsymbol{p}) \leq 1$, then $g : \mathbb{R}^n \to \mathbb{R}$ is a signed distance function estimate.

Distance representations can be directly ray-traced via various sphere tracing algorithms [1, 3, 7, 10]. The precision of the SDFE measures the slowdown of the sphere tracing algorithm; however, computing $q_{\max(g_1,g_2)}(\boldsymbol{x})$ for the intersection operation can be expensive.

The function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a footvector mapping if

1. $\boldsymbol{x} \mapsto \big\|\boldsymbol{f}(\boldsymbol{x})\big\|$ $(x \in \mathbb{R}^n)$ is a distance function
2. $\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{f}(\boldsymbol{x})) = \boldsymbol{0}$ for all $\boldsymbol{x} \in \mathbb{R}^n$

This means that $\boldsymbol{f}$ returns a vector pointing to one of the closest points on the surface it defines. Thus $\boldsymbol{x} + \boldsymbol{f}(\boldsymbol{x})$ is the corresponding footpoint, but note that the $\boldsymbol{f}$ function is not unique for a given shape. This is because some points will have more then one closest points from the geometry, each providing an allowed return value for the $\boldsymbol{f}$ function.

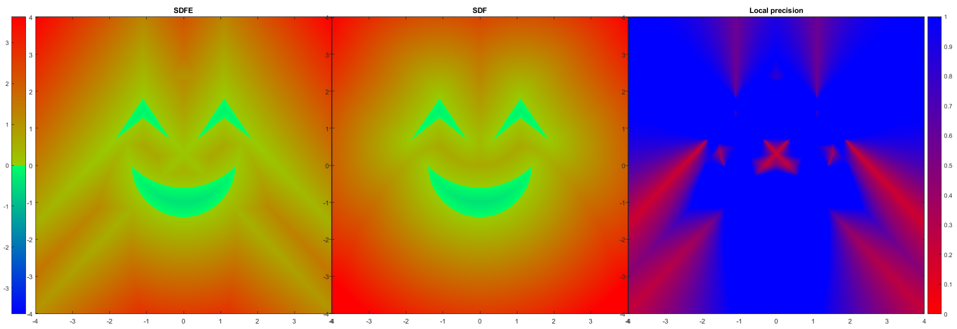


Figure 2: Left: 2D SDFE obtained through min and max set operations using transformations of a half-plane (line) and a circle primitive. Local precision is the ratio of the SDFE (left) and the exact SDF (middle) is displayed on the right signaling the slowdown of sphere tracing. Our footpoint iteration produced the middle image.

Similarly to SDFs, the footpoint representation requres volumetric, i.e. solid geometric information for the set-operations to be defined. Let us assume we can decide if $\boldsymbol{x}$ is inside $\boldsymbol{x} \in G \subseteq \mathbb{R}^n$ closed set or outside $\boldsymbol{x} \in \mathbb{R}^n \setminus G$, where the boundary set is $\partial G = \{\boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{f}(\boldsymbol{x})\| = 0\} \subseteq G$. For example, having the corresponding signed distance function $g : \mathbb{R}^n \to \mathbb{R}$ such that $G = \{g \leq 0\}$ and $\{\boldsymbol{f} = 0\} = \{g = 0\} = \partial G$ will allow the set operations.

3 Differential geometry and footpoint mappings

In this section, we investigate the relation between the derivatives of the footpoint mapping and the local differential geometry at the footpoint.

Let $G \subseteq \mathbb{R}^n$ denote a geometry of interest in either the plane or space, i.e. $n = 2, 3$, and $\operatorname{int} G \subseteq G$ its interior points. Let us assume that its boundary, ∂G, is a sufficiently smooth set in the sense of geometric continuity, that is, it can be covered by local parametrizations of the desired parametric smoothness [4].

The stationary condition of distance [12] states that the vector from the query position $\boldsymbol{x}$ to the footpoint $\boldsymbol{x}^*$ is perpendicular to the tangent entity of the shape, which is the tangent line in plane and the tangent plane in space. In other words, this means that the footvector mapping $\boldsymbol{f} = \boldsymbol{x}^* - \boldsymbol{x}$ is parallel to the curve or surface normal $\boldsymbol{n}^*$ at the footpoint $\boldsymbol{x}^*$.

Let $\boldsymbol{c}(t) : \mathbb{R} \to \mathbb{R}^2$ and $\boldsymbol{s}(u, v) : \mathbb{R}^2 \to \mathbb{R}^3$ be the parametric representation of ∂G in the plane and in space, respectively. In both cases, we denote any suitable implicit representation of G by $g : \mathbb{R}^n \to \mathbb{R}$ $(n = 2, 3)$. Using these notations, the perpendicularity of the footvector to ∂G is expressed as:

representation	**plane**	**space**
parametric	$\boldsymbol{f} \cdot \boldsymbol{c}' = 0$	$\begin{cases} \boldsymbol{f} \cdot \boldsymbol{s}_u = 0 \\ \boldsymbol{f} \cdot \boldsymbol{s}_v = 0 \end{cases}$
implicit	$\boldsymbol{f} \times \nabla g = \mathbf{0}$	$\boldsymbol{f} \times (\boldsymbol{s}_u \times \boldsymbol{s}_v) = \mathbf{0}$

where $\boldsymbol{c}'$ denotes differentiation with respect to the particular curve parameter t, $\boldsymbol{s}_u = \partial_u \boldsymbol{s}$, and $\boldsymbol{s}_v = \partial_v \boldsymbol{s}$.

A more intuitive characterization of the footvector mapping comes from observing the behaviour of its first degree Taylor expansion. Using $\boldsymbol{x} = [x, y, z]^T, \boldsymbol{x}_0 = [x_0, y_0, z_0]^T$, this is formally

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) + (x - x_0)\partial_1 \boldsymbol{f}(\boldsymbol{x}_0) + (y - y_0)\partial_2 \boldsymbol{f}(\boldsymbol{x}_0) + (z - z_0)\partial_3 \boldsymbol{f}(\boldsymbol{x}_0) \ .$$

Our goal is to quantify the interplay between the footpoint mapping and the differential geometry at the footpoint. The most natural setting for the study of the local differential geometry of shapes is the parametric representation. As such, in the following two subsections we derive our results for parametric curves in the plane and parametric surfaces in space.

3.1 Derivatives of the footpoint mapping: $\mathbb{R}^2$

In the case of parametric plane curves, the footvector mapping $\boldsymbol{f} : \mathbb{R}^2 \to \mathbb{R}^2$ is decomposed as

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) &= \boldsymbol{c}(t(\boldsymbol{x})) - \boldsymbol{x} \\ &= \boldsymbol{c} \circ t - \mathbf{id} \ , \end{aligned} \tag{3}$$

where $\boldsymbol{c} : \mathbb{R} \to \mathbb{R}^2$ is a parametrization of the boundary, $\boldsymbol{x} = [x, y]^T$ is the query position, and $t : \mathbb{R}^2 \to \mathbb{R}$ is the footparameter mapping that assigns the parameter value corresponding to the closest curve point to $\boldsymbol{x}$. In the second equation, $\mathbf{id}$ denotes the identity mapping of $\mathbb{R}^n$, i.e. $\partial_1 \mathbf{id} = \boldsymbol{e}_1$, $\partial_2 \mathbf{id} = \boldsymbol{e}_2$, where $\boldsymbol{e}_i$ are the canonical basis vectors of $\mathbb{R}^2$.

Let $i \in \{1, 2\}$ denote an arbitrary coordinate axis and let us take the partial derivative of $\boldsymbol{f} \cdot \boldsymbol{c}' = 0$ with respect to i as

$$\begin{aligned} 0 &= \partial_i \Big(\big(\boldsymbol{c} \circ t - \mathbf{id}\big) \cdot \boldsymbol{c}' \circ t \Big) \\ &= \partial_i \Big(\boldsymbol{c} \circ t - \mathbf{id} \Big) \cdot \boldsymbol{c}' \circ t + \Big(\boldsymbol{c} \circ t - \mathbf{id} \Big) \cdot \partial_i \boldsymbol{c}' \circ t \\ &= \Big(\boldsymbol{c}' \circ t \cdot \partial_i t - \boldsymbol{e}_i \Big) \cdot \boldsymbol{c}' \circ t + \Big(\boldsymbol{c} \circ t - \mathbf{id} \Big) \cdot \boldsymbol{c}'' \circ t \cdot \partial_i t \end{aligned}$$

After rearrangement, the partial derivative of the footparameter mapping is

$$\partial_i t = \frac{\boldsymbol{c}' \circ t \cdot \boldsymbol{e}_i}{\boldsymbol{c}' \circ t \cdot \boldsymbol{c}' \circ t + \boldsymbol{f} \cdot \boldsymbol{c}'' \circ t} \ .$$

Omitting the point of evaluation, the gradient of the footparameter mapping is

$$\nabla t = \frac{1}{\boldsymbol{c}' \cdot \boldsymbol{c}' + \boldsymbol{f} \cdot \boldsymbol{c}''} \boldsymbol{c}'$$

proving the simple intuition that the largest change in the footparameter happens when the query position is displaced parallel to the tangent line at the footpoint. More importantly, this equation also quantifies the amount of the largest change.

To give a geometric interpretation to this, let us consider the $\widehat{\boldsymbol{c}} : [0, L] \to \mathbb{R}^2$ arc-length parametrization of the boundary. This is done without loss of generality. The gradient of the footparameter mapping is then

$$\nabla t = \frac{1}{1 + \boldsymbol{f} \cdot \kappa \widehat{\boldsymbol{n}}} \widehat{\boldsymbol{t}} \ ,$$

where $\widehat{\boldsymbol{t}}, \widehat{\boldsymbol{n}}$ denote the Frenet frame unit tangent and normal vectors of the curve and κ is its curvature function. Noting that $\boldsymbol{f} \parallel \widehat{\boldsymbol{n}}$ and thus can be expressed as $\boldsymbol{f} = d \cdot \widehat{\boldsymbol{n}}$ for some $d \in \mathbb{R}$, the above becomes

$$\nabla t = \frac{1}{1 + d\kappa} \widehat{\boldsymbol{t}} \tag{4}$$

that is, a unit displacement of the query position along $\widehat{\boldsymbol{t}}$ results in a $\frac{1}{1+d\kappa}$ displacement in the footpoint parameter.

Equation (4) is also remarkable in the sense that it shows that the change in the footpoint parameter is the reciprocal of the change of the parametric speed of the original boundary curve offset by d. In Kallay's terms [9], it is a first order pull-back onto the parametrization of the offset curve.

To translate our results on the footparameter mapping to the footpoint mapping, let us consider the first degree Taylor expansion of Eq. (3) as

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) + (x - x_0)\partial_x \boldsymbol{f}(\boldsymbol{x}_0) + (y - y_0)\partial_y \boldsymbol{f}(\boldsymbol{x}_0) \ .$$

From Eq. (3), the partial derivatives of $\boldsymbol{f}$ are

$$\partial_i \boldsymbol{f} = \boldsymbol{c}' \circ t \cdot \partial_i t - \boldsymbol{e}_i \ ,$$

that is,

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) &\approx \boldsymbol{f}(\boldsymbol{x}_0) + (x - x_0)\big(\boldsymbol{c}'(t(\boldsymbol{x}_0))\partial_1 t(\boldsymbol{x}_0) - \boldsymbol{e}_1\big) + (y - y_0)\big(\boldsymbol{c}'(t(\boldsymbol{x}_0))\partial_2 t(\boldsymbol{x}_0) - \boldsymbol{e}_2\big) \\ &= \boldsymbol{f}(\boldsymbol{x}_0) - (\boldsymbol{x} - \boldsymbol{x}_0) + (x - x_0)\boldsymbol{c}'(t(\boldsymbol{x}_0))\partial_1 t(\boldsymbol{x}_0) + (y - y_0)\boldsymbol{c}'(t(\boldsymbol{x}_0))\partial_2 t(\boldsymbol{x}_0) \end{aligned}$$

This is written more succinctly omitting the evaluation points, denoting $\boldsymbol{f}_0 = \boldsymbol{f}(\boldsymbol{x}_0)$, $\nabla t_0 = \nabla t(\boldsymbol{x}_0)$, $\boldsymbol{c}_0' = \boldsymbol{c}'(t(\boldsymbol{x}_0))$, and substituting Eq. (3) as

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}_0 - (\boldsymbol{x} - \boldsymbol{x}_0) + \big((\boldsymbol{x} - \boldsymbol{x}_0) \cdot \nabla t_0\big)\boldsymbol{c}_0' \ ,$$

or, using arc-length parametrization,

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}_0 - (\boldsymbol{x} - \boldsymbol{x}_0) + \frac{(\boldsymbol{x} - \boldsymbol{x}_0) \cdot \widehat{\boldsymbol{t}}_0}{1 + d\kappa_0} \widehat{\boldsymbol{t}}_0 \ .$$

According to this, there is no change in the footpoint if $\boldsymbol{x} - \boldsymbol{x}_0$ is perpendicular to $\widehat{\boldsymbol{t}}$, but this only holds as long as the footpoint mapping is a function, that is, the footpoint is unique. As soon as we reach the cut locus, i.e. a point in the plane that has multiple closest points, the footpoint mapping becomes discontinuous and the footpoint can change arbitrarily along the circumference of the unbounding circle.

3.2 Derivatives of the footpoint mapping: $\mathbb{R}^3$

Now, let us consider the case when the geometry is a volume and $\boldsymbol{s} : \mathbb{R}^2 \to \mathbb{R}^3$ is the parametric representation of its boundary surface. The stationary condition of the footpoint is written as

$$\boldsymbol{f} \cdot \boldsymbol{s}_u = 0 \tag{5}$$

$$\boldsymbol{f} \cdot \boldsymbol{s}_v = 0 \ , \tag{6}$$

where the footpoint mapping is decomposed as

$$\boldsymbol{f} = \boldsymbol{s} \circ \boldsymbol{u} - \mathbf{id} \ , \tag{7}$$

that is,

$$\boldsymbol{f}(x, y, z) = \boldsymbol{s}(u(x, y, z), v(x, y, z))$$

with the two footparemeter mappings $u, v : \mathbb{R}^3 \to \mathbb{R}$, $\boldsymbol{u} = (u, v)$.

Differentiating Eq. (5) with respect to an arbitrary coordinate axis $i \in \{1, 2, 3\}$ yields

$$\begin{aligned} \partial_i(\boldsymbol{f} \cdot \boldsymbol{s}_u \circ \boldsymbol{u}) &= \partial_i\big((\boldsymbol{s} \circ \boldsymbol{u} - \mathbf{id}) \cdot \boldsymbol{s}_u \circ \boldsymbol{u}\big) \\ &= (\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ u \cdot \partial_i v - \boldsymbol{e}_i) \cdot \boldsymbol{s}_u \circ \boldsymbol{u} \\ &\quad + \boldsymbol{f} \cdot \big(\boldsymbol{s}_{uu} \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_{uv} \circ \boldsymbol{u} \cdot \partial_i v\big) \end{aligned}$$

Similarly, differentiating Eq. (6) is

$$\begin{aligned} \partial_i(\boldsymbol{f} \cdot \boldsymbol{p}_v \circ \boldsymbol{u}) &= (\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ u \cdot \partial_i v - \boldsymbol{e}_i) \cdot \boldsymbol{s}_v \circ \boldsymbol{u} \\ &\quad + \boldsymbol{f} \cdot \big(\boldsymbol{s}_{uv} \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_{vv} \circ \boldsymbol{u} \cdot \partial_i v\big) \end{aligned}$$

As such, the partial derivatives of the footparameter mappings with respect to the coordinate axes of $\mathbb{R}^3$ are found from the following system of six linear equations in the unknowns $\partial_i u, \partial_i v$, $i \in \{1, 2, 3\}$:

$$\begin{aligned} (\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ u \cdot \partial_i v - \boldsymbol{e}_i) \cdot \boldsymbol{s}_u \circ \boldsymbol{u} + \boldsymbol{f} \cdot \big(\boldsymbol{s}_{uu} \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_{uv} \circ \boldsymbol{u} \cdot \partial_i v\big) = 0 \\ (\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ u \cdot \partial_i v - \boldsymbol{e}_i) \cdot \boldsymbol{s}_v \circ \boldsymbol{u} + \boldsymbol{f} \cdot \big(\boldsymbol{s}_{uv} \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_{vv} \circ \boldsymbol{u} \cdot \partial_i v\big) = 0 \end{aligned}$$

Recalling that the footvector mappings are perpendicular to the tangent plane, we can make the substitution $\boldsymbol{f} = d \cdot \boldsymbol{m}$, where $d \in \mathbb{R}$ and $\boldsymbol{m}$ is the unit surface normal of $\boldsymbol{s}$, e.g. $\boldsymbol{m} = \frac{\boldsymbol{s}_u \times \boldsymbol{s}_v}{||\boldsymbol{s}_u \times \boldsymbol{s}_v||_2}$ at regular points of $\boldsymbol{s}$. The resulting system takes its final form as

$$\begin{aligned} \mathbb{E} \cdot \partial_i u + \mathbb{F} \cdot \partial_i v - \boldsymbol{e}_i \cdot \boldsymbol{s}_u \circ \boldsymbol{u} + d \cdot \mathbb{L} \cdot \partial_i u + d \cdot \mathbb{M} \cdot \partial_i v = 0 \\ \mathbb{F} \cdot \partial_i u + \mathbb{G} \cdot \partial_i v - \boldsymbol{e}_i \cdot \boldsymbol{s}_v \circ \boldsymbol{u} + d \cdot \mathbb{M} \cdot \partial_i u + d \cdot \mathbb{N} \cdot \partial_i v = 0 \end{aligned} \tag{8}$$

using the first and second fundamental forms of

$$\mathbb{E} = \boldsymbol{s}_u \cdot \boldsymbol{s}_u \ , \ \mathbb{F} = \boldsymbol{s}_u \cdot \boldsymbol{s}_v \ , \ \mathbb{G} = \boldsymbol{s}_v \cdot \boldsymbol{s}_v \ ,$$
$$\mathbb{L} = \boldsymbol{s}_{uu} \cdot \boldsymbol{m} \ , \ \mathbb{M} = \boldsymbol{s}_{uv} \cdot \boldsymbol{m} \ , \ \mathbb{N} = \boldsymbol{s}_{vv} \cdot \boldsymbol{m} \ .$$

It is possible to derive a concise solution to the system of six equations in Eq. (8) by using a special parametrization of $\boldsymbol{s}$ that is analogous to the arc-length or natural parametrization of curves. This parameterization takes the lines of curvature as the u, v parameter axes where the parameter lines are also arc-length. This can be done without loss of generality because lines of curvatures cover the surfaces simply, without gaps [11]. Let us denote this parametrization by $\widehat{\boldsymbol{s}}$.

The normal curvature of a curve on the surface with tangent vector $a \cdot \boldsymbol{s}_u + b \cdot \boldsymbol{s}_v$ is

$$\kappa(a,b) = \frac{\mathbb{L} \cdot a^2 + 2 \cdot \mathbb{M} \cdot a \cdot b + \mathbb{N} \cdot b^2}{\mathbb{E} \cdot a^2 + 2 \cdot \mathbb{F} \cdot a \cdot b + \mathbb{G} \cdot b^2} \tag{9}$$

If the surface is parameterized as $\widehat{\boldsymbol{s}}$, then

$$\mathbb{E} = 1 \ , \ \mathbb{F} = 0 \ , \ \mathbb{G} = 1 \ ,$$
$$\mathbb{L} = \kappa_1 \ , \ \mathbb{M} = 0 \ , \ \mathbb{N} = \kappa_2 \ ,$$

where κ_1, κ_2 are the principal curvature functions, i.e. the minima and maxima of normal section curvatures of Eq. (9) at every point on the surface.

Now, the system of equations in Eq. (8) simplifies to

$$\partial_i u \cdot (1 + d \cdot \kappa_1) = \boldsymbol{e}_i \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}$$
$$\partial_i v \cdot (1 + d \cdot \kappa_2) = \boldsymbol{e}_i \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}$$

giving us the partial derivatives of the footparameter mappings as a function of distance from the surface and the geometric invariants of the surface at the footpoint:

$$\partial_i u = \frac{\boldsymbol{e}_i \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} \qquad \partial_i v = \frac{\boldsymbol{e}_i \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} \tag{10}$$

Let us consider the first degree Taylor expansion of the footvector mapping as

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) + (x - x_0)\boldsymbol{f}_x(\boldsymbol{x}_0) + (y - y_0)\boldsymbol{f}_y(\boldsymbol{x}_0) + (z - z_0)\boldsymbol{f}_z(\boldsymbol{x}_0) \ ,$$

where, using Eq. (7), the $i \in \{1, 2, 3\}$ partial derivatives of $\boldsymbol{f}$ are

$$\boldsymbol{f}_i = \boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ \boldsymbol{u} \cdot \partial_i v - \boldsymbol{e}_i \ .$$

Substituting this into the Taylor expansion, we get

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) &+ (x - x_0)\Big(\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_1 u + \boldsymbol{s}_v \circ \boldsymbol{u} \cdot \partial_1 v - \boldsymbol{e}_1\Big) \\ &+ (y - y_0)\Big(\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_2 u + \boldsymbol{s}_v \circ \boldsymbol{u} \cdot \partial_2 v - \boldsymbol{e}_2\Big) \\ &+ (z - z_0)\Big(\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_3 u + \boldsymbol{s}_v \circ \boldsymbol{u} \cdot \partial_3 v - \boldsymbol{e}_3\Big) \end{aligned}$$

Assuming a natural parametrization of the surface, this is further developed using Eq. (10) as

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) &+ (x - x_0)\left(\widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_1 \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} + \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_1 \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} - \boldsymbol{e}_1\right) \\ &+ (y - y_0)\left(\widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_2 \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} + \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_2 \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} - \boldsymbol{e}_2\right) \\ &+ (z - z_0)\left(\widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_3 \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} + \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_3 \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} - \boldsymbol{e}_3\right)\end{aligned}$$

or, using the notational shorthands $\widehat{s}_{ui} = \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \boldsymbol{e}_i$, $\widehat{s}_{vi} = \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \boldsymbol{e}_i$,

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) - (\boldsymbol{x} - \boldsymbol{x}_0) + \sum_{i=1}^{3}((\boldsymbol{x} - \boldsymbol{x}_0) \cdot \boldsymbol{e}_i)\left(\widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \frac{\widehat{s}_{ui}}{1 + d \cdot \kappa_1} + \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \frac{\widehat{s}_{vi}}{1 + d \cdot \kappa_2}\right).$$

The first two terms of the approximation are interpreted similarly to the curve case. Since $\boldsymbol{f}(\boldsymbol{x}_0) = \boldsymbol{x}^* - \boldsymbol{x}_0$, $\boldsymbol{f}(\boldsymbol{x}_0) - (\boldsymbol{x} - \boldsymbol{x}_0) = \boldsymbol{x}^* - \boldsymbol{x}$, i.e. the approximation starts from the vector pointing to the footpoint of $\boldsymbol{x}_0$ from the new position $\boldsymbol{x}$. To distill the geometric meaning of the sum in the above approximation, let us rewrite $\boldsymbol{x} - \boldsymbol{x}_0$ in the spherical coordinate system about $\boldsymbol{x}_0$ with axes $\widehat{\boldsymbol{s}}_u$, $\widehat{\boldsymbol{s}}_v$, $\widehat{\boldsymbol{m}}$ as

$$\begin{aligned}\boldsymbol{x} - \boldsymbol{x}_0 &= l \cdot \left(\cos\alpha \sin\beta \cdot \widehat{\boldsymbol{s}}_u + \sin\alpha \sin\beta \cdot \widehat{\boldsymbol{s}}_v + \cos\beta \cdot \widehat{\boldsymbol{m}}\right) \\ &= l \cdot \left(\Delta x \cdot \widehat{\boldsymbol{s}}_u + \Delta y \cdot \widehat{\boldsymbol{s}}_v + \Delta z \cdot \widehat{\boldsymbol{m}}\right),\end{aligned}$$

where $l \geq 0, \alpha \in [0, 2\pi), \beta \in [0, \pi]$. This yields

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}) &\approx \boldsymbol{x}^* - \boldsymbol{x} + l \cdot \sum_{i=1}^{3}(\Delta x \cdot \widehat{s}_{ui} + \Delta y \cdot \widehat{s}_{vi} + \Delta z \cdot \widehat{m}_i) \cdot \frac{\widehat{s}_{ui}}{1 + d \cdot \kappa_1} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \\ &\quad + l \cdot \sum_{i=1}^{3}(\Delta x \cdot \widehat{s}_{ui} + \Delta y \cdot \widehat{s}_{vi} + \Delta z \cdot \widehat{m}_i) \cdot \frac{\widehat{s}_{vi}}{1 + d \cdot \kappa_2} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \\ &= \boldsymbol{x}^* - \boldsymbol{x} + \left(\frac{l}{1 + d \cdot \kappa_1} \cdot \sum_{i=1}^{3}(\Delta x \cdot \widehat{s}_{ui}^2 + \Delta y \cdot \widehat{s}_{vi}\widehat{s}_{ui} + \Delta z \cdot \widehat{m}_i\widehat{s}_{ui}) \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}\right) \\ &\quad + \left(\frac{l}{1 + d \cdot \kappa_2} \cdot \sum_{i=1}^{3}(\Delta x \cdot \widehat{s}_{ui}\widehat{s}_{vi} + \Delta y \cdot \widehat{s}_{vi}^2 + \Delta z \cdot \widehat{m}_i\widehat{s}_{vi}) \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}\right) \\ &= \boldsymbol{x}^* - \boldsymbol{x} + \frac{l \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} \cdot \left(\Delta x \sum_{i=1}^{3} \widehat{s}_{ui}^2 + \Delta y \sum_{i=1}^{3} \widehat{s}_{vi}\widehat{s}_{ui} + \Delta z \sum_{i=1}^{3} \widehat{m}_i\widehat{s}_{ui}\right) \\ &\quad + \frac{l \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} \cdot \left(\Delta x \sum_{i=1}^{3} \widehat{s}_{ui}\widehat{s}_{vi} + \Delta y \sum_{i=1}^{3} \widehat{s}_{vi}^2 + \Delta z \sum_{i=1}^{3} \widehat{m}_i\widehat{s}_{vi}\right)\end{aligned}$$

Note that $\widehat{\boldsymbol{s}}_u, \widehat{\boldsymbol{s}}_v, \widehat{\boldsymbol{m}}$ form an orthonormal basis, that is, for example $\widehat{\boldsymbol{s}}_u \cdot \widehat{\boldsymbol{s}}_u = \sum_{i=1}^{3} \widehat{s}_{ui}^2 = 1$ and $\widehat{\boldsymbol{s}}_u \cdot \widehat{\boldsymbol{s}}_v = \sum_{i=1}^{3} \widehat{s}_{ui}\widehat{s}_{vi} = 0$. Carrying out the resulting simplifications gives us the final form of the first degree Taylor expansion of the footvector mapping as

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) &\approx \boldsymbol{x}^* - \boldsymbol{x} + \frac{l \cdot \Delta x}{1 + d \cdot \kappa_1} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} + \frac{l \cdot \Delta x}{1 + d \cdot \kappa_2} \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \\ &= \boldsymbol{x}^* - \boldsymbol{x} + \cos\alpha \sin\beta \frac{l}{1 + d \cdot \kappa_1} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} + \sin\alpha \sin\beta \frac{l}{1 + d \cdot \kappa_2} \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \end{aligned} \tag{11}$$

Changing the query position $\boldsymbol{x}$ along the footpoint surface normal $\widehat{\boldsymbol{m}}$, i.e. when $\beta \in \{0, \pi\}$, does not alter the footpoint until we pass the cut locus of the geometry. The footvector is scaled according to the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_0$.

The largest change in the footpoint mapping occurs when the query position is displaced parallel to the tangent plane at the footpoint, in the direction of the smaller principal curvature in magnitude. In this case $\beta = \frac{\pi}{2}$ and Eq. (11) becomes

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^* - \boldsymbol{x} + l \cdot \left(\frac{\cos\alpha}{1 + d \cdot \kappa_1} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} + \frac{\sin\alpha}{1 + d \cdot \kappa_2} \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \right)$$

Interestingly, the magnitude of the change is only equal to the change in the planar footvector mapping of a normal section curve when we are moving $\boldsymbol{x}$ parallel to either of the principal curvature directions. In all other cases, the two quantities will be different since the normal section curvature is $\cos^2 \alpha\kappa_1 + \sin^2 \alpha\kappa_2$ by Euler's theorem.

4 Footpoint Intersection Iteration

Let $G_1 \subseteq \mathbb{R}^n$ and $G_2 \subseteq \mathbb{R}^n$ be two objects with the foot mapping $\boldsymbol{f}_1 : \mathbb{R}^n \to \mathbb{R}^n$ and $\boldsymbol{f}_2 : \mathbb{R}^n \to \mathbb{R}^n$, respectively. Our task is to produce a foot mapping $\boldsymbol{f}_{12} : \mathbb{R}^n \to \mathbb{R}^n$ with $G_{12} = G_1 \cup G_2$ or $G_{12} = G_1 \cap G_2$ similar to Eq. (1). This paper only describes the intersection since the complement geometry has the same foot mapping and $G_{12} = G_1 \cup G_2 = \mathbb{R}^n \setminus \big((\mathbb{R}^n \setminus G_1) \cap (\mathbb{R}^n \setminus G_2)\big)$.

If $\boldsymbol{x} \in G_1 \cap G_2$, then the intersection approximation is precise in Eq. (1), so

$$\boldsymbol{f}_{12}(\boldsymbol{x}) = \begin{cases} \boldsymbol{f}_1(\boldsymbol{x}) & \text{if } \|\boldsymbol{f}_1(\boldsymbol{x})\| \le \|\boldsymbol{f}_2(\boldsymbol{x})\| \\ \boldsymbol{f}_2(\boldsymbol{x}) & \text{otherwise} \end{cases} \qquad (\boldsymbol{x} \in G_1 \cap G_2) . \tag{12}$$

If the closest point to G_1 from $\boldsymbol{x} \notin G_1 \cap G_2$ is inside the G_2 set, then that point is the closest point to $\boldsymbol{x}$ in the $G_1 \cap G_2$ intersection. Thus,

$$\boldsymbol{f}_{12}(\boldsymbol{x}) = \begin{cases} \boldsymbol{f}_1(\boldsymbol{x}) & \text{if } \boldsymbol{x} + \boldsymbol{f}_1(\boldsymbol{x}) \in G_2 \\ \boldsymbol{f}_2(\boldsymbol{x}) & \text{if } \boldsymbol{x} + \boldsymbol{f}_2(\boldsymbol{x}) \in G_1 \end{cases} \qquad \big(\boldsymbol{x} \in \mathbb{R}^3 \setminus (G_1 \cap G_2)\big) . \tag{13}$$

However, this still leaves some $\boldsymbol{f}_{12}(\boldsymbol{x})$ values for us to define via iterative algorithms. The idea of this naive midpoint approach is to step closer to the intersection and re-evaluate $\boldsymbol{f}_{12}$:

$$\boldsymbol{f}_{12}(\boldsymbol{x}) := \frac{\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})}{2} + \boldsymbol{f}_{12}\left(\boldsymbol{x} + \frac{\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})}{2}\right) \qquad \text{if not (12) or (13).} \tag{14}$$

One can stop evaluating the recursion when one of Eq. (12) or Eq. (13) yield a value or after a predefined number of iterations. Figure 3 illustrates the convergence.

Since $\|\boldsymbol{f}_1\|$ and $\|\boldsymbol{f}_2\|$ are distance functions, there are no surface points inside the corresponding unbounding spheres $\mathcal{K}_{\|\boldsymbol{f}_i(\boldsymbol{x})\|}(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathbb{R}^3 : d(\boldsymbol{x}, \boldsymbol{y}) < \|\boldsymbol{f}_i(\boldsymbol{x})\|\}$:

$$\{\boldsymbol{f}_i = \boldsymbol{0}\} \cap \mathcal{K}_{\|\boldsymbol{f}_i(\boldsymbol{x})\|}(\boldsymbol{x}) = \emptyset \quad (\boldsymbol{x} \in \mathbb{R}^3, i = 1, 2) \ .$$

The point $\boldsymbol{x}$ is not in the intersection set $G_1 \cap G_2$ in the recursive Eq. (14). Therefore,

$$\begin{aligned} \big(G_1 \cap G_2\big) \cap \big(\mathcal{K}_{\|\boldsymbol{f}_1(\boldsymbol{x})\|}(\boldsymbol{x}) \cup \mathcal{K}_{\|\boldsymbol{f}_2(\boldsymbol{x})\|}(\boldsymbol{x})\big) &= \emptyset \\ \big(G_1 \cap G_2\big) \cap \mathcal{K}_{\max(\|\boldsymbol{f}_1(\boldsymbol{x})\|, \|\boldsymbol{f}_2(\boldsymbol{x})\|)}(\boldsymbol{x}) &= \emptyset \end{aligned}$$

Which means that we can take a larger heuristic step without overstepping with the following bisector iteration:

$$\boldsymbol{f}_{12}(\boldsymbol{x}) := \frac{\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})}{\|\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})\|} + \boldsymbol{f}_{12}\left(\boldsymbol{x} + \frac{\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})}{\|\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})\|}\right) \quad \text{if not (12) or (13).} \tag{15}$$

Figure 3 compares Eq. (14) and Eq. (15) iterative methods, with the upcoming deltoid iteration. This bisector iteration was sufficiently robust for 3D scenes as well. Figure 4 demonstrates the generated surface as a function of the iteration number.

5 Deltoid iteration

For this heuristics, we take advantage of the fact that the footvectors, $\boldsymbol{f}_1(\boldsymbol{x})$ and $\boldsymbol{f}_2(\boldsymbol{x})$, are perpendicular to the surfaces. First, we look for a

$$\boldsymbol{v}(f_1(\boldsymbol{x}), f_2(\boldsymbol{x})) = \alpha \cdot \boldsymbol{f}_1(\boldsymbol{x}) + \beta \cdot \boldsymbol{f}_2(\boldsymbol{x})$$

vector which is in the same plane as the footvectors. Second, the $\boldsymbol{x}+\boldsymbol{v}(\boldsymbol{f}_1(\boldsymbol{x}), \boldsymbol{f}_2(\boldsymbol{x}))$ should lie on each of the tangent planes at the footpoints.

Note that this quadrilateral is not necessarily a deltoid. To be precise, the evaluation point $\boldsymbol{x}$, the footpoints $\boldsymbol{x} + \boldsymbol{f}_1(\boldsymbol{x})$ and $\boldsymbol{x} + \boldsymbol{f}_2(\boldsymbol{x})$, and the next heuristic evaluation point $\boldsymbol{x} + \boldsymbol{v}(\boldsymbol{f}_1(\boldsymbol{x}), \boldsymbol{f}_2(\boldsymbol{x}))$ forms a right angular cyclic quadrilateral.

Therefore, the desired vector $\boldsymbol{v}(\boldsymbol{a}, \boldsymbol{b}) = \alpha \cdot \boldsymbol{a} + \beta \cdot \boldsymbol{b}$ has to be perpendicular to both $\boldsymbol{a} := \boldsymbol{f}_1(\boldsymbol{x})$ and $\boldsymbol{b} := \boldsymbol{f}_2(\boldsymbol{x})$, which means the following:

$$\begin{aligned} (\boldsymbol{v} - \boldsymbol{a}) \cdot \boldsymbol{a} = 0 \\ (\boldsymbol{v} - \boldsymbol{b}) \cdot \boldsymbol{b} = 0 \end{aligned} \quad \Longleftrightarrow \quad \begin{bmatrix} \boldsymbol{a}^\top \\ \boldsymbol{b}^\top \end{bmatrix} \cdot \boldsymbol{v} = \begin{bmatrix} \boldsymbol{a}^\top \\ \boldsymbol{b}^\top \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{a} & \boldsymbol{b} \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} aa \\ bb \end{bmatrix} \ .$$

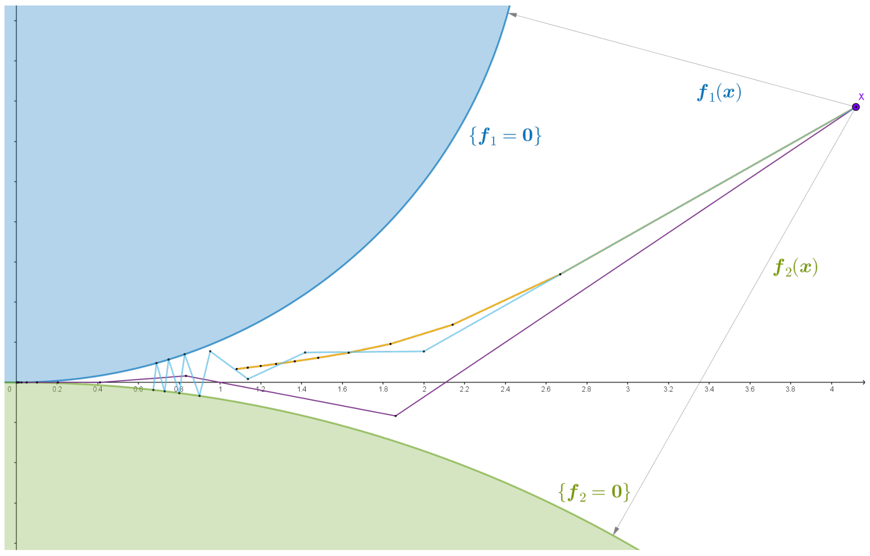

$\boldsymbol{f}_1(\boldsymbol{x})$
$\{\boldsymbol{f}_1 = \boldsymbol{0}\}$
$\boldsymbol{f}_2(\boldsymbol{x})$
$\{\boldsymbol{f}_2 = \boldsymbol{0}\}$

Figure 3: Comparison of the midpoint and deltoid footpoint iterations in a 2D scene where F and G are tangential circles, and the intersection is a single point. The midpoint method in Eq. (14) is the yellow iteration, the blue iteration is improved version from Eq. (15), the purple is the deltoid method from Section 5. The deltoid method converges much faster because of the surface linear approximation and the unrestricted step size.

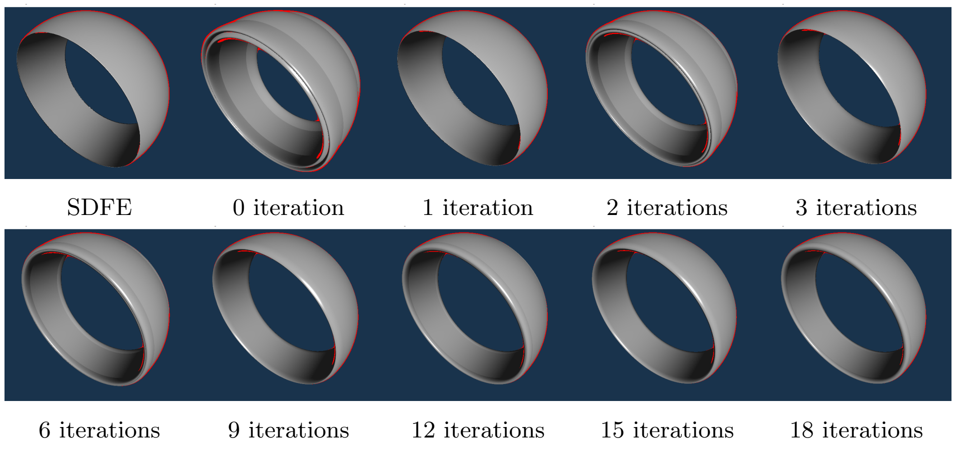


Figure 4: Convergence of the bisector iteration for an offset of a challenging intersection of a sphere and a cylinder. The iteration quickly starts to bounce from one surface to the other, resulting in similar error patterns in every second image, but converges nevertheless.

Where the dot products $aa = \boldsymbol{a} \cdot \boldsymbol{a}, bb = \boldsymbol{b} \cdot \boldsymbol{b}, ab = \boldsymbol{a} \cdot \boldsymbol{b}$. Solving the equation for α and β assuming $d := aa \cdot bb - ab \cdot ab \neq 0$ gives:

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}aa & ab\\ ba & bb\end{bmatrix}^{-1} \cdot \begin{bmatrix}aa\\ bb\end{bmatrix} = \frac{1}{d}\begin{bmatrix}bb & -ab\\ -ab & aa\end{bmatrix} \cdot \begin{bmatrix}aa\\ bb\end{bmatrix} = \frac{1}{d}\begin{bmatrix}bb \cdot (aa - ab)\\ aa \cdot (bb - ab)\end{bmatrix}. \qquad (16)$$

For the deltoid footpoint iteration, we substitude $\boldsymbol{v} = \alpha \cdot \boldsymbol{a} + \beta \cdot \boldsymbol{b}$ from Eq. (16) into Eq. (14), so $\boldsymbol{g}_{12}(\boldsymbol{x}) := \boldsymbol{v} + \boldsymbol{g}_{12}(\boldsymbol{x} + \boldsymbol{v})$ unless Eq. (12) or Eq. (13) provide a foot vector.

The presented algorithms are visualized in 2D on Figure 3. In 2D, the deltoid iteration was the most efficient and was used to produce Figure 2. However, this method diverged and produced too many artifacts in 3D to be applicable, so the bisector iteration was used in our test scenes.

6 Shader code generation from CSG trees

To represent CSG trees in code, we first define the type of a tree node as the union of

- some primitive shape types, like Box, Sphere or Cylinder, and
- some standard CSG operations, like Move, Rotate, Union and Intersect.

Each of these node types contains the specific parameters necessary to describe the object (size, color) or the operation (vector, angle) as fields. Once we have the union type (called Expr), we can build any CSG tree, where the nodes are all Expr objects: the leaves are instances of the primitive shape types, and the internal nodes are operations.

Now we can traverse this tree in a bottom-up or top-down manner to collect (or distribute) information about it. For example, we can count the number of red objects in the tree using this algorithm:

```
def alg(node):
    if node in [Box, Sphere, Cylinder]:
        return 1 if node.color is "red" else 0
    elif node in [Move, Rotate]:
        return alg(node.child)
    elif node in [Union, Intersect]:
        return alg(node.child_1) + alg(node.child_2)
```

Even though the algorithm is run from the root of the tree, what it actually does is breaking down and converting the tree into a single number, starting from the bottom. It begins with the leaf nodes because it can directly convert those. Then for each internal node, it first recursively converts the subtrees, then combines the obtained partial results into a single value using some node-specific logic.

We can use the same approach to derive much more complex information from the tree, like a program code that renders the represented model. We just need to change the so-called carrier type – the type of the value each subtree is reduced into, and the node-specific reduction logic. Our carrier type is a structure that contains a block of code and the name of the register that holds the result of the computation.

The generated block of code for the primitive shapes is simple: it's a function that takes a $\boldsymbol{x} \in \mathbb{R}^3$ position as an argument, and returns four scalars: the footvector and the signed distance value. Code generation in internal nodes combines the code generated for their subtrees. E.g. for a Union node this means that the resulting function executes the functions corresponding to the subtrees, collects their results, then calculates and returns the footvector and the distance for the union.

The intersection operation needs to evaluate its arguments multiple times, so we had to generate actual functions to calculate the subtrees, couldn't just directly calculate the results. Since we cannot define higher order functions in GPU shader code, each occurrence of the intersection operation had to be specialized for its actual arguments as a separate function. Writing all these specializations all by hand would have been tedious and error prone, this is one of the main reasons why we decided to generate the shared code.

The code generation also made it possible to implement optimizations which would be difficult to do manually. Such optimization is pushing down the Move and Rotate operations to the leaf nodes. The intuitive implementation of a CSG evaluator would first compute operand of a Move or a Rotate, and then apply the operation on the result. However, we can render the primitive objects in the leaf nodes with the same cost, regardless of their position or orientation, so we rather aggregate these operations while traversing down the tree, and apply them directly on the leaf objects.

This recursive approach to code generation is well known. What we recognized and used to our advantage is that the CSG model representation is a much simpler tree than usual syntax trees, we do not need any sophisticated state management to process it, which resulted in clean and reusable code generation.

7 Exact offset surfaces

All variants of the footpoint iteration algorithm are heuristic optimization iterations. For this reason, there are cases when the iteration does not converge to the correct solution. In essence, there is a trade-off between robustness and speed. Although we can visualize the above surfaces in real-time on an Nvidia 1080ti GPU, the visualization is expensive and three to ten times slower than the signed distance function representation.

Figure 5 demonstrates the most important advantage of our algorithm, that is, the offset operation will be exact afterwards. With enough iterations, the surface will be smooth because there is no voxelization. For simple surfaces such as the intersection of two objects, the raytracing of the offset surface can be real-time. For more complex scenes, we avoided running the iterative approach for the union operation because the standard minimum distanced union operation yields exact

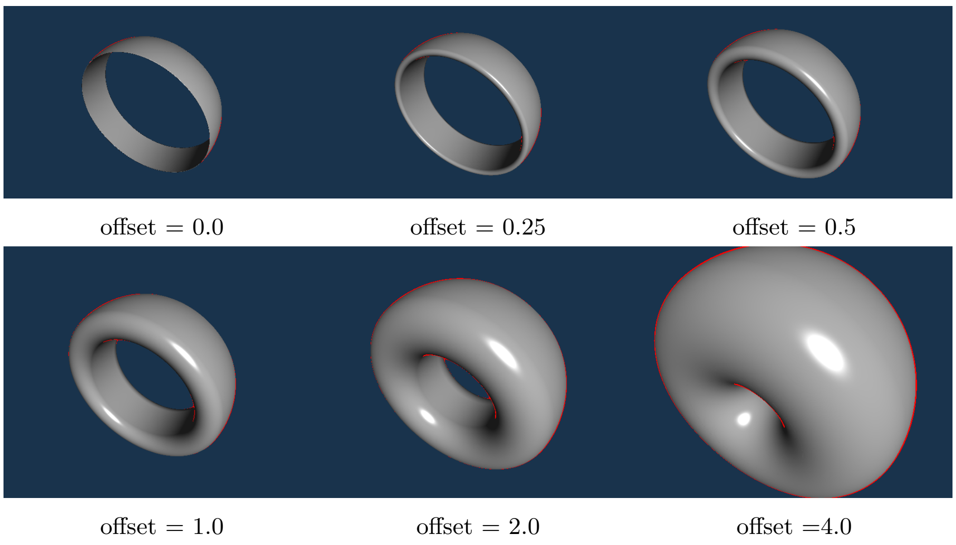


Figure 5: Different offsets of the intersection of a cylinder and a slightly larger sphere. Our bisector algorithm does a 100 iterations for each evaluation to produce the precise offset. The offset can be changed in real-time.

results outside the surface. Unless the result of the union is intersected with another object, the positive offset surface is correct.

Figure 6 showcases a few example surfaces and the drawbacks of the proposed method. Because the method is heuristic, the closest point of the intersection surface is not always found. In such cases, the surface may present artifacts or even appear elsewhere. Scene complexity both increases the likelihood of failure and the evaluation speed. When there are multiple intersection operations, the iterations are correctly inserted into each other by the code generator. This leads to an exponential slowdown in the number of nested intersection operations.

8 Conclusion

This paper proposed a footvector based representation of shapes. Section 3 provides a theoretical background for this, connecting the partial derivatives of the footvector mapping with the local differential geometry at the footpoint. The practicality of this representation, however, is provided by the iterative algorithms that make this representation closed under set theoretic and offset operations.

Offset operation of a signed distance function is as easy as subtracting the offset value from it, yet it is only precise on the CSG tree leafs, so-called primitives, in practise. This is because the SDF of combined objects are only signed distance lower bounds causing the offset surface to appear as if the offset was applied to the arguments of the set operation instead. In this paper, we devised algorithms

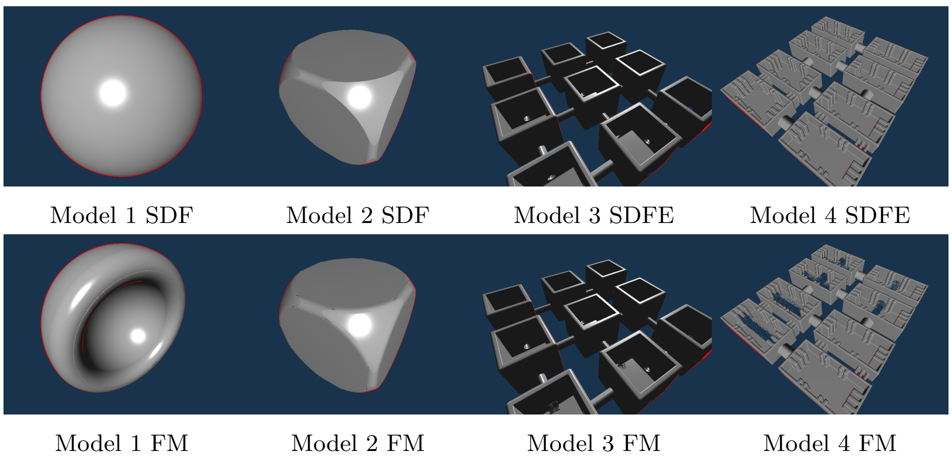


Figure 6: Several example scenes showcasing the strengths and limitations of the proposed methods. The SDFE of Model 1 completely deletes the subtracted sphere, yet the footpoint mapping offsets the correct surface. The improvements in smoothness is visible on Model 2, with some convergence artifacts. The iterative distance function of Model 3 and 4 introduce even more errors.

in Section 4 and 5 that iterate on input functions to produce the SDF of objects. Footvector mapping representations extended the distance information and provide search directions for the intersection operation iterations.

In two dimensions, the deltoid iteration outperformed the rest of the methods by a large factor. Computing the SDF in Figure 2 with the midpoint approach was about ten times slower compared to the deltoid method whilst achieving similar accuracy. Note that the iterations had to be nested to produce the CSG tree of set-operations causing exponential slowdown with CSG tree depth.

In three dimensions, our iterative methods are capable of producing high quality offset surfaces of intersection or difference of objects. The resulting footvector mapping can be visualized in real-time as a signed distance function despite the extra iterations within each intersection operation. Note that the expensive function evaluation time can amortized with better sphere tracing algorithms, such as enhanced sphere tracing [1] or quadric tracing which is an unpublished algorithm for reducing the number of function evaluations by pre-cacheing values.

For rendering purposes the SDF had to be implemented in shader code which does not support higher order functions. Hence, a CSG code generator was designed that created efficient implementations for our test scenes. In three dimensions, the bisector method performed the best because the deltoid iteration often did not converge to the right solution. Nevertheless, for most simple cases, the bisector method converged without artifacts, producing accurate offset surfaces.

References

[1] Bálint, Csaba and Valasek, Gábor. Accelerating Sphere Tracing. In Diamanti, Olga and Vaxman, Amir, editors, *EG 2018 - Short Papers*. The Eurographics Association, 2018. DOI: `10.2312/egs.20181037`.

[2] Bálint, Csaba, Valasek, Gábor, and Gergó, Lajos. Operations on signed distance functions. *Acta Cybernetica*, 24(1):17–28, May 2019. DOI: `10.14232/actacyb.24.1.2019.3`.

[3] Bán, Róbert, Bálint, Csaba, and Valasek, Gábor. Area Lights in Signed Distance Function Scenes. In Cignoni, Paolo and Miguel, Eder, editors, *Eurographics 2019 - Short Papers*. The Eurographics Association, 2019. DOI: `10.2312/egs.20191021`.

[4] do Carmo, Manfredo P. *Differential geometry of curves and surfaces*. Prentice Hall, 1976.

[5] Fabbri, Ricardo, Costa, Luciano Da F., Torelli, Julio C., and Bruno, Odemir M. 2D euclidean distance transform algorithms: A comparative survey. *ACM Comput. Surv.*, 40(1), February 2008. DOI: `10.1145/1322432.1322434`.

[6] Foley, James David. *12.7 Constructive Solid Geometry*. In *Computer Graphics: Principles and Practice*, pages 533–558. Addison-Wesley Professional, 1990.

[7] Hart, John C. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1996. DOI: `10.1007/s003710050084`.

[8] Hoffmann, Christoph M. *Boolean Operations on Boundary Representation*. In *Geometric and Solid Modeling: An Introduction*, pages 67–110. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[9] Kallay, Michael. A geometric Newton–Raphson strategy. *Computer Aided Geometric Design*, 18(8):797–803, 2001. DOI: `10.1016/S0167-8396(01)00070-X`.

[10] Keinert, Benjamin, Schäfer, Henry, Korndörfer, Johann, Ganse, Urs, and Stamminger, Marc. Enhanced Sphere Tracing. In Giachetti, Andrea, editor, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2014. DOI: `10.2312/stag.20141233`.

[11] Martin, R.R. Principal Patches - A New Class of Surface Patch Based on Differential Geometry. In ten Hagen, P.J.W., editor, *Eurographics Conference Proceedings*. The Eurographics Association, 1983. DOI: `10.2312/eg.19831003`.

[12] Patrikalakis, Nicholas M. and Maekawa, Takashi. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[13] Sethian, James A. Fast marching methods. *SIAM Rev.*, 41(2):199–235, June 1999. DOI: `10.1137/S0036144598347059`.

[14] Valasek, Gábor. Generating distance fields from parametric plane curves. In *Annales Mathematicae et Informaticae 48*, pages 83–91, 03 2018.