www.cad-journal.net
Computer-Aided Design AND APPLICATIONS™

Distance Field Generation by Proxies

Anna Lili Horváth[1], Gábor Valasek[2], Róbert Bán[3]

[1]Eötvös Loránd University, srirm5@inf.elte.hu
[2]Eötvös Loránd University, valasek@inf.elte.hu
[3]Eötvös Loránd University, rob.ban@inf.elte.hu

Corresponding author: Anna Lili Horváth, srirm5@inf.elte.hu

Abstract. Surfaces represented by distance functions can be rendered robustly and offer efficient means to evaluate various spatial queries on them. However, constructing and representing distance functions for general surfaces is a difficult problem. We propose a general method for this task by extending geometric distance fields to 3D. Such fields are grids of geometric proxies that have closed-form distance functions and approximate the distance function of the original surface at a higher order than mere distance samples. We show how first and second-order proxies can be constructed from parametric and signed distance function inputs, as well as consider the generalization to arbitrary implicitly defined surfaces.

Keywords: Signed distance function, Volume models, Ray casting
DOI: https://doi.org/10.14733/cadaps.2025.805-824

1 INTRODUCTION

The signed distance function (SDF) of an $F \subset \mathbb{E}^n$ geometry is an implicit function that maps the signed distance of the argument to the closest boundary point of F. The sign determines whether the query position is inside the volume or not. The SDF is a powerful representation of shapes, possessing all the flexibility of general implicit functions [1], as well as encoding global geometric information about the scene, simplifying tasks such as collision detection.

Despite its advantages, the exact SDF of a general complex scene cannot be expressed in closed-form. As such, the evaluation of the distance is carried out either procedurally or via approximations in practice. The former is usually reduced to a numerical distance minimization on the particular input representation while the latter uses an approximation to the input itself and treats the problem as a function approximation task.

Although these algorithms can satisfy precision constraints and thus provide sufficiently accurate numerical substitutions to the real distance, they have to interface with every geometric representation involved in the problem. Consequently, the implementation of such algorithms is intertwined with the combinatorial complexity of the modeling system.

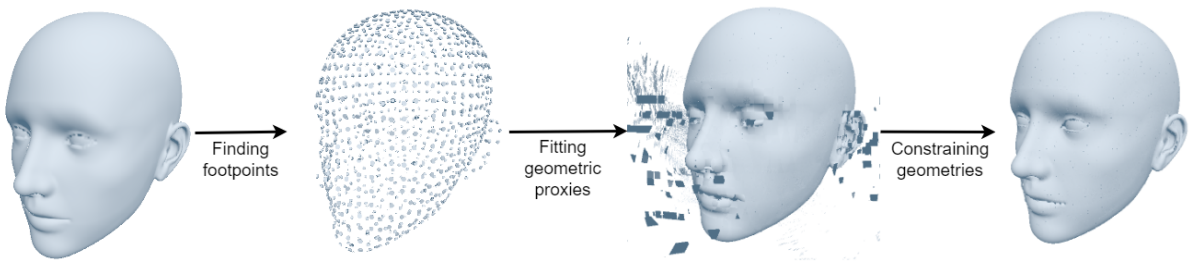


Figure 1: An outline of our presented method for constructing distance fields of general surfaces. First, we acquire a set of sample points from the input geometry then we fit appropriately chosen geometric proxies that have G^n connection to the original input, where n is the order of the field. Our approach takes samples along the grid, thus a 3D texture can be utilized to store the data, which then can be decoded to a distance approximation. In the example, we used a simple nearest sampling method. As we only ensure the necessary approximation locally at the samples, in the last step we have to restrict our proxies to a small area using geometric constraints.

Our paper proposes the use of geometric proxies that have G^n connection to the input geometry and possess efficient signed distance queries. Due to the geometrically continuous connection with the original geometry, their distance functions are local approximations to the original surface. This approach requires a minimal number of coefficients for a given order of accuracy.

We propose a method to construct a field of proxies for general inputs and present specific algorithms for parametric surfaces and SDFs. We also show that our approach ensures generality as the evaluation of the data becomes independent from the complexity of the input. The outline of our proposed method can be seen in Figure 1.

We overview the results of prior work in Section 2 and summarize the notational and theoretical background in Section 3.

Our first result is presented in Section 4 where we show that the derivatives of the SDF at a point only depend on the geometric invariants at the footpoint and the distance to the boundary. Based on this, Section 4 establishes a set of geometric primitives that we employ as first and second-order proxies in lieu of the input geometry, and Section 5 shows how we build a geometric distance field. Methods for decoding the field are presented in Section 6.

Our empirical results regarding accuracy are summarized in Section 7 and we summarize our findings in Section 8.

2 PREVIOUS WORK

Our paper discusses a discrete, approximate signed distance field framework and we focus on previous work in this area.

High-performance graphical applications utilize 2D and 3D distance fields for font rendering [7], collision detection, and a variety of visual effects, including global illumination [15, 8]. Due to the demands of the domain, efficient evaluation and the capability to utilize GPU hardware interpolation are of special interest and this restricts the class of investigated representations. Usually, this simplifies to function sample values, occasionally augmented by gradients.

However, it is possible to encode and query data that is of a geometric nature. Loop and Blinn[10] proposed such a solution to rendering vector art composed of cubic Bézier segments. The SDF to the boundary is approximated using a first order expansion which facilitates anti-aliased contour rendering. This

can be considered as an algebraic approximation to the SDF. In contrast, we utilize geometric proxies instead of polynomial fits to SDF values.

Song et al. [11] showed that classic Hermite interpolation may be applied to the problem of approximating the SDF. They showed that the derivatives of the SDF can be computed for both parametric boundary and implicit volume representations. Our work extends this by carrying out Hermite interpolation in distance space at the sample positions but relying on simple geometric proxies to do so. Moreover, our work is also compatible with input representations that do not allow derivative interrogation. We propose a variant that fits cubic algebraic polynomials to the input and uses the differential geometric invariants of the resulting approximations to construct proxies.

This approach was presented for 2D shapes with SDF input in [2]. Our method is a generalization of the method in 3D and for general input.

3 THEORETICAL BACKGROUND

Let $\mathbb{E}^n$ denote the n-dimensional Euclidean space and $\|.\|_2$ the Euclidean norm. The derivatives of $f : \mathbb{E}^n \to \mathbb{R}$ are written as $\partial_1 f, \partial_2 f, \ldots, \partial_n f$, while single variable function derivatives are denoted by f'.

An $\boldsymbol{r} \in \mathbb{R} \to \mathbb{E}^n$ parametric curve is regular at $t \in \mathbb{R}$ if $\boldsymbol{r}'(t) \neq \mathbf{0}$. An $\boldsymbol{r}(u, v) : \mathbb{R}^2 \to \mathbb{E}^n$ surface parametrization is regular at $(u_0, v_0) \in \mathbb{R}^2$ if its partial derivatives are linearly independent, i.e. $\partial_u \boldsymbol{s}(u_0, v_0) \times \partial_v \boldsymbol{s}(u_0, v_0) \neq \mathbf{0}$.

The distance function $\hat{f} : \mathbb{E}^n \to \mathbb{R}$ of an $F \subset \mathbb{E}^n$ shape is defined as $\hat{f}(\boldsymbol{x}) = d(\boldsymbol{x}, \partial F)$, where $d(\boldsymbol{x}, \partial F) = \inf_{\boldsymbol{y} \in G} \|\boldsymbol{x} - \boldsymbol{y}\|_2$ and $\partial F := \overline{F} \cap \overline{(\mathbb{E}^n \setminus F)}$ is the boundary of F. The signed distance function is defined as $\hat{f}(\boldsymbol{x}) = \operatorname{sgn}(\boldsymbol{x}) \cdot d(\boldsymbol{x}, \partial F)$ where

$$\operatorname{sgn}(\boldsymbol{x}) = \begin{cases} -1 & \text{if } \boldsymbol{x} \in F, \\ 1 & \text{if } \boldsymbol{x} \notin F. \end{cases}$$

The sign defines an inside-outside partitioning of the space.

We refer to all boundary points that realize the minimum distance as footpoints to $\boldsymbol{x}$. We denote the set of footpoints to $\boldsymbol{x}$ as $P(\boldsymbol{x}) = \{\boldsymbol{y} \in \partial F \mid \|\boldsymbol{y} - \boldsymbol{x}\|_2 = d(\boldsymbol{x}, F)\}$.

The DF is continuous and its gradient is of unit length at regular points. The cut locus is the set of points where the footpoint is not unique; the gradient of the DF is not defined at these points.

Two curves are $G^n, n \geq 1$ at a common point $\boldsymbol{x}$ if there exists a regular reparametrization with respect to which they are C^n at $\boldsymbol{x}$. Similarly, two surfaces are G^n if there exists a regular reparametrization after which the two surfaces are C^n. For the sake of practical considerations, we restrict our discussion to orientation preserving reparametrizations.

The arc-length or natural parametrization of curves is such that the traversal of the curve is of unit speed. We denote natural parametrization by hats, i.e. $\hat{\mathbf{r}} : [0, L] \to \mathbb{E}^n$, where $\|\hat{\mathbf{r}}'\|_2 \equiv 1$. Every regular parametrization of a curve can be written as an $\boldsymbol{r} = \hat{\mathbf{r}} \circ s : [a, b] \to \mathbb{E}^n$ composition where $s(t) = \int_a^t \|\boldsymbol{r}'(x)\|_2 dx : [a, b] \to [0, L]$, $L = s(b)$ denoting the arc-length of the curve. Note that for regular parametrizations, $s(t)$ is strictly monotonously increasing, thus it has an inverse s^{-1}.

Similarly, we consider $\boldsymbol{r}(u, v) : [a, b] \times [c, d] \to \mathbb{E}^3$ arbitrarily parametrized regular surfaces as reparametrizations of natural surface parametrizations, i.e. $\boldsymbol{r} = \hat{\mathbf{r}} \circ \boldsymbol{u}$. The $\hat{\mathbf{r}} : [0, L_1] \times [0, L_2] \to \mathbb{E}^n$ natural parametrization is defined such that the parameter lines are lines of curvatures traversed at unit speed.

The Frenet-Serret formulas quantify how the derivatives of an arc-length parameterized curve depend on the geometric invariants and only those, i.e. the curvature and torsion functions. The above natural parametrization of surfaces possesses a similar property. In particular, all of its mixed partial derivatives vanish (see Lemma 16 in [12]). As such, all partial derivatives of $\hat{\mathbf{r}}(s, t)$ depend on the geometric invariants of the lines of curvatures and only on those.

4 HIGHER ORDER DISTANCE FUNCTION RECONSTRUCTION

Our goal is to interpolate the SDF and its derivatives up to order n at an arbitrary point. Let us first show that geometric Hermite interpolation of order n results in an order n SDF reconstruction. Once that is established, we focus on shapes that are practical in the sense that the evaluation of their SDF is inexpensive.

In the case of parametric curves, let $t : \mathbb{E}^n \to \mathcal{P}(\mathbb{R})$ denote the foot-parameter mapping that assigns the parameter of the closest points on the curve to the query position, as shown in [13]. Since the SDF is only differentiable where $|t(\boldsymbol{x})| = 1$, we can assume that t is a real-valued function at the point of interest. Then the SDF, up to sign, is written as $\hat{f} = \|\boldsymbol{r} \circ t - \boldsymbol{id}\|_2$, where $\boldsymbol{id}$ is the identity mapping, i.e. $\boldsymbol{id}(\boldsymbol{x}) = \boldsymbol{x}$.

Let us write the $\boldsymbol{r} : [a, b] \to \mathbb{E}^n$ arbitrarily parametrized regular curve as the composition of its arc-length parametrization and the $s : [a, b] \to [0, L]$ arc-length function: $\boldsymbol{r} = \hat{\mathbf{r}} \circ s$. Then the unsigned DF is

$$\hat{f} = \|\boldsymbol{r} \circ t - \boldsymbol{id}\|_2 = \|(\hat{\mathbf{r}} \circ s) \circ (s^{-1} \circ \hat{t}) - \boldsymbol{id}\|_2 = \|\hat{\mathbf{r}} \circ \hat{t} - \boldsymbol{id}\|_2 \ , \tag{1}$$

where $\hat{t} : \mathbb{E}^n \to [0, L]$ is the footparameter mapping on the arc-length parametrization.

Similarly, the SDF of a regular parametric surface can be written using the footparameter mapping and its inverse. Discarding the sign again, the distance function is expressed as $\hat{f} = \|\boldsymbol{r} \circ \boldsymbol{t} - \boldsymbol{id}\|_2$, where $\boldsymbol{t} : \mathbb{E}^n \to \mathcal{P}(\mathbb{R}^2)$ is the surface footparameter mapping. Wherever the SDF is differentiable, this is further developed as

$$\hat{f} = \|(\hat{\mathbf{r}} \circ \boldsymbol{u}) \circ (\boldsymbol{u}^{-1} \circ \hat{\mathbf{t}}) - \boldsymbol{id}\|_2 = \|\hat{\mathbf{r}} \circ \hat{\mathbf{t}} - \boldsymbol{id}\|_2 \tag{2}$$

where $\boldsymbol{r} = \hat{\mathbf{r}} \circ \boldsymbol{u}$, with $\boldsymbol{u} : \mathbb{R}^2 \to \mathbb{R}^2$ being the regular reparametrization of the surface from natural parametrization, and $\hat{\mathbf{t}}$ is the footparameter mapping on the natural parametrization.

These prove that the n-th derivative of the SDF only depends on the distance between the query position and boundary and the natural parametrization of the shape. Quantitatively, the latter is equivalent to the geometric invariants, such as normal vectors, principal curvatures, etc., at the footpoint up to order n. This implies that G^n continuous surfaces possess C^n continuous connection of their signed distance fields.

As a consequence, curves and surfaces with the proper continuity properties can be used to approximate the SDF instead of polynomials. This means that we can use any geometry that is G^n continuously connected at the footpoint. Besides this, we have the following expectations towards the proxies: (i) the SDF of the proxy should be efficient to evaluate and (ii) the proxy should have a low memory cost.

In three-dimensions, the simplest G^1 surface is the tangent plane at the footpoint, which can be defined by one of its points and its normal vector. In our case, we can use the $P(\boldsymbol{x}_i)$ footpoint itself, and the corresponding normal vector to describe the proxy. Taking into consideration that, by definition, the normal at the footpoint is parallel to the vector from the footpoint to the $\boldsymbol{x}_i$ query point, the geometry can be described with three scalar values: the displacement vector $P(\boldsymbol{x}_i) - \boldsymbol{x}_i$.

For the G^2 geometric proxy, we have to use a shape that is able to represent the principal curvatures and directions of the surface. A geometry that is suitable for this is the torus as it is able to store any combination of curvatures as shown in Figure 2. If the torus is defined by its center, axes, and two radii as shown in Figure 3 the SDF can be obtained with a simple formula, as presented in [3].

However, this representation cannot represent the degenerate cases of cylinders and planes conveniently, where one or two of the curvatures is zero. Thus, we represent the torus by the footpoint, the principal curvatures, and the axis, see Figure 4.

It is important to note that we assume that our footpoint is either on the inner or outer circle of the torus, as highlighted in Figure 2. This way the torus will be uniquely defined, while without this assumption there would be multiple tori approximating the surface making the evaluation difficult. It can also be seen that for any curvature configuration, there exists a torus thet connects to the surface in one of the said points and approximates in order 2.

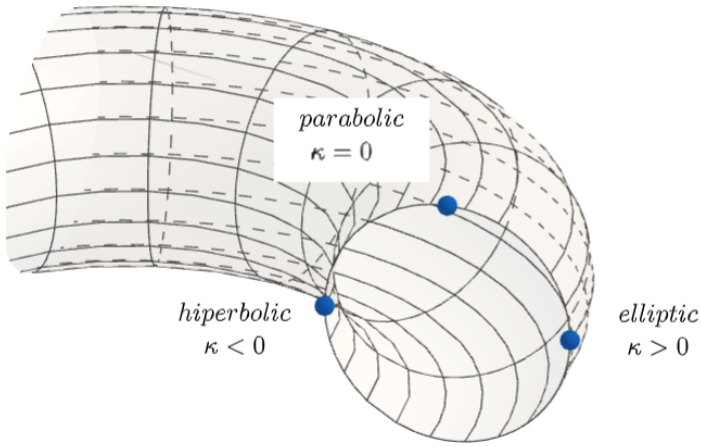


Figure 2: Classification of the points of the torus according to curvature configurations, $\kappa = \kappa_1 \cdot \kappa_2$ is the product of the two principal curvatures.

The sign of the curvatures will be given in relation to the surface normal, thus when decoding the field it is necessary to know this vector. It is either parallel to the $P(\boldsymbol{x}_i) - \boldsymbol{x}_i$ vector or the opposite of it, therefore we need an additional bit, representing the values of 1 or -1 encoding the direction.

As such, the torus can be stored with 9 scalar values on the computer. In particular, the footpoint $P(\boldsymbol{x}_i)$, the two principal curvatures $\kappa_r \in \mathbb{R}$ and $\kappa_R \in \mathbb{R}$, one of the principal axes $\boldsymbol{t} \in \mathbb{R}^3$ and the orientation of the correct surface normal $s \in \{1, -1\}$. This can be reduced to 8 scalars by encoding s in the length of $\boldsymbol{t}$, then it has to be decoded when reading the field.

5 FIELD GENERATION

In the previous section, we discussed how to construct geometric proxies that approximate the SDF of a shape up to the second order at regular points, i.e. where the footpoint is unique. Let us now investigate how to construct a globally defined approximation from these one-point proxies that retain the order of accuracy of the individual geometries.

In general, the proxies may have an arbitrary topology. Here, the scattered data interpolation toolbox [4] offers a variety of tools to extract a globally defined function from the samples. The proxies may be generated by sampling the input geometry, e.g. on a regular grid in the parameter space of parametric surfaces, then combined based on proximity to the query position. The simplest such approach is to take the distance from the closest proxy.

This method makes the generation of the field simple and provides an approximation with fewer proxies. However, as shown in the previous section, the order n approximation is ensured only locally, meaning that we will get the desired approximation order only in query points where the footpoint is close to the samples taken. Such an unstructured set of samples requires careful acceleration efforts to provide high performance.

For this reason, we choose the approach of using a 3D grid of proxies. We defined a set of sample points on a grid and approximated the distance function at them. While this approach can make the proxies redundant, the evaluation of the field becomes simple as we can find the closest sample fast.

Specifically, we define our field as a three dimensional texture with a given physical extent and texel

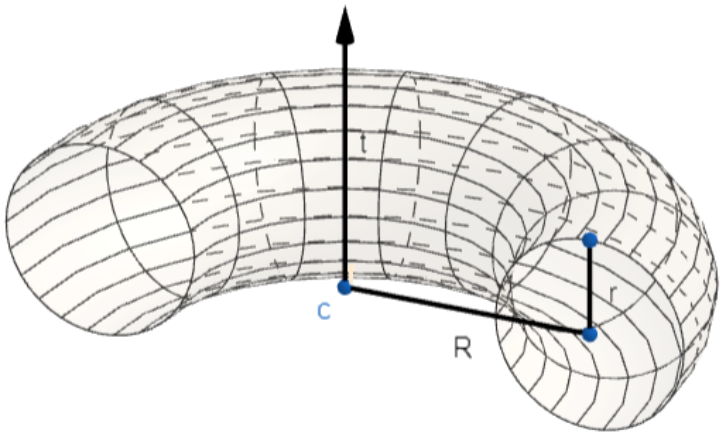


Figure 3: Defining parameters of the general torus representation. The torus is described by its $\boldsymbol{c}$ center, r and R radii and $\boldsymbol{t}$ axis.

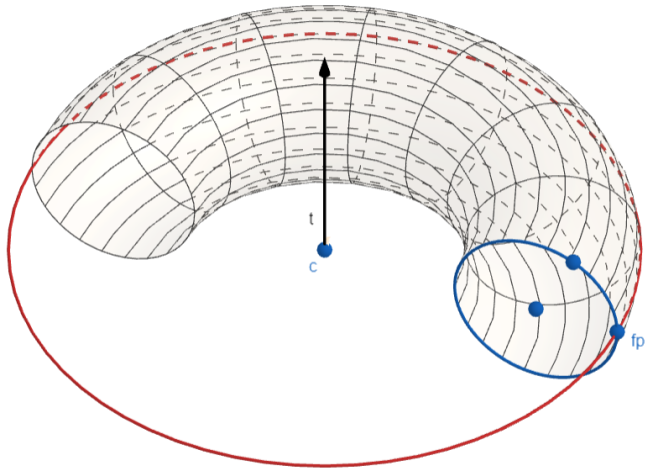


Figure 4: Defining parameters of the footpoint representation chosen such that both cylinders and planes are stored efficiently. The torus is described by a $\boldsymbol{p}$ surface point, the κ_r and κ_R principal curvatures at that point and the $\boldsymbol{t}$ axis.

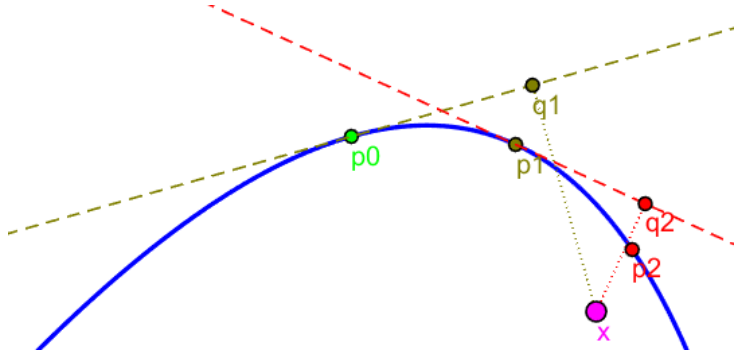


Figure 5: An example of the geometric Newton-Rhapson method in 2D. We are looking for the closest point to $\boldsymbol{x}$ and our initial guess is $\boldsymbol{p}_0$. The figure shows two iterations of the method. We approximate the curve with lines and take a closest point from them $(\boldsymbol{q}_1, \boldsymbol{q}_2)$, then calculate the step in the original parameter space obtaining a better approximation of the closest point $(\boldsymbol{p}_1, \boldsymbol{p}_2)$.

resolution. For every cell of the grid, we take the center as our $\boldsymbol{x}_i$ sample position and find the corresponding footpoint from the surface where we fit the proxy.

For any representation on which closest point query and differential geometric invariant data evaluation is possible, the following general algorithm can be used to generate proxies. For each cell of the 3D field grid:

1. find one of the closest surface points to the center of the cell
2. calculate the necessary differential geometric invariants at the footpoint and get the defining qualities of the proxy geometry

In certain cases, evaluating the curvatures or gradients may be infeasible or computationally too expensive. In what follows, we demonstrate how we address this and the closest point search for parametric and implicitly defined surface inputs.

5.1 Parametric Surface Input

Given a parametric surface $\boldsymbol{s} : \mathbb{R}^2 \to \mathbb{E}^3$, we can find the footpoint $\boldsymbol{p} = P(\boldsymbol{x})$ to $\boldsymbol{x} \in \mathbb{E}^3$ by solving for the (u, v) parameter pair that minimizes the distance from $\boldsymbol{s}(u, v)$ to $\boldsymbol{x}$. For example, this can be done by using the geometric Newton-Rhapson method [9]. An outline of the method in 2D can be seen in Figure 5. The 3D version follows a similar approach but utilizes the tangent plane instead. A crucial problem is the initial guess as the iteration needs a starting point that is close to the exact result to converge. We took a regular grid in the parameter space of the surface and chose the closest of the grid points to our query point as the starting point of the iterations.

Once the closest point and its (u, v) parameters have been determined, we proceed to the geometric invariant computation. For the tangent plane proxy, we can take the displacement vector as $\boldsymbol{d} = P(\boldsymbol{x}) - \boldsymbol{x}$.

For the second-order proxy, we have to calculate the derivatives $(\partial_u \boldsymbol{s}, \partial_v \boldsymbol{s})$ and get the normal as their cross product $\boldsymbol{n} = \frac{\partial_u \boldsymbol{s} \times \partial_v \boldsymbol{s}}{\|\partial_u \boldsymbol{s} \times \partial_v \boldsymbol{s}\|_2}$. The s sign we can set to 1 or -1 based on whether $\boldsymbol{n}$ is pointing in the same direction as $\boldsymbol{d}$.

After this, we can get the principal curvature values with the following formulas, where κ_1, κ_2 are the curvature values needed. The first fundamental form is expressed using

$$\mathbb{E} = \partial_u \boldsymbol{s} \cdot \partial_u \boldsymbol{s}, \quad \mathbb{F} = \partial_u \boldsymbol{s} \cdot \partial_v \boldsymbol{s}, \quad \mathbb{G} = \partial_v \boldsymbol{s} \cdot \partial_v \boldsymbol{s}, \tag{3}$$

and the second is

$$\mathbb{L} = \partial_{uu}\boldsymbol{s} \cdot \boldsymbol{n},\ \mathbb{M} = \partial_{uv}\boldsymbol{s} \cdot \boldsymbol{n},\ \mathbb{N} = \partial_{vv}\boldsymbol{s} \cdot \boldsymbol{n}\ . \tag{4}$$

The principal curvatures are then

$$\kappa_1, \kappa_2 = h \pm \sqrt{h^2 - g}$$

where the Gauss and mean curvatures are

$$g = \frac{\mathbb{L} \cdot \mathbb{M} - \mathbb{N}^2}{\mathbb{E} \cdot \mathbb{G} - \mathbb{F}^2}$$
$$h = \frac{\mathbb{L} \cdot \mathbb{G} - 2 \cdot \mathbb{M} \cdot \mathbb{F} + \mathbb{N} \cdot \mathbb{E}}{2 \cdot (\mathbb{E} \cdot \mathbb{G} - \mathbb{F}^2)}\ .$$

Since we assumed that the footpoint is on the inner or outer arc of the torus, its axis should be the direction of the bigger curvature.

The axis is obtained from its tangent

$$\lambda = -\frac{\mathbb{M} - \kappa \cdot \mathbb{F}}{\mathbb{N} - \kappa \cdot \mathbb{G}}\ ,$$

where κ is the larger curvature value.

5.2 SDF Input

For an $\hat{f} \in \mathbb{E}^3 \to \mathbb{R}$ SDF, the footpoint can be obtained from the gradient at our query point $\boldsymbol{x} \in \mathbb{E}^3$ and the distance value as $P(\boldsymbol{x}) = \boldsymbol{x} - \nabla\hat{f}(\boldsymbol{x}) \cdot \hat{f}(\boldsymbol{x})$

For the first-order case, the displacement vector is given by $-\hat{f}(\boldsymbol{x})\nabla\hat{f}(\boldsymbol{x})$. The normal is parallel to the gradient at the footpoint, thus it can be written as $\boldsymbol{n} = \frac{\nabla\hat{f}(P(\boldsymbol{x}))}{\|\nabla\hat{f}(P(\boldsymbol{x}))\|_2}$, and the s can be obtained as presented previously.

For second-order fields, the curvature values can be computed analytically as described in [6]. However, for a more general setting, we obtain the curvatures for an approximate surface at the footpoint. This only imposes value-evaluation constraints on the input, simplifying the integration of our method.

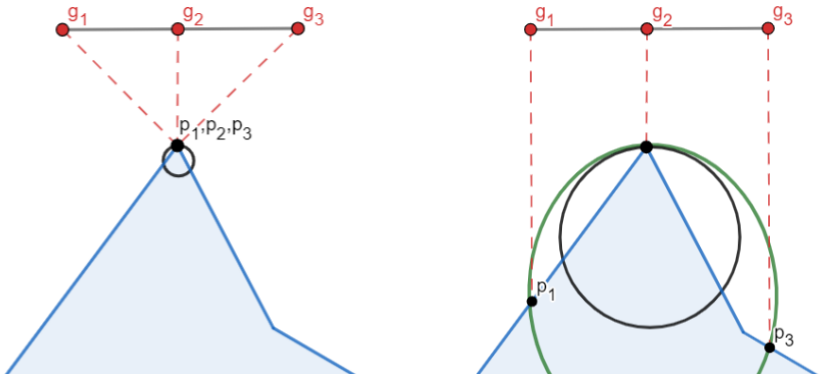
We take samples around the footpoint by taking a small grid on the tangent plane and projecting its $\boldsymbol{g}_{ij}$ points on the surface. We took two options into consideration

- taking the $P(\boldsymbol{g}_{ij})$ footpoints to each
- projecting the points orthogonally on the surface

Upon comparing the two methods, it became evident that the second approach provides a superior approximation. This observation is illustrated in Figure 6, where the selection based on the footpoint yields a degenerate result, whereas the orthogonal projection produces a smoother outcome. Consequently, we choose the orthogonal projection method as presented in Figure 7. which gives evenly spread samples ensuring a smooth approximation.

After constructing this set of $\boldsymbol{p}_{ij}$ surface points, we convert the points to a local coordinate system defined by the footpoint as the origin, an $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ arbitrary base of the tangent plane and the $\boldsymbol{n}$ normal as the base vectors.

We can fit an algebraic surface to these grid points, as demonstrated in [5]. It is important to emphasize that when employing orthogonal projection, the local x and y coordinates of the $\boldsymbol{p}_{ij}$ points will align with the local coordinates of the points on the tangent grid. Consequently, in the local base, only the z coordinates may differ for different sample points.



(a) All footpoints will be the same, the fitting of the torus will be numerically unstable

(b) With orthogonal projection we get an approximate result

Figure 6: An example for the difference between getting surface points by finding the footpoints and projecting orthogonally

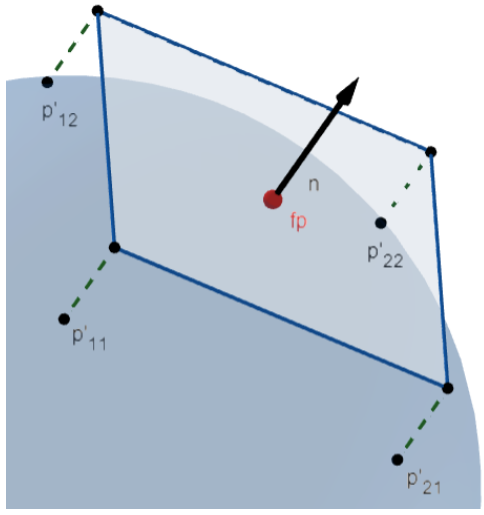


Figure 7: We approximate the surface around the footpoint by taking a set of surface points. We obtain these points by taking a small grid on the tangent plane at the footpoint and projecting its points orthogonally on the surface.

This converts the surface fitting into a linear system of equations, for which the Moore-Penrose pseudoinverse gives an optimal least square approximation. As our points form a grid with a fixed size and resolution, all the necessary data can be calculated as a preprocessing step prior to the fitting, making the algorithm easy to parallelize.

We can calculate the Weingarten matrix of the surface from the obtained coefficients as presented in [5]. An important quality of this matrix is that its eigenvalues correspond to the principal curvatures of the surface and the eigenvectors are the principal directions.

Thus we obtained the κ_r, κ_R curvature values and the axis can again be chosen as the direction of κ_r.

We also considered general implicit surfaces which are the generalization of SDFs, therefore the algorithm for obtaining the proxy geometries can be used in the same way. However, finding the footpoint is a more difficult problem.

We used the Newton method, but this algorithm only converges if our initial guess is close enough to the real footpoint. In our implementation, we initialized the guess by taking steps in the opposite direction of the gradient from our query point and started the iteration from different points.

6 DECODING THE FIELD

Once the geometric distance field is computed, our method becomes agnostic to the original input representation. All subsequent computations are carried out on the grid of proxies.

The field is stored in a 3D texture. When reading the field, we take the data from the encapsulating sample cell of our query point and calculate the distance function. Since the data stored at samples is not defined in the same global coordinate system, we cannot rely on hardware-accelerated trilinear filtering, we have to carry out this task manually. First, this requires us to evaluate the distance to a single proxy.

For the plane, we can use a general distance function. We can read the displacement vector $\boldsymbol{d}$ from the field, which is parallel to the normal at the footpoint thus $\boldsymbol{n} = \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|_2}$. The footpoint can be calculated from the $\boldsymbol{x}_i$ center position of the sample cell as $\boldsymbol{p} = \boldsymbol{x}_i + \boldsymbol{d}$. With this the distance of $\boldsymbol{x}$ from the plane is

$$d_{plane} = (\boldsymbol{x} - \boldsymbol{p}) \cdot \boldsymbol{n} \ .$$

.

For the second-order field, we can use a closed-form formula to calculate the distance from the torus as presented in [3]. To do so, we have to convert from our footpoint-based representation of the torus to the one used in the formula.

If both $\kappa_r = 0$ and $\kappa_R = 0$, the geometry is a plane. In this case, we need the footpoint, which is explicitly stored, and the normal at the footpoint, which we can calculate from the footpoint and the center position of the query texel ($\boldsymbol{x}_i$) augmented with the sign s as

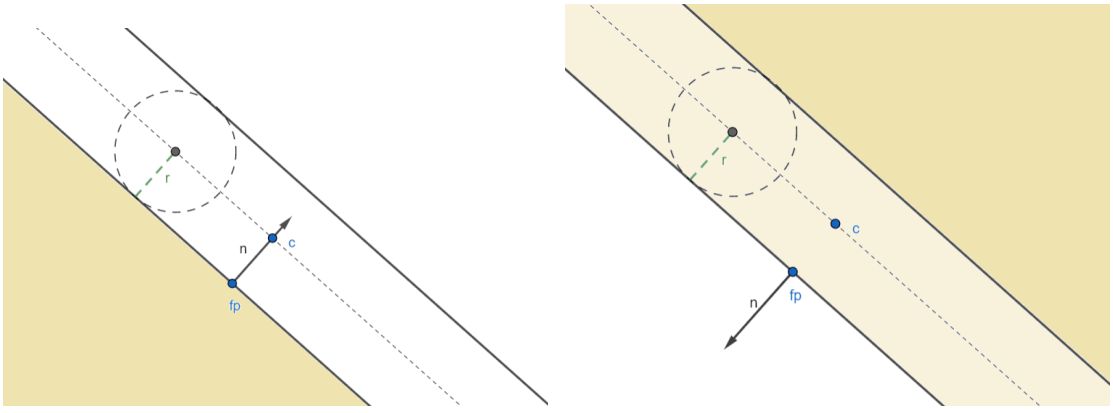
$$\boldsymbol{n} = s \cdot \frac{\boldsymbol{p} - \boldsymbol{x}_i}{\|\boldsymbol{p} - \boldsymbol{x}_i\|_2} \ . \tag{5}$$

After calculating this, we can use the SDF of the plane as written previously.

When $\kappa_R = 0$, i.e. the footpoint is a parabolic point, the proxy geometry is an infinite cylinder. The SDF of the cylinder can be calculated from a **c** point on the axis two vectors perpendicular to the axis (**n** the original surface normal and **t** the direction of κ_r are such) and the radius (r),

$$d_{cylinder} = \left\| \begin{bmatrix} \langle \mathbf{p} - \mathbf{c}, \mathbf{n} \rangle \\ \langle \mathbf{p} - \mathbf{c}, \mathbf{t} \rangle \end{bmatrix} \right\|_2 - r \ . \tag{6}$$

Here, $r = \frac{1}{|\kappa_r|}$ and we can get c with Equation (7) depending on whether the cylinder osculates the surfaces from the inside or outside. The two cases can be seen in Figure 8.



(a) $\kappa_r > 0$, the cylinder touches the surface from the outside

(b) $\kappa_r < 0$, the cylinder touches the surface from the inside

Figure 8: Calculating the axis point of the cylinder. Based on the sign of κ_r we have to step in the direction of $\boldsymbol{n}$ or in the opposite direction

$$\boldsymbol{c} = \begin{cases} \boldsymbol{p} + r \cdot \boldsymbol{n}, & \text{if } \kappa_r > 0 \\ \boldsymbol{p} - r \cdot \boldsymbol{n}, & \text{if } \kappa_r < 0 \end{cases} \tag{7}$$

When both $\kappa_r \neq 0$ and $\kappa_R \neq 0$ the geometry stored is a torus. For the simple SDF discussed previously we need to obtain the **t** axis, the **c** center and, the two r, R radii.

As discussed before, the sign of the curvatures is given in relation to the original surface normal at the footpoint, which is not necessarily the normal of the torus, see examples in Figure 9 (c) and (d). The correct normal can be calculated from the footpoint and the query texel center with the s sign as shown in 5.

The axis is already given, and $r = \frac{1}{\kappa_r}$. Let $R' = \frac{1}{|\kappa_R|}$ be the radius of the circle on which the footpoint is. This is not the radius R, but we know that $|R - R'| = r$. If the footpoint is an elliptic point then $R < R'$, if it is a hyperbolic point then $R' < R$, thus

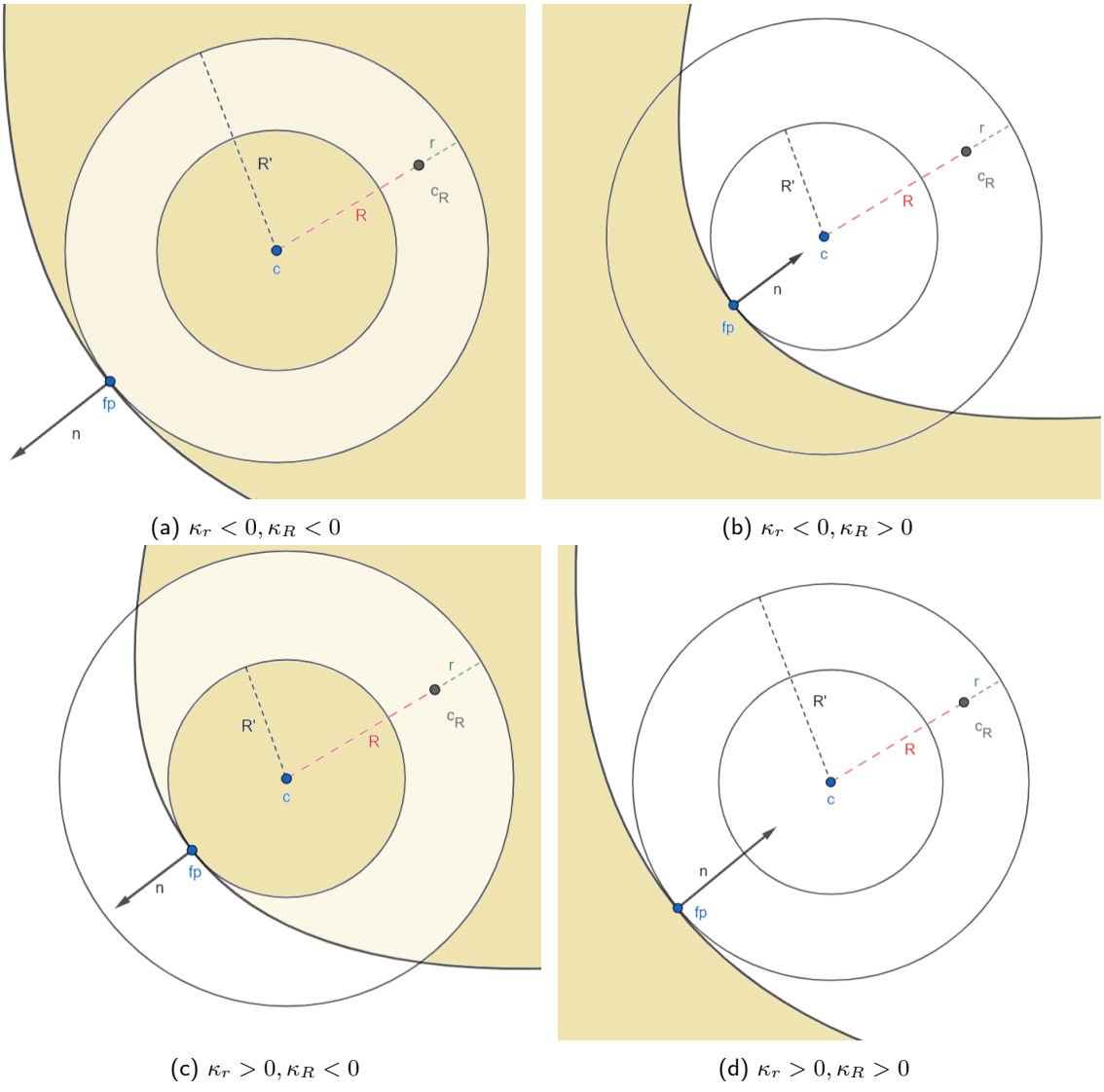
$$R = \begin{cases} \frac{1}{|\kappa_R|} + r, & \text{if } \kappa_r \cdot \kappa_R < 0 \\ \frac{1}{|\kappa_R|} - r, & \text{if } \kappa_r \cdot \kappa_R > 0 \end{cases} . \tag{8}$$

The position of c can be calculated from **fp** and R', depending on the sign of the curvatures:

$$\mathbf{c} = \begin{cases} \mathbf{fp} + \mathbf{n} \cdot \frac{1}{|\kappa_R|}, & \text{if } \kappa_R > 0 \\ \mathbf{fp} - \mathbf{n} \cdot \frac{1}{|\kappa_R|}, & \text{if } \kappa_R < 0 \end{cases} . \tag{9}$$

The different orientations of the torus and the surface can be seen in Figure 9.

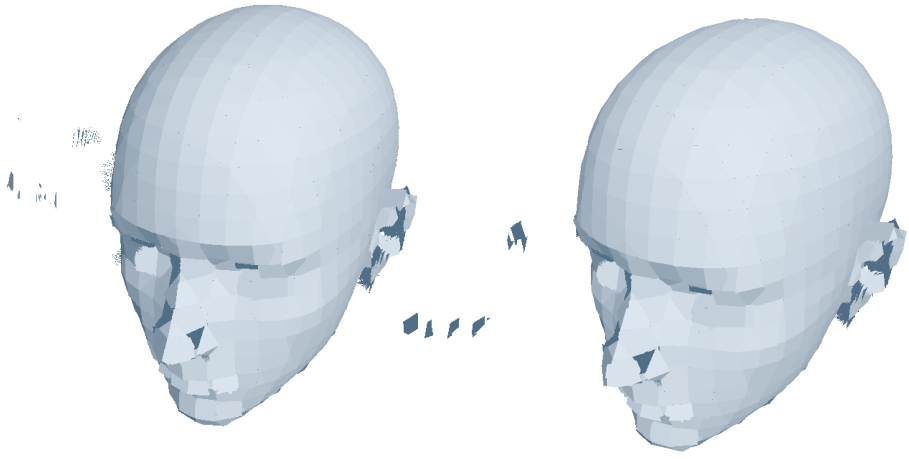
After decoding the field we can calculate the unsigned distance from the proxy by taking the absolute value of the distance function. This change is needed because the input surface does not necessarily have an inside outside partitioning, and even when it does possess one the proxy is not always touching the surface from the inside, see Figure 9 (c) and (d).



(a) $\kappa_r < 0, \kappa_R < 0$ (b) $\kappa_r < 0, \kappa_R > 0$

(c) $\kappa_r > 0, \kappa_R < 0$ (d) $\kappa_r > 0, \kappa_R > 0$

Figure 9: The orientation of the torus and the surface (in yellow). The relationship between $R' = \frac{1}{\kappa_R}$ and R is described by the sign of κ_R.



(a) An example of error with unconstrained proxies

(b) The corrected image with the constrained geometries.

Figure 10: An example of error, when we do not constrain our geometries. The approximation is only relevant around the footpoint, thus in query points far from the original shape we can get incorrect distance values, causing the floating artifacts. We constrained the geometries with a sphere around the footpoint.

Concluding these results, the correct unsigned distance based on the curvatures is

$$d = \begin{cases} |d_{plane}|, & \text{if } \kappa_r = 0 \text{ and } \kappa_R = 0 \\ |d_{cylinder}|, & \text{if } \kappa_R = 0 \text{ and } \kappa_r \neq 0 \\ |d_{torusz}|, & \text{if } \kappa_r \neq 0 \text{ and } \kappa_R \neq 0 \end{cases} . \tag{10}$$

An important consideration is that we approximate the surface only in a localized patch around the footpoint, therefore the proxies may give incorrect distances for points far away from the footpoint, causing errors as shown in Figure 10. This necessitates a constraint on the geometry to a smaller area. A straightforward approach that we used, is to limit geometries to the intersection of the surface and a sphere, with the radius of the sphere chosen appropriately.

In our implementation, our heuristic choice was to set the radius of the intersecting sphere as the size of a cell of the field. This decision was proven to be efficient in practice after comparing with other possible radius values, e. g. small constant numbers or multiples of the cell size.

For smoother connection between the proxies, we can use trilinear interpolation. To do this we calculate the distances to the proxies in the 8 closest texels and interpolate the distance values. This approach, however, does not respect the derivatives even at the footpoint. Therefore, we can use a Hermite interpolation method instead as shown in [14].

Using the blending functions $h_3(t) = -2t^3 + 3t^2$ for first-order data and $h_5(t) = 6t^5 - 15t^4 + 10t^3$ for second-order data instead of the general trilinear weights on the distances from the 8 geometries the derivative will be reconstructed correctly.

Sphere - Order 1 field

Res.	Norm	Parametric		SDF		Implicit	
		nearest	cubic Hermite	nearest	cubic Hermite	nearest	cubic Hermite
16^3	1	0.0019	0.003	**0.0017**	0.0032	**0.0017**	0.0027
	2	0.0000093	0.000015	**0.000001**	**0.000001**	0.0000062	0.000012
	∞	0.069	0.044	0.05	**0.032**	0.062	0.043
32^3	1	0.0005	0.0008	0.0008	0.0013	**0.00033**	0.00053
	2	0.0000022	0.000013	**0**	**0**	0.00000026	0.00000052
	∞	0.033	0.29	0.029	**0.02**	0.029	0.021

Sphere - Order 2 field

Res.	Norm	Parametric		SDF		Implicit	
		nearest	quintic Hermite	nearest	quintic Hermite	nearest	quintic Hermite
16^3	1	0.00000016	**0.0000001**	0.0000073	0.000057	0.00000032	0.0000006
	2	0.00000000091	0.00000000000028	**0**	**0**	0.0000000000004	0.00000000000056
	∞	0.02	**0.000012**	0.00031	0.0021	0.00021	0.0002
32^3	1	0.0000017	0.00012	0.000068	0.000049	**0.00000015**	0.00000019
	2	0.0000000027	0.000014	**0**	**0**	**0**	**0**
	∞	0.0033	0.29	0.00034	0.0035	0.00021	**0.00013**

7 RESULTS

The precision of our method was tested in various 3D environments. We evaluated the exact distance on a grid of resolution 217^3 and used it as a ground truth. During testing, we evaluated geometric fields of various resolutions on the same grid. Then we took the difference as the error. We compared error vectors using the average $\|\cdot\|_1$, $\|\cdot\|_2$, and $\|\cdot\|_\infty$ norms. Our fields were stored in 16-bit float textures.

We evaluated our method on basic shapes that are common in CAGD and also possess parametric and distance formulations. Our test shapes are shown on Figure 11.

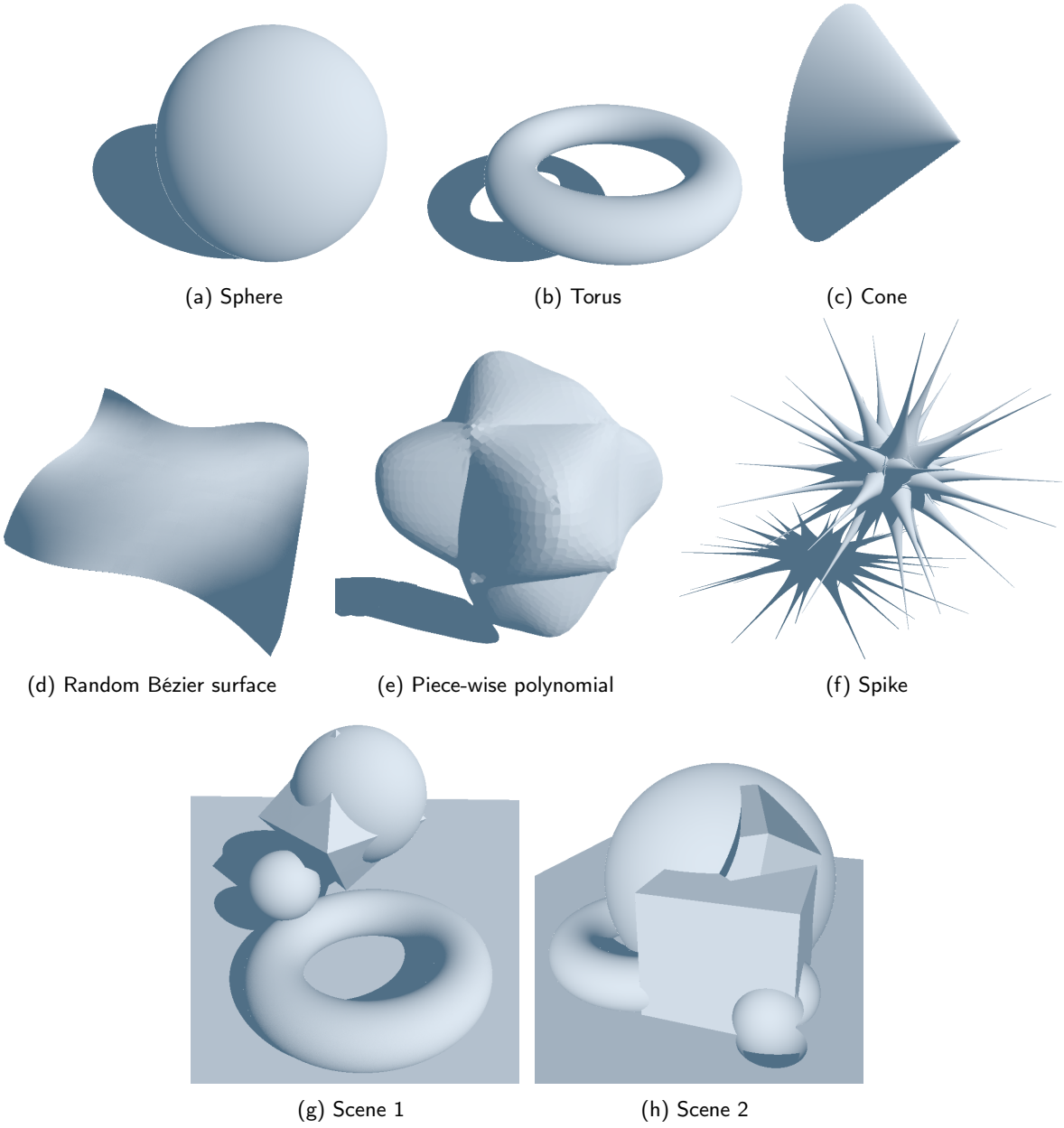
These results show that our algorithm for parametric surfaces and SDFs has a similarly good precision while on more complex shapes, the implicit surfaces perform poorly in comparison.

This can be attributed to large errors in the footpoint caused by the Newton method as it needs an initial guess close to the expected result to converge to the exact footpoint. However, for implicit surfaces, finding a point close to the real footpoint and ensuring the stability of the method is a difficult problem, which we can not address in this work. From the test it seems that our approach to constructing the field is correct, highlighted by the precision of the sphere, however the convergence of numerical optimization algorithms is beyond the scope of this paper, therefore making the method work for general implicit surfaces is an area of future research.

We conducted further measurements for parametric surfaces and SDF input to test their individual behavior on different shapes.

For parametric surfaces, we tested our method on a cone geometry, illustrated in Figure 11 (c),on random Bézier surfaces and on piece-wise polynomials consisting of multiple connected Bézier patches. Examples are shown in Figure 11 (d) and (e). As we do not have exact SDFs, we calculated our ground truth with the Newton-Rhapson method, and then used the same approach described above. The results can be seen in Table 2.

For SDFs, we conducted tests on shapes with small details and scenes combined with CSG operations from



(a) Sphere (b) Torus (c) Cone

(d) Random Bézier surface (e) Piece-wise polynomial (f) Spike

(g) Scene 1 (h) Scene 2

Figure 11: The 3D shapes used for testing. (a) and (b) was used for general comparison and performance tests, (c) and (d) for testing the method on parametric input, and (e) and (f) for SDF input

Torus - Order 1 field

Res.	Norm	Parametric		SDF		Implicit	
		nearest	cubic Hermite	nearest	cubic Hermite	nearest	cubic Hermite
16^3	1	0.15	0.16	**0.002**	0.0035	0.31	0.31
	2	0.036	0.35	**0.000001**	**0.000001**	0.13	0.13
	∞	0.31	0.31	0.068	**0.032**	0.73	0.72
32^3	1	0.05	0.04	**0.00069**	0.0011	0.24	0.28
	2	0.0039	0.0025	**0**	**0**	0.075	0.099
	∞	0.19	0.11	0.037	**0.017**	0.61	0.67

Torus - Order 2 field

Res.	Norm	Parametric		SDF		Implicit	
		nearest	quintic Hermite	nearest	quintic Hermite	nearest	quintic Hermite
16^3	1	0.00033	0.00045	**0.00031**	0.00033	0.26	0.26
	2	0.0000004	0.0000033	**0**	**0**	0.093	0.091
	∞	0.11	0.057	0.028	**0.019**	0.68	0.79
32^3	1	0.00023	0.00024	0.00023	**0.00022**	0.27	0.27
	2	0.000001	0.00001	**0**	**0**	0.095	0.096
	∞	0.11	0.36	0.0061	**0.0041**	0.7	0.83

Table 1: Accuracy measurements of the fields generated from different inputs, on given resolution with different filtering methods.

Cone

Res.	Norm	G1		G2	
		nearest	cubic H.	nearest	quintic H.
16^3	1	0.00089	0.0013	0.00055	**0.00052**
	2	0.0000039	0.0000055	0.0000012	**0.0000011**
	∞	0.085	0.087	0.082	**0.08**
32^3	1	0.00036	**0.00033**	0.00049	0.00046
	2	**0.00000099**	0.000001	0.000001	0.000001
	∞	**0.092**	**0.092**	**0.092**	0.093
64^3	1	0.00037	**0.00034**	0.00045	0.00043
	2	**0.00000077**	0.00000089	0.0000011	0.0000011
	∞	**0.073**	0.11	0.11	0.1

Random Bézier surface

Res.	Norm	G1		G2	
		nearest	cubic H.	nearest	quintic H.
16^3	1	**0.031**	0.065	0.039	0.042
	2	**0.0028**	0.014	0.004	0.015
	∞	**0.18**	0.54	**0.18**	0.67
32^3	1	**0.02**	0.063	0.023	0.044
	2	**0.0011**	0.015	0.0013	0.015
	∞	**0.14**	0.51	**0.14**	0.67
64^3	1	**0.012**	0.064	0.017	0.044
	2	**0.00037**	0.015	0.00071	0.016
	∞	**0.11**	0.52	0.12	0.67

Piece-wise polynomial (6 patches)

Res.	Norm	G1		G2	
		nearest	cubic H.	nearest	quintic H.
16^3	1	**0.028**	0.055	0.044	0.059
	2	**0.0021**	0.0097	0.005	0.0089
	∞	**0.11**	0.33	0.13	0.17
32^3	1	**0.022**	0.043	0.028	0.041
	2	**0.0014**	0.0057	0.0021	0.0042
	∞	0.084	0.17	**0.082**	0.11

Table 2: Accuracy measurements of the field generated for parametric Bézier surfaces. The table shows the results with nearest sampling and blended Hermite interpolation.

Spike

Res.	Norm	G1		G2	
		nearest	cubic H.	nearest	quintic H.
16^3	1	0.0083	0.011	**0.0048**	0.0059
	2	0.000006	0.000005	0.000005	**0.000004**
	∞	0.29	0.29	0.29	**0.28**
32^3	1	0.0022	0.0034	**0.0017**	0.002
	2	0.000002	0.000002	0.000002	0.000002
	∞	0.29	0.29	0.29	0.29
64^3	1	**0.00071**	0.001	0.00097	0.001
	2	0.000001	0.000001	0.000001	0.000001
	∞	0.29	**0.28**	0.29	**0.28**

Scene 1

Res.	Norm	G1		G2	
		nearest	cubic H.	nearest	quintic H.
16^3	1	0.003	0.0042	**0.0027**	0.0028
	2	0.000003	**0.000002**	0.000004	0.000003
	∞	0.27	**0.25**	0.27	0.27
32^3	1	**0.001**	0.0015	0.0013	0.0012
	2	**0.000001**	**0.000001**	0.000002	0.000002
	∞	0.27	**0.26**	0.27	0.27
64^3	1	**0.00056**	0.00067	0.00065	0.00063
	2	0.000001	0.000001	0.000001	0.000001
	∞	0.25	0.25	0.25	0.25

Scene 2

Res.	Norm	G1		G2	
		nearest	cubic H.	nearest	quintic H.
16^3	1	**0.0039**	0.0047	0.0085	0.0082
	2	0.000004	**0.000003**	0.000009	0.000007
	∞	0.38	0.26	0.29	**0.25**
32^3	1	**0.0016**	0.0018	0.0038	0.0036
	2	**0.000002**	**0.000002**	0.000004	0.000004
	∞	0.25	**0.2**	0.25	0.21
64^3	1	**0.001**	**0.001**	0.0035	0.0033
	2	**0.000001**	**0.000001**	0.000003	**0.000001**
	∞	0.24	**0.22**	0.24	0.23

Table 3: Accuracy measurements of the field generated for SDFs. The table shows the results with nearest sampling and blended Hermite interpolation.

simple shapes as shown on Figure 11 (f), (g) and (h). The results are presented in Table 3.

From these, we can conclude that the fields have good precision even on small resolutions without any filtering. For more difficult shapes, the first-order field offers better accuracy than the second-order or is around the same precision. This can be attributed to the first-order field being more robust, while the second-order field can be more dependent on the errors of the input. In the case of the SDF input, it is also affected by the size and orientation of the small grid taken when approximating the surface around the footpoint.

The data also shows that the maximum error does not significantly decrease with bigger resolution, furthermore, we can see big maximum errors even when the average and second norm are small. This can be caused by small errors in the footpoint, as the proxies are very sensitive to the footpoint position and even small inaccuracies can cause discontinuities and incorrect proxies.

To further validate this we conducted measurements, where we took a sphere for which we can calculate the footpoint with high accuracy. After finding the footpoints we took one of them and added a small noise by hand, then continued the proxy calculation as usual. This produced the same result when compared to the field generated with correct footpoints: while the average error remained minimal there was a large increase

Res.	Parametric	SDF	Implicit	Res.	Nearest	Hermite
16	0.06 ms	0.03 ms	0.01 ms	16	0.68 ms	3.12 ms
32	0.20 ms	0.07 ms	0.05 ms	32	0.75 ms	3.19 ms
64	1.33 ms	0.39 ms	0.32 ms	64	0.80 ms	3.39 ms

Table 4: The time of generating (left) and evaluating (right) fields.

in the maximum. When the error vector added to the footpoint had a length of 0.01 the maximum became 4 times bigger. This shows that the footpoint inaccuracies play a significant role in this phenomenon.

Lastly from the measurements, it is clear that the trilinear interpolation does not fit this approach. The interpolation is conducted on the distance values thus we lose a lot of geometrical data. For this reason, we are working on a filtering technique that respects the properties of the geometries.

We also conducted performance tests. When measuring the time of field generation, we used the shapes from Figure 11 (a) and (b) for comparison between the inputs on different resolutions. The result was the average performance on the different shapes as shown in the left part of Table 4. The test was conducted on NVIDIA GTX 1650 laptop GPU.

For testing the performance of the evaluation, we rendered the fields using a sphere tracing algorithm. Here, we did not need to differentiate between the different input geometry types and, the results were calculated as the average of the performance on different shapes. Our results are shown in the right part of Table 4.

The performance tests show that the field generation of the parametric surfaces is a bit slower, this can be attributed to needing to calculate an initial guess for the geometric Newton-Rhapson method.

8 CONCLUSION

In summary, we presented a novel method to approximate the signed distance function (SDF) of general surfaces with a small number of scalars.

We showed that approximation is possible up to a given order with appropriately defined proxy geometries. We described our reasoning for the specific proxies chosen and showed an algorithm for obtaining them for different inputs.

We introduced how to decode the fields and gave examples of filtering methods that reconstruct the SDF derivatives.

Our results show that geometric fields provide high accuracy even on small resolution, at a good performance. From our tests, it is clear that in general, the first-order field provides small average errors even on complex surfaces. The second-order field demonstrated its utility with more general curved surfaces and lower resolutions. However, compared to the G1 field, its approximation accuracy is lower, primarily due to potential numerical errors introduced during field construction. Future work could focus on developing a more robust field-generating method to enhance its applicability for general use.

Another area of further research is to improve the stability and precision of the footpoint finding methods to extend the concept for general implicit surfaces and reduce error for the other inputs as we showed that a part of the errors can be attributed to inaccuracies of the footpoint.

Lastly, to find a filtering technique that fits the concept of the geometric fields we plan on constructing a filtering method that combines multiple proxy geometries while keeping the geometric interpretation of the field.

ACKNOWLEDGEMENTS

Supported by the ÚNKP-23-2 and ÚNKP-23-4 New National Excellence Program of the Ministry for Culture and Innovation from the National Research, Development and Innovation Fund.

Anna Lili Horváth, https://orcid.org/0009-0006-2956-9227
Gábor Valasek, https://orcid.org/0000-0002-0007-8647
Róbert Bán, https://orcid.org/0000-0002-8266-7444

REFERENCES

[1] Bajaj, C.; Blinn, J.; Cani, M.P.; Rockwood, A.; Wyvill, B.; Wyvill, G.: Introduction to Implicit Surfaces. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. ISBN 1-55860-233-X.

[2] Bán, R.; Valasek, G.: First Order Signed Distance Fields. In A. Wilkie; F. Banterle, eds., Eurographics 2020 - Short Papers. The Eurographics Association, 2020. ISBN 978-3-03868-101-4. ISSN 1017-4656. http://doi.org/10.2312/egs.20201011.

[3] Eberly, D.: Fitting 3d data with a torus, 2020. https://www.geometrictools.com/Documentation/TorusFitting.pdf. Accessed: 2023-05-13.

[4] Franke, R.; Nielson, G.M.: Scattered data interpolation and applications: A tutorial and survey. In H. Hagen; D. Roller, eds., Geometric Modeling, 131–160. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991. ISBN 978-3-642-76404-2.

[5] Goldfeather, J.; Interrante, V.: A novel cubic-order algorithm for approximating principal direction vectors. ACM Trans. Graph., 23(1), 45–63, 2004. ISSN 0730-0301. http://doi.org/10.1145/966131.966134.

[6] Goldman, R.: Curvature formulas for implicit curves and surfaces. Computer Aided Geometric Design, 22(7), 632–658, 2005. ISSN 0167-8396. http://doi.org/https://doi.org/10.1016/j.cagd.2005.06.005. Geometric Modelling and Differential Geometry.

[7] Green, C.: Improved alpha-tested magnification for vector textures and special effects. In ACM SIGGRAPH 2007 Courses, SIGGRAPH '07, 9–18. ACM, New York, NY, USA, 2007. ISBN 978-1-4503-1823-5. http://doi.org/10.1145/1281500.1281665.

[8] Hu, J.; Yip, M.K.; Alonso, G.E.; Gu, S.; Tang, X.; Jin, X.: Efficient real-time dynamic diffuse global illumination using signed distance fields. Vis. Comput., 37(9–11), 2539–2551, 2021. ISSN 0178-2789. http://doi.org/10.1007/s00371-021-02197-0.

[9] Kallay, M.: A geometric Newton–Raphson strategy. Computer Aided Geometric Design, 18(8), 797–803, 2001. ISSN 0167-8396. http://doi.org/https://doi.org/10.1016/S0167-8396(01)00070-X.

[10] Loop, C.; Blinn, J.: Rendering Vector Art on the GPU, chap. 25. Addison-Wesley Professional, 2007. https://developer.nvidia.com/gpugems/gpugems3/contributors.

[11] Song, X.; Jüttler, B.; Poteaux, A.: Hierarchical Spline Approximation of the Signed Distance Function. In 2010 Shape Modeling International Conference, 241–245. IEEE, Aix-en-Provence, France, 2010. ISBN 978-1-4244-7259-8. http://doi.org/10.1109/SMI.2010.18.

[12] Valasek, G.: Nonlinear Geometric Models. Ph.D. thesis, Eötvös Loránd University, 2016. http://doi.org/10.15476/ELTE.2015.123.

[13] Valasek, G.; Bálint, C.; Leitereg, A.: Footvector representation of curves and surfaces. Acta Cybernetica, 25(2), 555–573, 2021.

[14] Valasek, G.; Bán, R.: Higher Order Algebraic Signed Distance Fields. Computer-Aided Design and Applications, 1005–1028, 2023. ISSN 16864360. http://doi.org/10.14733/cadaps.2023.1005-1028.

[15] Wright, D.: Dynamic occlusion with signed distance fields. In Advances in Real-Time Rendering in Games. Epic Games (Unreal Engine), SIGGRAPH, 2015.