EUROGRAPHICS 2025 / A. Bousseau and A. Dai
(Guest Editors)

Volume 44 (2025), Number 2

Implicit UVs: Real-time semi-global parameterization of implicit surfaces

Baptiste Genest, Pierre Gueth, Jérémy Levallois and Stephanie Wang

Adobe Research

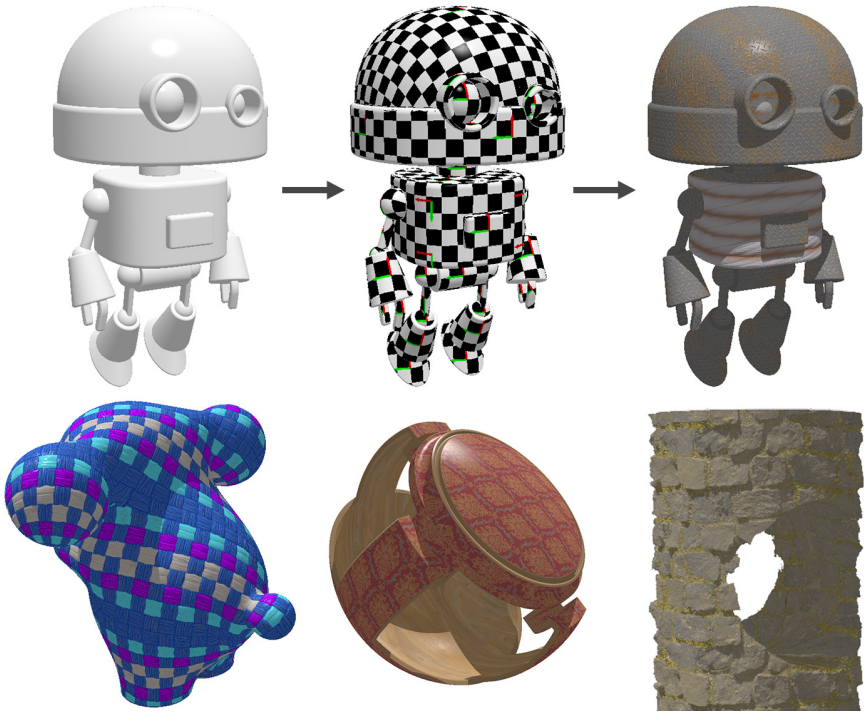


Figure 1: *Given an implicit surface (top left), we generate parameterizations with multiple seeds (top middle, seeds marked with red and green frames) and use them to apply rendering maps (top right). We can merge adjacent fields interactively, hence extending local uv-patches into semi-global parameterizations on smooth regions (bottom left). Our method implicitly segments the surface by sharp features and follows the geometry of the shapes, enabling the texturing of complex objects (bottom middle). Finally, since we never mesh the surface, we can still use the modeling advantages of implicit shapes, for example, to perform topological changes in real time (bottom right).*

Abstract

Implicit representations of shapes are broadly used in computer graphics since they offer many valuable properties in design, modeling, and animation. However, their implicit and volumetric nature makes applying 2D textures fundamentally challenging. We propose a method to compute point-wise and parallelizable semi-global parameterizations of implicit surfaces for texturing, rendering, and modeling purposes. Our method not only defines local patches of parameterization, but also enables the merging of multiple adjacent patches into large and spatially coherent ones that conform to the geometry. Implemented in shaders into a sphere-tracing pipeline, our method allows users to edit the uv-fields with real-time visualization. We demonstrate how to add rendering details (texture, normal, displacement, etc.) using our parameterization, as well as diffusing quantities on the surface and extending modeling tools with implicit shell maps. Furthermore, the textured objects remain implicit and can still be used in a modeling pipeline.

CCS Concepts

• ***Computing methodologies*** → ***Shape modeling;***

1 Introduction

Implicit surfaces are one of the most ubiquitous ways to represent shapes on a computer. Contrary to *explicit* representations, like a triangular mesh, which describes the $\mathbb{R}^3$-embedding of a collection of connected triangles, *implicit* representation only encodes a subset $M \subset \mathbb{R}^3$, defined as the set of zeros of a scalar field $f : \mathbb{R}^3 \to \mathbb{R}$,

$$M = \left\{ x \in \mathbb{R}^3 \,|\, f(x) = 0 \right\}. \tag{1}$$

Note that while it is possible that M contains volumetric regions (3D), curves (1D), or isolated points (0D), if we consider slight perturbations to f's value, M is usually a surface (2D). This scalar field f gives us a way to *query* the location of M. If, additionally, f satisfies the *Eikonal equation*,

$$\|\nabla f\| \equiv 1, \tag{2}$$

then f becomes a *Signed Distance Field (SDF)*, meaning that there exists a volumetric region Ω where $\partial\Omega = f^{-1}(0)$ and

$$f(x) = \begin{cases} -d(x, \partial\Omega), & \text{if } x \in \Omega, \\ d(x, \partial\Omega), & \text{else.} \end{cases} \tag{3}$$

When a surface M is encoded with an SDF f, an important technique, *sphere tracing* [Har96], computes the intersection of a ray and the surface by iterating

$$x_{k+1} = x_k + f(x_k)\ell, \tag{4}$$

where ℓ is the direction of the ray and $f(x_k) = d(x_k, M)$ is the largest "safe" (not intersecting with M) step size. If f can be evaluated in parallel, a GPU implementation in shader language of Eq. 4 [JQ14] can significantly speed up ray-tracing to achieve real-time rendering. This technique became the default rendering method for implicit surfaces because it enables 3D geometry editing in a fully implicit setting with real-time visualization.

We can model a scalar field f on a computer by storing either its values on a voxelized grid or its analytic expression. For example, we can store the values $f(x_{ijk}) = \|x_{ijk} - q\| - r$ of an SDF of a sphere $B_q(r)$ on the voxels x_{ijk} of a 3D grid, or we can store the sphere center $q \in \mathbb{R}^3$ and its radius $r > 0$. These two methods each have their trade-offs. While voxelized functions are more expressive and can handle freestyle edits, analytic functions enjoy a minimal memory footprint and infinite resolution. In this paper, though our method is agnostic to the type of functions (SDFs or non-SDFs) and the storage method (voxel or analytical expression), we focus on real-time geometry modeling and use analytic SDFs in most of our examples.

Many well-known design tools build on implicit representations. In *constructive solid geometry (CSG)*, one models the intersection of two shapes A and B by computing

$$f_{A\cap B} = \max(f_A, f_B), \tag{5}$$

where $f_\bullet$ denotes the SDF of $\bullet$. In *inverse rendering*, researchers often use implicit representation to model arbitrary geometry [VSJ22] because changing the surface topology can be made differentiable by changing f's value.

Implicit representation enjoys many appealing features, but what

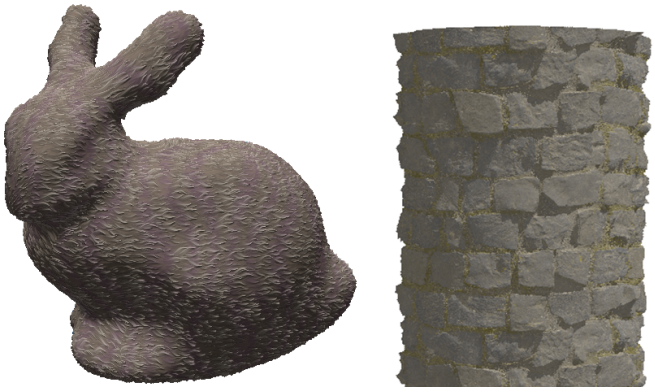


Figure 2: *We can render a bunny (left) and a cylinder (right) with color texture, normal map, and displacement in a sphere-tracing pipeline.*

are some of its drawbacks when compared to explicit representations? A mesh, for example, *explicitly* encodes a surface, an object intrinsically 2D but living in 3D. One can exploit this 2-dimensional nature to compute a 2D parameterization, for example using CETM [SSP08] or boundary-first flattening [SC17], so there exists a global mapping between the surface M and the uv-plane $\mathbb{R}^2$. This parameterization can be used to apply additional rendering properties

$$x \in \mathrm{M} \mapsto (u, v) \in \mathbb{R}^2 \mapsto \mathcal{A}(u, v), \tag{6}$$

where $\mathcal{A} : \mathbb{R}^2 \to Q$ could be color ($Q = [0, 255]^3$), normal ($Q = \mathbb{S}^2$), or displacement height ($Q = \mathbb{R}$). Since implicit surfaces are merely a subset in $\mathbb{R}^3$, one can only query the surface in a volumetric manner. Without first converting the implicit surface into a mesh (via *marching cube* [LC98] or *dual contour* [JLSW02]), one cannot easily parameterize the surface to add rendering details. This meshing step is computationally expensive and can introduce unwanted smoothed features. Furthermore, it depletes the advantage of using analytic SDFs for real-time modeling if the users have to generate the mesh to add details. We thus devise a fast and point-wise method to generate semi-global uv-coordinates on implicit surfaces for texturing purposes while avoiding meshing. By moving along the surface M using a second-order approximation of the shape, we build local parameterizations and provide a way to join multiple adjacent parameterized patches together into a common coordinate system. Built into a sphere-tracing renderer, our method runs in real-time on analytic SDFs and allows users to edit the uv-fields with instantaneous feedback. Finally, since we never mesh the surface, the final textured object remains implicit and can be used in a modeling pipeline, for example to perform topological operations on the shape in real-time, to deform it, or to exploit symmetries.

We summarize our contributions into the following list:

- A fully point-wise and parallelizable local parameterization built from an approximation of the geodesic path between the query point and a reference point.
- Each parameterization patch is defined on a smooth region delimited by sharp features.

- A smooth blending of multiple local parameterization patches into a *common coordinate system* for parameterizing a larger smooth portion of the surface. Our interpolation scheme is continuous and only blends near the interface between seeds.
- This parameterization system can be used to
  - Apply textures of color, normal, and displacement maps in a sphere-tracing rendering pipeline (see Fig. 2).
  - Diffuse and interpolate scalar and vector values on the surface.
  - Encode a complex (possibly explicit) shape as a displacement on a simple implicit surface.
  - Morph some other implicit geometry in the three-dimensional shell around the surface using shell maps [PBFJ05].
- The evaluation of this system runs in parallel per pixel and can be implemented on GPU.

2 Previous works

Purely implicit approaches

We first review techniques that directly aim at computing coordinates or adding details on implicit surfaces without discretizing the surface.

Many such techniques make heavy assumptions on the structure of the function f. For example, analytical fields, which are structured as the skeleton of simple shapes are used to compute coordinates and blending textures [TW99], add fine geometric details on the surface from a noise defined on the surface [ZBL*12], or from displacement maps on shapes with canonical coordinates, such as ellipsoids [SP91]. Another structured representation is proposed by Arquès et al. [AMP00] where an implicit surface is defined by a large number of blended discoid primitives. Since this representation is very dense in terms of sampling, they can encode complex geometry and store coordinates at each seed, enabling the use of textures.

Another approach is to define a flow from the surface to an enclosing volume, such as a sphere, on which coordinates are known [ZGVdF98] [TW99]. However, such a mapping process induces a lot of distortion and is valid mostly for simple, blob-like shapes.

Decal style that works on arbitrary geometry but only locally

On the other hand, some approaches work without assuming *any* structure on the surface, which can then be applied on *any* shape representation, including implicit surfaces. Indeed, in the work of De Groot et al. [DGWB*14], the authors propose a way to define a local coordinate system around some points in order to apply decals on generic surfaces. They do so by only using the Euclidean distance and an orientation frame carried by each point. This technique has been extended in several works, such as by using curvature information to improve distance estimation [RASS16], or by enhancing the interaction between the decals [NMBS21]. However, despite its efficiency and simplicity, the coordinate computation is too simple to be applied on topologically complex shapes at a large scale, hence being mostly restricted to decals.

Can be applied to implicit surfaces but need discretization

To bypass the absence of an explicit surface, some works first discretize the shape, using contouring [LC98] [JLSW02] to compute coordinates by applying some variations of the shortest path algorithm. In the work of Pederson et al. [Ped95], the authors subdivide the shape into parameterized patches, whose borders are geodesics. To compute the geodesic between two points, they first compute the shortest path on the contoured shape, which is then refined by a curve shortening flow restrained on the implicit surface (which is a common way to compute such curves [YZ10]). Finally, the authors match the derivatives of the coordinates between the patches in order to tile the surface. A variation of Djikstra's algorithm is also proposed in the work of Schmidt et al [SGW06], which also propagates exponential coordinates from a seed. Such a system provides an intuitive and controllable way to compute robust coordinates on implicit surfaces. However, since the shape is discretized, the output is not an implicit surface anymore and one loses the benefits of such a representation.

Dense representations

When the implicit surface is stored in a voxel grid, the geometry editing is simplified, and users can add fine details [EB17] [DHM12]. For example, by defining a coordinate system via a shell of particles, the work of Brodersen et al. [BMPB08] enables the use of shell maps [PBFJ05] to carve or add geometric details by morphing some other geometry on the surface of the shape. By fully embracing such volumetric representation, the work of Tarini [Tar16] bypasses the need of a surface by also computing and storing uv coordinates in voxels. In a similar way, the field of differentiable rendering may also encode shapes in such a volumetric representation [VSJ22]. However, although voxel-based approaches offer very high accuracy, the cost of such accuracy in terms of grid resolution often precludes real-time processing and control.

3 A local parameterization around a reference point

The goal of this paper is to extend what is possible when rendering implicit surfaces using sphere tracing. In such a pipeline, the GPU computes, in parallel, for each pixel of the screen the intersection between its ray and the surface. For each ray-surface intersection $x \in \mathrm{M}$, we would like to query additional rendering attributes $\mathcal{A}(u,v)$ (color, specularity, etc., see Sec. 5.2). We design a pointwise method of deriving uv-coordinates that is implementable in a GPU renderer.

In this section, we first create a local parameterization around a single reference point $p \in \mathrm{M}$. Recall that a local parameterization is a mapping from a neighborhood U of p to $\mathbb{R}^2$:

$$\begin{aligned} \varphi_p : U \subset M &\to \mathbb{R}^2 \\ x &\mapsto (u,v). \end{aligned} \tag{7}$$

We use a approximation of the exponential map (Sec. 3.1) together with an ambient heuristic (Sec. 3.3) to approximate the geodesic path between the query point x and the reference point p. We explain how geodesics embody local parameterization in Sec. 3.4. In Sec. 3.6, we lay out our algorithm and provide additional implementation details.

3.1 Local approximation of the exponential map.

3.1.1 Geodesics

Geodesics are the generalization of straight lines to a (curved) manifold. A curve $\gamma : [0,T] \to \mathrm{M}$ is a geodesic if it is a critical point of the following energy

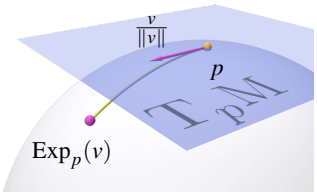
$$E(\gamma) = \int_0^T ||\dot{\gamma}(t)||^2 dt. \tag{8}$$

where $||.||$ is the standard Euclidean norm in $\mathbb{R}^3$. Such a curve is parameterized by arc length, i.e., $\|\dot{\gamma}\| \equiv 1$. We can understand geodesics as the locally shortest paths on a surface. A geodesic path beginning with point $p \in \mathrm{M}$ with an initial direction $v \in \mathrm{T}_p\mathrm{M}$ is the critical point of $E(\gamma)$ with the following *initial values*:

$$\begin{cases} \gamma_p^v(0) = p \\ \dot{\gamma}_p^v(0) = \frac{v}{||v||}. \end{cases} \tag{9}$$

The *exponential map* $\mathrm{Exp}_p : \mathrm{T}_p\mathrm{M} \to \mathrm{M}$ maps any vector v to the solution of Eq. 9 at $t = \|v\|$:

$$\mathrm{Exp}_p(v) = \gamma_p^v(||v||). \tag{10}$$



(See inset figure.) From the arc length parameterization, the *geodesic* distance between p and $\mathrm{Exp}_p(v)$ is exactly $d_M(p, \mathrm{Exp}_p(v)) = ||v||$.

3.1.2 First order approximation

An implicit surface $\mathrm{M} = f^{-1}(0)$ is uniquely defined by the function f; therefore, all its geometric properties can be inferred from f and its derivatives. For example, the gradient of f gives rise to the unit normal of M:

$$n(p) = \frac{\nabla f(p)}{||\nabla f(p)||}. \tag{11}$$

The tangent plane $\mathrm{T}_p\mathrm{M}$ is the orthogonal complement of $n(p)$,

$$\mathrm{T}_p\mathrm{M} = n(p)^\perp = \nabla f(p)^\perp. \tag{12}$$

Given an initial direction $v \in \mathrm{T}_p\mathrm{M}$, we can locally approximate the exponential map $\mathrm{Exp}_p(v)$ *à la* Forward Euler method:

$$\gamma_p^v(\tau) = p + \tau v. \tag{13}$$

However, one can check with a simple Taylor expansion of f that Eq. 13 only stays on M up to $o(\tau)$:

$$\begin{aligned} f(p+\tau v) &= f(p) + \langle v, \nabla f(p) \rangle \tau + o(\tau) \\ &= o(\tau), \end{aligned}$$

since $f(p) = 0$ and $v \perp \nabla f(p)$. This first order approximation requires very small time steps to keep the curve close to the surface. Since evaluating f is the most costly part of the pipeline, this can significantly slow down the process. Next, we proceed to describe a second-order approximation in order to bypass this limitation.

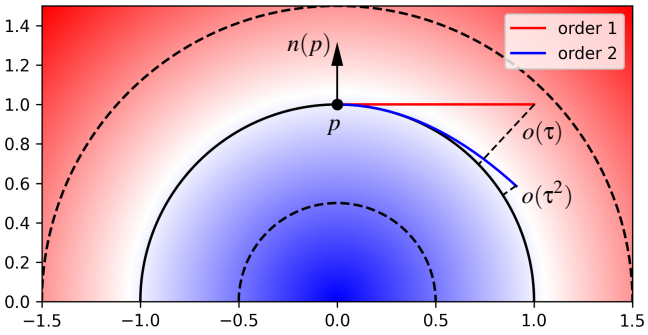


Figure 3: *Comparison of first vs second order schemes for Exp_p. A second-order approximation of the level set allows one to take larger, hence fewer, steps. The walk remains significantly closer to the surface than with the first-order method. The dotted-lines illustrate the size of the error and the location of the projection.*

3.1.3 Second order approximation

A second-order approximation of the exponential map on an implicit surface M is a path γ_p^v such that

$$f(\gamma_p^v(\tau)) = o(\tau^2). \tag{14}$$

To build such γ_p^v, we must use second-order geometric quantities of M. Let $n : \mathrm{M} \to \mathbb{S}^2$ denote the Gauss map as given in Eq. 11. The *shape operator* at p is defined as the differential of n,

$$\begin{aligned} S_p : \mathrm{T}_p\mathrm{M} &\to \mathrm{T}_p\mathrm{M} \\ v &\mapsto D_v n(p) = \left.\frac{\partial n}{\partial v}\right|_p. \end{aligned} \tag{15}$$

The eigenvalues κ_1, κ_2 and eigenvectors v_1, v_2 of S_p are the principal curvatures and principal curvature directions of M at p. For implicit surfaces, we can compute the shape operator S_p from f with

$$S_p = \left(I - n(p)n(p)^T\right) \frac{H_f(p)}{||\nabla f(p)||}, \tag{16}$$

where $H_f(p)$ denotes the Hessian of f at p. The shape operator gives a local quadratic approximation of the surface M,

$$\begin{aligned} \gamma_p^v(\tau) &= p + \tau v + \frac{\tau^2}{2} \langle v, S_p(v) \rangle \, n(p) \\ &= p + \tau v - \frac{\tau^2}{2||\nabla f(p)||} \left\langle v, H_f(p) v \right\rangle \, n(p). \end{aligned} \tag{17}$$

One can use the Taylor expansion of f at p, composed with the above, to verify that Eq. 14 is satisfied.

Evaluating the full Hessian in Eq. 17 would be pretty expensive (e.g. using finite difference, f needs to be evaluated 18 times!). Luckily, we only need the term $\langle v, H_f v \rangle = (D_v)^2 f$. The three-point finite difference scheme,

$$\langle v, H_f v \rangle = (D_v)^2 f = \frac{f(p-hv) - 2f(p) + f(p+hv)}{h^2} + o(h^2), \tag{18}$$

only requires 2 additional evaluations of f since we always need

$f(p)$. For highly curved surfaces or those C^1 but not C^2 (differentiable but not twice differentiable), one can re-project the approximation of $\mathrm{Exp}_p(v)$ onto M using:

$$\Pi_f(x) = x - f(x)\frac{\nabla f(x)}{||\nabla f(x)||^2}. \tag{19}$$

As shown in Fig. 3, the second order approximation permits larger steps and, therefore, fewer evaluations are needed to move a fixed distance T, even on irregular surface.

3.2 The log map as a local parameterization

As described previously, the exponential map Exp_p maps the tangent plane $\mathrm{T}_p\mathrm{M}$ to the surface M. If we select an orthonormal basis $e_1, e_2 \in \mathrm{T}_p\mathrm{M}$, we can then map $\mathbb{R}^2$ to M by

$$\begin{aligned} \mathbb{R}^2 &\to \mathrm{M} \\ (u,v) &\mapsto \mathrm{Exp}_p(ue_1 + ve_2). \end{aligned} \tag{20}$$

In other words, given a 2D coordinate (u,v), we can use Eq. 20 to deduce the 3D point location $\mathrm{Exp}_p(ue_1+ve_2)$ on M. This process describes the inverse of our goal Eq. 7. We therefore build the inverse of the exponential map, the *logarithmic (log) map*. It is defined locally on a neighborhood $U \subset \mathrm{M}$ near p:

$$\mathrm{Log}_p : U \to \mathrm{T}_p\mathrm{M}, \tag{21}$$

such that $\mathrm{Exp}_p(\mathrm{Log}_p(x)) = x$. As illustrated on Fig. 4, using the same parameterization of $\mathrm{T}_p\mathrm{M} \to \mathbb{R}^2$ for the fixed orthonormal frame $e_1, e_2 \in \mathrm{T}_p\mathrm{M}$, the Log map effectively maps U to $\mathbb{R}^2$, which is our goal!

Computing the inverse map is always harder than the forward map. The log map takes a query point $x \in \mathrm{M}$ and finds the velocity $v = \dot{\gamma}(0) \in \mathrm{T}_p\mathrm{M}$ of the geodesic γ from $p = \gamma(0)$ to $x = \gamma(T)$. Unlike the initial value problem posed in Eq. 9, here we need to solve the optimization problem with fixed *boundary values*:

$$\min_{\gamma(0)=p,\gamma(T)=x} E(\gamma). \tag{22}$$

In the following subsection, we explain how we use a heuristic to approximate the solution to this problem.

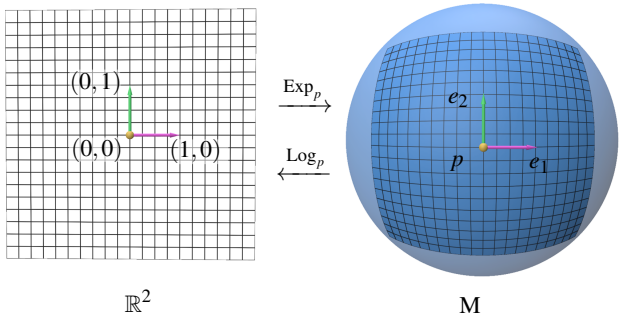


Figure 4: *With a selected orthonormal frame $e_1, e_2 \in T_pM$, the exponential map Exp_p maps a flat plane to the surface, while the log map Log_p provides each nearby point x with 2D coordinates.*

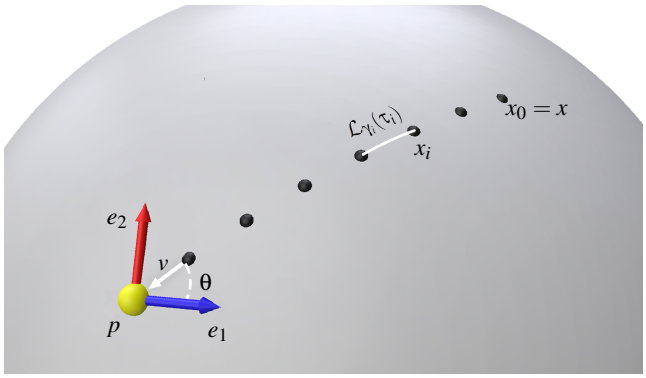


Figure 5: *Following a gradient descent of the Euclidean distance while adhering to the surface yields an approximated geodesic between x and p. We calculate the coordinates from the arc length of the curve and its incoming angle at endpoint p with a fixed frame e_1, e_2.*

3.3 Approximating the log map with ambient heuristic

Large-scale approaches to computing geodesics commonly use global methods to explore large portions of the geometry all at once [CLPQ20]. However, we want to avoid meshing the surface and to be able to run the queries in parallel, so we resort to locally approximating the geodesic for each query point x.

In a fashion similar to the $A^\star$ search algorithm, we follow a heuristic to guide the path. We approximate the geodesic from x to p with a gradient flow of the Euclidean ambient distance $d(y,z) = \|y-z\|$ while adhering to the surface M:

$$\begin{cases} \dot{\gamma}(t) &= \Pi_{\mathrm{T}_{\gamma(t)}\mathrm{M}}\left(\nabla d\left(\gamma(t), p\right)\right) \\ \gamma(0) &= x, \end{cases} \tag{23}$$

The gradient of Euclidean distance ∇d has a closed form,

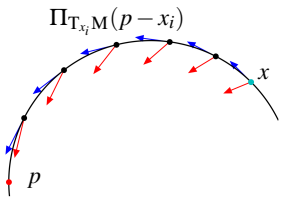
$$\nabla d(\gamma(t), p) = \frac{\gamma(t) - p}{\|\gamma(t) - p\|}. \tag{24}$$

So does the projection operator $\Pi_{\mathrm{T}_{\gamma(t)}\mathrm{M}}$ on the tangent plane $\mathrm{T}_{\gamma(t)}\mathrm{M}$,

$$\Pi_{\mathrm{T}_{\gamma(t)}\mathrm{M}} = I_3 - n(\gamma(t))n(\gamma(t))^T. \tag{25}$$

Furthermore, we can use the exponential map defined in Eq. 17 to keep the discretized sequence on the surface:

$$x_{i+1} = \mathrm{Exp}_{x_i}(-\tau_i \nabla^{\mathrm{M}}_{x_i} d(x_i, p)). \tag{26}$$



Here τ_i denotes the step size at step i. Our approach doesn't fall into the IVP-BVP (initial value problem vs boundary value problem) dichotomy presented in [CLPQ20] since we use a local geodesic tracing technique to approximate the geodesic between two points.

There is no guarantee that our approximation (Eq. 26) converges to an actual geodesic between x and p. It is also possible that our

calculated path from x to y differs from that from y to x. However, we only use this admittedly flawed approximation of geodesics only to build a coordinate system. From here on, we sometimes omit the adjective “approximated" and refer to these curves as “geodesics."

3.4 From geodesics to coordinates

The previous subsection lets us approximate the geodesic curve γ from x to p. Although we cannot convert the curve γ directly to uv-coordinates in T_pM, we draw inspiration from the conversion of polar to Cartesian coordinates:

$$(u,v) = r(\cos(\theta),\sin(\theta)). \tag{27}$$

Indeed, here we can estimate the length $r = \mathcal{L}(\gamma)$ and an angle θ between the direction of γ at its endpoint p and the fixed frame $e_1, e_2 \in T_pM$. See Fig. 5 for an illustration.

We compute $\mathcal{L}(\gamma)$ by measuring each of the N curves γ_i, joining $x_i = \gamma_i(0)$ and $x_{i+1} = \gamma_i(\tau_i)$, from Eq. 26:

$$\mathcal{L}(\gamma) \approx \sum_{i=0}^{N-1} \mathcal{L}(\gamma_i). \tag{28}$$

We then approximate each $\mathcal{L}(\gamma_i)$ using *Simpson's rule*:

$$\mathcal{L}(\gamma) = \int_0^\tau \|\dot\gamma(s)\| ds \approx \frac{\tau}{6}\Big(||\dot\gamma(0)|| + ||\dot\gamma(\frac{\tau}{2})|| + ||\dot\gamma(\tau)||\Big). \tag{29}$$

To estimate the incoming angle θ at p, we first calculate its unit tangent direction at the end point $p = \gamma(T)$:

$$v = \frac{\dot\gamma(T)}{\|\dot\gamma(T)\|}. \tag{30}$$

Since the frame $e_1, e_2 \in T_pM$ is orthonormal, we have the following relation:

$$(\cos(\theta),\sin(\theta)) = \big(\langle -v, e_1\rangle, \langle -v, e_2\rangle\big). \tag{31}$$

Finally, we compute the log map in (u,v) coordinates using:

$$\mathrm{Log}_p^{\mathrm{uv}}(x) = \mathcal{L}(\gamma)\big(\langle -v, e_1\rangle, \langle -v, e_2\rangle\big). \tag{32}$$

The curve-based parameterization has several advantages compared to those using direct estimators for implicit decals [DGWB*14] [RASS16]. Since we walk on the surface, we obtain a much more meaningful estimation of the geodesic distances, as displayed in Fig. 9, and we can accumulate additional geometric information along the curve as explained in the following subsection.

3.5 Geometric information accumulated along the curve

3.5.1 Parallel transport

Parallel transport refers to the way a tangent vector evolves when transported along a curve γ from one point of the surface to another. Here, we can build the mapping from the tangent plane of the seed T_pM to the one of the query points T_xM.

On a classical triangular mesh [CDS10], the smallest rotation matrix between two neighboring faces F_i and F_j gives a coherent discretization of the parallel transport between $T_{F_i}M$ and $T_{F_j}M$. This rotation is the same as the one that rotates the face normals n_i to n_j along the dihedral angle, given by the following formula:

$$R_{n_i,n_j} = I_3 + J + \frac{1}{1+\langle n_i, n_j\rangle} J^2, \tag{33}$$

where J is the skew-symmetric matrix such that $Ju = (n_i \times n_j) \times u$ for all $u \in \mathbb{R}^3$. We take inspiration from this formula and build the parallel transport matrix from reference point p to query point x by accumulating the rotation at each iteration between x_i and x_{i+1}:

$$R_{T_pM \to T_xM} = \prod_{i=1}^{n-1} R_{n(x_{i+1}),n(x_i)}. \tag{34}$$

From here on, use the notation $R_{p\to x} = R_{T_pM\to T_xM}$ interchangeably for simplicity. We illustrate the accumulation process in Fig. 6. This will be useful to apply normal maps in Sec. 5.2.1, to diffuse vector quantities in Sec. 5.3, and to compute optimal frames in Eq. 51 when merging multiple patches in Sec. 4.4.

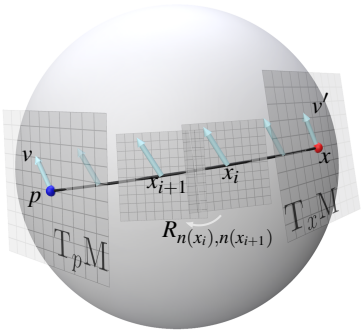


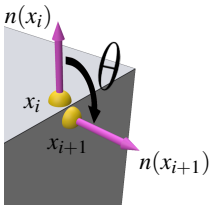
Figure 6: *We can discretize parallel transport along the curve built by Alg. 1. We store the product of each rotation matrix $R_{n(x_{i+1}),n(x_i)}$ into one rotation matrix $R_{p\to x}$. A vector $v \in T_pM$ is then transported to $v' \in T_xM$ by $v' = R_{p\to x}v$.*

3.5.2 Detecting sharp features

Until this point, we have built our theory on the assumption that the function f is sufficiently smooth. However, models found in the wild often contain sharp features where the normal is discontinuous, e.g. the 90-degree edges of a box. Our geodesic tracer would be unstable near these sharp edges and corners because there's no well-defined tangent plane to project the distance gradient, hence, the uv-patch cannot cross them in a robust way. We argue that this property does not diminish the usefulness of our method. In the context of 3D design, sharp features often divide the shape into semantically separate regions, each having their own uv island. For example, in Fig. 1, sharp features signal different components made of different materials in the robot.

Since our parameterization is based on geodesics connecting query points x and a reference point p, we can detect sharp features by testing for normal discontinuity along the curve:

$$n(x_i)\cdot n(x_{i+1}) < 1 - \epsilon_2. \tag{35}$$



Here $\epsilon_2 > 0$ is the tolerance to normal changes. This test may give out false positives, i.e., detecting sharp features if the shape smoothly but rapidly curves. One may use different tolerance ϵ_2 at different regions to accommodate the geometry.

3.6 Implementation

We sum up our method for a patch of local parameterization around a single reference point p in Alg. 1.

3.6.1 Limits of ambient heuristics

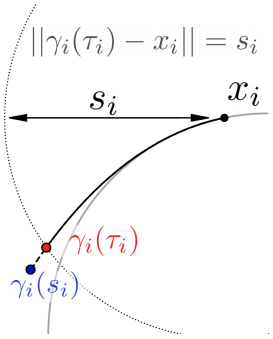
The gradient flow of the Euclidean distance doesn't always produce a valid curve between the query point x and the reference point p on the surface. Since we do not perform a global search to find the optimal path, the flow might get stuck in a local minimum. These local minima are the points x such that $p-x$ is co-linear with $n(x)$. However, since the function $x \mapsto d(x,p)$ is a continuous function on M with a global minimizer p, every point in the attraction basin of p must be able to follow the gradient flow and reach p.

3.6.2 Adaptive step size

One key ingredient in ensuring the continuity of the local parameterization is requiring that the algorithm takes the same number of steps at all query points. Take the case that x and x' are two nearby query points. Because we run Alg. 1 on x and x' independently, if one uses N steps and the other uses $N-1$ steps, that missing step can introduce discontinuities in their uv-coordinates. Using a fixed number of iterations with adaptive step size also helps evenly allocate computation time for each core of the parallel computation.

To fully utilize every iteration of the N steps, we must pick the step size τ_i of the gradient descent to maximize our chances of obtaining an almost-geodesic curve γ that reaches the reference point p. If the step size is too large, we can deviate from the level set M if it is highly curved; if it is too small, we might not reach the seed before the iterations run out. We set the time steps so that each iteration moves by the Euclidean distance between the current location and the goal $\|x_i - p\|$, divided by the remaining number of steps:

$$s_i := \|x_{i+1} - x_i\| = \frac{\|x_i - p\|}{N-i}. \quad (36)$$



To infer the step size τ_i required for the above equation, we take from Eq. 17 that $x_{i+1} = \gamma_i(\tau_i) = x_i + \tau_i v + \frac{\tau_i^2}{2}\alpha n$ for some scalar $\alpha \in \mathbb{R}$ and $||v|| = ||n|| = 1, v \perp n$. We then solve $\|\gamma_i(\tau_i) - x_i\|^2 = s_i^2$ for τ_i:

$$\tau_i = \sqrt{\frac{\sqrt{1+s_i^2\alpha^2}-1}{\frac{\alpha^2}{2}}} \leq s_i. \quad (37)$$

Note when the surface is flat, $H_f = 0$ and therefore $\alpha = 0$, we recover $\tau_i = s_i$. When the surface M is smooth, the fact that each query point takes the same number of steps with position-dependent step size makes the resulting log map depend continuously on the input. When the surface is merely C^1 and not

Algorithm 1: Log Map around a single reference point

Data: scalar field: F, query point $x \in$ M, reference point p, frame $(e_1, e_2) \in \mathrm{T}_p\mathrm{M}^2$, number of iterations N, maximum distance D, tolerance parameters ϵ_1, ϵ_2
Result: uv coordinates: $(u,v) = (r\cos(\theta), r\sin(\theta))$, parallel transport rotation matrix $R_{\mathrm{T}_p\mathrm{M}\to\mathrm{T}_x\mathrm{M}}$

```
if d(x,p) > D then
    return fail;    // exceeds maximal distance
end
x_0 = x;
r_0 = 0;
R_0 = I_3;
// Fixed number of iterations
for i ∈ [0, N-1] do
    s_i = d(x_i,p)/(N-i);
    τ_i = ComputeTimeStep(s_i);    // adaptive time step (Sec. 3.6.2)
    // Compute direction
    v_i = Π_{n_{x_i}^⊥}(p - x_i);
    if ||v_i|| < ε_1 then
        return fail;    // heuristic fails (Sec. 3.6.1)
    end
    γ_i = Γ(F, x_i, v_i/||v_i||);    // build geodesic (Eq. 17)
    x_{i+1} = γ_i(τ_i);
    if n_{i+1} · n_i < 1 - ε_2 then
        return fail;    // sharp feature (Sec. 3.5.2)
    end
    // Accumulate rotation from TM_{x_{i+1}} to TM_{x_i}
    R_{i+1} = R_i R_{n_{i+1}→n_i};
    r_{i+1} = r_i + L_{γ_i}(τ_i);
    if r_{i+1} + d(x_{i+1}, p) > D then
        return fail;    // exceeds maximal distance
    end
end
if d(x_N, p) < r_min then
    r = r_N + d(x_N, p);
    // Use x_{N-1} - p instead of x_{N-1} - x_N for numerical robustness
    t = Π_{n_p^⊥}(x_{N-1} - p);
    return (u,v) = (t/||t|| · re_1, t/||t|| · re_2), R = R_N R_{n_p→n_N}
end
return fail;    // exceeds maximal steps
```

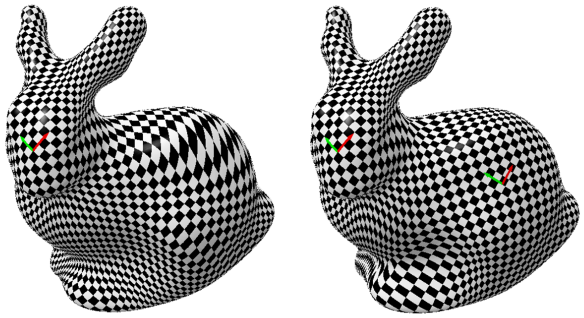


Figure 7: *Alg. 1 can parameterize a large area with one seed on a relatively smooth surface. However, all log maps accumulate distortion with curvature [SGW06]. With just one seed at the bunny's head, the parameterization gets distorted on its back (left). We propose a way to alleviate this issue with multiple patches of parameterization. Users may add a seed to the region where the parameterization of the first seed gets distorted and merge the two patches to get a smooth semi-global parameterization (right).*

C^2 (differentiable but not twice-differentiable), we found that reprojecting each x_i using Eq. 19 is enough in practice.

4 Semi-global parameterization using multiple patches

The previous section describes the method to compute a local parameterization patch given a reference point p and an orthonormal frame $e_1, e_2 \in \mathrm{T}_p\mathrm{M}$. We call the tuple $(p, e_1, e_2) \in M \times \mathrm{T}_p\mathrm{M}^2$ a *seed* for a patch of parameterization. In this section, we investigate using multiple seeds to reduce distortion (Fig. 7) and cover more area (Fig. 8).

4.1 Finding the geodesic-nearest seed

If there are n seeds $\{(p_i, e_{1i}, e_{2i})\}_{i=1}^n$, a query point x can move towards any of the n seeds to obtain its uv-coordinates. Previous

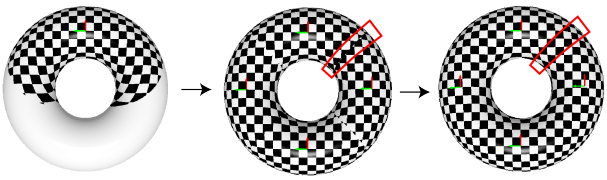


Figure 8: *Since one seed is often insufficient to parameterize a large area of the implicit surface (left), we can place multiple seeds to cover more area (middle). This, however, creates discontinuity between each of the parameterization patches (middle, discontinuity highlighted with a red box). We use the geodesic information between seeds to merge the patches and generate a semi-global parameterization without discontinuity (right, no texture discontinuity in the red box).*

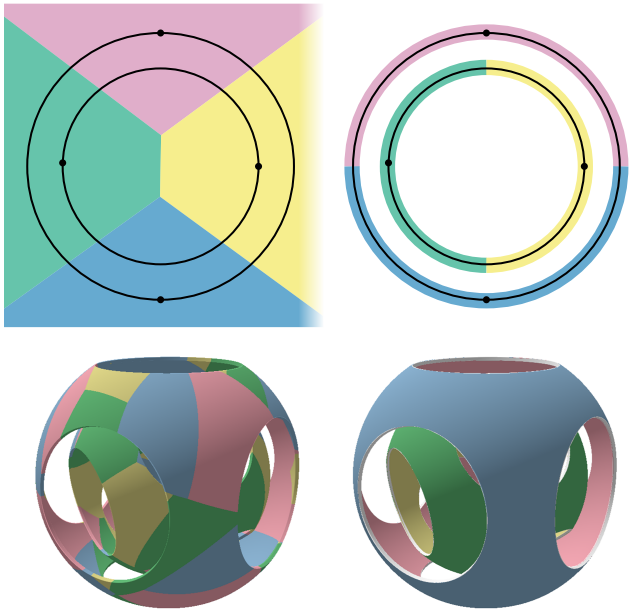


Figure 9: *We illustrate in 2D the Euclidean Voronoï cells (top left) and the geodesic Voronoï cells (top right) with the same seed placement on the same geometry. parameterization using the Euclidean nearest seed [DGWB*14] [RASS16] is only valid near the seeds and can produce erroneous results with too few seeds (bottom left). We use the geodesic nearest seed and the segmentation by sharp feature, which incorporates the geometry of the surface and produces favorable results (bottom right) with the same few seeds used in the bottom left.*

works use the seed closest to the query point measured by the Euclidean distance [DGWB*14] [RASS16]. This strategy can incur significant errors because a small Euclidean distance between x and p_i does not guarantee that x is semantically associated with p_i or even connected to p_i on the surface (Fig. 9 left). We replace the Euclidean distance $d(x, p_i)$ with the geodesic distance $d_M(x, p_i) = \mathcal{L}(\gamma)$ computed following Alg. 1. The *Voronoï cell* of a seed p_i, defined by the set of x where p_i is the closest seed to x, is the region where we compute the uv-coordinates based on (p_i, e_{1i}, e_{2i}). Using the geodesic closest seed implies partitioning the surface with the *geodesic Voronoï cells*, which are always simply connected. As a result, our strategy produces coherent parameterization on implicit surfaces regardless of geometry, and we can use much fewer seeds compared to previous works.

Since the geodesic distance is always longer than the Euclidean distance

$$d_M \geq d, \tag{38}$$

we can use the Euclidean distance to speed up the subroutine in finding the geodesic-nearest seed. We first find the n_{euc} closest seeds w.r.t. the Euclidean distance to the query point x, sorted by ascending order. Starting with the Euclidean closest seed, we call Alg. 1 to compute the geodesic distance $d_M(x, p_i)$ between x and each seed p_i. According to Eq. 38, if a seed p_i has a larger Eu-

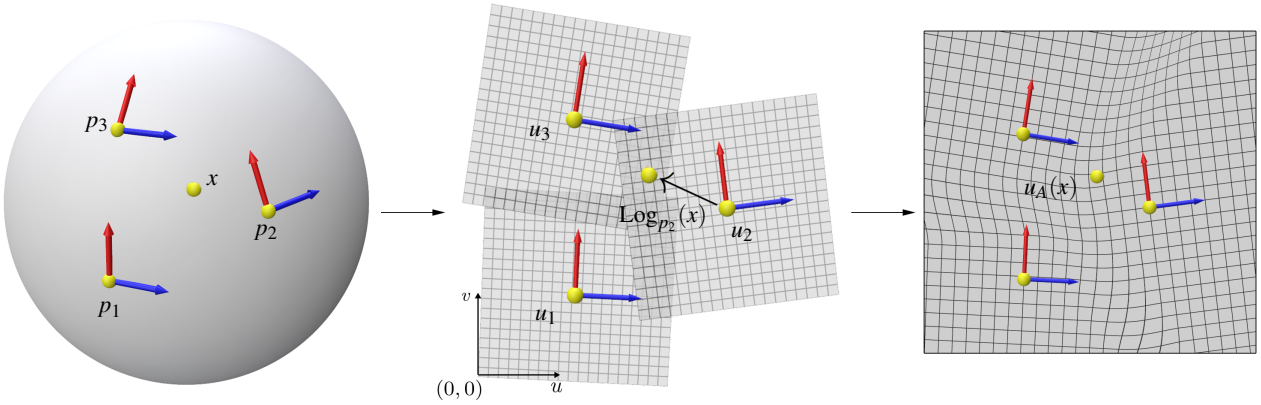


Figure 10: *To combine multiple uv-fields, we wish to take weighted averages of coordinates at a point x, coming from adjacent seeds (left). Since each Log map Log_{p_i} is expressed in its own referential, we first assign each seed p_i an offset coordinate $u_i \in \mathbb{R}^2$ in a common uv-space (middle). In this common coordinate system, we can blend the coordinates at the interfaces between the seeds using compact support interpolation (right).*

clidean distance $d(x,p_i)$ than the current minimal geodesic distance $d_M(x,p^*)$, p_i would have no chance to be closer to x than the current best seed p^*. Therefore, we can terminate the iteration early with very few calls to Alg. 1.

4.2 Merging patches overview

Existing techniques that generate a global uv-field on the entire implicit surface [SP91] [Tig01] can facilitate texture application but suffer from singularities and significant distortion since they are defined generically. On the other hand, techniques using local coordinate systems are much more controllable and have low distortion near each seed [DGWB*14] [SGW06], but they only permit small and localized texture patches ("decal" style). To benefit from the best of both worlds, we propose a way to merge multiple local coordinate fields into a more global one while maintaining quality and control. Moreover, since even exact log maps [SGW06] accumulate distortion with curvature, our method can be viewed as extending the valid region of the log map.

To generate a more global parameterization, a natural idea is to interpolate the uv-fields from different seeds:

$$u(x) = \frac{\sum_{i=1}^{n} w_i(x)\mathrm{Log}_{p_i}^{\mathrm{uv}}(x)}{\sum_{i=1}^{n} w_i(x)}. \tag{39}$$

In particular, one can use the Voronoï diagram of the seeds to guide the interpolation. To keep our evaluation fully point-wise, real-time, and implicit by nature, we cannot afford to compute interpolation weights as in natural neighbors techniques [BU06]. Therefore, we look to kernel-based techniques, i.e. using the distance to each seed to calculate the interpolation weight. However, interpolating uv-fields is more involved than interpolating fixed data, such as colors (Fig. 11). Since each parameterization $\mathrm{Log}_{p_i}^{\mathrm{uv}}$ is centered around $(u,v) = (0,0)$ at its seed p_i, simply taking a weighted sum of different uv-fields will result in incoherent texture. In particular, the uv-coordinates for every seed are all the same, $(0,0)$, even though they are scattered at different places on the surface! To express each uv-field involved in the interpolation in a common coordinate system, we assign each seed p_i with an offset location u_i in a mutually agreeable manner (Fig. 10 middle). We modify Eq. 39 with uv-offsets:

$$u(x) = \frac{\sum_{i=1}^{n} w_i(x)(\mathrm{Log}_{p_i}^{\mathrm{uv}}(x) + u_i)}{\sum_{i=1}^{n} w_i(x)}. \tag{40}$$

We discuss the blending weights w_i in Sec. 4.3 and the uv-offsets u_i in Sec. 4.4. We illustrate the complete pipeline in Fig. 10.

In practice, we build a graph (V,E) where the vertices are the seeds location $V = \{p_i\}_{i=1}^n$ and an edge is included $E_{ij} \in E$ if the user selects p_i and p_j for merging. Each edge is weighted by the geodesic distance $d_M(p_i,p_j)$. We also attach the uv-offset u_i to each seed $(p_i, e_{1i}, e_{2i})_i)$ into tuples,

$$(p_i, e_{1i}, e_{2i}, u_i) \in M \times (\mathrm{T}_p\mathrm{M})^2 \times \mathbb{R}^2, i = 1, \cdots, n. \tag{41}$$

We package these tuples together with the weighted graph into an *Atlas* data structure $A = (\{(p_i, e_{1i}, e_{2i}, u_i)\}_i, E)$ to be sent to GPU for rendering the textured surface. See Alg. 2 for the complete implementation for computing the blended uv.

Figure 11: *Techniques like triplanar mapping directly blend the color, resulting in blurred texture and incoherence (left). We extend several uv-fields into a common uv-coordinate system, enabling structured textures on a large scale (right).*

Finally, the location of the seams of the final parameterization is very easy to predict since it lies at the Voronoï frontier between two seeds, i and j, that are not merged together, i.e. if $(i,j) \notin E$. Note that one could use this system on the colors between unmerged fields to make the seams less visible.

4.3 Compactly supported weights

One could use the Gaussian kernel based on the geodesic distance between points. However, though it provides controllable and smooth interpolation, it has poor locality, i.e., the weights remain nonzero even if the points are far away. Since we want to avoid frequent calls to Alg. 1, we prefer to blend only at the Voronoï frontier (boundary of Voronoï cells) between two seeds. We first describe a kernel that only blends in the narrow band of the Voronoï frontier (Sec. 4.3.1). We then describe how to blend amongst more than two seeds (Sec. 4.3.2). Finally, we share a simple test using the Euclidean distance to check if a given seed p_i is needed to compute the final parameterization of the query point x (Sec. 4.3.2).

4.3.1 Approximation of the distance to the geodesic Voronoï frontier

In a purely Euclidean setting, one can evaluate the signed distance from a query point x to the Voronoï frontier $V_{i,j}$ between two seeds p_i and p_j by projecting the vector $x-p_i$ to the line connecting p_i and p_j:

$$d^{\text{Euc}}_{V_{i,j}}(x) = \left\langle x-p_i, \frac{p_j-p_i}{||p_j-p_i||} \right\rangle - \frac{||p_j-p_i||}{2}. \tag{42}$$

To replace the inner product structure with only length information, we first replace $\langle x-p_i, p_j-p_i\rangle$ with $||p-p_i||||p_j-p_i||\cos(\alpha)$ and use the cosine law,

$$\cos(\alpha) = \frac{l_0^2+l_1^2-l_2^2}{2l_0l_1}, \tag{43}$$

where $l_0 = d(x,p_i), l_1 = d(p_j,p_i), l_2 = d(x,p_j)$. Combining these formulas gives:

$$\begin{aligned} d^{\text{Euc}}_{V_{ij}}(x) &= \frac{||x-p_i||^2+||p_j-p_i||^2-||x-p_j||^2}{2||p_j-p_i||} - \frac{||p_j-p_i||}{2} \\ &= \frac{||x-p_i||^2-||x-p_j||^2}{2||p_j-p_i||}. \end{aligned} \tag{44}$$

To get an approximation of the *geodesic* distance to the geodesic Voronoï frontier, we replace all the Euclidean distances with the geodesic ones:

$$d_{V_{ij}}(x) \approx \frac{d_M(x,p_i)^2 - d_M(x,p_j)^2}{2d_M(p_j,p_i)}. \tag{45}$$

We use this approximated distance as a weight function to blend the uv-fields from these two seeds. Given a blending width $2\sigma > 0$, we define the weight using a smooth blending function ω (see Fig. 12):

$$w_{i,j}(x) = \begin{cases} 1, & d_{V_{ij}}(x) < -\sigma \\ 0, & \sigma < d_{V_{ij}}(x) \\ \omega\left(\frac{1}{2} - \frac{d_{V_{ij}}(x)}{2\sigma}\right), & -\sigma \le d_{V_{ij}}(x) \le \sigma. \end{cases} \tag{46}$$

Algorithm 2: Compute blended uv from an atlas

Data: Atlas $A = (\{(p_i, e_{1i}, e_{2i}, u_i)\}_i, E)$, query point $x \in M$, bandwidth $\sigma > 0$; parameters for Alg. 1: number of steps N, maximal distance $D > 0$, tolerance ϵ_1, ϵ_2
Result: Blended coordinates: $u(x) \in \mathbb{R}^2$

```
{p_{i_1}, ..., p_{i_n}} = SortSeed(x, A);          // sorted in ascending ||x - p_{i_k}||
L_1 = LogMap(x, p_{i_1}, N, D, ε_1, ε_2);          // Alg. 1
L* = L_1;
i* = i_1;
d*_M = ||L_1||;                                    // +∞ if Alg. 1 failed
// Find the geodesic nearest seed p_{i*}
for k ∈ [2, n] do
    if ||x - p_{i_k}|| ≤ d*_M then
        L_k = LogMap(x, p_{i_k}, N, d*_M, ε_1, ε_2);
        if ||L_k|| ≤ d*_M then
            i* = i_k ;                             // current best index
            d*_M = ||L_k|| ;                       // current geodesic distance
            L* = L_k ;                             // current uv
        end
    end
end
if |Neighbors(E, i*)| = 0 then
    // No blending required
    return L* + u_{i*} ;                           // offseted uv-coordinates
end
else
    // Compactly supported blending
    L = (0, 0);
    w_{i*} = 1;
    W = 0;
    for j ∈ Neighbors(E, i*) do
        d_{j,i*} = d_M(p_j, p_{i*}) ;              // edge weight of e_{ji*} ∈ E
        if (d*_M)^2 + 2σ d_{j,i*} ≥ ||x - p_j||^2 then
            // passes test from Sec. 4.3.3
            L_j = LogMap(x, p_j, N, d*_M + 2σ);
            w_{i*,j} = w(d*_M, ||L_j||, d_{j,i*});  // blending weight by Eq. 46
            w_{i*} = min(w_{i*}, w_{i*,j});        // Eq. 47
            w_{j,i*} = 1 - w_{i*,j};               // Eq. 48
            // Numerator of Eq. 40
            L = L + (w_{j,i*})(L_j + u_j);
            // Denominator of Eq. 40
            W = W + w_{j,i*};
        end
    end
    return (L + w_{i*} L*) / (W + w_{i*});
end
```

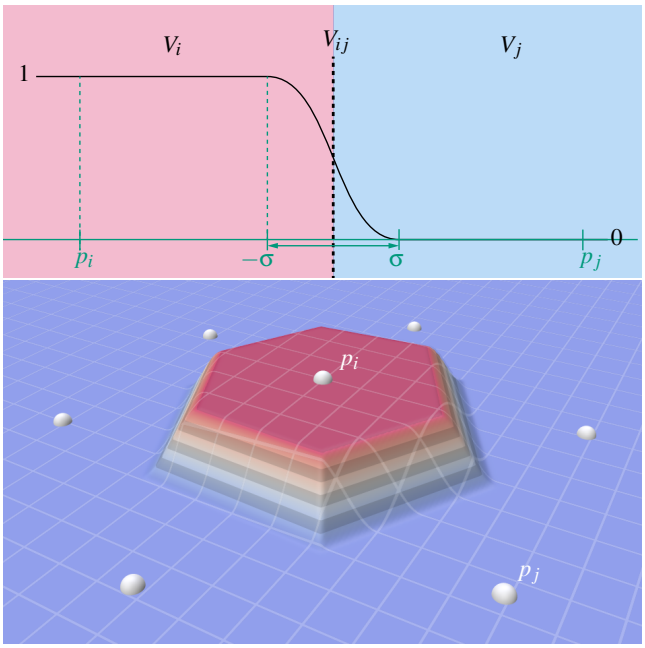


Figure 12: *Blending weight $w_{i,j}$ between two seeds p_i, p_j (top) and blending weight $w_i = \min_{j,e_{ij}\in E} w_{i,j}$ in relation to all surrounding seeds (bottom). The final weight w_i is continuous, compactly supported, and only takes values other than 0 or 1 on a narrow band near the Voronoï frontiers between seeds.*

To be more precise, we use $\omega(t) = 3t^2 - 2t^3$, but any smooth function with $\omega(0) = 0, \omega(1) = 1$ could work. In practice, we set $\sigma = \frac{1}{3}\min_{e_{ij}\in E} d_M(p_i, p_j)$ for all seed pairs. Intuitively, $w_{i,j}$ is the smoothed indicator function of the Voronoï half-space $H_{i,j} = \{x \in \mathrm{M} \mid d_M(x,p_i) < d_M(x,p_j)\}$.

4.3.2 Combining weights from multiple seeds

If a query point x falls in the blending area of more than two seeds, we need a weight $w_i(x)$ for each of these relevant seeds p_i. Since the Voronoï cell V_i is the intersection of all the Voronoï half-spaces $V_{i,j}$, we take inspiration from $\min(1_A, 1_B) = 1_{A\cap B}$, where 1_A is the indicatrix function of A, and propose the following weight function:

$$w_i(x) = \min_{j\,:\,e_{ij}\in E} w_{i,j}(x), \tag{47}$$

We plot the graph of this function in Fig. 12 (bottom). This formula is continuous and only depends on distances to nearby seeds selected for blending. Furthermore, to implement things efficiently in shaders, if p_i is the closest seed to x, we skip Eq. 47 for other seeds p_j and use the follow approximation:

$$w_j(x) \approx w_{j,i}(x) = 1 - w_{i,j}(x). \tag{48}$$

This assumption makes the evaluation linear in the number of seeds to blend (no need to compute w_{j_1,j_2}) and prevents caching. This assumption might not hold at the intersection of multiple Voronoï frontiers, but we found the discontinuity barely visible in all of our examples. The final blending then follows Eq. 40 using these weights.

4.3.3 A cheap test to avoid unnecessary computations

Suppose p_i is the closest seed to the query point x. If $w_{i,j}(x) = 1$ for a neighbor seed p_j, $e_{ij} \in E$, we would not need to compute $\mathrm{Log}^{uv}_{p_j}(x)$ since it will not influence the final uv at x. This happens when $d_{V_{ij}}(x) < -\sigma$, i.e. x is sufficiently far away from to p_j. We can use the Euclidean distance and Eq. 45 to make a cheap test:

$$\begin{aligned} w_{i,j}(x) = 1 &\iff \frac{d_M(x,p_i)^2 - d_M(x,p_j)^2}{2d_M(p_i,p_j)} < -\sigma \\ &\iff d_M(x,p_i)^2 + 2\sigma d_M(p_i,p_j) < d_M(x,p_j)^2. \end{aligned} \tag{49}$$

Since $||x-p_j||^2 \le d_M(x,p_j)^2$, we can test for the sufficient condition

$$d_M(x,p_i)^2 + 2\sigma d_M(p_i,p_j) < ||x-p_j||^2, \tag{50}$$

which saves us a call to Alg. 1 to compute $d_M(x,p_j)$ when this condition is met.

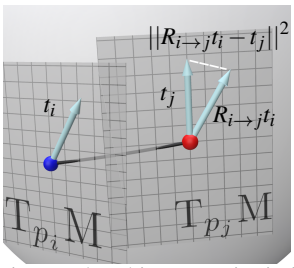
4.4 Optimal frames and uv-offsets

Even on a flat plane, two seeds with inconsistent frames $(e_1,e_2) \neq (e'_1,e'_2)$ can cause the merged parameterization to have severe distortion. We therefore optimize the frame placement and uv-offsets together to minimize distortion. We let the user specify the position of the seeds $\{p_i\}_{i=1}^n$ together with the blending graph E that indicates what seeds should be blended.

The *distortion* of a parameterization can be understood as a measurement of how it differs from an isometry (a locally rigid map). In other words, we will measure how the parametrized surface behaves differently from the uv-plane. We start by observing that, on a flat plane, parallel transport is always the identity map $R_{x\to y} = I_3$. Since we restrict our attention to ortho-normal frames, given e_1, we can uniquely determine e_2 by a 90° rotation on the tangent plane, using the formula $e_2 = n \times e_1$. We introduce the optimization variables $t_i \in \mathbb{R}^3$ for the frame vector e_1 at each seed. To penalize distortion, we select the optimal frames by minimizing a simple Dirichlet energy on the blending graph A, weighted by the inverse squared geodesic distance to favor interactions with closest neighbors:

$$E_{\text{frame}}\left(\{t_i\}_{i=1}^n\right) = \sum_{e_{ij}\in E} \frac{||R_{i\to j}t_i - t_j||^2}{d_M(p_i,p_j)^2}. \tag{51}$$

Here we use $R_{i\to j} = R_{p_i\to p_j} = R_{\mathrm{T}_{p_i}\mathrm{M}\to\mathrm{T}_{p_j}\mathrm{M}}$ to simplify the notation. Since this energy is invariant by global rotations, we fix one frame t_1 for one seed p_1.



If M is a flat plane, the global minimizer of the energy would be constant field $t_i = t, i = 1,\cdots,n$ and $E_{\text{frame}} = 0$. Once the optimal $\{t_i\}_{i=1}^n$ are computed, we set e_{1i} by projecting t_i to $\mathrm{T}_{p_i}\mathrm{M}$ and normalizing it. When M is curved, the optimal frame field will change as little as possible between two neighboring seeds. This energy is similar to the one used in the work of Knöppel et al. [KCPS13], except that

we apply it to a very small graph A and we fix one frame instead of finding the smallest energy eigenvector.

Once we computed the frames, we then use Alg. 1 to compute the uv-coordinates in reference to all neighboring seeds that require blending $\{\text{Log}^{\text{uv}}_{p_i}(p_j)\}_{e_{ij}\in E}$. Again, if M is flat, the uv-offsets u_i are trivial:

$$\text{Log}_{p_j}(p_i) = u_i - u_j. \tag{52}$$

We follow the same spirit and select the uv-offsets that minimize deviation from the above equation:

$$E_{\text{offset}}\left(\{u_i\}_{i=1}^n\right) = \sum_{e_{ij}\in E} \frac{||u_i - u_j - \text{Log}_{p_j}(p_i)||^2}{d_M(p_i,p_j)^2}. \tag{53}$$

Similarly, we fix the uv-offset $u_0 = (0,0)$ for one seed p_0 to remove redundant degrees of freedom.

Both energies E_{frame} and E_{offset} are quadratic and can be minimized by solving a linear system. The corresponding matrices are standard graph Laplacians, except that the frame energy matrix is defined by block where the parallel transport matrices are stored in the cross terms:

$$L^{\nabla}_{ij} = -\frac{R_{i\to j}}{d_M(p_i,p_j)^2}, \tag{54}$$

in a fashion similar to the discrete connection Laplacians [KCPS13].

It is important to note that the main advantage of our method resides in the fact that the linear systems are of size proportional to the number of seeds in the blending graph, a small number in practice. In our experience, we never needed more than 10 seeds per patch, whereas linear systems used to parameterize meshes are often applied on meshes with several thousands of vertices. We need to update the frames and uv-offsets if the user makes any change to the implicit surface, moves a seed, or changes the blending graph, but the systems' size makes them instantaneously solvable. Hence, the editing of the merged uv-field is also real-time!

4.5 Detecting topological obstruction

By definition, there exists no continuous and global mapping between geometries of different topologies. For example, we cannot map a cylinder to a plane without cutting the cylinder first. This is why placing three seeds on a cylinder and merging their uv-fields

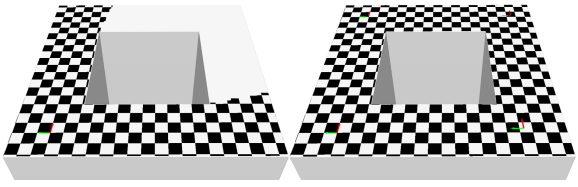


Figure 13: *The parameterization merging also solves some issues of the Euclidean heuristic: even if each local uv-field cannot cross the hole in the center (left), merging parameterizations from four seeds allows us to parameterize around it (right).*

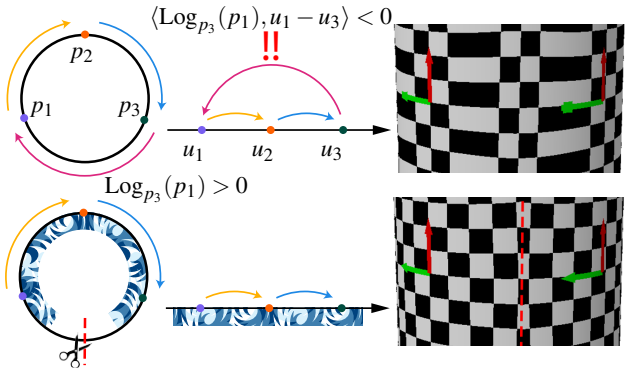


Figure 14: *In 1D, a circle cannot be parametrized by a straight line without inversion (top left, top middle). This is why a cylinder cannot be parametrized by a flat plane of texture (top right) without texture inversion. We detect this inversion between seeds and suggest a cut at their Voronoï frontier (bottom row).*

would not yield a valid global mapping. A seam must exist between at least two seeds. We want to detect such topological obstructions when the user inputs the merging graph so we can alert them that they are about to create an incoherent uv-field (See Fig. 14). Even though we avoid global operations to ensure a real-time pipeline, we can reliably detect uv defects caused by topology by checking if the global uv (Eq. 40) has a negative Jacobian, i.e. $\det(J) < 0$. It turns out we only need to check for

$$\left\langle \text{Log}_{p_i}(p_j), u_j - u_i \right\rangle < 0, \tag{55}$$

for each blending edge $E_{ij} \in E$ after computing the uv-offsets.

5 Applications

5.1 Remaining in a fully implicit setting

Why remain in implicit (point-wise) settings when one could just convert to meshes and parameterize them [SGW06] [Ped95]? We recount the advantages of remaining in a fully implicit setting First, the instantaneous computation per pixel per frame allows users to modify *any* parameter of our method or of the underlying model and get immediate feedback. This includes parameters of Alg. 1, seed placement, merging graph, etc... Artists can have the highest level of control during the design process. Another advantage is that the output remains implicit, so one could use our method at any node of a CSG construction tree to build independent textured objects before combining them. One can also perform topological changes on textured shapes, which is difficult on meshes (Fig. 18 right). Finally, while a high-resolution mesh is required for deformation, our method can generate parameterization on implicit surfaces warped by $\omega : \mathbb{R}^3 \to \mathbb{R}^3$ [SJNJ19] with no additional cost. Indeed, one can first texture a shape $\text{M} = \{f = 0\}$ in a rest pose, then, when rendering the deformed shape $\omega(\text{M}) = \{f \circ \omega^{-1} = 0\}$ with sphere-tracing, when an intersection x is found, i.e., $f(\omega^{-1}(x)) = 0$, Alg. Eq. 2 can be called on the function f directly but queried at the point $\omega^{-1}(x)$. In this way, as illustrated on Fig. 15, the walk Eq. 23 is performed on the rest pose while the uv appropriately follows the shape from M to $\omega(\text{M})$. In a similar fashion, if an object is

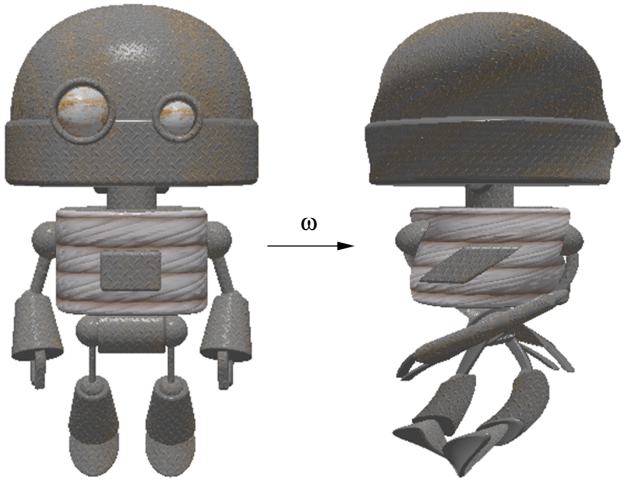


Figure 15: *When rendering an implicit shape deformed by a warping operator* ω*, one can compute the uv-coordinates on the rest pose (left) and transfer the texture to the deformed pose (right). A highly deformed mesh would show rough edges where the resolution is not high enough, while implicit surfaces can fully preserve their geometry.*

symmetric, for example w.r.t. the yz-plane, one can symmetrize the texturing by querying the uv at $(|x|, y, z)$, hence dividing the number of required seeds by 2.

5.2 Rendering details on implicit surfaces

In this section, we review several ways to use Implicit UV to add details on implicit surfaces. Applying color texture is straightforward. Given a texture picture $\mathcal{A} : \mathbb{R}^2 \to [0,255]^3$, the color at query point $x \in \mathrm{M}$ is computed by

$$x \mapsto u(x) \mapsto \mathcal{A}(u(x)). \tag{56}$$

To apply normal and displacement maps, we utilize the parallel transport from Sec. 3.5.1.

5.2.1 Normal maps

Recall that normal maps $N : \mathbb{R}^2 \mapsto \mathbb{S}^2$ are textures that contain three channels (n_1, n_2, n_3) consisting of the 3D coordinates of the material's normal vector at each point of the texture. They capture sub-resolution details that may change how light reflects off the surface. Normal maps were previously only available to meshes and other explicit representations due to their requirement for parameterizations. We aim to extend them to implicit surfaces rendered with sphere-tracing. Since normals are vectors, they are stored in coordinates (n_1, n_2, n_3) with a fixed basis $\{t, b, n\}$. At a given location x on a surface with the local tangent vector $t(x)$, bi-tangent vector $b(x)$, and normal vector $n(x)$, the normal map augments the rendering normal into

$$\tilde{n}(x) = n_1 t(x) + n_2 b(x) + n_3 n(x). \tag{57}$$

On meshes, the standard procedure uses the uv-coordinates on a triangle to estimate the basis $t(x)$ and $b(x)$ and uses the triangle normal for $n(x)$. The frame $\{t(x), b(x), n(x)\}$ is stored in a TBN matrix and

$$\tilde{n}(x) = \mathrm{TBN}(x) N(u, v). \tag{58}$$

On implicit surfaces, the tangent frame t_i and normal n_i at seed p_i are a part of the Atlas data structure. Therefore, we transport this frame from the nearest seed p_i to the query point x to obtain the rendering normal:

$$\tilde{n}(x) = R_{p_i \to x} \mathrm{TBN}_i N(u(x)). \tag{59}$$

5.2.2 Displacement maps

When f is an SDF, we can also use our method to add high frequency details by modifying the distance field query. Unlike most common methods for implicit surfaces, our method does not involve warping and is suited for structured displacement maps.

The idea is very simple: we want to displace the surface M, in the normal direction $n(x)$ by an amount $h(x)$. The problem is loading the height map $h : \mathbb{R}^2 \to \mathbb{R}$ from a 2D image on query points on a curved geometry $x \in \mathrm{M} \subset \mathbb{R}^3$.

To illustrate the process, observe that the graph of a function $h : \mathbb{R}^2 \to \mathbb{R}$, $\mathrm{M} = \{(x, y, z) \mid z = h(x, y)\}$ can be seen as the 0-level set of a function $f_h : \mathbb{R}^3 \to \mathbb{R}$ by

$$f_h(x, y, z) = z - h(x, y). \tag{60}$$

The above can also be seen as displacing the (x, y) plane $P_{xy} = \{(x, y, z) \mid z = 0\}$ by h in its normal direction e_z. The (undisplaced) signed distance function to P_{xy} is $d(x, y, z) = z$ to $P_{x,y}$, the parameterization of P_{xy} is

$$\begin{aligned} u : P_{xy} &\to \mathbb{R}^2 \\ (x, y, 0) &\mapsto (x, y). \end{aligned} \tag{61}$$

Considering the projection to P_{xy}, $(x, y) = \Pi(x, y, z)$, we can rewrite Eq. 60 as:

$$f_h(x, y, z) = d(x, y, z) - (h \circ u \circ \Pi)(x, y, z). \tag{62}$$

We can directly generalize Eq. 62 to curved surfaces defined by SDFs. Given an SDF $f : \mathbb{R}^3 \to \mathbb{R}$, our parameterization system $u(x)$, and a height map $h : \mathbb{R}^2 \to \mathbb{R}$, the modified SDF is

$$f_h(\xi) = f(\xi) - h \circ u \circ \Pi_f(\xi), \tag{63}$$

where the projection Π_f can be computed by:

$$\Pi_f(\xi) = \xi - f(\xi) \nabla f(\xi), \tag{64}$$

when ξ is close to the surface $\mathrm{M} = f^{-1}(0)$. We demonstrate this computation in 2D in Fig. 16.

To accelerate the computation, we simplify the function f_h away from the ϵ-shell of the original surface by

$$f_h(\xi) \approx \begin{cases} f(\xi) - \epsilon & \text{if } |f(\xi)| > \epsilon\tau \\ f(\xi) - (h \circ u \circ \Pi_f)(\xi) & \text{else,} \end{cases} \tag{65}$$

for some multiplier $\tau > 1$.

Zanni et al. had a similar expression in [ZBL*12], except they only considered noise functions defined on the surface instead of the height map term $h \circ u$. Sclaroff et al. also had an alternative formulation for displacement using warping [SP91]. Compared to previous works, our formulation generalizes more easily to other codimensions since it does not require normals on the shape. For example, as displayed in the inset, applying this displacement to the SDF of a 1D object (here a line) yields the surface of revolution.

Approximating normals from displacement

When no normal map is precomputed, we can still avoid using finite difference schemes on f_h to compute the normal of the displaced SDF.

We start from the fact that the vector $\nabla f(x)$ can be represented from any ortho-normal basis $\{e'_1(x), e'_2(x), e'_3(x)\}$:

$$\nabla f(x) = \sum_{i=1}^{3} D_{e'_i} f(x) e'_i(x). \tag{66}$$

Similar to Sec. 5.2.1, we use the parallel-transported frame $e'_1 = R_{p_i \to x} e_1, e'_2 = R_{p_i \to x} e_2$ from the nearest seed $(p_i, (e_1, e_2)_i)$ and the normal $e'_3 = n_f(x)$ for this ortho-normal basis. We know $D_{n(x)} f_h(x) = 1$ because Eq. 63 and $D_n \Pi_f = 0$ and $D_n f(x) = 1$ as f is an SDF. The true derivatives w.r.t. $e'_1(x), e'_2(x)$ need to consider the change of the frame $e'_1(x), e'_2(x)$, but we ignore it by assuming that the variations from the height map h dominates the derivatives. We thus obtain, for $\xi \in f_h^{-1}(0)$ on the height-displaced surface and $x = \Pi_f(\xi) \in f^{-1}(0)$,

$$\nabla f_h(\xi) \approx \nabla f_h(x) \approx \nabla f(x) + e'_1 \partial_1 h(u(x)) + e'_2 \partial_2 h(u(x)). \tag{67}$$

We approximate the partial derivatives $\partial_1 h, \partial_2 h$ using finite difference schemes on $\mathbb{R}^2$. Finally, we normalize ∇f_h to obtain the unit normal vector.

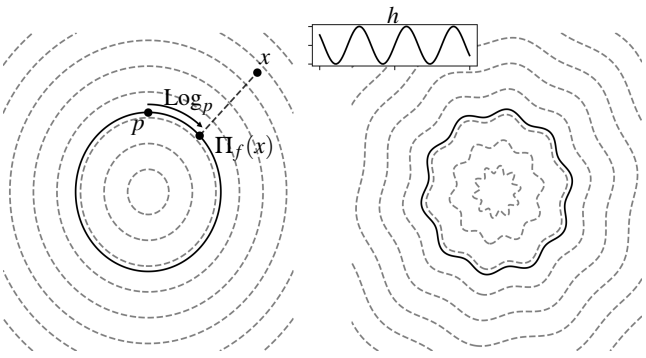


Figure 16: *In order to apply a displacement in the normal direction of a signed distance field (left), we first project the query point x onto the surface $\Pi_f(x)$, then compute the uv (here with one seed p) then subtract the height at said uv (right).*

It's worth noting that while this approximation is acceptable for visualization, it cannot provide reliable bounds for the Lipschitz constant $\max_{\xi \in \mathbb{R}^3} \|\nabla f_h(\xi)\|$ of the displaced SDF f_h. Computing the exact Lipschitz bound would involve knowing the curvature of the surface and parameterization distortion *a priori*. We cannot derive these pieces of information from our point-wise and seed-placement-dependent parameterization easily. However, we found that using a multiple of the standard Lipschitz constant used for height maps [GGP*15]

$$\lambda_h = \max_{x \in \mathbb{R}^2} \sqrt{1 + ||\nabla h(x)||^2}, \tag{68}$$

is usually enough in practice.

5.3 Interpolation and diffusion of scalar and vector quantities

One can use our fast estimate of geodesics and parallel transport to diffuse scalar or vector quantities. This can be used for painting / designing textures, for example.

Diffusion

Diffusion of quantities $\{a_i\}_{i=1}^N$ from N point sources $\{p_i\}_{i=1}^N$ is often solved with the *heat equation*:

$$\begin{cases} \partial_t u(x,t) = \Delta u(x,t), & t > 0 \\ u(x,0) = g(x), & t = 0, \end{cases} \tag{69}$$

where the initial value is the *Dirac-delta* of the quantities to be diffused,

$$g(x) = \sum_{i=1}^{N} a_i \delta_{p_i}(x). \tag{70}$$

The usual way to solve this problem on a meshed surface is to discretize the Laplacian Δ (e.g. using the cotan Laplacian) and then perform a time integration. Of course, here we want to avoid meshing the surface and solving the global problem.

We can mimic the solution to the heat equation by exploiting the geodesic distance between the source points and a query point $x \in M$. The fundamental solution to the heat equation is the *heat kernel*,

$$K(x,t) = \frac{1}{4\pi t} \exp\left(\frac{-||x||^2}{4t}\right). \tag{71}$$

By linearity, the solution to Eq. 69 is then $u(x,t) = \sum_i K(x - p_i, t) a_i$. We substitute $\|\cdot\|$ in the heat kernel with the geodesic distance and approximate diffusion on a curved surface M by

$$u(x,t) \approx \sum_{i=1}^{N} \frac{1}{4\pi t} \exp\left(\frac{-d_M(x,p_i)^2}{4t}\right) a_i \tag{72}$$

Interpolation

In a similar fashion, we can use kernel methods to interpolate values instead of diffusing them. For example, one can use a Gaussian

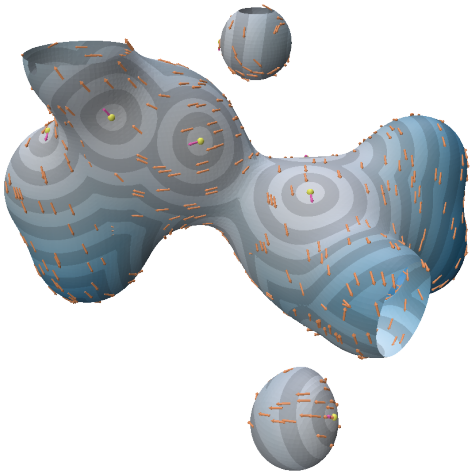


Figure 17: *Tangent vector field interpolation on a metaball surface by kernel methods using the approximated geodesic distance.*

Figure 18: *Conversion of a textured mesh into an implicit shape as a displacement of a sphere (middle). One can then trivially apply standard implicit modeling operators, here a time-dependant twist and a subtraction by a cylinder (right).*

kernel,

$$u_\sigma(x) = \frac{\sum_{i=1}^N \exp\left(-d_M(x,p_i)^2\sigma^{-1}\right) a_i}{\sum_{i=1}^N \exp\left(-d_M(x,p_i)^2\sigma^{-1}\right)}, \tag{73}$$

to interpolate the data $\{a_i\}$.

Diffusion on tangent spaces

We can use Eq. 72 to diffuse $\mathbb{R}^3$-vectors, but the result wouldn't be a field of tangent vectors even if the inputs are $(v_i \in \mathrm{T}_{p_i}\mathrm{M}, i = 1,\cdots,N)$. Therefore, one must use parallel transport to first bring all the vectors $v_i \in \mathrm{T}_{p_i}\mathrm{M}$ to the tangent space $\mathrm{T}_x\mathrm{M}$ at the query point x. We use the R matrices built by Alg. 1 for our point-wise interpolator:

$$v_\sigma(x) = \frac{\sum_{i=1}^N \exp\left(-d_M(x,p_i)^2\sigma^{-1}\right) R_{p_i\to x}(v_i)}{\sum_{i=1}^N \exp\left(-d_M(x,p_i)^2\sigma^{-1}\right)}. \tag{74}$$

The result of this interpolator is displayed in Fig. 17.

5.4 Encoding complex geometries as displacement maps of simple implicit surfaces

The displacement map presented in Eq. 65 can be used to add local geometric details on an implicit surface. However, one could also use it to encode the entire shape as a deformation of a simple one, as displayed on Fig. 18. This approach appears to have many advantages since the complexity of the uv computation is proportional to the complexity of the shape we walk on. For simple shapes like spheres, the displacement computation is almost free. Of course, one direct limitation is that the target shape T must belong to the set of shapes $\Phi_n(B)$ that are a deformation in the normal direction of the base shape B, i.e.:

$$\Phi_n(B) = \{S \in \Sigma^2(\mathbb{R}^3),\ \text{s.t.}\ , \exists h : B \to \mathbb{R}^+, S = B + hn_B\}, \tag{75}$$

where $\Sigma^2(\mathbb{R}^3)$ is the set of all 2-manifolds of $\mathbb{R}^3$ and $B + hn_B = \{x + h(x)n_B(x), x \in B\}$. For example, when $B = \mathbb{S}^2$ is a sphere centered at the origin, $\Phi_n(\mathbb{S}^2)$ is the set of radial heights. For applications like character animation, this limitation is not that restrictive, since the head could be a deformed sphere and the arms and legs deformed cylinders. The conversion process is straightforward since it only requires computing ray intersection queries from a sample of B to T and then storing the ray's length in the height map at the uv computed by Alg. 1. The conversion process is parallel and can be made almost instantaneous with a BVH structure when T is a mesh. Note that one can also capture the normals and the colors of the shape T in other maps. The resulting shape can then be rendered in real-time in a sphere-tracing pipeline while benefiting from the modeling possibilities offered by implicit approaches, such as animation or topological changes. This could be an interesting way to encode shapes since it benefits from the advantages of both explicit and implicit representations while being efficient, controllable, and lightweight without training neural networks.

5.5 Implicit shell maps

Getting access to uv-coordinates on a surface enables querying 2D data, but one can also use them to define a 3D coordinate system that follows the geometry of said surface. Such a coordinate system is known as a *shell map* [PBFJ05], and it is defined on an offset of the original surface, a *shell space*. An ϵ-shell space is defined as :

$$M^\epsilon = \{x \in \mathbb{R}^3 \text{ s.t. } d(x,M) \leq \epsilon\}. \tag{76}$$

We exploit the fact that, when M is the 0 iso-surface of a signed distance function f, such an offset is trivially defined as $M^\epsilon = \{-\epsilon \leq f \leq \epsilon\}$. In such a shell, we implicitly define a shell map from a 2D coordinate system on M, $x \in M \mapsto (u(x), v(x)) \in \mathbb{R}^2$ in the following way:

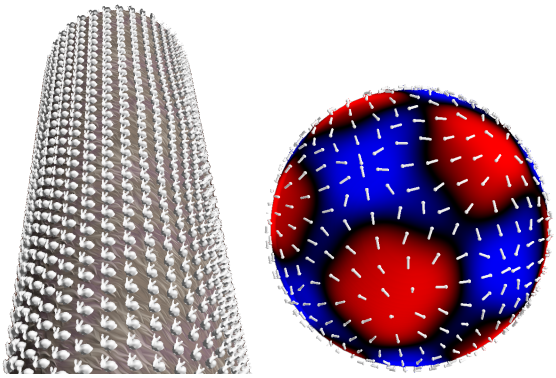
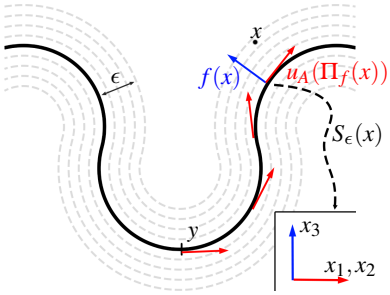


Figure 19: *A large number of bunnies can by instantiated on a cylinder by using a modular operator on the computed uv (left). A tangent vector field (here, the gradient of a scalar field) is visualized by placing 3D arrows and using a modular operator on the uv for the duplication (right).*

$$S_\epsilon : M^\epsilon \to \mathbb{R}^2 \times [-1,1]$$
$$x \mapsto \begin{pmatrix} u(\Pi_f(x)) \\ v(\Pi_f(x)) \\ f(x) \end{pmatrix} \epsilon^{-1} \tag{77}$$



As an application, one can copy the geometry induced by another implicit function g in the shell-space, by defining a new function f_g^ϵ as:

$$f_\epsilon^g(x) = \min(f(x), \epsilon g(S_\epsilon(x)) \tag{78}$$

Defining shell maps on implicit representations has been explored. In [BMPB08], the uvs and the shell coordinates are defined using a cage of particles placed on the surface. However, since the surface is represented by voxel grids, their method is computationally heavy and not suited for real-time processing and evolving geometries. Our method remains fully implicit. Therefore, we can directly use the above embedding by using our Atlas system in a sphere-tracing pipeline as displayed in Fig. 19.

Performance

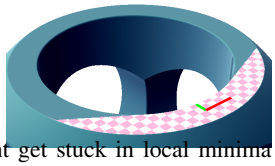
The performance of our pipeline is highly dependent on all of the following factors:

- Geometry of the surface $M = f^{-1}(0)$,
- Evaluation complexity of f,
- Screen resolution,
- Distance between camera and M,
- Number of steps N, number of seeds n, seed placement, number of blending pairs $|E|$, blending width σ,

We implemented our method in Windows and ran all our experiments on a laptop with an RTX 4080 laptop GPU on a Full HD screen. In our experience, most color-textured (including normal and roughness maps) examples achieved more than 40 frames per second, including the blue alien toy (Fig. 1 bottom left), bunny (Fig. 7), and torus (Fig. 8). Even for models as complex as the robot in Fig. 1 (top row), which consists of 42 seeds and 9 blending pairs, the fully color-textured visualization still reaches 25 fps. After adding displacement maps, the performance of rendering the column (Fig. 2 right) dropped from 60 fps to 40 fps and the bunny (Fig. 2 left) to 10 fps, though the sphere-displaced-to-head model (Fig. 18) remained at 60 fps.

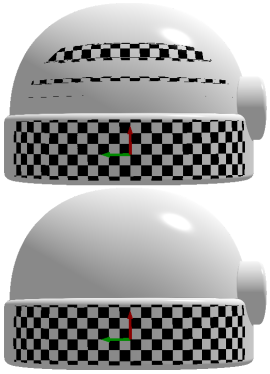
6 Limitations and Future works

In this paper, we propose a tool to perform *implicit* queries of *explicit* data. We use an ambient space heuristic to compute an approximation of the geodesic between two points and use it to derive the 2D coordinates in a point-wise manner. Our method enables one to apply additional details using texture mapping while still benefiting from the modeling advantages of implicit surfaces.



The limitations of our method mainly stem from our heuristical approximation of the log map. Since no global shortest path computation is performed, the gradient descent might get stuck in local minima when the geometry has a lot of concavities. While the merging system can help to overcome such locality, as displayed in Fig. 13, the segmentation imposed by the sharp edges remains. For example, as displayed on the inset, narrow regions delimited by sharp features may require many seeds for full coverage.

The ambient heuristic and the point-wise nature of our method can also affect the merging of uv-fields. In the scenario where two seeds p_i, p_j are selected for merging, if a query point x can reach one but not the other, even though we can compute the geodesic between p_i and p_j, we still cannot evaluate the merged uv-coordinates for x.



Since the method is fully point-wise, complete spatial coherence is hard to guarantee. For example, the sharp feature detection might not give spatially coherent results when the number of steps N for Alg. 1 is too low. As displayed in the inset, for $\epsilon_2 = 0.015$, the jump between the two parts of the head is too small to be detected by some points for $N < 14$, but is uniformly detected when $N \geq 14$.

While the performance of rendering a fully textured model remains real-time on most shapes, displacement mapping (Sec. 5.2.2) and implicit shell maps (Sec. 5.5) are often slower. These two operations are highly dependent on the complexity of the base SDF because they involve recomputing the UV at each sphere-tracing iteration in the shell. Therefore, complex SDFs might not be rendered in real-time when displaced. Having a way to start the computation of a coordinate from the one of the previous iterations could greatly

speed up the process. Using accelerating structures on the seeds and multi-scale rendering could also greatly improve the performance.

The interactivity granted by the implicit nature of our method allows one to place seeds and modify the uv-fields easily. An interesting future work could be the automatic placement of the seeds and the generation of the merging graph, for example using a variational approach.

Finally, the displacement of a simple shape into a more complex one (Sec. 5.4) appears to be a promising representation of shapes, falling in the intersection of the implicit and explicit categories. Fitting a complex geometry into a simple base SDF, possibly using a small neural network [CB24], with a height map using our method could potentially express a wide range of shapes.

Acknowledgements

We thank Lucien Dupont for allowing us to use the Robot Model [Dup22] and Benjamin Lindsay for the editorial assistance. The head model was created by "Jungle Jim" [Jim24].

References

[AMP00] ARQUÈS D., MICHELIN S., PIRANDA B.: Implicit surface driven by an atlas of discoids. *Graphicon'00* (2000), 256–261. 3

[BMPB08] BRODERSEN A., MUSETH K., PORUMBESCU S., BUDGE B.: Geometric texturing using level sets. *IEEE Transactions on Visualization and Computer Graphics 14*, 2 (2008), 277–288. 3, 16

[BU06] BOBACH T., UMLAUF G.: Natural neighbor interpolation and order of continuity. In *VLUDS* (2006), pp. 69–86. 9

[CB24] COIFFIER G., BÉTHUNE L.: 1-lipschitz neural distance fields. In *Computer Graphics Forum* (2024), Wiley Online Library, p. e15128. 17

[CDS10] CRANE K., DESBRUN M., SCHRÖDER P.: Trivial connections on discrete surfaces. *Computer Graphics Forum (SGP) 29*, 5 (2010), 1525–1533. 6

[CLPQ20] CRANE K., LIVESU M., PUPPO E., QIN Y.: A survey of algorithms for geodesic paths and distances. *arXiv preprint arXiv:2007.10430* (2020). 5

[DGWB*14] DE GROOT E., WYVILL B., BARTHE L., NASRI A., LALONDE P.: Implicit decals: Interactive editing of repetitive patterns on surfaces. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 141–151. 3, 6, 8, 9

[DHM12] DU S.-P., HU S.-M., MARTIN R. R.: Semiregular solid texturing from 2d image exemplars. *IEEE Transactions on Visualization and Computer Graphics 19*, 3 (2012), 460–469. 3

[Dup22] DUPONT J.: Robot Design. https://www.shadertoy.com/view/ftGcW1, 2022. [Online; accessed 17-September-2024]. 17

[EB17] EYIYUREKLI M., BREEN D. E.: Detail-preserving level set surface editing and geometric texture transfer. *Graphical Models 93* (2017), 39–52. 3

[GGP*15] GÉNEVAUX J.-D., GALIN E., PEYTAVIE A., GUÉRIN E., BRIQUET C., GROSBELLET F., BENES B.: Terrain modelling from feature primitives. In *Computer Graphics Forum* (2015), vol. 34, Wiley Online Library, pp. 198–210. 14

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

[Jim24] JIM J.: 18x Stylized Men's Heads. https://skfb.ly/oVnzp, 2024. [Online; accessed 17-September-2024]. 17

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 339–346. 2, 3

[JQ14] JEREMIAS P., QUILEZ I.: Shadertoy: Learn to create everything in a fragment shader. In *SIGGRAPH Asia 2014 Courses*. 2014, pp. 1–15. 2

[KCPS13] KNÖPPEL F., CRANE K., PINKALL U., SCHRÖDER P.: Globally optimal direction fields. *ACM Trans. Graph. 32*, 4 (2013). 11, 12

[LC98] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*. 1998, pp. 347–353. 2, 3

[NMBS21] NIYAZOV A., MELLADO N., BARTHE L., SERRANO M.: Dynamic decals: Pervasive freeform interfaces using constrained deformable graphical elements. *Proceedings of the ACM on Human-Computer Interaction 5*, ISS (2021), 1–27. 3

[PBFJ05] PORUMBESCU S. D., BUDGE B., FENG L., JOY K. I.: Shell maps. *ACM Transactions on Graphics (TOG) 24*, 3 (2005), 626–633. 3, 15

[Ped95] PEDERSEN H. K.: Decorating implicit surfaces. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), pp. 291–300. 3, 12

[RASS16] ROCHA A., ALIM U., SILVA J. D., SOUSA M. C.: Decal-maps: Real-time layering of decals on surfaces for multivariate visualization. *IEEE transactions on visualization and computer graphics 23*, 1 (2016), 821–830. 3, 6, 8

[SC17] SAWHNEY R., CRANE K.: Boundary first flattening. *ACM Transactions on Graphics (ToG) 37*, 1 (2017), 1–14. 2

[SGW06] SCHMIDT R., GRIMM C., WYVILL B.: Interactive decal compositing with discrete exponential maps. In *ACM SIGGRAPH 2006 Papers*. 2006, pp. 605–613. 3, 8, 9, 12

[SJNJ19] SEYB D., JACOBSON A., NOWROUZEZAHRAI D., JAROSZ W.: Non-linear sphere tracing for rendering deformed signed distance fields. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia) 38*, 6 (Nov. 2019). doi:10/dffm. 12

[SP91] SCLAROFF S., PENTLAND A.: Generalized implicit functions for computer graphics. *ACM Siggraph Computer Graphics 25*, 4 (1991), 247–250. 3, 9, 14

[SSP08] SPRINGBORN B., SCHRÖDER P., PINKALL U.: Conformal equivalence of triangle meshes. In *ACM SIGGRAPH 2008 papers*. 2008, pp. 1–11. 2

[Tar16] TARINI M.: Volume-encoded uv-maps. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 1–13. 3

[Tig01] TIGGES M.: *Two-dimensional texture mapping of implicit surfaces*. University of Calgary, 2001. 9

[TW99] TIGGES, WYVILL: A field interpolated texture mapping algorithm for skeletal implicit surfaces. *1999 Proceedings Computer Graphics International* (1999), 25–32. 3

[VSJ22] VICINI D., SPEIERER S., JAKOB W.: Differentiable signed distance function rendering. *ACM Transactions on Graphics (TOG) 41*, 4 (2022), 1–18. 2, 3

[YZ10] YU H., ZHANG J. J.: Geodesic computation on implicit surfaces. *International Journal of Information Science and Computer Mathematics 2*, 1 (2010), 33–49. 3

[ZBL*12] ZANNI C., BARES P., LAGAE A., QUIBLIER M., CANI M.-P.: Geometric details on skeleton-based implicit surfaces. In *Eurographics 2012-33rd Annual Conference of the European Association for Computer Graphics* (2012), Eurographics, pp. 49–52. 3, 14

[ZGVdF98] ZONENSCHEIN R., GOMES J., VELHO L., DE FIGUEIREDO L. H.: Controlling texture mapping onto implicit surfaces with particle systems. In *Proceedings of Implicit Surfaces' 98* (1998), ACM New York, USA, pp. 131–138. 3