Fast Texture Mapping for Triangle Soups using Electrostatic Monopole Field Lines

Alexander Schier[*], Stefan Hartmann[†], and Reinhard Klein[‡]

October 10, 2018

We present an efficient approach for the parameterization of triangle soups. Our technique tackles the problem by first approximating the triangle soup by a watertight 2-manifold offset mesh proxy. By establishing point correspondences between the triangle soup and the shape proxy by tracing electrostatic field lines in time $O(kn)$ instead of $O(kn^2)$, we can transfer properties computed on the surface, such as UV-coordinates even for large input meshes. The technique can process large triangle soups by replacing the computational intensive physical calculations by approximations developed initially for large-scale physical simulations. Thus it is possible to decrease the time complexity of the initial charge distribution algorithm from $O(n^3)$ to $O(n)$. We demonstrate our method on a multitude of challenging triangle soups and focus on the transfer of UV-coordinates during our experiments. An intensive run-time analysis and a comparison of our results to state of the art techniques in standard modeling tools concludes our study.

Published in Computers & Graphics, Volume 77, (Pages 140-155).
DOI: https://doi.org/10.1016/j.cag.2018.09.020

[*]schier@uni-bonn.de
[†]hartmans@cs.uni-bonn.de
[‡]rk@cs.uni-bonn.de

1 Introduction

Surface parameterizations are essential for many different applications in computer graphics, such as texture mapping, attribute transfer, and re-meshing. In the last three decades, a considerable amount of individual mesh parameterization techniques that focus on preservation of different mesh properties, like distances, areas, and angles, have been proposed to the computer graphics community (see [26] for a detailed introduction). Typically such algorithms require watertight 2-manifold triangle meshes and are not applicable for the parameterization of triangle soups, i.e., collections of triangles that do not necessarily have common vertices or edges and might have gaps or overlaps. Such mesh configurations hinder the task of computing smooth texture mappings. However, they are not rare. Examples are CAD objects that are composed of different parts that are not necessarily connected, see, e.g., Fig. 1, but rely on a common parameterization as otherwise noticeable discontinuities might be present. In this paper, we tackle the problem of computing parameterizations for triangle soups that has received far less attention within the computer graphics community in recent years. Having available a 2-manifold representation of the input mesh would allow the use of well-defined geometry processing algorithms in order to compute different kinds of mesh properties and even the use of existing high-quality surface parameterization techniques. In order to achieve our desired goal, we propose to solve the problem using two steps. First, we approximate the shape of the triangle soup by a surface that is watertight and 2-manifold. Such an approximation allows us to use state of the art algorithms for surface parameterization. Second, we establish unique point-to-point correspondences between the triangle soup and the shape proxy. Finding a high-quality mapping between the triangle soup and the shape proxy is not a trivial problem and although different techniques that establish correspondences exist, ambiguous correspondences based on distances might be found (see Fig. 2). In our approach, we tackle the problem of finding unique point correspondences by tracing field lines that originate from an electrostatic monopole, i.e., the charged input mesh, towards the shape proxy. The field lines follow a vector field that assigns a unique direction to each point in space, eliminating ambiguous correspondences by design. As the field lines eventually will hit the surface of the shape proxy and do not intersect each other, we can compute an *injective* mapping between the triangle soup and the shape proxy.

Existing approaches such as Degener and Klein [16], that aim for similar goals make use of dipole fields. Such fields rely on two different charges with opposite sign to establish correspondences between the triangle soup and the shape proxy. Although dipole fields are simple to establish and can provide a *bijective* mapping, they have a few significant drawbacks. First, the complexity of tracing the field lines for k steps is in $O(k(n+m)^2)$, while n and m being the number of faces of the triangle soup and the shape proxy respectively. Depending on the accuracy of the shape proxy, m might be much larger than n and making the field line tracing procedure even more computational intensive as each field line depends on the charges of all triangles. Second, the field lines strongly depend on the accuracy and the quality of the shape proxy and needs to be recomputed every time the shape proxy changes. For these reasons, we opted against a dipole field in favor of a monopole field, i.e., we only charge the triangle soup. Such an electrostatic monopole induces field lines that do not depend on a surrounding object and are significantly faster to calculate because they need a smaller amount of charged points distributed in space. Wang et al. [54] propose electrostatic monopole fields for harmonic parameterization of the space around objects. However, the computational complexity prevents the method from being used for large triangle meshes. An indispensable requirement on monopole fields is a physically exact charge distribution, i.e., an even potential on the surface of the object. Typically, the solution of a *dense* linear system provides the initial charge distribution. However, this results in a computational complexity that is in $O(n^3)$ and thus intractable for large triangle meshes. In contrast to Wang et al., we compute the initial charge distribution by leveraging techniques initially developed for large-scale physical simulations where the charge and potential fields are approximations, that do provide enough accuracy for establishing point correspondences and thus drastically reduce the complexity of the initial charge distribution from $O(n^3)$ to $O(n)$. In particular, the contributions of our paper are:

- We introduce a technique for the transfer of arbitrary surface properties, e.g., UV-coordinates, from a watertight 2-manifold shape proxy to a given triangle soup.
- We efficiently compute correspondences between a triangle soup and a mesh proxy by tracing field lines from an electrostatic monopole to the proxy surface, leveraging fast multipole methods to trace field lines in $O(kn)$ instead of $O(kn^2)$ needed in previous work.
- We show how the complexity of computing the initial charge distribution of the monopole can be reduced from $O(n^3)$ in previous work to $O(n)$ making the method applicable for large mesh objects.

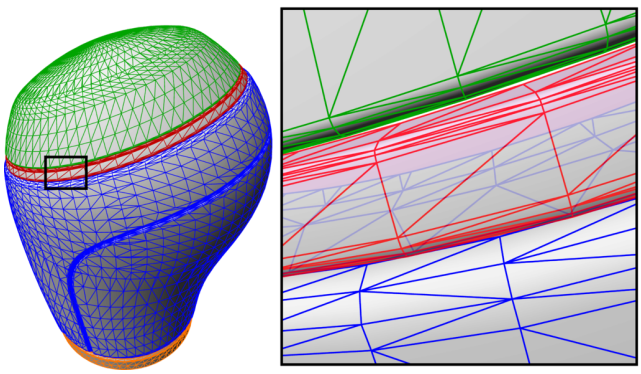


Fig. 1: A gear selector model illustrates that CAD models are composed of multiple simple parts, which do not form a common manifold mesh and contain gaps and overlaps.

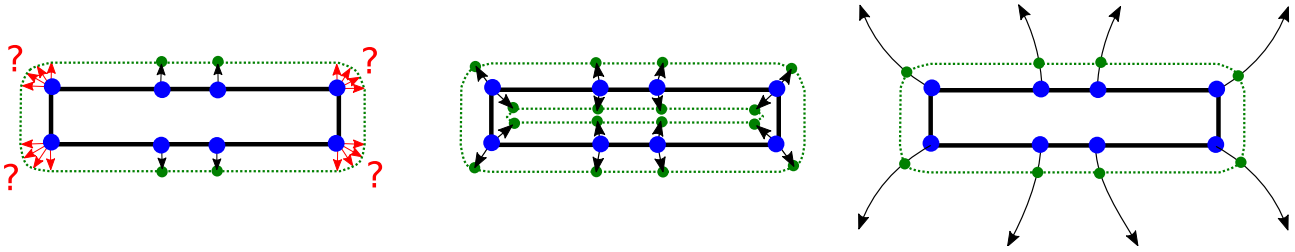


Fig. 2: Left: A distance based mapping, which cannot guarantee unique point correspondences. Middle: A dipole field defines a bijective mapping between a triangle soup and a two-sided offset mesh proxy. Right: A monopole field defines field lines in the whole space, which provide an injective mapping to any enclosing offset mesh proxy.

2 Related Work

Our work is related to three different but related fields, which are mesh parameterization, finding point correspondences, and generating mesh proxies.

Mesh Proxies For our parameterization method, we need to generate an enclosing offset mesh proxy. The generation can be done in various ways using approaches of different complexity. Wood et al. [57] generate isosurfaces from distance volumes in a two-step approach. They start with a coarse triangulation that provides a watertight mesh for the second step. The second step iteratively refines the triangulation using a force-based approach that adaptively fits the mesh to the data. Shen et al. propose a method to interpolate and to approximate implicit surfaces from a polygon soup [50] using a moving least-squares formulation. Cohen-Steiner et al. [13] describe mesh proxies using variational shape approximation and Wu and Kobbelt [58] use such a surface approximation for structure recovery. Yan et al. [59] propose a semantic segmentation based on the quadric fitting approach.

The method of Kazhdan et al. [27] extracts isosurfaces from octrees and Calderon and Boubekeur [11] describe an approach of building bounding proxies using mesh simplification with regularization of the input shape based on asymmetric morphological closing. The nested cages method [44] can be used to create a set of proxy meshes, in which each coarser triangulation completely encloses all finer ones.

Although we focus on using a simple distance-based approach, we can exploit the advantages of the techniques discussed above. As we only need to compute the field lines once, we can reuse them to test for intersections on a set of hierarchical proxies.

Parameterizations For surface parameterization a considerable amount of different techniques has been introduced (see, e.g., [19] for a detailed introduction) with focus on different properties of a mesh, as no parameterization can conserve all properties like angles, areas, and edge lengths. The surveys of [49] and [26] provide a good overview of the different approaches and algorithms that were introduced in recent years.

Common approaches include *"Most Isometric Parameterizations"* (MIPS) [24], *"Angle Based Flattening"* (ABF++) [48], *"As Rigid As Possible"* (ARAP) and *"As Similar As Possible"* (ASAP) parameterizations [34] using local-global algorithms, and the technique for minimal metric distortion proposed by Degener and Klein [15]. The *AMIPS* algorithm by Fu et.al [21] extends the MIPS algorithm using an inexact block coordinate descent method to avoid local minima and provides a maximum distortion control mechanism, such that single triangles with a significant distortion can be avoided. Fu et al. also describe an algorithm for *inversion-free mapping by simplex assembly* [20], which disassembles the mesh into its triangles and then optimizes for affine transformations instead of vertex positions. The reassembly is achieved by edge constraints, which ensure that the vertices of adjacent triangles stay connected.

Seamless parameterization have been tackled by Miles and Zorin [37] using *incremental flattening*. The technique of Rabinovich et al. [43] uses *locally injective mappings* and compatibility constraints to achieve seamless texture mappings.

By treating the boundary of a surface patch first and constructing the conformal map once the boundary is known, the *Boundary First Flattening* method [46] can reduce the cost for parameterization by a factor 30 to 50 to the next-fastest boundary-controlled method.

The *Autocuts* algorithm [42] approaches mesh parameterization in a different way. Instead of first computing seams and then parameterizing the individual charts, the authors propose an energy function that jointly optimizes for seams and a low distortion parameterization. Although by this algorithm the mesh is considered as a triangle soup, connectivity information between vertices is still necessary. In contrast, our algorithm does not require connectivity information. Instead, the triangle soup is used to generate a watertight offset mesh proxy, which then can be parameterized using any parameterization technique at hand.

Aigerman and Lipman use a generalization of Tutte embeddings to euclidean *orbifolds* in recent work [2, 3, 1], which results in a sparse linear system for producing bijective approximations of conformal mappings. A generalization of the Euclidean orbifold approach is the *Harmonic Global Parameterization with Rational Holonomy* (HGP) method by Bright et. al [9], which can handle arbitrary topologies and numbers of cone points.

Point Correspondences There are many different approaches to finding point correspondences, which often have entirely different goals, requirements on the input, e.g., given landmark points, and methodologies how they find the corre-

spondences. There are also many different metrics to measure the quality of point correspondences. A survey of different methods for shape correspondence can be found in [52], reviews of different methods for mesh morphing can be found in [29, 5], different similarity measures for shape matching are compared in [53], and in a more recent survey [8] summarizes trends in shape similarity assessment.

Point correspondences by flattening can be found using *Möbius voting* [33]. The meshes are embedded in the complex plane and Möbius transformations are computed using three fixed landmark points to establish the correspondences together with confidence values. *G-Flattening*, introduced by Aigerman et al. [4], computes seamless bijective mappings from charts flattened to the plane. Their technique uses an optimization procedure which is invariant to the cuts used to produce the initial charts.

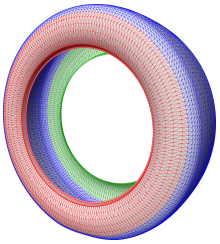
Using *heat kernel maps* [39], isometries on compact manifolds of any dimension and genus can be found with only one given landmark point by utilizing the solution of the heat equation. The *Blended intrinsic maps* (BIM) [28] method combines different nearly isometric maps such as *Möbius maps* and h*eat kernel maps* to compute a mapping between two genus zero surfaces. These maps are weighted to minimize the error in an optimized final map. An extension of the method for meshes of higher genus described by Sandilands et al. [45] uses electrostatic potential isosurfaces to find genus zero approximations of the object to be able to use a blended map.

The *Functional Maps* [38] approach utilizes a map between function spaces on two different shapes. By choosing the indicator function, the functional map is a simple permutation of the vertices, and by choosing more sophisticated bases, the representation can be simplified and becomes more robust. A typical choice for the basis are the eigenfunctions of the Laplace-Beltrami operator which establishes the connection to the heat kernel maps approach. A detailed introduction to functional maps can be found in [12]. An extension of functional maps by using the Green's functions of the Laplace-Beltrami operator to embed shapes into their own function space to calculate correspondences between non-isometric shapes with the constraint that the mapping function approximates a conformal map is given in [10].

Finally our approach is related to the texture mapping approach using *dipole field lines* of Degener and Klein [16] but employs *monopole fields* to get more accurate results. Our technique significantly reduces the number of charges that are required to compute the potentials. We replace their simple speedup heuristic of truncating the list of charges to consider by a sound physical approximation using the fast multipole method, which has well-known error bounds. Besides, we avoid using a two-sided mesh proxy, because our field lines exactly match the physical model and thus always extend to the outside, intersecting any enclosing mesh. In a recent paper by Barill et al. [6], another field approximation algorithm based on Taylor expansion is used for fast calculation of winding numbers of point clouds and triangle soups by using dipoles.

3 Motivation

There is a vast amount of meshes that have highly irregular triangles and non-standard topologies. Such meshes are typically used in game production and often occur in CAD applications for product design. For the latter, the individual parts are modeled using NURBS (see, e.g., [40]), and their representations are used in spray-cast and injection-molding production pipelines. At the same time, these CAD object parts need to be used for product visualization. For efficient visualization these NURBS surface are triangulated, resulting in highly irregular triangle meshes that are typically non-manifold. In contrast to geometry processing, these "bad" meshes are acceptable to be used in visualization pipelines. Especially for visualization of such objects, e.g., the tire composed of three surfaces shown in the inset and the gear selector shown in Fig 1, a globally consistent UV-layout of a specific material is necessary. Standard modeling tools typically provide robust mesh parameterization techniques, but fail to handle these type of meshes when they are composed of multiple components (see Fig. 18).



Artists in the game and entertainment industry usually are skilled to create nice looking models by composing and reshaping multiple mesh primitives, which are already similar to the part of the mesh they are forming, but do not consider the concept of watertight 2-manifold meshes. Such a workflow also results in meshes, which must be considered triangle soups, as they often are non-continuous and non-manifold.

Our technique avoids the hard problem of creating a parameterization for such meshes from scratch by providing a robust method to transfer arbitrary surface properties, e.g., a parameterization, from a manifold mesh to a triangle soup. We generate a 2-manifold mesh proxy from the input, which can be parameterized using one of the many existing algorithms for high-quality parameterization of 2-manifold meshes and use point correspondences to transfer the resulting parameterization onto the input triangle soup.

4 Overview

Our process consists of two pipelines which can be executed in parallel (see Fig. 3). The first pipeline (top row) generates a watertight 2-manifold offset mesh proxy from the input triangle soup and then computes a parameterization for it. The second pipeline (bottom row) computes an electrostatic field of the input triangle soup and traces its field lines. Afterwards, the results of both pipelines are used to assign UV-coordinates to the triangle soup based on the point correspondences found by intersecting the field lines emerging from the vertices of the triangle soup with the triangles of the mesh proxy.

The paper is organized as follows: In section 5 we start by showing how we generate offset mesh proxies for triangle soups and in section 6 we shortly discuss seam generation and texturing for the proxy meshes using existing algorithms for 2-manifold meshes. In section 7, we describe the efficient cal-

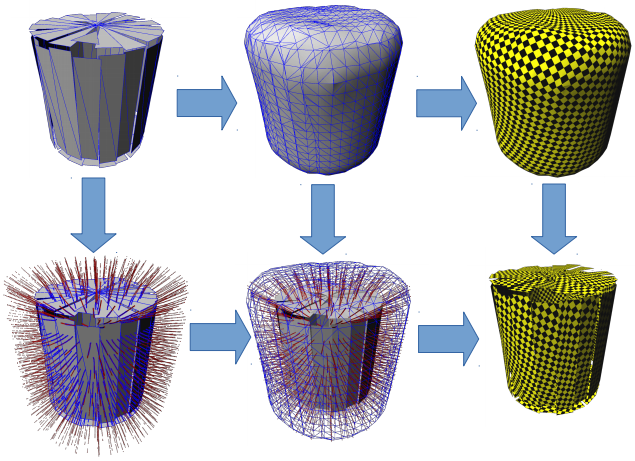


Fig. 3: In our parameterization process, we create an offset mesh proxy for the triangle soup which can be parameterized using conventional techniques as shown in the top row. To find the point correspondences between the triangle soup and the offset mesh proxy, we calculate the electrostatic field and then trace electrostatic field lines until they intersect the offset mesh proxy. Then we interpolate the UV-coordinates at the intersection points and use them to parameterize the triangle soup (bottom row).

culation of the electrostatic monopole field and how we trace its field lines in order to establish point correspondences on the offset mesh proxy, what is the main result of the paper. A post-processing step to reduce artifacts at the texture seams is presented in section 8. We evaluate our results in section 9 by presenting extensive benchmarks of the involved algorithms and a comparison with standard algorithms. In section 10, we show results for a multitude of meshes both from shape benchmark sets and real-world models and compare our results with automatic parameterizations using the Autodesk Maya® 3D editor in section 11.

5 Generating an Offset Mesh Proxy

A crucial step for our technique is computing a watertight offset mesh, which approximates the shape of the triangle soup. For our experiments, a distance-based approach has proven to be sufficient. However, depending on the triangle soup, other more sophisticated algorithms might be used as well, e.g., to reduce the number of triangles while preserving as much detail as possible.

In our approach, we define the isosurface of the distance-based mesh proxy as $S = \{\vec{x} \in \mathbb{R}^3 | d(\vec{x}) = o\}$, where $\mathrm{d}(\vec{x})$ is the distance of a point $\vec{x}$ to the nearest point on the triangle soup and o is the offset distance. The choice of the offset distance parameter $o \in \mathbb{R}^+$ is a trade-off between the accuracy of the shape approximation and the needed mesh resolution. An additional objective is to avoid capturing unwanted surface noise, which can be realized by choosing a larger offset distance or smoothing of the generated offset mesh proxy.

To triangulate the offset surface S, we use the marching cubes algorithm [35]. By calculating the distance using a bounding volume hierarchy (BVH) [7] of the primitives, the complexity of one evaluation is in $\mathcal{O}(\log n)$ and computing the BVH has complexity $\mathcal{O}(n \log n)$. Afterwards, the resulting manifold mesh is decomposed into its connected components, which are usually one outer component and one or more inner components, that may vanish depending on the offset parameter. A triangle soup or mesh with multiple components might have multiple outer components as well, if there are gaps which are larger than $2o$.

The inner and outer components of the mesh are identified using BVHs of the mesh components to test each vertex of a component if it is inside any other mesh component. We discard the inner components to map the vertices only to the outer parts of the mesh proxy. The number of intersection tests needed can be reduced by first testing if the *bounding boxes* of the mesh components are overlapping. For non-intersecting meshes, there is a good chance that the bounding boxes do not intersect either and thus we can skip the mesh intersection tests. If the bounding boxes are intersecting, we can stop when the first *mesh* intersection is found, what often only requires a few intersection tests.

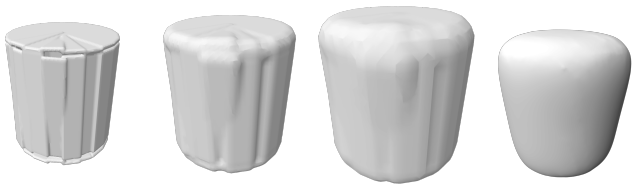


Fig. 4: We generated different mesh proxies for the triangle soup in Fig. 3. The first one fits the shape too closely to form a single mesh and cannot be used for a common parameterization of the triangle soup. The other mesh proxies have a larger offset parameter, what results in watertight meshes, which enclose the triangle soup. The rightmost mesh proxy was smoothed using Laplacian smoothing.

Fig. 4 shows different mesh proxies for the triangle soup in Fig. 3. The first proxy tightly fits the triangle soup but still contains some holes. Larger offset distances lead to smoother shape approximations, which are easier to parameterize. Surface noise, like in the second mesh in figure 4, can be reduced by Laplacian smoothing [18]. Smoothing artifacts, such as shrinking, can be compensated by choosing a larger offset parameter o and do not impose a problem on our technique. For example in Fig. 3 we have chosen the largest offset mesh proxy for better illustration. Typically we would choose the tightest feasible mesh proxy, which captures the geometry of the underlying manifold without reflecting the mesh artifacts of the triangle soup, to get a more accurate texture mapping.

6 Texturing the Offset Mesh Proxy

Once we have generated a suitable offset mesh proxy, we can texture it using one of the many available algorithms for texturing smooth watertight meshes, e.g., [31, 25, 24, 32, 15, 48,

34, 21, 20, 43, 46, 42]. For our examples we used [15] for parameterization together with the segmentation algorithm in [32] to generate the seams needed by the algorithm, but any algorithm for parameterization of 2-manifold meshes can be used.

For complex meshes, automatic seam generation may produce cuts that lead to visible seams on the surface. We allow the user to adjust such cuts by an additional semi-automated solution for interactive seam generation. The user selects a set of vertices, which are used to generate a path along the proxy surface using Dijkstra's algorithm [17]. To simplify finding paths along edges with high mesh curvature[1], we use a cost function

$$\begin{aligned}\text{cost}([\vec{x},\vec{y}]) := \|\vec{x}-\vec{y}\| + \alpha\left(1 - \frac{\Delta\vec{x}+\Delta\vec{y}}{2\max_z(\Delta\vec{z})}\right) \\ + \beta\|\Delta\vec{x}-\Delta\vec{y}\|\end{aligned}$$

which does not only depend on the total edge length but also on the mesh curvature and the change of the curvature along the path. α and β are weight parameters for the mean curvature and the change of curvature respectively. To calculate the mesh Laplacian, we use the discrete cotangent Laplacian [41] of the mesh vertices. When searching paths between a sequence of user-defined points, we assign edges used by previous segments a weight of ∞ to avoid self-intersections in the path. In a single segment, Dijkstra's algorithm prevents self-intersections by design.

7 Texture Mapping by Tracing Electrostatic Field Lines

For texture mapping between the triangle soup and an offset mesh proxy, we create a seamless mapping between the triangle soup and the offset mesh proxy by tracing electrostatic field lines from the vertices of the triangle soup to the offset mesh. For this purpose, we will consider the triangle soup as a virtually charged conductor with a charge distribution resulting in a constant electrostatic surface potential. We will first have a look at the physical model and its advantages for texture mapping and then discuss the differences between the dipole fields used in [16] and our monopole field approach.

Afterwards, we turn to the discretization of the physical model, define the relevant equations for triangle soups, and show how to reduce the complexity of field line tracing and the needed preparation step to linear time by using algorithms initially developed for large-scale physical simulations.

7.1 Physical Background

The governing equation for electrostatic fields is Coulomb's law, which describes the force between two point charges q_1 and q_2 with distance r as

$$F = k_e\frac{q_1 q_2}{r^2}$$

[1]For general meshes, seams are the least visible at edges of negative curvature. For the technical objects we focus on, edges with high positive curvature are good candidates for seams as well.

By integration over a charged surface Γ, we get the potential field ϕ and the electrostatic field $\vec{E}$

$$\phi(\vec{x}) = \quad k_e\int_\Gamma \sigma(\vec{y})\frac{1}{\|\vec{x}-\vec{y}\|}\mathrm{dy} \tag{1}$$

$$\vec{E}(\vec{x}) = -k_e\int_\Gamma \sigma(\vec{y})\frac{\vec{x}-\vec{y}}{\|\vec{x}-\vec{y}\|^3}\mathrm{dy} \tag{2}$$

where $\sigma(\vec{y})$ is the surface charge density at a point $\vec{y}$ on the surface and k_e is Coulomb's constant[2].

Note that the electrostatic field $\vec{E} = -\nabla\phi$ is the negative gradient of the potential field and ϕ is a harmonic function, what has several useful implications. A harmonic function has by the minimum principle no real local minima, so tracing its gradient cannot get stuck. As field lines follow the electrostatic field, they cannot cross each other because this would require two different field vectors at the intersection point. Furthermore, the gradient $\nabla\phi$ is orthogonal to the isosurfaces of the potential function ϕ, what implies that two field lines cannot merge into one as this would require a tangential component. For these reasons, we can ensure an *injective* mapping between the triangle soup and any enclosing surface.

We will drop k_e for the rest of the paper as it is only a multiplicative constant and we are not interested in the physical value of the forces.

7.2 Field Lines for Texture Mapping

For our method, we assume, that a conductor connects all triangles of the triangle soup, even when there is no mesh connectivity between them. When the physical model is applied, the charges rearrange instantly to minimize potential differences and because all charges are located at the triangles, the triangle surfaces must be the global maximum of the potential function.

Such a model allows for a parameterization of the whole space by *electric coordinates* [54], which identify each point uniquely by its potential and the field line passing through it.

Because the monopole field lines can be traced outside of the triangle soup towards infinity, we are guaranteed to find point correspondences on *any* enclosing mesh. Thus for every vertex of the triangle soup, we find a correspondence on the enclosing offset mesh proxy which can be used to map UV-coordinates from the offset mesh triangle to the triangle soup vertex. By the property that field lines do not cross each other, the map will not introduce flipped triangles.

7.3 Dipole versus Monopole Fields

Degener and Klein [16] use a dipole field with two opposite charges to induce field lines between a mesh and a close offset mesh proxy. Their approach uses a two-sided mesh proxy as the charged mesh proxy *defines* the direction of the dipole field lines emerging from the triangle soup. Thus both sides must be considered as possible candidates for texture mapping.

When using an electrostatic *monopole* field, where only the triangle soup is charged, the field is independent of the offset mesh proxy. As a consequence, not only the number of

[2]$k_e = 8.988 \cdot 10^9 Nm^2C^{-2}$

charge points needed is reduced and thus the computational time, but it also allows for optimizing offset meshes to better approximate the triangle soup without considering how the offset mesh will affect the field lines. As there is no attracting force guiding the field lines in the direction of the offset mesh, we need to follow the physical model closely to get meaningful field lines. Thus we have to calculate a surface charge density that induces a constant non-zero potential on the triangle surfaces which follows from the principle of minimum energy. Note, that only a constant potential on the surface guarantees that there are no local minima in the electric field and field lines can be traced from the surface towards points at infinity without being attracted to local extrema. See Fig. 2 for examples of point correspondences using a dipole field (middle) and a monopole field (right).

7.4 Potential Calculation on Triangles

In a triangle soup, the surface is defined by the union $\Gamma := \bigcup_{i=1}^{N} T_i$ of its triangles. We approximate the surface charge by piecewise constant charges $q_i = \int_{T_i} \sigma(\vec{x})\mathrm{dx}$ on the triangles and measure the potential at the triangle barycenters $\vec{c}_i$.

The potential field and the electrostatic field integrals from equation 1 and 2 are approximated (omitting k_e) by

$$\phi_i := \sum_{j=1}^{N} q_j I_{ij} \approx \quad \phi(\vec{c}_i) \tag{3}$$

$$-\vec{E}_i := \sum_{j=1}^{N} q_j \vec{J}_{ij} \approx -\vec{E}(\vec{c}_i) \tag{4}$$

using a m-point quadrature rule for the numeric integrals with Gauss quadrature points $\vec{y}_{jk}$ and the corresponding weights w_k on the triangle T_j:

$$I_{ij} := \sum_{k=1}^{m} w_k \frac{1}{\|\vec{c}_i - \vec{y}_{jk}\|}, \quad \vec{J}_{ij} := \sum_{k=1}^{m} w_k \frac{\vec{c}_i - \vec{y}_{jk}}{\|\vec{c}_i - \vec{y}_{jk}\|^3}$$

We are restricted to use quadrature rules which do not use the center as quadrature point to avoid division by zero[3].

7.5 Fast Charge Calculation

We need to calculate a charge distribution, which yields a constant non-zero potential at the surface, to induce the field lines, that will later be used to establish the point correspondences.

The linear system to solve for triangle charges which induce a prescribed potential field ϕ on the centers is:

$$\begin{pmatrix} I_{11} & \dots & I_{1n} \\ \vdots & \ddots & \vdots \\ I_{n1} & \dots & I_{nn} \end{pmatrix} \vec{q} = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_n \end{pmatrix} \tag{5}$$

[3]From the physical point of view, the quadrature points are equivalent to point charges at which the potential and electrostatic fields have a singularity making it impossible to measure the value there.

In [54], the authors solve the system directly, which is not feasible for larger meshes since the system matrix is dense and thus a solver needs $\mathcal{O}(n^3)$ operations in order to solve the linear system and has a memory requirement of $\mathcal{O}(n^2)$.

Instead of solving for the exact potential, an approximation is sufficient in our use case. So we use the *robin hood method for non-local charge transfer* [30] to calculate an approximate charge distribution, which minimizes the deviation from a constant surface potential.

The algorithm initializes the potential field from an arbitrary non-zero charge field and then divides the charge transfer process in discrete steps which iteratively balance the potentials until a steady state is reached. In each iteration, the two triangles that have the largest and the smallest potential exchange the charge that is necessary to reach the same potential at both triangles. Because their charge also influences the potential of other triangles, the error of neighboring triangles decreases as well and thus we get exponential convergence. As the charge transfer is calculated analytically, the total charge of the triangle soup is conserved exactly even when coarse triangulations are used.

In the initialization step of the algorithm, the surface potential induced by the initial charge distribution is calculated. In general, the complexity is $\mathcal{O}(n^2)$, as each triangle charge influences the potential of all other triangles. We accelerate the computation by using an approximate potential calculated using the fast multipole method (FMM) [22], which reduces the complexity to $\mathcal{O}(n)$.

During the iteration, the potentials of all triangles need to be recalculated after each charge transfer. Instead of using the fast multipole method in each step, the potentials are only updated by subtracting the influence of the old charge and adding the influence of the new one. The amount of charge to transfer from the triangle T_m with the maximum potential to the triangle T_n with the minimum potential is given by

$$\delta q := \frac{\phi_m - \phi_n}{I_{mm} + I_{nn} - I_{mn} - I_{nm}}. \tag{6}$$

After the charge transfer the potential ϕ_i of each triangle T_i is updated by

$$\begin{aligned} \phi_i^{(new)} &:= \phi_i^{(old)} + \left(q_n^{(new)} - q_n^{(old)}\right) I_{in} \\ &\quad + \left(q_m^{(new)} - q_m^{(old)}\right) I_{im} \end{aligned} \tag{7}$$

$$= \phi_i^{(old)} + \delta q I_{in} - \delta q I_{im} \tag{8}$$

As the update step only needs constant time, the complexity of a single iteration step lies in $\mathcal{O}(n)$ and the involved constants are small. By recalculating the integrals I_{ij} each time instead of storing them, the memory requirement lies in $\mathcal{O}(n)$. Furthermore, both steps of the algorithm can easily be parallelized.

For the algorithm to converge, the mesh triangles should have similar areas. If the mesh contains triangles, which have a significantly larger area than the other triangles, we use a local refinement before calculating the charge points. We only trace the vertices of the original mesh, so the number of field

lines is not increased. The computed charges q_i can now be used to define the approximated electrostatic field in $\mathbb{R}^3$ as

$$\vec{E}(\vec{x}) := -\sum_{j=1}^{N} q_j \sum_{k=1}^{m} w_k \frac{\vec{x} - \vec{y}_{jk}}{\|\vec{x} - \vec{y}_{jk}\|^3}. \tag{9}$$

7.6 Field Line Tracing

Using equation 9, we can trace particles from the triangle soup vertices along the electrostatic field to find their field lines. Because the field has a discontinuity at the triangle surfaces, we place the particles at a point $\vec{x}_{(0)}$ with a small offset into the outer normal direction, at which the field is well-defined. To trace the field lines, we use an explicit Euler scheme with a timestep δ_t:

$$\vec{x}_{(i+1)} := \vec{x}_{(i)} + \delta_t \frac{\vec{E}\left(\vec{x}_{(i)}\right)}{\left\|\vec{E}\left(\vec{x}_{(i)}\right)\right\|} \tag{10}$$

We normalize the field vector $\vec{E}$ to get an evenly spaced step size independent from the distance to the charge points. The timestep size δ_t is chosen, such that the field lines hit the offset mesh proxy after a given number of steps. For common offset mesh proxies, $50 - 100$ steps resulted in useful field lines. When the proxy is chosen to be far from the object, the number of steps should be increased to achieve the needed accuracy.

Point correspondences between the triangle soup and the offset mesh proxy can now be established by finding the intersections of the field lines with the offset mesh triangles. When an intersection is found, we store the index of the intersected offset mesh triangle together with the barycentric coordinates of the intersection point on the triangle.

Because the field is harmonic and thus has no real local minima, all particles which are not inside closed objects will eventually hit the offset mesh and thus establish a point correspondence.

Stability of the Euler Scheme When using the explicit Euler scheme, the timestep δ_t must satisfy the CFL condition [14] to avoid numeric artifacts. To increase the stability of the integration scheme, we implemented the second-order midpoint method that is also known as the modified Euler method. This technique is stable and allows for an adaptive step size. We compare the results of both methods in section 9.5.

Time complexity As calculating $\vec{E}$ for a single particle using equation 9 lies in $\mathcal{O}(n)$, we would have a complexity of $\mathcal{O}(k \cdot n^2)$ when tracing all field lines for k steps. To reduce the complexity we use a fast multipole approximation for equation 9 and achieve an overall time complexity of $\mathcal{O}(k \cdot n)$. Because the particles do not influence each other and the point charges do not change over time, the field line tracing can easily be parallelized, and the fast multipole data structures only need to be calculated once.

After tracing the field lines, we use the stored indices and barycentric coordinates to interpolate the texture coordinates at the intersection points and assign them to the corresponding vertices of the triangle soup.

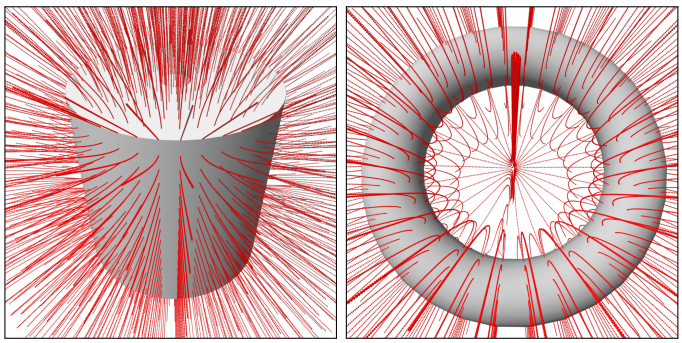


Fig. 5: Left: The electrostatic field lines of a cylinder have little curvature. Right: The field lines of a torus have high curvature and a singularity at the center, which make it necessary to use a tight offset mesh proxy to avoid skewing the resulting texture.

8 UV-Correction at Chart Boundaries

The triangle intersection points are determined by the triangle index and the barycentric coordinates. This information allows for a unique mapping between the original shape and the proxy surface. However, to compute a texture mapping, we need to take the genus zero charts of the offset mesh proxy into account. We already optimized the seams on the offset mesh proxy to be less visible in section 6, but triangles with vertices mapped to different charts of the offset mesh still introduce texture artifacts as the UV-coordinates of neighboring triangles may lie far apart.

Fig. 6 shows on the left side a parameterization of an offset mesh for a cube, which is divided into six charts each of them having genus zero. When the vertices of the cube are projected onto these charts, triangles near the seams have their vertices mapped onto different charts of the offset mesh proxy as seen in the middle of Fig. 6.

As such situations only occur for triangles near a UV-chart boundary, it is often sufficient to move the affected vertices of a triangle a small distance to make all vertices lie on the same chart. The resulting triangle texture is slightly stretched, but for most parameterizations the effect is negligible, and the error gets smaller when the offset mesh proxy is refined.

When the vertices of a triangle are mapped to different charts of the offset mesh proxy, we calculate for each vertex the closest point on the two other charts and choose the chart which minimizes the difference between the area of the transformed triangle and the area of the original triangle. The resulting positions of the triangle in the UV-space are shown on the right side of Fig. 6. A direct comparison of the visible artifacts in the texture of a cube with a coarse triangulation is shown in Fig. 7.

9 Evaluation & Discussion

The main advantages of using monopole fields instead of dipole fields are *performance* and *flexibility*.

First, we neither need to compute the potential of the offset mesh triangles nor consider them when tracing the field

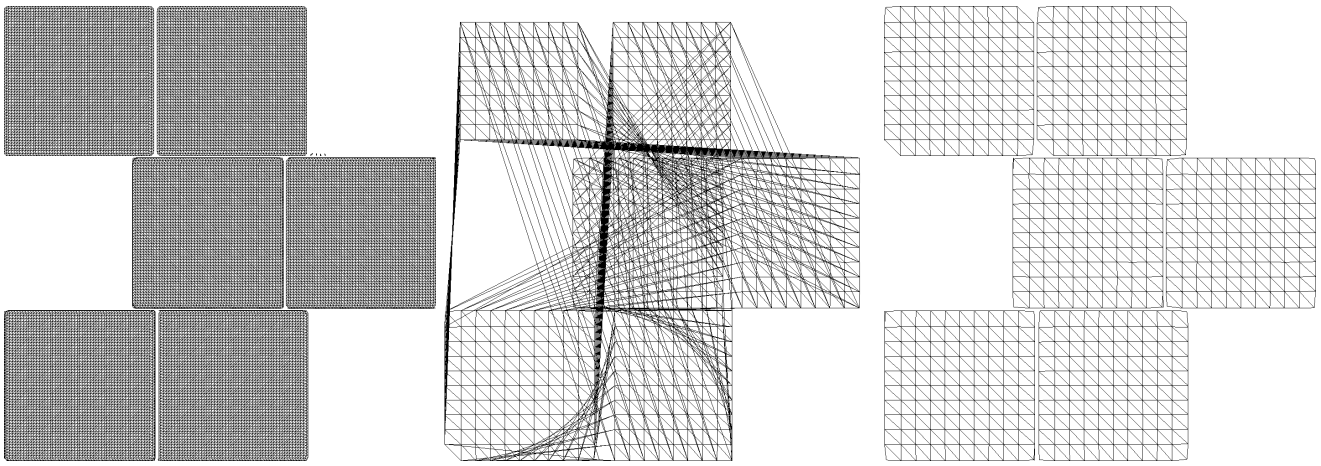


Fig. 6: Left: Simple tri-planar UV-coordinates for an offset mesh proxy of a cube. Middle: The UV-coordinates of the cube after mapping the vertices onto the mesh proxy. Right: The UV-coordinates of the cube after we applied our UV-fixing procedure.

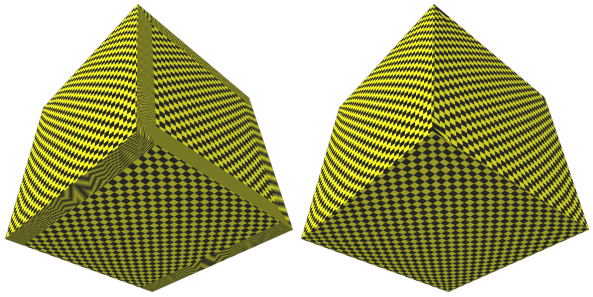


Fig. 7: Left: The triangle mapping causes texture artifacts near texture seams on the offset mesh proxy. Right: Our UV-correction method is able to remove the distortion by allowing a small deformation of the mapped triangle.

lines of the triangle soup. As a consequence, the computational burden is significantly reduced. Therefore, we are now able to use detailed shape approximations without increasing the time needed for field line tracing. Second, in contrast to Degener and Klein, the field is independent of the offset mesh proxy, what allows us to interchange different offset mesh proxies without costly recalculation of the field. Third, the decrease in computational complexity of tracing the field lines to $\mathcal{O}(kn)$ allows us to process larger triangle soups, which were infeasible to compute using the approach of Degener and Klein.

In this section, we compare the run-time of our method against previous approaches. All experiments were performed on a Intel Core i7-3740QM CPU with a 2.7 GHz quad core processor (3.70 GHz turbo frequency on a single core) and 32 GB RAM.

9.1 Offset Mesh Proxies

For texturing, an offset mesh proxy which is a good approximation of the original shape is crucial. However, for smaller offset distances, which capture more details of the shape, we need to increase the number of samples when extracting the surface using the marching cubes algorithm. As a direct consequence, smaller offset distances lead to mesh proxies with a higher triangle count. In Table 1 we list the relation between offset distance, required resolution (determined by the sampling theorem [47]), and the number of triangles and vertices generated for a simple cylinder of radius 1.0 and height 2.0, to emphasize how fast the number of triangles and vertices is growing when the offset distance is getting smaller.

Table 1: The relation between the offset distance from the cylinder and the needed resolution for the marching cubes algorithm according to the sampling theorem.

Offset Distance [%]	Needed Samples	Vertices	Triangles
10	13	576	1148
5	40	6894	13784
3	67	21376	42748
2	100	50086	100168
1	200	210814	421624

9.2 Monopole vs. Dipole Potential Calculation

To compare the computational time needed by the potential calculation, we used a torus with 5000 and a torus with 2450 triangles, each equipped with two different offset mesh proxies with a distance of 5% of the torus diameter, triangulated using 40 and 80 samples.

In Table 2 and Fig. 8, we show the time needed to calculate the potentials induced from a given charge field by using a direct calculation using equation 3 and by using fast multipole methods (FMM) implemented with the *exafmm* library [60]. For the multipole acceptance criterion, we used values of $\theta = 0.1$ and $\theta = 0.4$, where a larger value of θ results in a coarser approximation of the electrostatic field. To approximate the

integrals, we used 6th-degree Gauss quadrature, leading to $12|F|$ charge points, where $|F|$ is the number of triangles in the triangle soup. Both methods were parallelized to use all cores of the CPU.

Table 2: Time needed to calculate the surface potentials.

	Total Triangles	Full Calculation	FMM ($\theta = 0.1$)	FMM ($\theta = 0.4$)
Monopole	5000	1.352	0.721	0.178
Dipole (40)	20312	24.476	6.192	0.855
Dipole (80)	65256	324.395	23.259	2.771
Monopole	2450	0.302	0.168	0.074
Dipole (40)	17754	17.519	4.838	0.743
Dipole (80)	62590	283.918	18.984	2.944

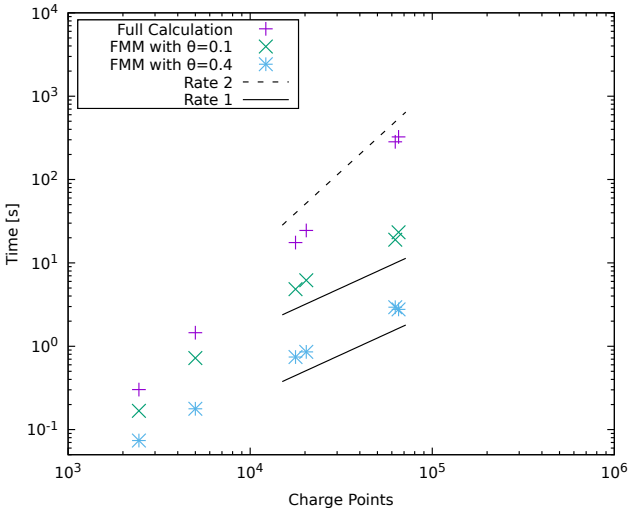


Fig. 8: The time needed to calculate the surface potentials using the simple $\mathcal{O}(n^2)$ approach and the time needed when using a fast multipole methods approximation.

As the number of charges for a *monopole* field only depends on the number of triangles of the triangle soup, the calculation is much faster than the calculation for *dipole* fields, because dipole algorithms need to calculate the potential of the offset mesh triangles as well. In our example, the total number of triangle potentials which need to be calculated for a dipole field is by a factor 4 to 24 larger than the number of potentials needed for a monopole field. For both monopole and dipole calculations, the full potential calculation has quadratic complexity while the fast multipole approximation has linear complexity as depicted in Fig. 8.

9.3 Robin Hood Iteration

The calculation of the charge field is done using our implementation of the robin hood iteration (see section 7.5) with a parallelized potential update step from equation 8, which uses all cores of the CPU. We tested the iteration with different shape primitives and measured the time needed to converge to a maximum-norm error below 10^{-13}, which is the best accuracy we were able to achieve on all meshes.

The iteration converged in about 2 seconds on a torus with 5000 triangles, on a sphere with 4900 triangles, and on a cylinder with 5900 triangles, and we observed an exponential convergence rate. The time needed to achieve the desired accuracies is listed in Table 3 and the convergence rates are plotted in Fig. 9.

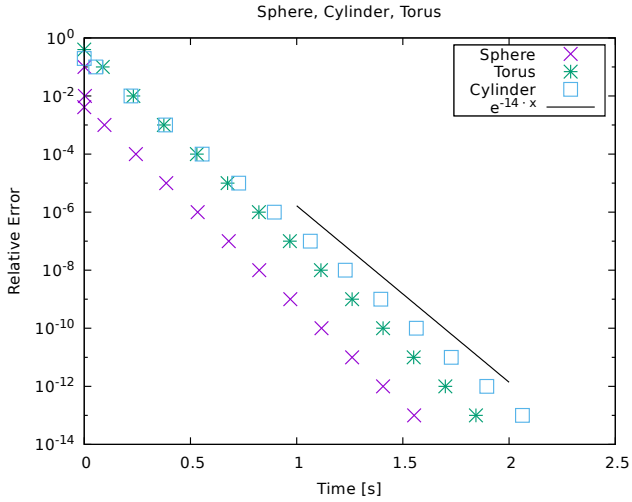


Fig. 9: Semi-logarithmic plot of the maximum norm error of the potential on smooth objects versus the time needed by the robin hood iteration.

For objects with sharp features, the iteration consumed much more time to converge, e.g., on a cube with 4900 triangles the error remained larger than 10^{-6} for about 350 seconds before it drops rapidly below 10^{-13} in the last few iterations. Fig. 10 shows rescaled plots of the convergence behavior and the time needed to achieve the desired accuracies is listed in Table 3.

To measure the speedup of approximating the charge field using the robin hood method, we compared the time needed for solving the full equation system in equation 5 with the time needed by the robin hood algorithm to converge to a potential error smaller than 10^{-13} in the maximum norm. We chose a torus (radius 1.0, section radius 0.5) as an example because its analytic surface and its charge field are continuous and it is easy to generate regular triangulations of different resolutions.

To solve the linear system, we computed a LU-decomposition of the matrix with complete pivoting using libeigen [23] on a single core[4] and for the robin hood algorithm we used our own implementation. We measured both the time needed to converge when using a direct calculation of the initial potentials and the time needed when using a fast multipole calculation of the initial potentials with $\theta = 0.1$. The robin hood update step, the full calculation of initial potentials, and the fast multipole calculations were parallelized to use all cores of the CPU.

We observed a linear rate in the robin hood iteration. When we compute the full potential during the initialization

[4]The libeigen solver is not parallelized, but the time needed should only differ by a constant factor of about the number of cores, so the convergence rate is not changed.

Table 3: The time needed by the robin hood iteration to converge on the test objects.

Sphere (4900 Triangles)			Torus (5000 Triangles)			Cylinder (5900 Triangles)			Cube (4900 Triangles)		
Error	Iterations	Time [s]	Error	Iterations	Time [s]	Error	Iterations	Time [s]	Error	Iterations	Time [s]
$4.13{\cdot}10^{-3}$	0	0.00	$3.95{\cdot}10^{-1}$	0	0.00	$1.96{\cdot}10^{-1}$	0	0.00	$5.95{\cdot}10^{-1}$	0	0.00
$1{\cdot}10^{-1}$	1	0.00	$1{\cdot}10^{-1}$	137	0.09	$1{\cdot}10^{-1}$	67	0.05	$1{\cdot}10^{-1}$	14957	9.98
$1{\cdot}10^{-2}$	2	0.00	$1{\cdot}10^{-2}$	366	0.23	$1{\cdot}10^{-2}$	297	0.22	$1{\cdot}10^{-2}$	71754	54.62
$1{\cdot}10^{-3}$	142	0.10	$1{\cdot}10^{-3}$	595	0.38	$1{\cdot}10^{-3}$	526	0.38	$1{\cdot}10^{-3}$	169494	137.66
$1{\cdot}10^{-4}$	371	0.24	$1{\cdot}10^{-4}$	825	0.53	$1{\cdot}10^{-4}$	755	0.56	$1{\cdot}10^{-4}$	291756	254.70
$1{\cdot}10^{-5}$	600	0.39	$1{\cdot}10^{-5}$	1054	0.68	$1{\cdot}10^{-5}$	984	0.73	$1{\cdot}10^{-5}$	375936	336.23
$1{\cdot}10^{-6}$	829	0.53	$1{\cdot}10^{-6}$	1283	0.82	$1{\cdot}10^{-6}$	1213	0.90	$1{\cdot}10^{-6}$	402376	363.98
$1{\cdot}10^{-7}$	1058	0.68	$1{\cdot}10^{-7}$	1512	0.97	$1{\cdot}10^{-7}$	1442	1.06	$1{\cdot}10^{-7}$	402390	364.00
$1{\cdot}10^{-8}$	1287	0.82	$1{\cdot}10^{-8}$	1741	1.11	$1{\cdot}10^{-8}$	1671	1.23	$1{\cdot}10^{-8}$	402620	364.24
$1{\cdot}10^{-9}$	1516	0.97	$1{\cdot}10^{-9}$	1970	1.26	$1{\cdot}10^{-9}$	1900	1.40	$1{\cdot}10^{-9}$	402849	364.45
$1{\cdot}10^{-10}$	1745	1.12	$1{\cdot}10^{-10}$	2199	1.41	$1{\cdot}10^{-10}$	2129	1.56	$1{\cdot}10^{-10}$	403078	364.68
$1{\cdot}10^{-11}$	1974	1.26	$1{\cdot}10^{-11}$	2428	1.55	$1{\cdot}10^{-11}$	2358	1.73	$1{\cdot}10^{-11}$	403307	365.03
$1{\cdot}10^{-12}$	2203	1.41	$1{\cdot}10^{-12}$	2657	1.70	$1{\cdot}10^{-12}$	2587	1.90	$1{\cdot}10^{-12}$	403535	365.37
$1{\cdot}10^{-13}$	2431	1.55	$1{\cdot}10^{-13}$	2883	1.84	$1{\cdot}10^{-13}$	2814	2.06	$1{\cdot}10^{-13}$	403767	365.73

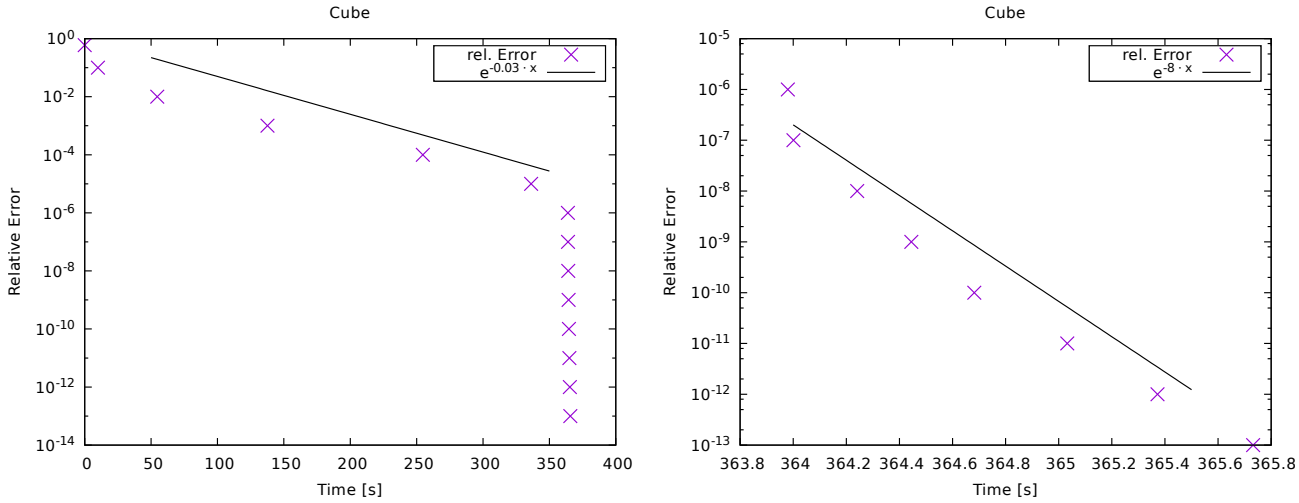


Fig. 10: Left: Semi-logarithmic plot of the convergence of the robin hood iteration on a cube with 4900 triangles. Right: The convergence plot rescaled to show the increased rate after 364 seconds.

step, the quadratic complexity starts to dominate the overall rate at about 5000 triangles. When using the fast multipole method for the initialization step, the overall rate stays linear independent of the size of the input.

The measured times are listed in Table 4 and the corresponding rates are plotted in Fig. 11. We did not solve the full system for more than 12800 triangles, as the computation time increases drastically. The libeigen solver showed a stable rate of $\mathcal{O}(n^3)$ for solving the system which matches the theoretic complexity. We assume that the rate stays constant for larger systems as well.

9.4 Field Tracing

To measure the time needed for field line tracing we used a torus of radius 1.0 and section radius 0.5, traced its field lines 50 steps, and compared the full calculation of the field vectors using equation 9 against using the fast multipole method approximation of the field with the parameters $\theta = 0.4$ and $\theta = 0.1$. The charge field, which induces the field lines, was calculated using fast multipole methods and the robin hood iteration in both cases. Table 5 lists the time needed for tracing the field lines of the torus mesh triangulated with different resolutions. The corresponding rates are plotted in Fig. 12. As we used 6th-degree Gauss quadrature, the number of charge points is $12|F|$ where F is the number of triangles.

We observed that the approximation with fast multipole methods yields a rate of 1, while the full calculation has the expected rate of 2. We did not measure the time of the full calculation for the last two triangulations, as it would have been too time-consuming. We assume the rate to remain stable because the algorithm only consists of a simple particle update step using equations 9 and 10 without any side effects.

9.5 Numeric Integration

For tracing the field lines, we need to calculate two different integrals, i.e., the time integration when tracing a particle and the field integration to calculate the field vector at the position of the particle. For both integrals, we use numeric approximations.

Table 4: A comparison between the time needed for calculating the charge distribution by solving the linear system in equation 5 and the time needed by the robin hood algorithm for calculating the charge distribution. We observe a time complexity of $\mathcal{O}(n^3)$ for solving the linear system and a complexity of $\mathcal{O}(n^2)$ when using the robin hood algorithm with a full potential initialization. When we combine the robin hood algorithm with fast multipole methods, the complexity is reduced to $\mathcal{O}(n)$.

Triangles	RH [s]	RH (FMM) [s]	Full [s]
800	0.14	0.22	0.33
1600	0.37	0.40	2.80
3200	1.01	0.76	22.56
6400	6.21	4.33	236.68
12800	14.15	5.43	1678.71
25600	46.93	12.19	—
51200	197.25	19.79	—
105800	918.38	36.94	—

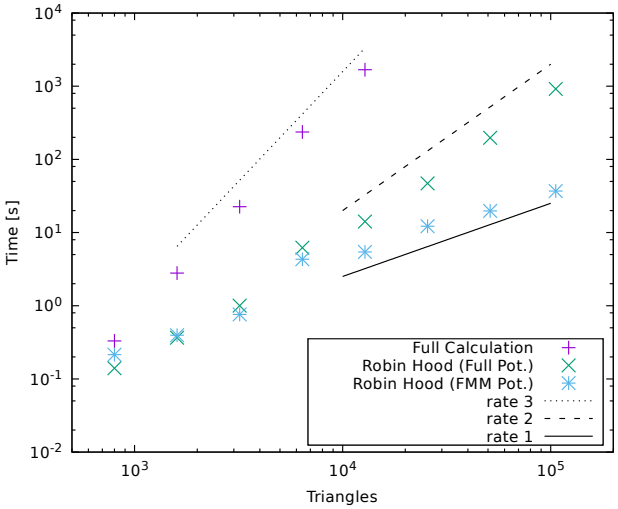


Fig. 11: The relation between the number of triangles and the time needed to converge shows a stable rate of 3 for solving the linear system, while the robin hood algorithm using a full initialization with equation 3 shows an asymptotic rate of 2 which is reduced to 1 when an initialization with fast multipole methods is used.

Time Integration For the time discretization, we use the explicit Euler scheme. In theory, this scheme might become unstable if the CFL condition [14] does not hold for the chosen timestep. We did not encounter such problems in practice when we chose the step size in a way that the field line needs about $50-100$ steps to reach the offset mesh. As an alternative, we implemented the midpoint method, which is stable and has a second-order convergence.

In Fig. 13 and Table 6 we show the convergence of the explicit Euler scheme and the midpoint method against the result of the midpoint method at the step size 10^{-4}, when tracing the 400 field lines of the torus in Fig. 5. An example of a field line traced one time using the explicit Euler scheme and the other time using the midpoint method is depicted in Fig. 14. Even when using a rather large timestep of $\delta_t = 0.1$, the final positions of the traced particles are close.

Field Integration We integrate the potential field using a Gauss integration over the triangles to get the field vectors at the positions of the traced particles.

The accuracy of this integration has a much more significant influence on the quality of the field lines than the time integration, especially for positions near the charged surface. Fig. 15 shows a pipe with 320 faces and 160 vertices as an example in which the field lines intersect solid faces when the integration method is not accurate enough. In our experience second-order Gauss quadrature is accurate enough for nearly convex objects, but for shapes with concavities or even tunnels, like the pipe in fig 15, we need to use higher order integration or finer triangulations to avoid numeric artifacts.

In section 7.4, we restricted the choice of the integration rule for the charge calculation to rules which do not include the barycenter of the triangle, to be able to calculate the potential there. While this avoids problems during the charge calculation, the singularities could cause problems in the field line tracing process as well. When necessary, special singularity treatment, e.g. [56] and [36], can be used in equations 3 and 4 to compensate for the singularities at the integration points.

10 Examples

We performed a multitude of experiments to compute parameterizations for complex objects including thin structures to showcase our technique. All experiments were performed using an own implementation as an Autodesk Maya® plugin. For the parameterization of the offset mesh proxies in our examples, we used the balanced isometric parameterization algorithm [15].

We created example triangle soups from models in the Princeton shape benchmark set [51] by duplicating all shared vertices and adding random noise to the vertex coordinates (see Table 7). Afterwards, we used the triangle soups as input for our algorithm and parameterized them. The results are shown in Fig. 16. To showcase our technique on objects having a higher genus, we used a *botijo* model with 13859 vertices and 29734 faces and genus 5 from the COSEG data set [55] and split it into a triangle soup with 89202 vertices and 29734 triangles. We used a tight offset mesh proxy with a distance of 0.2% and a marching cubes resolution of 250 voxels in each dimension because the handles cause a curved electrostatic field. In Fig. 17 we show the botijo with its field lines, the textured offset mesh proxy, and finally the textured triangle soup. The average distortion at the vertices is listed in Table 7. We expect that most triangle soups, which should be used for rendering, contain less noise.

As examples for real-world models, we parameterized a gear selector with 14356 triangles and 43068 vertices and a seat

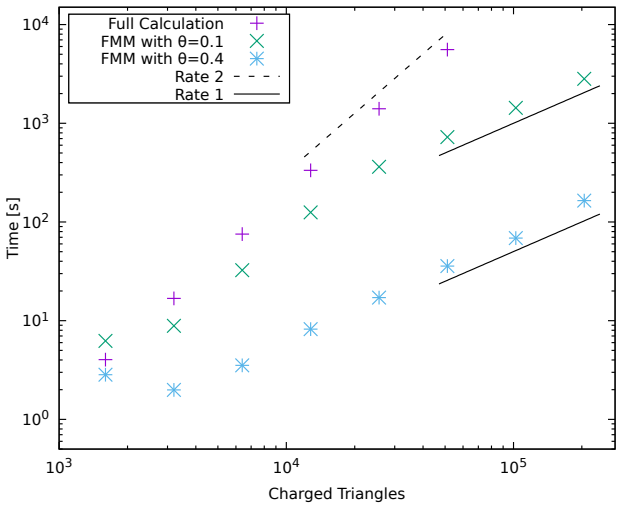


Fig. 12: The relation between the time needed for tracing the field lines of triangle soups with different numbers of triangles for 50 steps using the full calculation with equation 9 and using a fast multipole methods approximation. We observe the expected rate of 2 for the full calculation and asymptotic rates of 1 for the fast multipole method approximations.

belt buckle with 47661 triangles and 51875 vertices from a set of car parts which were provided by Volkswagen AG. The left side of Fig. 18 shows the final parameterization of the objects.

11 Comparison with Maya UV-Generators

In addition to the experiments from the previous section, we perform a comparison against UV-generators available in Autodesk Maya®. The available functions can generate planar, tri-planar, cylindrical, and spherical UV-coordinates for arbitrary triangle soups by mapping the triangles on standard parameterizations of these shapes.

We parameterized the gear selector and the seat belt buckle from our set of car parts using these functions, *manually* rescaled the generated components in UV-space to have the same texture resolution, and *manually* aligned them to have matching textures at the boundaries. The textured models are shown on the right in Fig. 18.

To texture the gear selector we choose cylindrical UV-coordinates for the handle and spherical UV-coordinates for the cap which provide useful approximations of the shapes. We see smaller artifacts and a singularity at the top, but the overall result for the gear selector is acceptable.

For the seat belt buckle, we used a tri-planar UV-mapping for each connected component on its own, as it seems to be the best built-in method for the mostly cuboidal shape. For the curved piece of the belt at the back, we used a tri-planar UV-mapping as well because none of the available generators can match the shape correctly. As it has an almost right angle in the middle, a tri-planar mapping yields a better result than expected, but we still see large artifacts inside the texture of

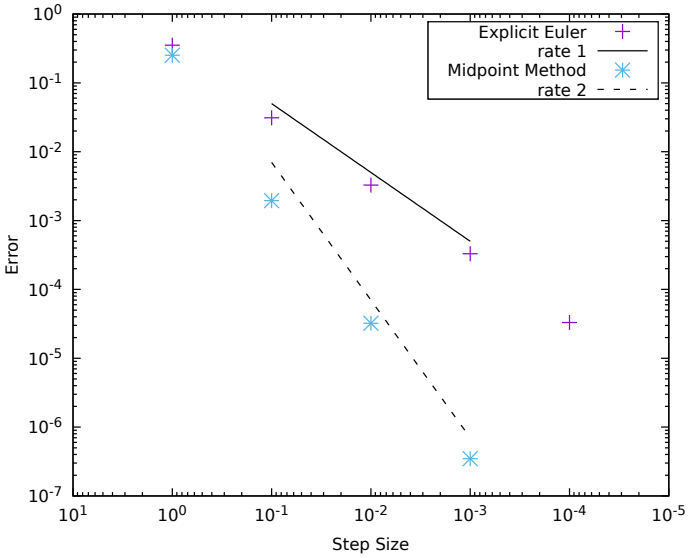


Fig. 13: Convergence of the field lines traced using the explicit Euler scheme and the midpoint method.

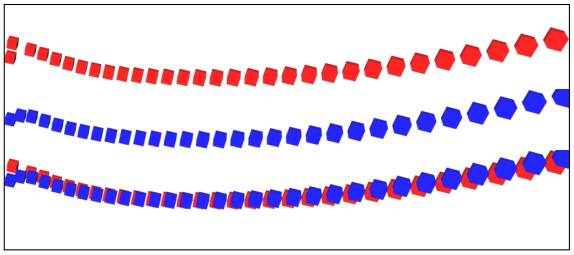


Fig. 14: The time integration using an explicit Euler scheme (top) and the midpoint method (middle) yield very similar results even with a large timestep of 0.1. At the bottom, both field lines (traced from left to right) are shown together. Note that the first position is not the same, as the particle is visualized after the first tracing step.

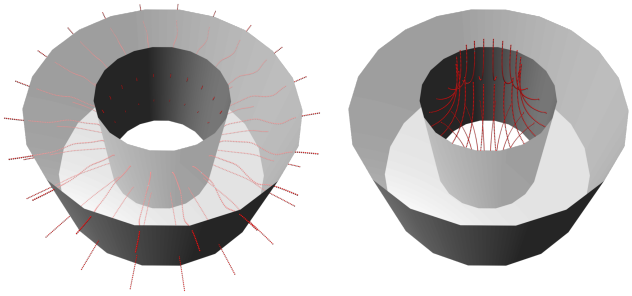


Fig. 15: The field lines of the inside of a pipe using a second-order Gauss integration (left) and using a fourth-order Gauss integration (right) for the potential field. While the second-order integration fails to prevent the field lines from intersecting solid faces. A fourth-order accurate integration method yields correct results.

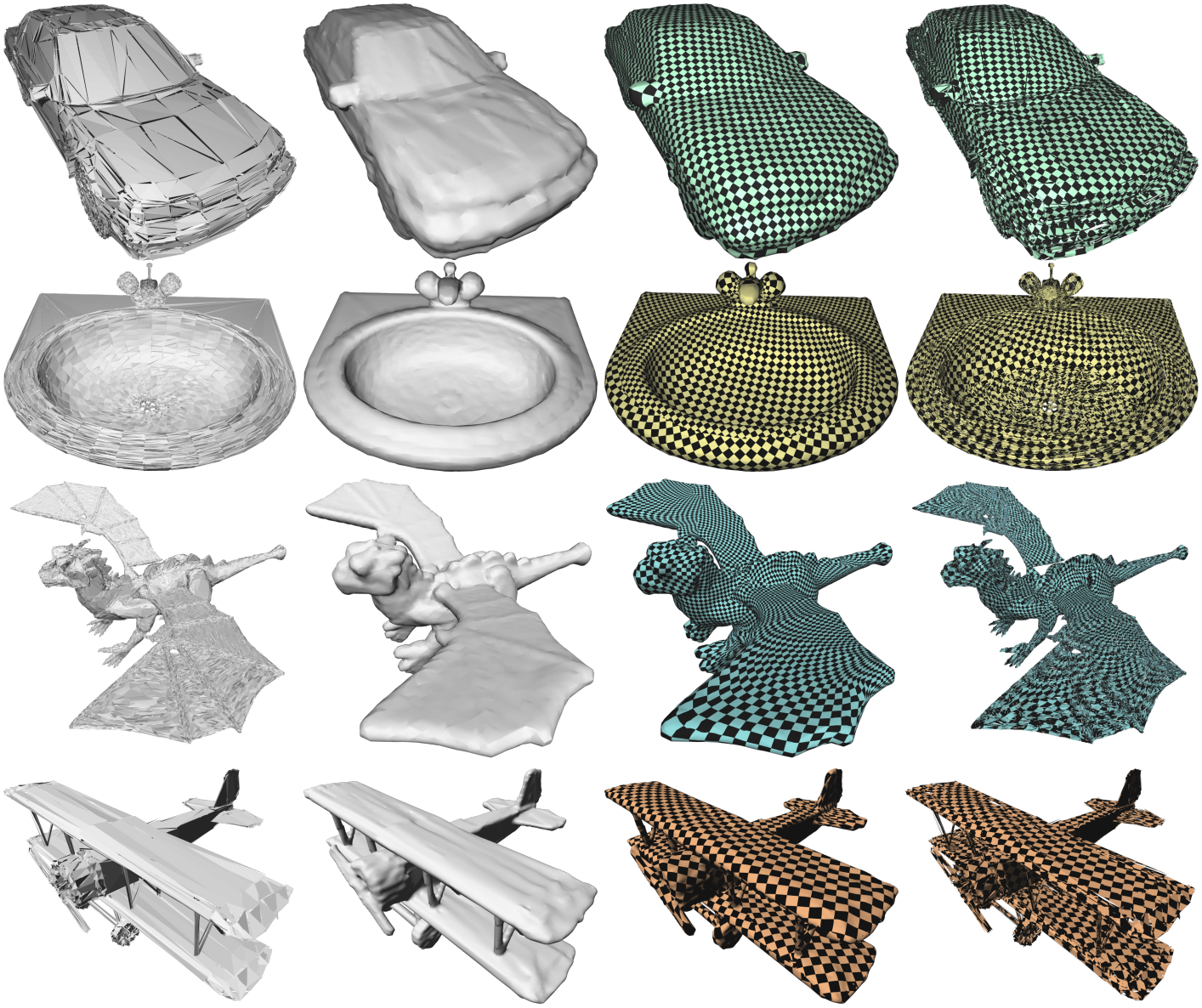


Fig. 16: We used examples from the Princeton shape benchmark models to generate triangle soups for testing our novel texture mapping method. The triangle soup (left) was generated by splitting the model into its triangles and adding random noise to the vertex positions. Then we created an offset mesh proxy and textured it using the parameterization algorithm in [15] (middle). The triangle soup with the projected texture is shown on the right side.

The input triangle soup | The triangle soup with field lines | The textured (smooth) offset mesh proxy | The triangle soup after texture mapping

Fig. 17: As a challenging example of a model of higher genus, we chose a *botijo* model from the COSEG data set and created a triangle soup from it by splitting the triangles and adding random noise to the vertex positions. Then we calculated the electrostatic field lines of the triangle soup and used them to find point correspondences on a textured 2-manifold mesh proxy of the *botijo*.

Table 5: We compared the time needed for tracing field lines of triangle soups with a different number of triangles for 50 steps using equation 9 and the time needed when using a fast multipole method approximation. We observed a computational complexity of $\mathcal{O}(n^2)$ for the full calculation which is reduced to $\mathcal{O}(n)$ when a fast multipole method approximation is used.

Triangles	FMM ($\theta = 0.4$) [s]	FMM ($\theta = 0.1$) [s]	Full [s]
800	2.18	2.53	0.94
1600	2.84	6.24	4.03
3200	1.99	8.87	16.83
6400	3.53	32.44	75.37
12800	8.22	124.96	333.97
25600	17.13	361.96	1402.52
51200	35.71	724.67	5584.04
102400	68.69	1434.35	—
204800	164.71	2820.83	—

Table 6: The error of the field line integration using the explicit Euler scheme and the midpoint method, measured by comparing the final particles positions of the traced field lines, using the result for the midpoint method at the finest timestep as reference.

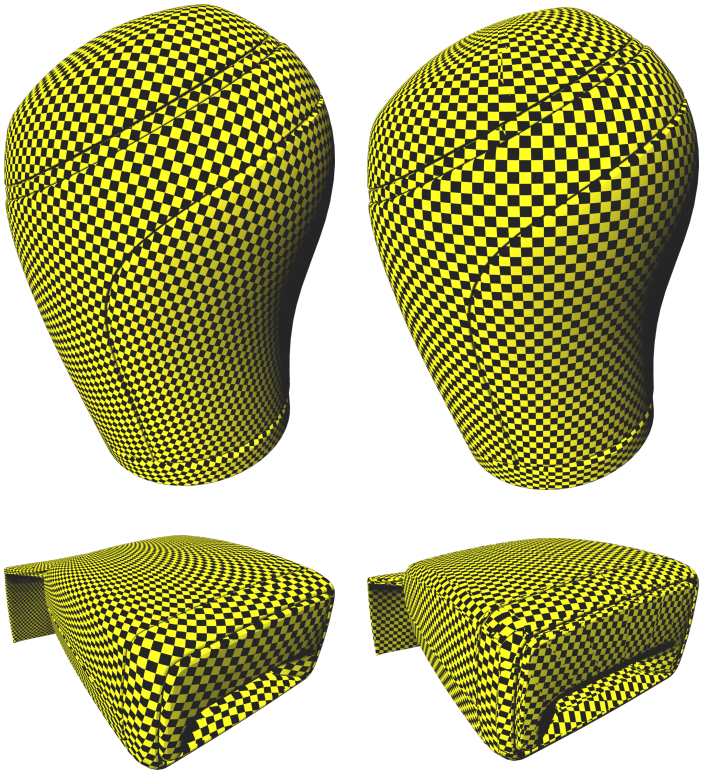
Step Size	Steps	Error (Explicit Euler)	Error (Midpoint Method)
$1{\cdot}10^{0}$	$1{\cdot}10^{0}$	$3.522{\cdot}10^{-1}$	$2.5148{\cdot}10^{-1}$
$1{\cdot}10^{-1}$	$1{\cdot}10^{1}$	$3.1137{\cdot}10^{-2}$	$1.9424{\cdot}10^{-3}$
$1{\cdot}10^{-2}$	$1{\cdot}10^{2}$	$3.2753{\cdot}10^{-3}$	$3.2174{\cdot}10^{-5}$
$1{\cdot}10^{-3}$	$1{\cdot}10^{3}$	$3.301{\cdot}10^{-4}$	$3.482{\cdot}10^{-7}$
$1{\cdot}10^{-4}$	$1{\cdot}10^{4}$	$3.3034{\cdot}10^{-5}$	—

each component of the model, which cannot be fixed without a substantial amount of manual work.

In comparison, the parameterization projected from an offset mesh proxy is smoother and contains fewer artifacts than the parameterization with Autodesk Maya®. In the case of the seat belt buckle, we get a continuous texture while the triplanar projection introduces many seams in the texture even at surfaces of low curvature. For more complex shapes, i.e., models of higher genus, there are no useful built-in methods for automatic parameterization at all.

12 Limitations & Future Work

The algorithm works best with triangle soups, which are an approximation of a 2-manifold watertight mesh, e.g., triangulated CAD models. Smaller artifacts and surface noise are handled correctly, but some edge cases require additional preprocessing.



Our Method Autodesk Maya®

Fig. 18: We created a parameterization for the gear selector in Fig. 1 and a seat belt buckle from a VW Tiguan model with our novel method (left). For comparison, we show UV-coordinates created by an expert using only the generators in Autodesk Maya® (right). While the gear selector can be parameterized using Autodesk Maya® with minor artifacts, the seat belt buckle shows severe artifacts which need manual correction.

Table 7: The distortion of the model given by the average distance of the perturbed vertices from the original positions. The values are rescaled with the diameter of the bounding box of the object.

	2-Norm	1-Norm	∞-Norm
Car	$4.52806 \cdot 10^{-3}$	$7.06919 \cdot 10^{-3}$	$3.53651 \cdot 10^{-3}$
Dragon	$1.49028 \cdot 10^{-3}$	$2.32749 \cdot 10^{-3}$	$1.16346 \cdot 10^{-3}$
Sink	$3.81522 \cdot 10^{-3}$	$5.95865 \cdot 10^{-3}$	$2.9774 \cdot 10^{-3}$
Plane	$2.46499 \cdot 10^{-3}$	$3.71417 \cdot 10^{-3}$	$2.05845 \cdot 10^{-3}$
Botijo	$9.62139 \cdot 10^{-4}$	$1.50191 \cdot 10^{-3}$	$7.51904 \cdot 10^{-4}$

12.1 Triangle Intersections

While our algorithm is robust against intersecting triangles in noisy meshes, it does not handle the case of triangle intersections, for which vertices are mapped to different parts of the offset mesh. Fig. 19 (top row) shows an example of two intersecting planes, each having a row of triangles which is cut in a way that the field lines of the vertices intersect different parts of the offset mesh proxy. As the field lines intersect the offset mesh far apart from each other, we observe severe texture artifacts.

To solve the problem, we refine the triangles at the intersection in a pre-processing step, such that for each triangle all vertices lie on the same side of the intersection and will be mapped to the same part of the offset mesh (see the bottom row of Fig. 19).

When refining the intersecting triangles, the number of field lines and thus the computation time is increased proportionally to the number of triangles, which are intersected in a way that requires refinement. For reducing the artifacts, only additional field lines need to be added, but we recommend using the new triangles for the charge calculation as well, because the potential field is also affected by such triangles.

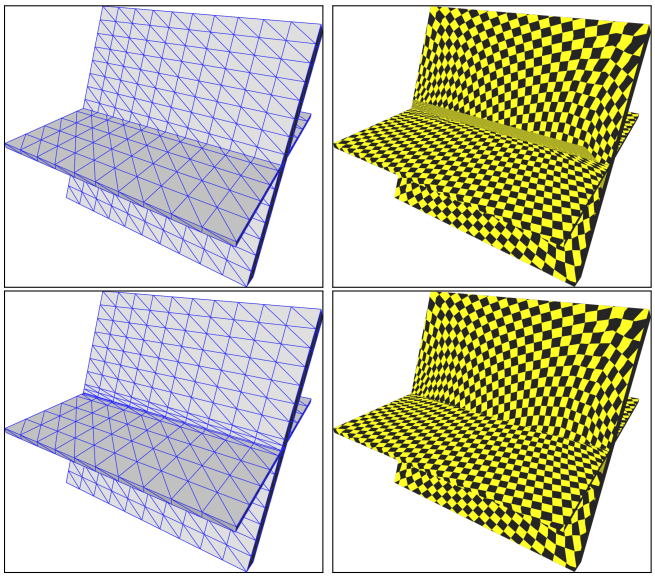


Fig. 19: Top: A plane is split by another plane. The vertices of the cut triangles are mapped onto different parts of the offset mesh proxy, which causes visible artifacts. Bottom: By refining the triangles at the intersection points, we can make sure that all vertices of the new triangles are mapped to the same part of the mesh proxy.

12.2 Narrow Surfaces

The second problem might occur at surfaces, which are close together, and thus induce a field with a strong curvature and possibly singularities. Fig. 21 shows an example of two plates forming a V-shape, which induces an electric field with large curvature even when the surfaces are flat. To avoid artifacts due to the curved field lines between the triangle soup and the

offset mesh (see Fig. 20 top row), we need to use a narrower offset mesh proxy, which intersects the field lines close to the original surface. The bottom row of Fig. 20 shows the result of using a tight offset mesh.

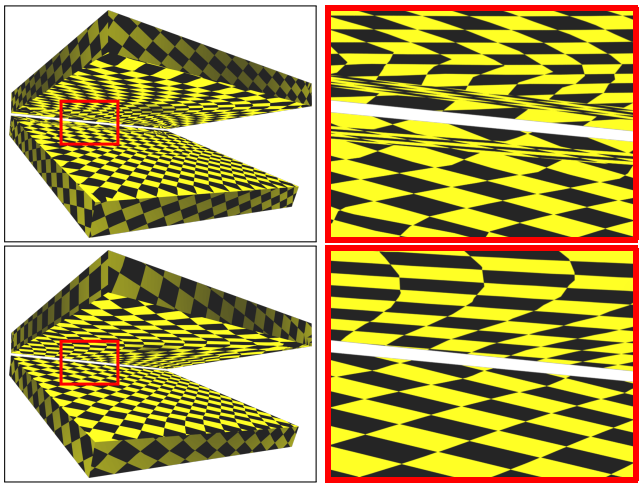


Fig. 20: Top: The offset mesh proxy is intersected by the field lines far from the original surface, what leads to artifacts in the texture because of the high curvature of the field lines. Bottom: Reducing the offset distance yields an improved result without artifacts.

To use a closer offset mesh, as needed to handle the narrow parts of the example, we neither need to increase the number of field lines, nor we need to recalculate the charge field. Thus replacing an offset mesh which causes artifacts by a more accurate one is rather cheap, because only the intersections with the field lines need to be recalculated.

Future work may address the increased number of offset mesh triangles by applying mesh simplification, or using a more sophisticated offset mesh proxy generation algorithm.

12.3 Non-Bijectivity

A dipole field as used by Degener et al. [16] guarantees *bijectivity*, because of the inherent symmetry of the approach. As each point on both surfaces is intersected by a unique field line, which only intersects exactly *one source* and *one target mesh point*, and field lines cannot cross each other, the mapping must be bijective and continuous. Note that by using a two-sided offset mesh proxy, there are two different target meshes and the triangle soup has one bijection to the outer mesh proxy and one bijection to the inner mesh proxy, see Fig. 2.

In our approach, we do not use a charged offset mesh proxy and thus get monopole field lines, which extend from the surface to infinity. We use the UV-coordinates of the *first intersection* of such field lines with the offset mesh proxy as point correspondences for the vertices. This approach is only *injective*, because each point of the offset mesh proxy can only be intersected by one field line, but a single field line may intersect the offset mesh proxy more than once, so the other intersection points are not in the image of the mapping. Fig. 22 shows an example of field lines emerging from a concavity,

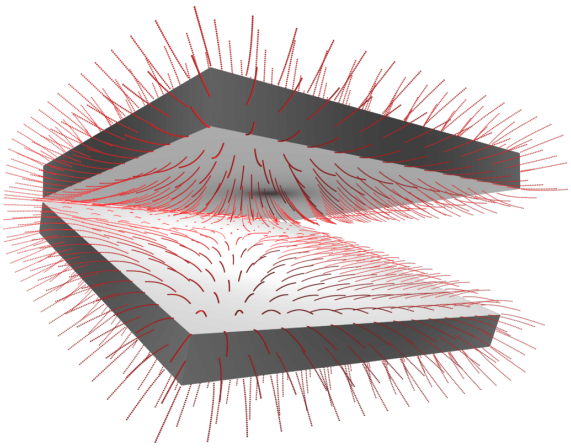


Fig. 21: The field lines between the two plates have a large curvature because both plates repel them and the field has a singularity at the center and the common edge of the plates.

which intersect an offset mesh proxy multiple times.

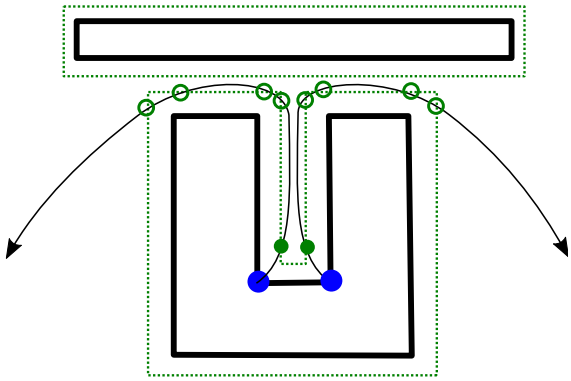


Fig. 22: Monopole field lines may intersect an offset mesh proxy more than once and thus cannot guarantee a bijective mapping.

12.4 Triangle Soups with Boundary

Our approach works best for triangle soups, which approximate watertight meshes. On triangle soups which approximate meshes with boundary, the field tracing becomes more sensitive to distortions and we might get ambiguities in the field.

In an approximation of a watertight mesh, the “inside” has a near zero field, which makes it easy to determine inside and outside and prevents tracing field lines in the wrong direction due to approximation errors. When the triangle soup is approximating a mesh with boundary, its field lines have several disadvantages: The field lines of boundary vertices often point in a direction tangential to the triangle, while a perpendicular direction would lead to a better mapping. For triangles which are part of a “one-sided” triangle soup, like a

plane, the vertices have field lines of similar strength in front and back direction, what makes it hard to use the field lines to determine which side of the triangle should be textured without using additional information like the surface normal orientation. An inconsistent choice of directions causes to severe distortion, as a the vertices of a triangle will be mapped to different sides of the corresponding offset mesh proxy.

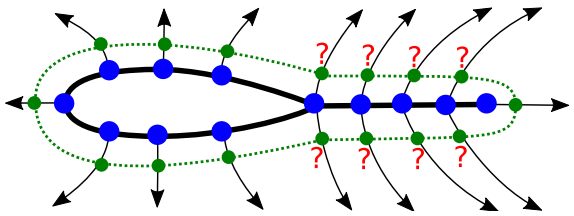


Fig. 23: While the texture mapping is well-defined and continuous on the left side, the one-sided boundary at the right side induces field lines in two directions with no clear indication which one should be used. Furthermore, the boundary vertex at the right causes a singularity in the field.

13 Conclusion

We presented a method to generate and trace field lines of electrostatic monopole fields in an efficient way to use them for finding point correspondences between triangle soups and enclosing 2-manifold meshes. This result proved to be useful for fully automatic texture mapping between triangle soups and automatically generated offset mesh proxies.

Our novel monopole field line tracing algorithm has a total complexity of $\mathcal{O}(k \cdot n)$ for tracing monopole field lines from n vertices of a triangle soup for k steps and $\mathcal{O}(k \cdot (n+m))$ for tracing dipole field lines, where m is the number of additional vertices on the enclosing mesh.

The reduced complexity is a significant improvement over previous approaches like [16], which have a complexity of $\mathcal{O}(k \cdot n^2)$ for tracing monopole field lines and $\mathcal{O}(k \cdot (n+m)^2)$ for tracing dipole field lines. Furthermore, previous approaches that use electrostatic *monopole* fields such as the technique of Wang et al. [54] need to solve a dense linear equation system, which has a complexity of $\mathcal{O}(n^3)$. Our technique reduces the complexity to $\mathcal{O}(n)$ by using approximations for the involved physical quantities.

We evaluated both convergence and run-time of our algorithm and tested it on a multitude of 3D models from shape benchmark sets and real-world CAD models. We have shown, that the technique can establish point correspondences for texture transfer. Besides, we discussed that an approximation of the underlying physics provides sufficient accuracy.

The evaluation of the fast multipole method and the robin hood algorithm has shown, that both hold their promise regarding computational complexity and quality of the approximation. The significant speedup achieved through these approximations enabled us to use field lines to parameterize large triangle soups. This might not be achieved using other techniques, as a significantly larger computational effort is needed.

By using monopole fields, we gained the flexibility to use different mesh proxies without the need for a time-consuming recomputation of the field lines. Furthermore, we are now able to experiment with different mesh proxy shapes without considering their influence on the electrostatic field.

Acknowledgments

We like to thank Volkswagen AG for providing us with parts of car models and the reviewers for their valuable feedback on the article.

References

[1] Noam Aigerman, Shahar Z Kovalsky, and Yaron Lipman. Spherical orbifold tu e embeddings. *ACM Transactions on Graphics (TOG)*, 36(4):90, 2017.

[2] Noam Aigerman and Yaron Lipman. Orbifold tutte embeddings. *ACM Trans. Graph.*, 34(6):190–1, 2015.

[3] Noam Aigerman and Yaron Lipman. Hyperbolic orbifold tutte embeddings. *ACM Trans. Graph.*, 35(6):217–1, 2016.

[4] Noam Aigerman, Roi Poranne, and Yaron Lipman. Seamless surface mappings. *ACM Trans. Graph.*, 34(4):72:1–72:13, July 2015.

[5] Marc Alexa. Recent advances in mesh morphing. In *Computer graphics forum*, volume 21, pages 173–198. Wiley Online Library, 2002.

[6] Gavin Barill, Neil Dickson, Ryan Schmidt, David I.W. Levin, and Alec Jacobson. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics*, 2018.

[7] Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.

[8] Silvia Biasotti, Andrea Cerri, A Bronstein, and M Bronstein. Recent trends, applications, and perspectives in 3d shape similarity assessment. In *Computer Graphics Forum*, volume 35, pages 87–119. Wiley Online Library, 2016.

[9] Alon Bright, Edward Chien, and Ofir Weber. Harmonic global parametrization with rational holonomy. *ACM Transactions on Graphics (TOG)*, 36(4):89, 2017.

[10] Oliver Burghard, Alexander Dieckmann, and Reinhard Klein. Embedding shapes with green's functions for global shape matching. *Computers & Graphics*, 68C:1–10, 2017.

[11] Stéphane Calderon and Tamy Boubekeur. Bounding proxies for shape approximation. *ACM Transactions on Graphics (TOG)*, 36(4):57, 2017.

[12] Frédéric Chazal, Maks Ovsjanikov, Etienne Corman, Michael Bronstein, Emanuele Rodola, Mirela Ben-Chen, Leonidas Guibas, and Alex Bronstein. Computing and processing correspondences with functional maps. In *ACM SIGGRAPH 2017 Courses*, 2017.

[13] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 905–914. ACM, 2004.

[14] Richard Courant, Kurt Friedrichs, and Hans Lewy. On the partial difference equations of mathematical physics. *IBM journal of Research and Development*, 11(2):215–234, 1967.

[15] Patrick Degener. Computing parameterizations of triangulated surfaces with minimal metric deformations. Diplomarbeit, Institut für Informatik II, Universität Bonn, 2003.

[16] Patrick Degener and Reinhard Klein. Texture atlas generation for inconsistent meshes and point sets. In *IEEE International Conference on Shape Modeling and Applications 2007 (SMI'07)*, pages 156–168. IEEE Computer Society, June 2007.

[17] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[18] David A Field. Laplacian smoothing and delaunay triangulations. *International Journal for Numerical Methods in Biomedical Engineering*, 4(6):709–712, 1988.

[19] Michael S Floater and Kai Hormann. Surface parameterization: a tutorial and survey. In *Advances in multiresolution for geometric modelling*, pages 157–186. Springer, 2005.

[20] Xiao-Ming Fu and Yang Liu. Computing inversion-free mappings by simplex assembly. *ACM Transactions on Graphics (TOG)*, 35(6):216, 2016.

[21] Xiao-Ming Fu, Yang Liu, and Baining Guo. Computing locally injective mappings by advanced mips. *ACM Transactions on Graphics (TOG)*, 34(4):71, 2015.

[22] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[23] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[24] Kai Hormann and Günther Greiner. Mips: An efficient global parametrization method. Technical report, ERLANGEN-NUERNBERG UNIV (GERMANY) COMPUTER GRAPHICS GROUP, 2000.

[25] Kai Hormann, Günther Greiner, and Swen Campagna. Hierarchical parametrization of triangulated surfaces. In *Proceedings of Vision, Modeling, and Visualization*, volume 1999, pages 219–226, 1999.

[26] Kai Hormann, Konrad Polthier, and Alia Sheffer. Mesh parameterization: Theory and practice. In *ACM SIGGRAPH ASIA 2008 Courses*, SIGGRAPH Asia '08, pages 12:1–12:87, New York, NY, USA, 2008. ACM.

[27] Michael Kazhdan, Allison Klein, Ketan Dalal, and Hugues Hoppe. Unconstrained isosurface extraction on arbitrary octrees. In *Symposium on Geometry Processing*, volume 7, pages 256–263, 2007.

[28] Vladimir Kim, Yaron Lipman, and Thomas Funkhouser. Blended intrinsic maps. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 30(4), July 2011.

[29] Francis Lazarus and Anne Verroust. Three-dimensional metamorphosis: a survey. *The Visual Computer*, 14(8):373–389, 1998.

[30] Predrag Lazić, Hrvoje Štefančić, and Hrvoje Abraham. The robin hood method–a novel numerical method for electrostatic problems based on a non-local charge transfer. *Journal of Computational Physics*, 213(1):117–140, 2006.

[31] Bruno Lévy and Jean-Laurent Mallet. Non-distorted texture mapping for sheared triangulated meshes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 343–352, New York, NY, USA, 1998. ACM.

[32] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *ACM transactions on graphics (TOG)*, volume 21, pages 362–371. ACM, 2002.

[33] Yaron Lipman and Thomas Funkhouser. Möbius voting for surface correspondence. *ACM Transactions on Graphics (TOG)*, 28(3):72, 2009.

[34] Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J Gortler. A local/global approach to mesh parameterization. In *Computer Graphics Forum*, volume 27, pages 1495–1504. Wiley Online Library, 2008.

[35] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[36] J Ma, V Rokhlin, and Stephen Wandzura. Generalized gaussian quadrature rules for systems of arbitrary functions. *SIAM Journal on Numerical Analysis*, 33(3):971–996, 1996.

[37] Ashish Myles and Denis Zorin. Global parametrization by incremental flattening. *ACM Transactions on Graphics (TOG)*, 31(4):109, 2012.

[38] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps: a flexible representation of maps between shapes. *ACM Transactions on Graphics (TOG)*, 31(4):30, 2012.

[39] Maks Ovsjanikov, Quentin Mérigot, Facundo Mémoli, and Leonidas Guibas. One point isometric matching with the heat kernel. In *Computer Graphics Forum*, volume 29, pages 1555–1564. Wiley Online Library, 2010.

[40] Les Piegl. On nurbs: A survey. *IEEE Comput. Graph. Appl.*, 11(1):55–71, January 1991.

[41] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics*, 2(1):15–36, 1993.

[42] Roi Poranne, Marco Tarini, Sandro Huber, Daniele Panozzo, and Olga Sorkine-Hornung. Autocuts: simultaneous distortion and cut optimization for uv mapping. *ACM Transactions on Graphics (TOG)*, 36(6):215, 2017.

[43] Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. Scalable locally injective mappings. *ACM Transactions on Graphics (TOG)*, 36(2):16, 2017.

[44] Leonardo Sacht, Etienne Vouga, and Alec Jacobson. Nested cages. *ACM Transactions on Graphics (TOG)*, 34(6):170, 2015.

[45] Peter Sandilands and Taku Komura. Model topology change with correspondence using electrostatics. In *Proceedings of the 20th ACM Symposium on Virtual Reality Software and Technology*, pages 41–44. ACM, 2014.

[46] Rohan Sawhney and Keenan Crane. Boundary first flattening. *ACM Transactions on Graphics (TOG)*, 37(1):5, 2017.

[47] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[48] Alla Sheffer, Bruno Lévy, Maxim Mogilnitsky, and Alexander Bogomyakov. Abf++: fast and robust angle based flattening. *ACM Transactions on Graphics (TOG)*, 24(2):311–330, 2005.

[49] Alla Sheffer, Emil Praun, Kenneth Rose, et al. Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision*, 2(2):105–171, 2007.

[50] Chen Shen, James F O'Brien, and Jonathan R Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *ACM Siggraph 2005 Courses*, page 204. ACM, 2005.

[51] Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. The princeton shape benchmark. In *Shape modeling applications, 2004. Proceedings*, pages 167–178. IEEE, 2004.

[52] Oliver Van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. A survey on shape correspondence. In *Computer Graphics Forum*, volume 30, pages 1681–1707. Wiley Online Library, 2011.

[53] Remco C Veltkamp and Michiel Hagedoorn. State of the art in shape matching. In *Principles of visual information retrieval*, pages 87–119. Springer, 2001.

[54] He Wang, Kirill A Sidorov, Peter Sandilands, and Taku Komura. Harmonic parameterization by electrostatics. *ACM Transactions on Graphics (TOG)*, 32(5):155, 2013.

[55] Yunhai Wang, Shmulik Asafi, Oliver van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Active co-analysis of a set of shapes. *ACM Transactions on Graphics (TOG)*, 31(6):165, 2012.

[56] DRSM Wilton, S Rao, AW Glisson, D Schaubert, O Al-Bundak, and C Butler. Potential integrals for uniform and linear source distributions on polygonal and polyhedral domains. *IEEE Transactions on Antennas and Propagation*, 32(3):276–281, 1984.

[57] Zoë J Wood, Peter Schröder, David Breen, and Mathieu Desbrun. Semi-regular mesh extraction from volumes. In *Proceedings of the conference on Visualization'00*, pages 275–282. IEEE Computer Society Press, 2000.

[58] Jianhua Wu Leif Kobbelt. Structure recovery via hybrid variational surface approximation. In *Computer Graphics Forum*, volume 24, pages 277–284. Wiley Online Library, 2005.

[59] Dong-Ming Yan, Wenping Wang, Yang Liu, and Zhouwang Yang. Variational mesh segmentation via quadric surface fitting. *Computer-Aided Design*, 44(11):1072–1082, 2012.

[60] Rio Yokota and Lorena A. Barba. A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems. *CoRR*, abs/1106.2176, 2011.