Non-linear sphere tracing for rendering deformed signed distance fields

DARIO SEYB, Dartmouth College
ALEC JACOBSON, University of Toronto
DEREK NOWROUZEZAHRAI, McGill University
WOJCIECH JAROSZ, Dartmouth College

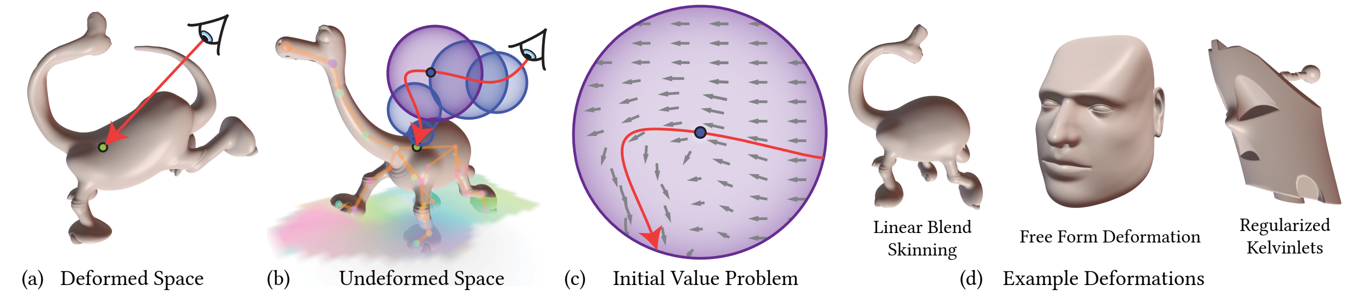


Fig. 1. We tackle the problem of rendering deformed signed distance fields (a), by phrasing sphere tracing in object space (b) as an initial value problem. Under non-linear deformation the straight deformed space ray becomes a curve, which we follow via numerical integration (c). We go to great lengths to avoid computing the *inverse deformation*. This enables us to easily apply many modern deformation techniques to signed distance fields (d).

Signed distance fields (SDFs) are a powerful *implicit* representation for modeling solids, volumes and surfaces. Their infinite resolution, controllable continuity and robust constructive solid geometry operations, coupled with smooth blending, enable powerful and intuitive sculpting tools for creating complex SDF models. SDF metric properties also admit efficient surface rendering with sphere tracing. Unfortunately, SDFs remain incompatible with many popular direct deformation techniques which re-position a surface via its *explicit* representation. Linear blend skinning used in character articulation, for example, directly displaces each vertex of a triangle mesh. To overcome this limitation, we propose a variant of sphere tracing for directly rendering deformed SDFs. We show that this problem reduces to integrating a non-linear ordinary differential equation. We propose an efficient numerical solution, with controllable error, which first automatically computes an initial value along each cast ray before walking conservatively along a curved ray in the undeformed space according to the signed distance. Importantly, our approach does not require knowledge, computation or even global existence of the inverse deformation, which allows us to readily apply many existing forward deformations. We demonstrate our method's effectiveness for interactive rendering of a variety of popular deformation techniques that were, to date, limited to explicit surfaces.

CCS Concepts: • **Computing methodologies** → **Ray tracing**; **Volumetric models**; *Animation.*

Additional Key Words and Phrases: sphere tracing, signed distance fields, deformation, non-linear ray tracing

Authors' addresses: Dario Seyb, dario.r.seyb.gr@dartmouth.edu, Dartmouth College; Alec Jacobson, jacobson@cs.toronto.edu, University of Toronto; Derek Nowrouzezahrai, derek@cim.mcgill.ca, McGill University; Wojciech Jarosz, wjarosz@dartmouth.edu, Dartmouth College.

ACM Reference Format:
Dario Seyb, Alec Jacobson, Derek Nowrouzezahrai, and Wojciech Jarosz. 2019. Non-linear sphere tracing for rendering deformed signed distance fields. *ACM Trans. Graph.* 38, 6, Article 229 (November 2019), 12 pages. https://doi.org/10.1145/3355089.3356502

1 INTRODUCTION

Explicit and implicit surface representations have unique and complementary advantages. Over the past decades, computer-aided design, interactive graphics, computer games and visual effects industries have gravitated toward explicit representations such as triangle meshes, NURBS or subdivision surfaces. While modeling an initial explicit surface can be tedious and error-prone, explicit surfaces are very easy to *deform*. For example, animating a triangle mesh of a video-game character is as simple as prescribing new vertex positions as a function of time.

In contrast, implicit surfaces boast a number of advantages over explicit surfaces during modeling, such as: infinite resolution, controllable continuity, trivially robust constructive solid geometry operations, domain repetition and smooth blending. Implicit representations are differentiable, tend to require less storage, and can simultaneously model volumes, solids and surfaces. These advantages are fueling a resurgence of interest in implicit functions, especially signed distance functions (SDFs), in computer vision and machine learning (due to differentiability), in "clay sculpting" VR content creation (due to globally classifying inside-outside regions), and in creative coding communities (due to platforms like ShaderToy that enable programmatic authoring of SDFs). While convenient for modeling, implicit functions are unfortunately not directly compatible with popular surface deformation techniques developed for animating explicit surfaces. A common but unsatisfactory solution is to *convert* an implicit surface into an explicit mesh (e.g., via marching cubes) before animation, inevitably losing surface information and many of the aforementioned advantages.

We propose a non-linear variant of sphere tracing for directly rendering deformed solids, volumes and surfaces defined as SDFs. We show how the problem can be cast as the numerical integration of an ordinary differential equation (ODE), and we provide an automatic construction to determine the initial value for this integration, before leveraging efficient ODE solvers. In this way, we maintain the strengths of SDFs while enabling the rich palette of real-time deformation and animation techniques (e.g., linear blend skinning, free-form deformations, Kelvinlets) that were previously only compatible with explicit representations. We demonstrate the effectiveness of our method with a prototypical implicit modeling/animation tool inspired by state-of-the-art VR content creation.

1.1 Related Work

Implicit surface modeling and rendering has a rich history in computer graphics [Bajaj et al. 1997; Blinn 1982; Fujita et al. 1990; Hart 1996; Wyvill et al. 1986b; Wyvill and Trotman 1990] with many applications [Jones et al. 2006; Pasko et al. 1995]. Beyond graphics, SDFs are also a natural representation for object reconstruction [Curless and Levoy 1996; Ilic and Fua 2006], tracking [Park et al. 2019; Schmidt et al. 2014; Taylor et al. 2017] and recognition tasks in vision/learning [Genova et al. 2019; Tulsiani et al. 2017]. They also play a key role in physical simulation for collision detection [Koschier et al. 2016] and fluid simulation [Sethian and Smereka 2003]. The ability to directly render deformed SDFs would enrich each of these application areas. Below, we focus on works related to rendering SDFs and other implicit surfaces, as well as the deformation techniques we would like to support.

Directly deforming implicits. For simple deformations (e.g. affine) it is sometimes possible to directly transform the parameters of the implicit function (e.g., a translated/scaled/rotated sphere is a general quadric) or transform rays by the inverse transformation (rays map to rays under affine transforms). Particle systems [Hart et al. 2002; Turk and O'Brien 1999; Witkin and Heckbert 1994] and composition trees [Wyvill et al. 1999] can also be used to directly deform or combine more complex implicit functions. Individual components of a compound implicit can be bound to animated affine transformations, simulating soft/deformable materials or simple articulated characters [Cani-Gascuel and Desbrun 1997; Desbrun and Gascuel 1995; Russell 1999; Wyvill et al. 1986a].While these tools can be used for some types of animation, the degrees of freedom in the animation are strongly tied to the underlying surface representation. Still, these approaches have witnessed a resurgence with the accessibility of recent implicit modeling tools for VR and AR [Brinx Software 2019; Evans 2015; Facebook Technologies 2019; Media Molecule 2019; Unbound Technologies 2019], and platforms like Shadertoy for creative coding [Korndorfer 2015]. While such methods can leverage blending of the constituent implicits to provide some smoothness [Gourmel et al. 2013], the resulting deformations remain limited compared to those available in professional animation pipelines.

Explicit forward deformation. The majority of deformation techniques used in animation – including smooth skinning [Jacobson et al. 2014], Kelvinlets [De Goes and James 2017], and free-form deformation (FFD) [Sederberg and Parry 1986] or cages [Joshi et al. 2007] – are *forward* deformation methods that map positions in undeformed space to the deformed space. Such deformations are easy to use in conjunction with explicit geometric representations, like subdivision surfaces or meshes, by simply applying the forward transform to (control) vertices. Implicit functions have also been used to push around (forward deform) explicit geometry for improved contact control and volume preservation [Vaillant et al. 2013, 2014]. We treat the converse problem of applying general forward deformations to implicit model representations, which has generally not been possible since there are no explicit world-space positions on which to apply the forward deformation function.

Deforming implicits via meshing. Meshing the implicit function using e.g., marching cubes [Lorensen and Cline 1987; Wyvill et al. 1986b] or related techniques [Ju et al. 2002; Kobbelt et al. 2001], enables any forward deformation technique and has currently been the dominant strategy in the aforementioned application areas. Unfortunately, this process is lossy, and it is hard to capture small details and sharp edges. This can be overcome by meshing at a higher resolution (at the cost of storage and speed), but it remains impractical for procedural fractal geometry [Barnsley et al. 1988; Ebert et al. 2003]. The mesh must also be recomputed any time the underlying implicit is modified, making it expensive to handle rapid editing updates, or time varying implicits (e.g., animated water).

Numerical root finding with inverse deformations. If we can efficiently compute the *inverse* of the deformation at any point in (world) space, then we could render the deformed surface directly by evaluating the implicit function at each step of a numerical root finding [Barr 1986] method, like ray marching [Perlin and Hoffert 1989] or interval arithmetic [Knoll et al. 2007, 2009; Mitchell 1990]. For SDFs, sphere tracing [Bálint and Valasek 2018; Hart 1996; Keinert et al. 2013; Reiner et al. 2011] is the preferred alternative to ray marching, since it provides automatic adaptive stepping and antialiasing. Unfortunately, it is not currently compatible with general non-linear deformations since they invalidate the SDF's distance metric, and appropriate Lipschitz bounds are not always easy to calculate. More importantly, even for simple deformations like linear FFDs (i.e., cubical voxel cages with trilinear interpolation), the inverse transformations simply do not exist (e.g., the forward transform is not bijective) or are not known.

One alternative is to use even simpler deformation building blocks which do admit analytic inverses. Shell mapping [Porumbescu et al. 2005], for instance, applies implicitly defined fine-scale detail to a mesh via a surrounding tetrahedral shell. Since the tets define a piecewise affine deformation of space, the inverse deformation within each tet only requires a 4×4 matrix inverse. Unfortunately, such piecewise affine deformations cause C^1 discontinuities at tet boundaries. Smooth shell mapping [Jeschke et al. 2007] reduces these discontinuities by replacing the tets with bilinear prisms, but this causes rays to bend, complicating the resulting inverse calculation.

Instead of computing inverses of a desired forward deformation, an alternative is to define a new class of *inverse* deformations which

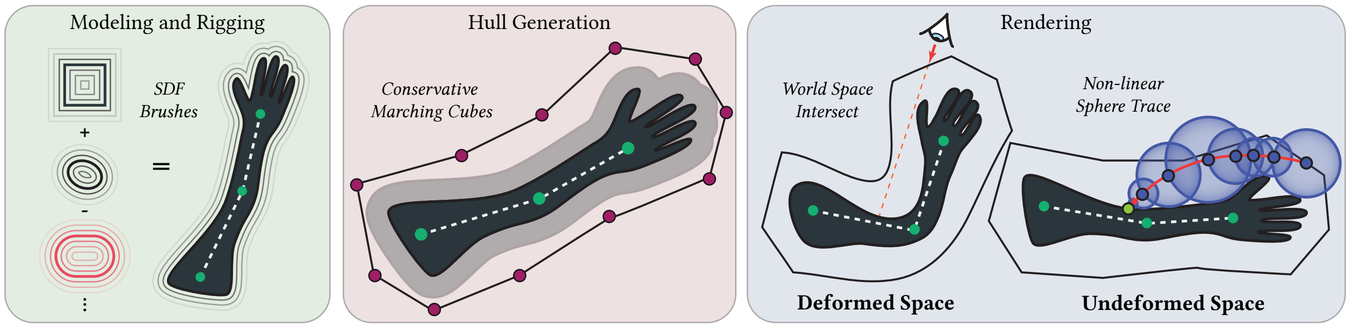


Fig. 2. Shown here is a high level overview over our method applied to linear blend skinning. After having modeled an implicit surface using a combination of *analytic brushes* and having defined an appropriate skeleton, we generate a *triangle mesh hull* that encloses the implicit surface via marching cubes at a low resolution. We then deform this hull with the chosen forward mapping deformation technique and rasterize it. The deformed space position retrieved from rasterization is easily transformed back to undeformed space via barycentric interpolation across the corresponding hull face (Section 3.2). A *non-linear sphere tracing* procedure is then started at the approximate undeformed space position (Section 3).

natively map from (deformed) deformed space to (undeformed) undeformed space. Unfortunately, these approaches tend to not be as versatile as the forward ones, since the resulting deformations create surface duplication for large deformations (space warping [Beier and Neely 1992; Chen et al. 2001]), are difficult to control artistically (level-set evolution methods [Osher and Sethian 1988]), or suffer from numerical dissipation and other degradation of the surface after repeated blending operations (variational warping [Sugihara et al. 2010; Turk and O'Brien 2005]) which needs to be corrected [Slavcheva et al. 2017].

Deformation via non-linear ray tracing. Non-linear ray tracing methods that account for light rays bending due to gravity or other forces [Gröller 1995; James et al. 2015; Satoh 2003] or from passing through media with a continuously-varying index of refraction [Berger et al. 1990; Cao et al. 2010; Gutierrez et al. 2005; Seron et al. 2004; Sloup 2003; Stam and Languénou 1996] can also be used to render deformed objects. These methods typically work directly with the forward deformation, but since they model specific physical phenomena, the resulting deformations are quite limited. Kurzion and Yagel [1995] proposed "ray deflectors" which deform space only locally for better artistic control. Our approach is conceptually similar, though we derive our "deflectors" or "attractors" to directly model a chosen forward deformation, like smooth skinning or Kelvinlets, and we operate in undeformed space where distances are preserved to enable efficient rendering with sphere tracing.

1.2 Overview

We pose non-linear sphere tracing as an application of parametric curve deformation [Barr 1984], which we express as an initial value problem using the Jacobian of the (inverse) deformation (Section 3). Neyret [1996] similarly locally linearized a deformation, effectively solving this initial value problem with Euler integration. We enable the use of arbitrary ODE solvers and crucially, we decouple the steps needed to faithfully represent the curved ray trajectories from the expensive SDF lookups needed by sphere tracing (Section 3.1), allowing real-time performance with low error. This approach, however, still requires computing the inverse at least once to seed the initial value, so it remains incompatible with complex deformations like smooth skinning where a unique inverse does not exist or is not known. We solve this problem (Section 3.2) by transitioning from world- to object-space at the boundary of a coarse hull mesh enclosing the implicit surface, entirely avoiding the need for (potentially non-existent) deformation inverses, while providing quantifiable error control. Figure 2 illustrates our approach, and we apply it to a combination of several common forward deformation approaches, like smooth skinning and Kelvinlets, in Section 6.

2 PROBLEM STATEMENT

We assume our geometry is represented by a *signed distance field* $S : \mathbb{R}^3 \mapsto \mathbb{R}$ for some subset of $\mathbb{R}^3$ and $\|\nabla S\| \approx 1$. How this mapping is described, which regions of space can be evaluated, and how much attention is paid to the gradient bound depends on the application.

This geometry is deformed by a *deformation function* D. In this work, we limit ourselves to *space deformations*, which means that D is a function $D : \mathbb{R}^3 \mapsto \mathbb{R}^3$ for a volumetric subset of $\mathbb{R}^3$ instead of for some set of explicit points on a surface. Luckily, many forward deformations used in practice either define space deformations or can be easily made to do so, as we will discuss in Section 4.

We wish to visualize the 0-isosurface of the distance field S directly. While some simple distance fields admit analytic ray-isosurface intersection methods, these do not exist in general. To be more specific, we want to find the point on a *ray* $\boldsymbol{x} : \mathbb{R} \mapsto \mathbb{R}^3$ with $\boldsymbol{x}(s) = \boldsymbol{p} + s\boldsymbol{\omega}$ which simultaneously lies on the isosurface $S(\boldsymbol{x}) = 0$. We use italic (e.g., s) for scalars, bold lowercase letters (e.g., $\boldsymbol{p}, \boldsymbol{x}$) for points in $\mathbb{R}^3$, and $\boldsymbol{\omega}$ always represents a unit-length direction vector in $\mathbb{R}^3$. Disregarding deformation for now, this task is equivalent to finding the roots of $S(\boldsymbol{x}(s)) = 0$. There are a few methods to do this and we build on sphere tracing [Hart 1996] which works very well for implicit surfaces with $\|\nabla S\| \approx 1$ and requires $\|\nabla S\| \leq 1$.

Sphere tracing is a form of ray marching that iteratively steps at points $\boldsymbol{x}_i$ along the ray, but uses the distance value returned by the SDF at each point as the next stepping distance (see Fig. 3):

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + |S(\boldsymbol{x}_i)|\,\boldsymbol{\omega} \quad \text{with} \quad \boldsymbol{x}_0 = \boldsymbol{p}. \tag{1}$$

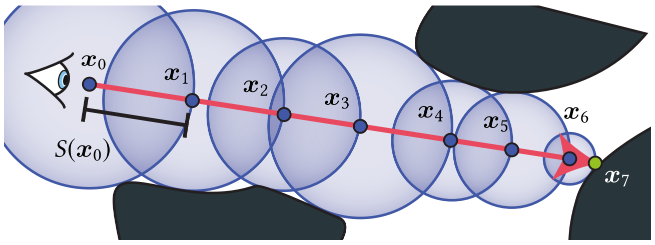


Fig. 3. Unbounding spheres along a straight ray.

Since $\|\nabla S\| \leq 1$, each evaluation of the SDF defines an "unbounding sphere" which is guaranteed to be intersection-free. The process continues until we reach some user-specified threshold $|S(\boldsymbol{x}_i)| < \epsilon$.

The common method to apply a deformation D to an SDF is to solve $(S \circ D^{-1})(\boldsymbol{x}(s)) = 0$ instead. We could accomplish this by still stepping along the ray in deformed space while transforming each point back into undeformed space to sample the distance field:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \left|(S \circ D^{-1})(\boldsymbol{x}_i)\right| \boldsymbol{\omega}. \tag{2}$$

While conceptually simple, this makes two limiting assumptions. Most importantly, D^{-1} has to be available and cheap to compute. While this is the case for affine transformations and even many non-linear ones, such as twists and bends [Barr 1984; Wyvill et al. 1998], it is not the case in general. Even for, e.g., a simple FFD based on tri-linear interpolation, an analytic inverse does not exist. Furthermore, even when D^{-1} is available, we have to ensure that $\left\|\nabla(S \circ D^{-1})\right\| \leq 1$ in order to use sphere tracing [Hart 1996]. Unfortunately, S gives us the distance from the surface in *undeformed space*, but we are stepping in *deformed space*. If D is *Lipschitz continuous* then we can step according to $|S(\boldsymbol{x}_i)|/\lambda$ if we know the associated *Lipschitz constant* $\lambda \geq \left\|\nabla(S \circ D^{-1})\right\|$. Again, for affine transformations this is trivial [Stander and Hart 1994], but for more complicated deformations Lipschitz constants can be tedious or even impossible to compute analytically. Moreover, assuming we know the Lipschitz constant λ for a given D, adjusting the distance function by this constant value for every point in space can significantly degrade tracing performance if λ is very large. This is particularly problematic if we have $\lambda \geq \left\|\nabla(S \circ D^{-1})(\boldsymbol{x})\right\| \gg 1$ for some small region $\boldsymbol{x} \in A$, but for most regions of space $\boldsymbol{x} \notin A$, we have $\left\|\nabla(S \circ D^{-1})(\boldsymbol{x})\right\| \approx 1$. Here, we would be forced to take unnecessarily small steps for $\boldsymbol{x} \notin A$ since $|S(\boldsymbol{x})|/\lambda \ll 1$. Computing local Lipschitz constants would solve this problem but requires a non-negligible computation whenever the deformation changes, i.e., every frame in most scenarios.

All of this complexity arises from the fact that we are tracing in *deformed space*, but the distance values we sample are defined in *undeformed space*. If we could trace in *undeformed space* instead, we would not have to worry about Lipschitz bounds. The issue now is that for non-linear transformations, the straight ray in *deformed space* is deformed to some, potentially complex, curve in *undeformed space* (Fig. 4).

3 NON-LINEAR SPHERE TRACING

Given these issues, we propose *Non-linear Sphere Tracing* (NLST). Building on Barr's [1984] formulation for deforming parametric curves/surfaces, we rewrite the ray equation as an integral

$$\boldsymbol{x}(s) = \boldsymbol{p} + s\boldsymbol{\omega} = \boldsymbol{p} + \int_0^s \boldsymbol{\omega}\, \mathrm{d}t. \tag{3}$$

Here $\boldsymbol{x}'(s) = \boldsymbol{\omega}$ is the first derivative of $\boldsymbol{x}$ with respect to the parameter s and is a constant in deformed space. Applying the inverse deformation D^{-1} to Eq. (3) gives us the ray expressed in undeformed space

$$\hat{\boldsymbol{x}}(s) = D^{-1}(\boldsymbol{p}) + \int_0^s J_{D^{-1}}(\boldsymbol{x}(t))\, \boldsymbol{\omega}\, \mathrm{d}t, \tag{4}$$

where we use $\hat{\bullet}$ to indicate the object-space equivalent of a world-space quantity $\bullet$. Note that we do not have to evaluate D^{-1} inside the integral, but only its Jacobian $J_{D^{-1}}$. Moreover, due to the inverse function theorem, the Jacobian of the *inverse deformation* at some world-space parameter location t is the inverse of the Jacobian of the *forward deformation* at a corresponding object-space parameter location $\hat{t}$ (i.e., $J_{D^{-1}}(\boldsymbol{x}(t)) = J_D^{-1}(\hat{\boldsymbol{x}}(\hat{t}))$), which is just a 3×3 matrix. So, as long as we have access to the Jacobian of our deformation function, and that Jacobian matrix is invertible in the region of space we are tracing through, we can evaluate the integrand. Finally, we reparameterize the integral (4) with respect to arc-length in *undeformed space* to arrive at our final *object-space ray integral*

$$\hat{\boldsymbol{x}}(\hat{s}) = D^{-1}(\boldsymbol{p}) + \int_0^{\hat{s}} \hat{\boldsymbol{\omega}}(\hat{\boldsymbol{x}}(t))\, \mathrm{d}t, \quad \text{with} \quad \hat{\boldsymbol{\omega}}(\hat{\boldsymbol{x}}) = \frac{J_D^{-1}(\hat{\boldsymbol{x}})\, \boldsymbol{\omega}}{\left\|J_D^{-1}(\hat{\boldsymbol{x}})\, \boldsymbol{\omega}\right\|}. \tag{5}$$

This allows us to consider $\hat{s}$ as a distance along $\hat{\boldsymbol{x}}$ and, in particular, it means that the parameter $\hat{s}$ and our signed distance field $S(\hat{\boldsymbol{x}})$ are defined in the same metric space. We are now left with two issues. First, we cannot compute the integral in Eq. (5) analytically (we do not even know $\hat{s}$ a priori!). Secondly, we still need to evaluate D^{-1} once, for the start point $\boldsymbol{p}$ of our ray in deformed space. We will tackle these problems in Sections 3.1 and 3.2, respectively.

3.1 A Joint Method for Root Finding and Ray Integration

Recall from Section 2 that we are trying to find the smallest world-space distance s such that $(S \circ D^{-1})(\boldsymbol{x}(s)) < \epsilon$. Using the results from the last section, this is equivalent to finding the object-space distance $\hat{s}$ such that $S(\hat{\boldsymbol{x}}(\hat{s})) < \hat{\epsilon}$. Now, the naïve extension of sphere tracing to non-linear sphere tracing is to step along $\hat{\boldsymbol{x}}$ in increments of $|S(\hat{\boldsymbol{x}})|$, which would allow us to evaluate points $\hat{\boldsymbol{x}}_i$ iteratively as

$$\hat{\boldsymbol{x}}_{i+1} = \hat{\boldsymbol{x}}_i + |S(\hat{\boldsymbol{x}}_i)|\, \hat{\boldsymbol{\omega}}(\hat{\boldsymbol{x}}_i), \quad \text{with} \quad \hat{\boldsymbol{x}}_0 = D^{-1}(\boldsymbol{p}). \tag{6}$$

While this would work reasonably well when $S(\hat{\boldsymbol{x}}_i)$ is small relative to the ray deformation, error would quickly accumulate when the ray is highly curved and $\hat{\boldsymbol{x}}_i$ is far from the surface (see Fig. 5 a). One way to account for the ray deformation is to reduce the step

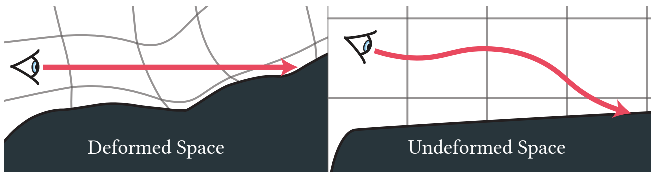


Fig. 4. A straight ray in deformed space (left) maps to a curve in undeformed space under deformation (right).

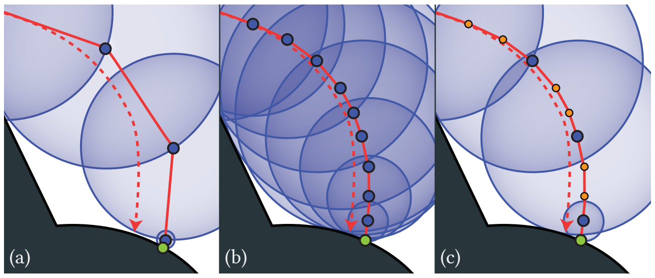
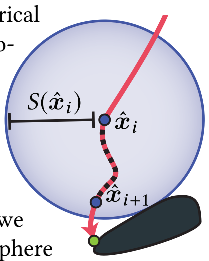


Fig. 5. With strongly deformed rays, the step size given by $S(\hat{\boldsymbol{x}}_i)$ might be too large to accurately sample the ray (a). Reducing the step size naïvely improves accuracy, but necessitates many more evaluations of S, degrading performances (b). Our method reproduces the ray to the same accuracy while only minimally increasing the number of evaluations of S (c).

size by some factor (Fig. 5 b), e.g., using $|S(\hat{\boldsymbol{x}}_i)|/10$ instead of $|S(\hat{\boldsymbol{x}}_i)|$ in Eq. (6). While this would allow us to reconstruct the deformed ray more accurately, reducing the step size sacrifices performance benefits that we hope to gain from sphere tracing, effectively falling back to naïve non-linear ray marching.

Our insight is that we can separate the step size requirements of the *ray integrator* from those of the *root finding process* by introducing *substeps* within each unbounding sphere without requiring additional SDF lookups. To achieve this, we can pose each linear sphere tracing step i from $\hat{\boldsymbol{x}}_i$ to $\hat{\boldsymbol{x}}_{i+i}$ in Eq. (6) more generally as the solution to a first-order ordinary differential equation $\boldsymbol{y}'(\hat{\boldsymbol{x}}) = \hat{\boldsymbol{\omega}}(\hat{\boldsymbol{x}})$ with initial condition $\boldsymbol{y}_0 = \hat{\boldsymbol{x}}_i$. Now we can compute the next sphere tracing step as $\hat{\boldsymbol{x}}_{i+1} = \mathrm{R}(\hat{\boldsymbol{x}}_i, S(\hat{\boldsymbol{x}}_i))$, where R is any ODE solver taking the initial condition and the target integration duration as parameters. Equation (6) is just the most simple instantiation of such a solver using a single forward Euler integration step. We found that first-order integrator proved impractical for most non-trivial deformations. Thanks to the common occurrence of ODEs, there is a wealth of research into higher-order solvers for many problem types [Butcher and Goodwin 2008]. We will discuss our particular choice, and how we control error due to numerical integration in Section 3.3. Importantly, a chosen solver might divide $S(\hat{\boldsymbol{x}}_i)$ into substeps as needed to reach a given error tolerance, but it never needs to *re-evaluate* $S(\hat{\boldsymbol{x}})$. Thanks to the arc-length parameterization of $\hat{\boldsymbol{x}}$, any correct solver will ensure that $\|\mathrm{R}(\hat{\boldsymbol{x}}_i, \hat{s}) - \hat{\boldsymbol{x}}_i\| \leq S(\hat{\boldsymbol{x}}_i)$, $\forall \hat{s} \leq S(\hat{\boldsymbol{x}}_i)$. That is, during each substep we are guaranteed to stay inside the unbounding sphere around $\hat{\boldsymbol{x}}_i$ and thus never violate the sphere tracing condition.



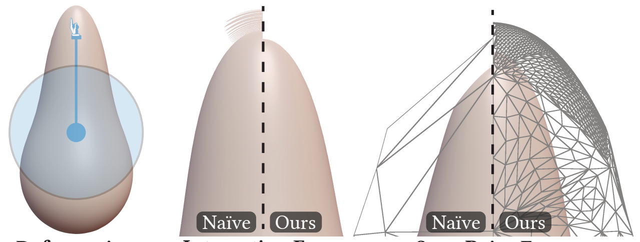
3.2 Finding the Undeformed Space Ray Start

As discussed in the last section, inverses are very expensive to compute or do not exist at all for most non-trivial forward deformation methods. While we avoid computing inverses along the ray thanks to our non-linear sphere tracing method, we still need to find the ray start point $\hat{\boldsymbol{x}}_0 = D^{-1}(\boldsymbol{p})$ in undeformed space. Since in general D might not be invertible or even well defined at the camera position, we propose to automatically generate a *low resolution explicit hull* that completely contains the surface in undeformed space. We then deform the hull vertices using the forward transform and intersect the deformed triangle mesh with the deformed space view ray. In our implementation we rasterize the hull mesh to find the primary intersection point, but ray tracing it via simple ray-triangle intersections is possible as well. This is a generalization of the bounding-box technique for determining an initial value proposed by Barr [1986]. Given the intersection point $\boldsymbol{p}$ in *deformed space* we use barycentric coordinates for the respective triangle to efficiently obtain the corresponding *object-space* location $\hat{\boldsymbol{p}}$. In fact, this is done automatically in hardware as object-space vertex positions can be interpolated from vertex to fragment shading units by the rasterizer. This effectively constructs the approximate inverse $\widetilde{D^{-1}}(\boldsymbol{p}) \approx D^{-1}(\boldsymbol{p})$.

Conservative marching cubes. To generate the hull mesh we use a version of the basic marching cubes algorithm [Lorensen and Cline 1987]. Marching cubes is a good fit because it is easy to implement efficiently on the GPU. We could employ alternative contouring methods (e.g., [Wyvill et al. 1986b]) as well, but did not explore them at this time. Additionally, while more sophisticated algorithms are available, we would not derive much benefit from them. At this stage in our algorithm, we do not have to worry about reproducing sharp features. We simply would like to generate a mesh that roughly encloses the underlying isosurface. By default marching cubes is not conservative, but if we offset the original isosurface by the marching cubes diagonal cell size we can guarantee that the hull completely contains the original surface. Note that this works if the surface is defined by an SDF because isosurface offsets directly correspond to distances. This is necessary because any part of the isosurface outside the hull would not be rendered, and hull faces cutting through the isosurface would reveal the inside of the object.

3.3 Principled Methods for Controlling Error

Our method has two possible sources of errors, the numerical integration of the trajectory and the approximation of the inverse via hull linearization. That is, when there is error we either do not follow the ray trajectory accurately or we start the trajectory at the wrong location in undeformed space. Note that while we are not necessarily guaranteed to hit the surface in the correct location, we *are* guaranteed to not step into the surface thanks to sphere tracing. Here we show how error can manifest itself in the rendered image.



We deform a sphere by a single Kelvinlets brush (left). When we use naïve Euler integration, the surface is not reproduced faithfully and in extreme cases numerical issues can cause ringing artifacts. These vanish when using an adaptive integration method (center). When the hull does not provide enough resolution, the error in the ray start point causes artifacts and the surface does not seem

C1 continuous. This is fixed by subdividing the hull as described in this section (right). Both of these types of error are particularly visible in motion and we refer to the supplemental video for a better impression of the resulting artifacts.

Choosing an appropriate ODE solver. In Section 3.1 we showed that we can use any existing ODE solver to accurately reproduce the ray, but we need to consider the solver we choose carefully, because much of our method's practicality depends on its performance. Under many forward deformations, the ray stays fairly straight in large regions of space while being strongly deformed in some small subset. Additionally, once we get close to the surface, $S(\hat{\boldsymbol{x}})$ becomes small and a simple forward Euler integrator as in Eq. (6) is often sufficient. If we employ a solver that treats all of these cases uniformly it will either not reproduce the ray correctly under strong deformation or incur a large overhead in the common simple case.

Hence we propose a hybrid approach. When $S(\hat{\boldsymbol{x}})$ is relatively large compared to $\hat{\epsilon}$ we use an adaptive Runge-Kutta integrator. There are many methods we can pick from and we evaluated Runge-Kutta-Fehlberg (RKF45) [Fehlberg 1970], Dormand-Prince (DP54) [Dormand and Prince 1980] or Bogacki-Shampine (BS23) [Bogacki and Shampine 1989]. While RKF45 and DP54 provide a high-order accuracy, the minimum number of J_D^{-1} evaluations they require is relatively high as well. In practice we have found BS23 to be accurate enough for our purposes while only requiring a minimum of four evaluations per step. Still, when we are close to the surface and $S(\hat{\boldsymbol{x}})$ is small anyway, we would like to avoid this additional overhead: we fall back to simple Euler integration once $S(\hat{\boldsymbol{x}})$ is smaller than $a \cdot \hat{\epsilon}$, where a is a user-chosen threshold (3 in our implementation).

Another question is whether an implicit solver could provide additional benefits. While we did not evaluate this rigorously we would not expect the implicit solver to perform well. The equations we are solving tend to not be particularly stiff and we did not observe any objectionable artifacts when using the Bogacki-Shampine solver. Since we enforce a *constant speed* along our ray, situations with runaway error as the effective step size increases rapidly are not possible. Additionally, using an implicit solver would incur a minimum performance overhead which is difficult to control adaptively, as opposed to the small minimum overhead of Bogacki-Shampine.

An adaptive error threshold. The advantage of using an adaptive integrator is that we can choose an error threshold that is specified as a distance in undeformed space and the integrator will add as many substeps as necessary to reach that threshold. The choice of error threshold is important because it greatly affects rendering quality as well as performance. If the threshold is too high, we will get artifacts which manifest as the surface "swimming", that is, seeming to be in a different place depending on view direction (see the supplemental video). Even during traditional sphere tracing we have to choose a threshold ϵ below which we terminate the ray. A common way to choose this threshold in a principled manner is called *cone tracing*. We construct a cone around the ray with an opening angle determined by the footprint of one pixel. We can then compute the error threshold as

$$\hat{\epsilon}(\hat{s}) = \hat{s}\hat{r_p}, \tag{7}$$

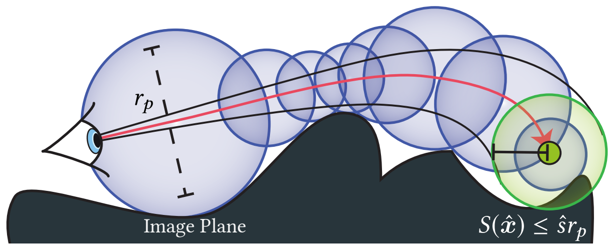


Fig. 6. When we step along the deformed ray we keep track of $\hat{s}$ which allows us to compute the cone radius. Once $S(\hat{\boldsymbol{x}})$ is less than the cone radius (green), we terminate the ray. This means that the intersection point is bounded by $\hat{\epsilon}$. We use the same error threshold for the adaptive integrator and hence this bound is preserved.

where $\hat{r_p}$ is the radius of one pixel in undeformed space. We can compute $\hat{r_p}$ as $\hat{r_p} = sr_p \left\| J_D^{-1} \right\|$, that is, the cone radius at the deformed space intersection point times the inverse Jacobian determinant. This leads to rays that terminate as soon as the pixel cone intersects the surface. We propose to apply this same threshold during adaptive integration.

Reducing hull linearization error via adaptive subdivision. The other source of error is the approximate inverse $\widetilde{D^{-1}}$ introduced in the last section. If the difference between our approximation and the actual inverse of D is too large, we will not start rays at the correct positions in undeformed space. When we deform the hull by deforming its vertices we are in effect sampling the deformation at a finite number of points and interpolating the samples linearly across faces. Since the deformation is in general non-linear this interpolation is not exact. The hull triangles should bend in deformed space but they stay flat. In Fig. 7 we show an example of this in two dimensions. The shape is bounded by a low resolution hull (Fig. 7 a) which is deformed by a strongly non-linear deformation. While for the hull vertices $D(\widetilde{D^{-1}}(\boldsymbol{p})) = \boldsymbol{p}$ holds, this is not true for positions on the segments lying between vertices. Instead of bending like the green ground truth shape, the hull segments stay piecewise linear (Fig. 7 b). This results in an error in the start point when we use $\widetilde{D^{-1}}$ to transform the world-space intersection point $\boldsymbol{p}$ to object-space $\hat{\boldsymbol{p}}$.

When we reduce the lengths of the individual linear segments our deformed hull approximates the ground truth much better (Fig. 7 c) and the error is greatly reduced. This suggests that to limit error we should limit the size of the triangles in our hull. One way would be to increase the grid resolution we use when running the marching cubes algorithm. This reduces the size of the grid cells and hence the size of the generated triangles as well. Of course, the issue with this is that it scales poorly as the resolution increases. When we double the grid resolution on each axis, we have to evaluate 8× more cells. Additionally, if the deformation is locally linear we do not have to worry about the size of hull triangles. The deformation can change continuously, but we would prefer not having to regenerate the hull every frame, particularly if we would have to run marching cubes at a high resolution. Our insight here is that to reduce the error from linearization, the only factor that matters is the *triangle size* and not the shape of the hull. A simpler solution to reduce the

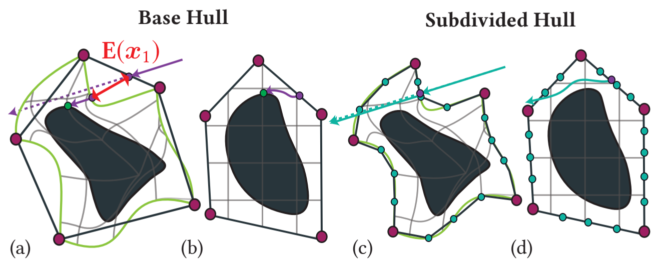


Fig. 7. Using linear interpolation to find $\widetilde{D^{-1}}(\boldsymbol{p}) \approx D^{-1}(\boldsymbol{p})$ assumes that $D(\boldsymbol{p})$ is linear. This is rarely the case in the deformations we discuss here. So for large triangles (a/b), $\widetilde{D^{-1}}(\boldsymbol{p}) \neq D^{-1}(\boldsymbol{p})$ and particularly $D(\widetilde{D^{-1}}(\boldsymbol{p})) \neq \boldsymbol{p}$. This leads to computing the wrong undeformed space ray start points, resulting in surface artifacts in the final image. For example, in (a) the ground truth ray represented by the dashed line does not hit the shape, but the ray NLST traces (b) generates a hit due to error in the start point. When we reduce the size of each linear segment, the error reduces as well and the ray traced by NLST corresponds closely to the ground truth (c/d).

size of triangles is to facet subdivide them. Subdivided triangles lie in the same plane as the original triangle *in undeformed space* and the added vertices are simply deformed by D just as the original vertices were. Furthermore, it is important to note that while other methods [Porumbescu et al. 2005; Taylor et al. 2017] linearize the deformation over the entire *volumetric* space, we only do so on a *2D manifold*. As such, we can more easily quantify the error introduced by linearization as

$$E(\boldsymbol{p}) = \left\| \boldsymbol{p} - D(\widetilde{D^{-1}}(\boldsymbol{p})) \right\|. \tag{8}$$

Hence the error over the surface of a deformed space triangle T is

$$E_T = \int_T E(\boldsymbol{p})\, \mathrm{d}\boldsymbol{p}. \tag{9}$$

As the area of T goes to 0, E_T will vanish since $E(\boldsymbol{p}) = 0$ by definition when $\boldsymbol{p}$ is a vertex of T. The error for any given triangle T can be approximated via numerical integration over Ts surface. To ensure that error stays below a given threshold we subdivide T based on the magnitude of E_T. In practice, this is done via *hardware tessellation* when rendering the hull mesh. This results in an adaptive subdivision scheme that reduces overall linearization error below a perceptible threshold at very little additional cost. We evaluate how the subdivision level affects performance and error in Fig. 10.

4 MAPPING DEFORMATION TECHNIQUES TO NLST

After having described *Non-linear Sphere Tracing* in the last section, we will now discuss what is required of a forward deformation technique to be able to use it in our framework.

One of the strengths of NLST is that it goes to great lengths to avoid computing inverses. This allows us to use any deformation technique for which we can compute D and J_D^{-1} in the space enclosed by the hull mesh. In fact, even if J_D is cumbersome to derive analytically, we found that a numerical approximation (e.g. via forward differences) is sufficient in many cases. We can therefore directly use a large class of deformation techniques by simply specifying the forward transform D, the same transform that is used when deforming an explicit surface. This includes many popular methods like FFDs, regularized Kelvinlets, and linear blend skinning (LBS) with weights defined using analytic distance falloff functions.

Mapping techniques that are not naturally defined as space deformations are slightly less trivial. The most common example of this is LBS with skinning weights defined only at vertices. This is problematic for our technique, because in that context we cannot evaluate D at any point in space

4.1 Linear Blend Skinning for NLST

The deformation D induced by linear blend skinning depends on a weight function W. As mentioned earlier, W is classically defined for each vertex of the mesh we want to deform and hence a *discrete subset* of undefomed space positions. For NLST, we do not know all the positions we will need to evaluate D for a priori. The main issue we have to solve now is to define W for a volumetric subset of undeformed space so we can evaluate it, and therefore the deformation function D, at every point inside the hull. We investigated two approaches, inspired by analytic vs. painted/stored weight maps used in traditional LBS.

Analytic weights. One option is to define bone weights *analytically*. That is, we define $W(\hat{\boldsymbol{x}})$ as some function that is reasonably fast to evaluate analytically and still somewhat user controllable. Since we already have the machinery to edit SDFs, using them as a basis for bone weights is a natural choice. For each bone B, we define an SDF S^B. We can then compute the (unnormalized) weight of this bone as some transformation of the value of S^B. This allows us to define weights automatically based on simple distance falloff functions, while still enabling more detailed manual refinement akin to weight painting, all while requiring very little additional storage.

Tabulated weights. To allow us to leverage the vast literature of more sophisticated, automatic methods for computing skinning weights, we also support tabulated/sampled definitions of W. Here we simply store skinning weights in a regular grid and W is evaluated by interpolating weights inside the relevant grid cell. This is akin to precomputing skinning weights at mesh vertices, though defined volumetrically. One issue is that some automated skinning techniques only compute weights over the surface [Baran and Popović 2007]. Luckily, more recent techniques such as bounded biharmonic weights (BBW) [Jacobson et al. 2011] are already defined volumetrically. While BBW traditionally uses the volumetric data only during pre-computation and discards it at runtime, we simply store the volumetric weights instead. We use the implementation of BBW by Jacobson et al. [2011] which accepts a triangle mesh, computes a tetrahedral mesh and generates weights on the tetrahedral vertices. We use the hull mesh we already use in our method as the input triangle mesh. Once we have the weights defined on tetrahedral vertices we rasterize them using barycentric interpolation to a regular 3D grid.

5 IMPLEMENTATION

To show the practicality of NLST we created a prototype modeling application using the Unity3D game engine [Unity Technologies 2019]. Thanks to the simplicity of our algorithm, the implementation is fairly straightforward and we are able to do all the necessary computations on the GPU. We support a variety of SDF representations but focus on regular grids (interpolated trilinearly or tricubically), SDFs computed procedurally in shader code, and SDFs defined parametrically via analytic brush lists. We generate the hull using the basic marching cubes algorithm in a compute shader at a user-defined grid resolution with grid bounds computed either automatically or by hand, depending on the SDF representation. To avoid transferring any data from the GPU to the CPU, we store the hull geometry in a compute buffer and rasterize it via a procedural draw call. We implemented NLST in the traditional rasterization pipeline using a tessellation shader for the adaptive hull subdivision (Section 3.3). The patch constant function computes the tessellation error over each original hull triangle and the domain shader transforms the resulting vertices from object to deformed space. The pixel shader implements the root finding algorithm, retrieving the undeformed space start point from an interpolated attribute. Once we have computed the undeformed space intersection point using NLST, we can trivially transform it to deformed space using D. We then recompute the camera space depth of this point and use it for depth testing to easily combine sphere traced objects with explicit geometry. We retrieve the surface normal from the SDF via finite differences and transform it to deformed space using the inverse transpose $(J_D^{-1})^\top$. Hence, we can do per fragment shading like we would in a traditional fragment shader. Finally, the shaded color is written to the target render texture and blended with the rest of the scene geometry. While we did not spend significant effort optimizing our implementation, it nonetheless easily runs at real-time frame rates on modern graphics cards.

6 EVALUATION AND RESULTS

We will evaluate NLST and compare it to existing methods for implicit surface deformation. Additionally, we will demonstrate NLST's ability to preserve many of the benefits of SDFs, all while enabling intuitive control over deformation.

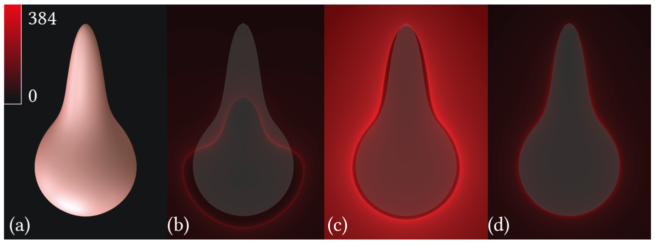


Fig. 8. Above we show the number of SDF evaluations for each pixel in red and the ground truth shape is overlaid in gray on the rendered images. When rendering a strongly deformed object (a), naive non-linear sphere tracing (b) completely fails to reproduce the ground truth shape. Artificially reducing the sphere tracing step size (c) solves this issue, but greatly increases the number of S evaluations (visualized in red). Our sub-steps do not require the evaluation of S, allowing us to accurately reproduce the ground truth at little additional cost (d).

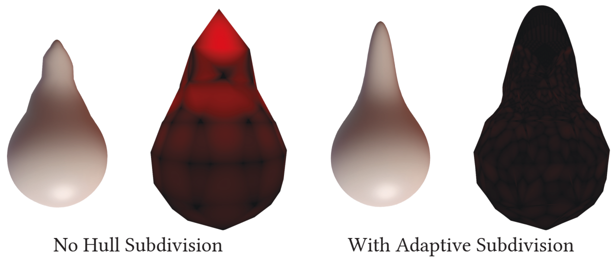


Fig. 9. When we do not subdivide the hull sufficiently, NLST does not reproduce the isosurface faithfully (left). This is due to the error $E(\boldsymbol{p})$, shown in red, in the undeformed space start point (middle left). Once we apply our adaptive subdivision scheme (middle right), error is reduced to imperceptible levels (right).

6.1 Evaluating Methods for Error Control

NLST includes several methods to control error. Figure 8 evaluates the impact of our conservative sub-stepping method. Since we can accurately reconstruct the ray in each unbounding sphere without having to reevaluate S, we maintain good accuracy while preserving performance in both strongly non-linearly deformations regions and in regions of linear deformation. The starting points of rays are approximate, especially for large hull triangles under non-linear deformation. Figure 9 shows the resulting artifacts for large hull triangles, and how this diminishes with our adaptive subdivision scheme. To validate this scheme, in Fig. 10 we rendered the deformed shape from Fig. 9 with a progressively smaller error threshold, showing that as the threshold gets smaller, error reduces as predicted. Notably, because we leverage the GPU's optimized tessellation shader stage, we did *not* measure any increase in render time, even with extremely dense tesselations.

6.2 Comparisons to Related Methods

We compared our method to several existing implicit surface deformation techniques.

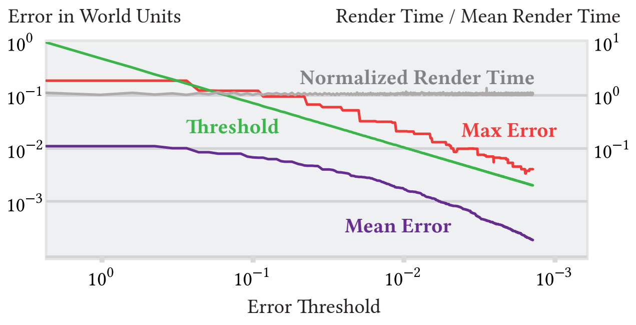


Fig. 10. We evaluate the performance impact and error characteristics of our subdivision method. As we decrease the error threshold, the number of rendered polygons increases. This does not affect render times since fragment shading is the bottleneck of our method. Note that our error heuristic successfully reduces both maximum *and* mean error.

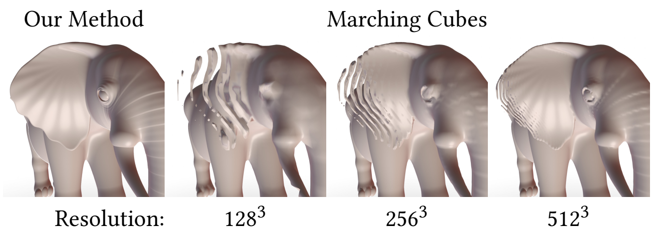


Fig. 11. For many models, running marching cubes at a resolution high enough to accurately reproduce their isosurface is not feasible. Here, the elephant ears are thin and not aligned with grid cells. This results in artifacts, including loss of topology preservation. As with original sphere tracing, our method guarantees the preservation of thin and sharp features. Note that even at the lowest resolution shown here, our method is more than twice as fast as marching cubes (see Table 1).

Isosurface extraction. As a baseline we test how we fare against isosurface extraction via marching cubes. Note that marching cubes is certainly not a state-of-the-art isosurface extraction technique. Here we use it as a representative approach which needs to remesh when the surface changes. Our main goal is to test whether the same algorithm we use to generate the hull could be used at a higher resolution to generate a surface for traditional rendering. We considered faster [Wyvill et al. 1986b] and more accurate meshing techniques [Ju et al. 2002], but they come with trade-offs. For example, dual contouring can produce non-manifold meshes in some situations which causes artifacts during lighting computations. Both approaches, like marching cubes, are also limited in their ability to reproduce small features as well as sharp creases and corners by the chosen grid resolution. Still, one of the advantages of isosurface extraction is that rendering is extremely fast if the isosurface does not change. But in an editing context we want to enable the user to quickly change both the deformation as well as the underlying geometry. Figure 11 shows that we can not rely on marching cubes in this scenario: computation time is simply too high at resolutions that approximate the isosurface with reasonable accuracy since the algorithm scales with the cube of the linear grid resolution. We still leverage marching cubes to construct our hull, but for this we rarely need to use resolutions higher than 32^3. While this would be inadequate if we where to display the hull to the user, it is accurate enough to serve as a starting point for NLST, particularly when combined with our adaptive subdivision scheme.

Table 1. We used an NVIDIA Titan RTX and rendered at 1080p resolution. Timings are complete frame times in milliseconds averaged while rotating the object under changing deformation.

		Marching Cubes on S×S×S grid					
Scene	NLST	S=32	64	128	256	512	1024
Dinosaur	10	6	8	26	150	2075	9689
Head	13	6	9	34	193	2326	12220
Elephant	10	6	8	26	150	1983	9385

Articulated distance fields. Taylor et al. [2017] proposed volumetrically linearizing deformations to enable articulated SDFs. While they applied this for hand tracking, we extend and compare to this approach for rendering. We automatically generate a tetrahedral mesh and sample the deformation at its vertices. This is conceptually similar to shell mapping [Porumbescu et al. 2005], but instead of rendering a thin shell derived from an objects surface, the tetrahedrons cover the whole volume of the object. The robot head in Fig. 12 (a) was constructed via CSG operations on SDFs. Isosurface extraction (b) needs to be dense to accurately reproduce the geometry and small-scale deformations, such as the one forming the nose of the robot. Even when extracted at 256^3 (in 11.2 s) there are still obvious artifacts where the isosurface has sharp features. Taylor et al. [2017] only linearize the *deformation*, preserving sharp *geometric* features. Unfortunately, to reproduce small-scale deformations without introducing discontinuity artifacts, the spatial linearization needs to be high resolution, which is not feasible in real-time rendering. The nose is therefore almost completely lost in (c). While the C^1 discontinuities in the deformation were not a concern for Taylor et al. [2017] in hand tracking, when used for rendering these discontinuities become problematic. Our method (d) is able to reproduce both sharp geometry features as well as small-scale deformations, producing an artifact free image.

6.3 Applying Forward Deformation Techniques

We also map several techniques for explicit surface deformation to our framework.

Free-form deformations & sculpting brushes. First we show the ease of applying FFDs within NLST. While Chen et al. [2001] proposed a method that implicitly defined volumes deformed by FFDs, our approach allows us to apply FFDs in a much more general framework without the need for a specialized approach. Figure 13 shows our implementation of FFDs applied to an implicitly defined head model. Since it is easy to compose different deformation techniques in NLST, we additionally deform using De Goes and James's [2017] sculpting brushes. These are a great fit for NLST because they are described analytically, hence are fast to evaluate on the graphics card, while being sufficiently complex that they do not have an analytic inverse.

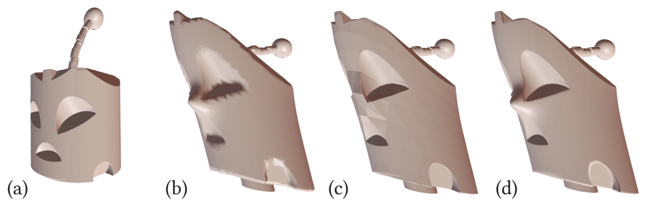


Fig. 12. Starting with an analytically described distance field (a) we apply a forward deformation composed of two Kelvinlet brushes via isosurface extraction (b), linearizing the deformation (c) and our method (d). Our method combines the support for small scale deformations that a dense mesh provides with the accurate reproduction of sharp features typical of direct implicit surface rendering techniques.

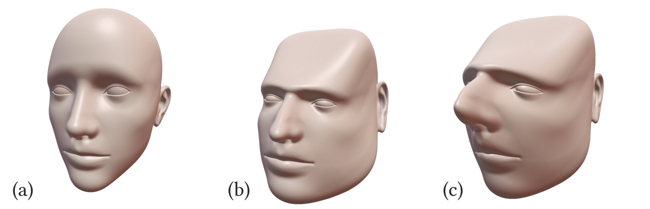


Fig. 13. We apply cubic free form deformation to a signed distance field (a), resulting in a deformed shape (b). A benefit of our method is that combining multiple deformations is trivial. We show this by applying a "Regularized Kelvinlet" [De Goes and James 2017] edit to the result of free form deformation (c).

Linear blend skinning. To show that NLST supports even more complex deformation methods, we implement LBS as discussed in Section 4.1. Figure 14 shows a dinosaur posed in progressively more extreme poses. Note that for the pose shown in Fig. 14 (c), the head and the tail of the character almost meet. In traditional implicit surface deformation techniques this would be practically impossible to reproduce because of the extremely strong space compression it causes. Our hull meshing can robustly provide undeformed space start points relatively close to the isosurface, allowing NLST to handle this case correctly. Finally, we want to point out that we can now apply SDF operations at *any* point. In Fig. 14 (d) we add procedural surface detail *before* deformation (and hence it deforms with the surface) and subtract mass from the character *after* LBS allowing us to interact with the geometry in deformed space.

Volumetric modeling. Lastly we apply NLST to volume rendering. Figure 15 shows a density function derived from the value of S and modified by noise, creating a cloudy appearance. Under deformation, space is stretched and compressed. As pointed out by Stander and Hart [1994], we need to take this into account when rendering our deformed volume and adjust the sampled density accordingly. Luckily this is trivial in NLST, since we compute the Jacobian of the deformation function already.

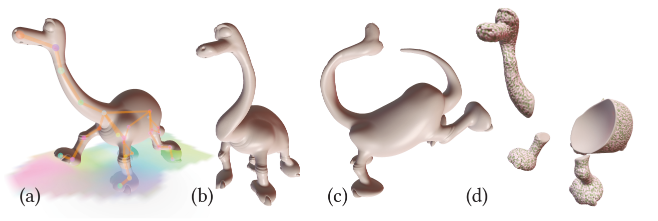


Fig. 14. We generate bounded biharmonic weights [Jacobson et al. 2011] volumetrically based on a user defined skeleton and the hull mesh (a). We can then use the weights and the skeleton to derive D. NLST treats this deformation like any other deformation function and we can pose the character (b). In (c) we show a pose where the head and the tail of the character meet and the deformation is *globally* non-bijective. We can still render this pose with NLST, since *locally* (within the head and within the tail), the deformation is invertible. In (d) we manipulate the underlying SDF before deformation (for texture detail) and after (for geometry manipulation).

7 CONCLUSIONS, LIMITATIONS AND FUTURE WORK

We have presented and evaluated *Non-linear Sphere Tracing* and showed how to use our method to easily incorporate many deformation techniques in an implicit modeling framework. We hope enriching SDFs with these capabilities inspires a broader interest in revisiting implicit representations in settings where explicit representations have become the *de facto* standard.

Of course, our technique is not without downsides. We inherit the benefits, but also the limitations, of all root finding methods, and sphere tracing in particular. Since we have to evaluate the SDFs along the ray, rendering performance tends to be worse than ray tracing or rasterizing explicit surfaces. Especially at grazing angles, intersections are expensive to evaluate and the number of maximum sphere tracing steps has to be limited. This can lead to artifacts for concave objects where NLST runs out of iterations before hitting the surface and a fragment is discarded mistakenly. Keinert et al. [2014] introduce improvements to traditional sphere tracing to alleviate this issue. These could be applied to NLST as well.

Moreover, sphere tracing is only applicable to SDFs, so we cannot deform more general implicits for which efficient-to-evaluate distances/distance bounds are unavailable. While we technically do support any implicit surface as long as its gradient is less than 1, the performance of sphere tracing rapidly degrades under those conditions. Being able to support more general implicit surfaces in our framework is an interesting avenue for future work.

Due to our implicit workflow, we never "apply" or bake our deformations, as you could on explicit vertices. While this creates a non-destructive editing/deformation workflow, it also means we have to pay the performance penalty for deformations during rendering which can degrade when layering many deformations. While we support editing the surface after one set of deformations as shown in Fig. 14, we would like to investigate supporting an arbitrary number of deformation/surface editing layers. An additional downside is that our method does not support the evaluation of the implicit function corresponding to the deformed surface at an arbitrary point in deformed space, even just due to the fact that we support self-intersecting geometry which classical implicit function representations do not. This limits NLST's applicability to problems such as collision detection. Still, in the supplemental video we do show a particle simulation that efficiently collides particles with a deformed SDF, but extra care has to be taken to track particle positions in undeformed space.

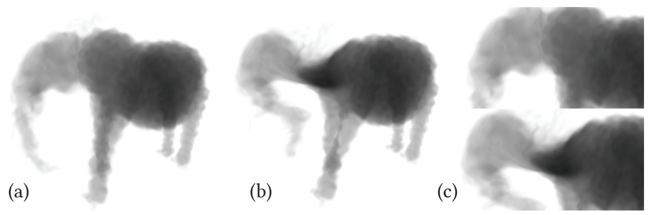


Fig. 15. We deform a volume with a density function derived from an SDF with added noise (a). NLST correctly accounts for volume compression and expansion, making compressed areas darker (b,c).

In Section 3 we pointed out that to trace through some region of space, we need to be able to compute J_D^{-1} for points along the ray. This imposes one restriction on the particular instances of deformations that we support, namely, they have to be *locally foldover-free*. In future work, it would be interesting to try to leverage recent work on immersions of self-intersecting solids to allow overlaps [Li and Barbič 2018].

ACKNOWLEDGMENTS

We thank Alex Evans, Milan Grajetzki and the members of the Dartmouth Visual Computing Lab for helpful discussions, and the reviewers for their insightful feedback. The dinosaur and elephant model, created by Iniqo Quilez, as well as the human head model, created by Thomas Hooper, are shared under the Creative Commons Attribution-NonCommercial-ShareAlike license. The robot head model was created by Milan Grajetzki. This work was partially supported by the National Science Foundation (Grant 1844538) and the Canada Research Chairs program.

REFERENCES

Chandrajit Bajaj, Jim Blinn, Brian Wyvill, Marie-Paule Cani, Alyn Rockwood, and Geoff Wyvill. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann.

Csaba Bálint and Gábor Valasek. 2018. Accelerating Sphere Tracing. *Proceedings of Eurographics Short Papers* (2018), 4 pages. https://doi.org/10/gfz542

Ilya Baran and Jovan Popović. 2007. Automatic Rigging and Animation of 3D Characters. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 26, 3 (July 2007), 72. https://doi.org/10/d2ck5v

Michael F. Barnsley, Robert L. Devaney, Benoit B. Mandelbrot, Heinz-Otto Peitgen, Dietmar Saupe, Richard F. Voss, Yuval Fisher, and Michael McGuire. 1988. *The Science of Fractal Images* (1st ed.). Springer-Verlag. https://doi.org/frdznz

Alan H. Barr. 1984. Global and Local Deformations of Solid Primitives. *Computer Graphics (Proceedings of SIGGRAPH)* 18, 3 (July 1984), 21–30. https://doi.org/10/fcwvgw

Alan H. Barr. 1986. Ray Tracing Deformed Surfaces. *Computer Graphics (Proceedings of SIGGRAPH)* 20, 4 (Aug. 1986), 287–296. https://doi.org/10/cpqr6g

Thaddeus Beier and Shawn Neely. 1992. Feature-Based Image Metamorphosis. *Computer Graphics (Proceedings of SIGGRAPH)* 26, 2 (July 1992), 35–42. https://doi.org/10/crjpph

M. Berger, T. Trout, and N. Levit. 1990. Ray Tracing Mirages. *IEEE Computer Graphics & Applications* 10, 3 (May 1990), 36–41. https://doi.org/10/cfbfc3

James F. Blinn. 1982. A Generalization of Algebraic Surface Drawing. *Computer Graphics (Proceedings of SIGGRAPH)* 16, 3 (July 1982), 273. https://doi.org/10/fgvzkf

P. Bogacki and L.F. Shampine. 1989. A 3(2) Pair of Runge-Kutta Formulas. *Applied Mathematics Letters* 2, 4 (1989), 321 – 325. https://doi.org/10/cwcdkx

Brinx Software. 2019. MasterpieceVR. https://www.masterpiecevr.com/

John Charles Butcher and Nicolette Goodwin. 2008. *Numerical methods for ordinary differential equations*. Vol. 2. Wiley Online Library. https://doi.org/10/fhv3h9

M. Cani-Gascuel and M. Desbrun. 1997. Animation of Deformable Models Using Implicit Surfaces. *IEEE Transactions on Visualization and Computer Graphics* 3, 1 (Jan. 1997), 39–50. https://doi.org/10/c6bqg2

Chen Cao, Zhong Ren, Baining Guo, and Kun Zhou. 2010. Interactive Rendering of Non-Constant, Refractive Media Using the Ray Equations of Gradient-Index Optics. *Computer Graphics Forum* 29, 4 (2010), 1375–1382. https://doi.org/10/fbff4n

Huawei Chen, Jürgen Hesser, and Reinhard Männer. 2001. Fast Volume Deformation Using Inverse-Ray-Deformation and FFD. In *GraphiCon*.

Brian Curless and Marc Levoy. 1996. A Volumetric Method for Building Complex Models from Range Images. In *Annual Conference Series (Proceedings of SIGGRAPH)*. ACM Press, New York, NY, USA, 303–312. https://doi.org/10/crn3vr

Fernando De Goes and Doug L. James. 2017. Regularized Kelvinlets: Sculpting Brushes Based on Fundamental Solutions of Elasticity. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 36, 4 (July 2017), 40:1–40:11. https://doi.org/10/gfz56k

Mathieu Desbrun and Marie-Paule Gascuel. 1995. Animating Soft Substances with Implicit Surfaces. In *Annual Conference Series (Proceedings of SIGGRAPH)*. ACM, New York, NY, USA, 287–290. https://doi.org/10/b96ndx

J. R. Dormand and P. J. Prince. 1980. A Family of Embedded Runge-Kutta Formulae. *J. Comput. Appl. Math.* 6, 1 (March 1980), 19–26. https://doi.org/10/cfw5fc

David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Kenneth Perlin, and Steven Worley. 2003. *Texturing and modeling: a procedural approach* (3rd ed.). Morgan Kaufmann, San Francisco, CA, USA.

Alex Evans. 2015. Learning from Failure: a Survey of Promising, Unconventional and Mostly Abandoned Renderers for "Dreams PS4", a Geometrically Dense, Painterly UGC Game. *ACM SIGGRAPH Course Notes*, Article 2 (2015). https://doi.org/10/gf2v8v

Facebook Technologies. 2019. Oculus Medium. https://www.oculus.com/medium/

E. Fehlberg. 1970. Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme. *Computing* 6, 1 (March 1970), 61–71. https://doi.org/10/cc7qv5

Takushi Fujita, Katsuhiko Hirota, and Kouichi Murakami. 1990. Representation of splashing water using metaball model. *Fujitsu* 41, 2 (1990), 159–165. (in Japanese).

Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T. Freeman, and Thomas Funkhouser. 2019. Learning Shape Templates with Structured Implicit Functions. *arXiv:1904.06447 [cs]* (April 2019). arXiv:cs/1904.06447

Olivier Gourmel, Loic Barthe, Marie-Paule Cani, Brian Wyvill, Adrien Bernhardt, Mathias Paulin, and Herbert Grasberger. 2013. A Gradient-based Implicit Blend. *ACM Transactions on Graphics* 32, 2 (2013), 12. https://doi.org/10/gf6wk7

Eduard Gröller. 1995. Nonlinear Ray Tracing: Visualizing Strange Worlds. *The Visual Computer* 11, 5 (May 1995), 263–274. https://doi.org/10/ffcq74

Diego Gutierrez, Adolfo Muñoz, Oscar Anson, and Francisco J. Seron. 2005. Non-Linear Volume Photon Mapping. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*. Eurographics Association, 291–300. https://doi.org/10/gfzngk

J.C. Hart, E. Bachta, W. Jarosz, and T. Fleury. 2002. Using Particles to Sample and Control More Complex Implicit Surfaces. In *Shape Modeling International*. https://doi.org/10/dfw2ss

John C. Hart. 1996. Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer* 12, 10 (Dec. 1996), 527–545. https://doi.org/10/b3q2p6

S. Ilic and P. Fua. 2006. Implicit Meshes for Surface Reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 2 (Feb. 2006), 328–333. https://doi.org/10/ctgm5g

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 30, 4 (July 2011), 78:1–78:8. https://doi.org/10/ckcmsj

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. 2014. Skinning: Real-Time Shape Deformation. In *ACM SIGGRAPH Course Notes*. https://doi.org/10/gf2ng4

Oliver James, Eugénie von Tunzelmann, Paul Franklin, and Kip S. Thorne. 2015. Gravitational Lensing by Spinning Black Holes in Astrophysics, and in the Movie Interstellar. *Classical and Quantum Gravity* 32, 6 (Feb. 2015), 065001. https://doi.org/10/gdvj4r

Stefan Jeschke, Stephan Mantler, and Michael Wimmer. 2007. Interactive Smooth and Curved Shell Mapping. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*, Jan Kautz and Sumanta Pattanaik (Eds.). The Eurographics Association, 351–360. https://doi.org/10/gfz557

M. W. Jones, J. A. Baerentzen, and M. Sramek. 2006. 3D Distance Fields: A Survey of Techniques and Applications. *IEEE Transactions on Visualization and Computer Graphics* 12, 4 (July 2006), 581–599. https://doi.org/10/bwnmjs

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic Coordinates for Character Articulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 26, 3 (July 2007). https://doi.org/10/bqj5jk

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual Contouring of Hermite Data. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 21, 3 (July 2002), 339–346. https://doi.org/10/bdg3sp

Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. 2013. Improved Ray Casting of Procedural Distance Bounds. *Journal of Graphics Tools* 17, 4 (Oct. 2013), 127–138. https://doi.org/10/gfz54s

Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. 2014. Enhanced Sphere Tracing. In *STAG: Smart Tools & Apps for Graphics*. 8. https://doi.org/10/gfz549

A. Knoll, Y. Hijazi, C. Hansen, I. Wald, and H. Hagen. 2007. Interactive Ray Tracing of Arbitrary Implicits with SIMD Interval Arithmetic. In *Proceedings of IEEE Symposium on Interactive Ray Tracing*. 11–18. https://doi.org/10/fkxrdv

A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. 2009. Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic. *Computer Graphics Forum* 28, 1 (2009), 26–40. https://doi.org/10/d5s7kh

Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. 2001. Feature Sensitive Surface Extraction from Volume Data. In *Annual Conference Series (Proceedings of SIGGRAPH) (SIGGRAPH '01)*. ACM, New York, NY, USA, 57–66. https://doi.org/10/cbh7f9

Johann Korndorfer. 2015. The Timeless Way of Building Geometry - How to create content with Signed Distance Functions. https://www.youtube.com/watch?v=s8nFqwOho-s

Dan Koschier, Crispin Deul, and Jan Bender. 2016. Hierarchical Hp-Adaptive Signed Distance Fields. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Computer Animation*. Eurographics Association, Goslar Germany, Germany, 189–198.

Yair Kurzion and Roni Yagel. 1995. Space Deformation Using Ray Deflectors. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, Patrick M. Hanrahan and Werner Purgathofer (Eds.). Springer-Verlag, 21–30. https://doi.org/10/gfz54w

Yijing Li and Jernej Barbič. 2018. Immersion of Self-Intersecting Solids and Surfaces. *ACM Transactions on Graphics* 37, 4 (2018). https://doi.org/10/gd52q5

William E. Lorensen and Harvey E. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proceedings of SIGGRAPH)* 21, 4 (Aug. 1987), 163–169. https://doi.org/10/ft9gsh

Media Molecule. 2019. Dreams PS4. https://www.mediamolecule.com/games/dreams

Don Mitchell. 1990. Robust Ray Intersection with Interval Arithmetic. In *Proceedings of Graphics Interface*, Vol. Halifax. 68–74. https://doi.org/10/gfz56m

Fabrice Neyret. 1996. *Local Illumination in Deformed Space.* Technical Report RR-2856. INRIA.

Stanley Osher and James A Sethian. 1988. Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. *J. Comput. Phys.* 79, 1 (Nov. 1988), 12–49. https://doi.org/10/cq9w6r

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. *arXiv:1901.05103 [cs]* (Jan. 2019). arXiv:cs/1901.05103

A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. 1995. Function Representation in Geometric Modeling: Concepts, Implementation and Applications. *The Visual Computer* 11, 8 (Aug. 1995), 429–446. https://doi.org/10/fsqzrw

Ken H. Perlin and Eric M. Hoffert. 1989. Hypertexture. *Computer Graphics (Proceedings of SIGGRAPH)* 23, 3 (July 1989), 253–262. https://doi.org/10/fdmsxd

Serban D. Porumbescu, Brian Budge, Louis Feng, and Kenneth I. Joy. 2005. Shell Maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 24, 3 (July 2005), 626. https://doi.org/10/d4bh4g

Tim Reiner, Gregor Mückl, and Carsten Dachsbacher. 2011. Interactive Modeling of Implicit Surfaces Using a Direct Visualization Approach with Signed Distance Functions. *Computers & Graphics* 35, 3 (June 2011), 596–603. https://doi.org/10/fsnj24

Kenneth B. Russell. 1999. *IMPS: Implicit Surfaces for Interactive Animated Characters.* Masters Thesis. Massachusetts Institute of Technology.

Tetsu R. Satoh. 2003. Symplectic Ray Tracing: A New Approach to Non-Linear Ray Tracing by Using Hamiltonian Dynamics. In *Visualization and Data Analysis*, Vol. 5009. International Society for Optics and Photonics, 277–286. https://doi.org/10/fr5tg6

Tanner Schmidt, Richard Newcombe, and Dieter Fox. 2014. DART: Dense Articulated Real-Time Tracking. In *Robotics: Science and Systems*, Vol. 2. Robotics: Science and Systems Foundation. https://doi.org/10/gf2dr2

Thomas W. Sederberg and Scott R. Parry. 1986. Free-Form Deformation of Solid Geometric Models. *Computer Graphics (Proceedings of SIGGRAPH)* 20, 4 (Aug. 1986), 151–160. https://doi.org/10/cb8rr3

F. J. Seron, D. Gutierrez, G. Gutierrez, and E. Cerezo. 2004. Visualizing Sunsets through Inhomogeneous Atmospheres. In *Proceedings of Computer Graphics International (CGI)*. 349–356. https://doi.org/10/fg79hz

J. A. Sethian and Peter Smereka. 2003. Level Set Methods for Fluid Interfaces. *Annual Review of Fluid Mechanics* 35, 1 (2003), 341–372. https://doi.org/10/ffqv25

Miroslava Slavcheva, Maximilian Baust, and Slobodan Ilic. 2017. Towards Implicit Correspondence in Signed Distance Field Evolution. In *Proceedings of the International Conference on Computer Vision (ICCV)*. https://doi.org/10/c935

J. Sloup. 2003. Visual Simulation of Refraction Phenomena in the Earth's Atmosphere. In *Proceedings on Seventh International Conference on Information Visualization (IV)*. 452–457. https://doi.org/10/czt7cs

Jos Stam and Eric Languénou. 1996. Ray Tracing in Non-Constant Media. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, Xavier Pueyo and Peter Schröder (Eds.). Springer-Verlag, 225–234.

Barton T. Stander and John C. Hart. 1994. A Lipschitz Method for Accelerated Volume Rendering. In *Proceedings of the 1994 Symposium on Volume Visualization (VVS '94)*. ACM, New York, NY, USA, 107–114. https://doi.org/10/dxj3vz

Masamichi Sugihara, Brian Wyvill, and Ryan Schmidt. 2010. WarpCurves: A Tool for Explicit Manipulation of Implicit Surfaces. *Computers & Graphics* 34, 3 (June 2010), 282–291. https://doi.org/10/dqnmqj

Jonathan Taylor, Vladimir Tankovich, Danhang Tang, Cem Keskin, David Kim, Philip Davidson, Adarsh Kowdle, and Shahram Izadi. 2017. Articulated Distance Fields for Ultra-Fast Tracking of Hands Interacting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 36, 6 (Nov. 2017), 244:1–244:12. https://doi.org/10/gcqbht

Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. 2017. Learning Shape Abstractions by Assembling Volumetric Primitives. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 1466–1474. https://doi.org/10/gfz56d

Greg Turk and James F. O'Brien. 1999. *Variational Implicit Surfaces.* Technical Report GIT-GVU-99-15. Georgia Institute of Technology.

Greg Turk and James F O'Brien. 2005. Shape Transformation using Variational Implicit Functions. In *ACM SIGGRAPH Course Notes*. ACM, 13. https://doi.org/10/b6hfjf

Unbound Technologies. 2019. Unbound. http://unbound.io/

Unity Technologies. 2019. Unity3D. https://unity.com/

Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. 2013. Implicit Skinning: Real-Time Skin Deformation with Contact Modeling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 32, 4 (July 2013), 125:1–125:12. https://doi.org/10/gfz54q

Rodolphe Vaillant, Gäel Guennebaud, Loïc Barthe, Brian Wyvill, and Marie-Paule Cani. 2014. Robust Iso-Surface Tracking for Interactive Character Skinning. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 33, 6 (Nov. 2014), 189:1–189:11. https://doi.org/10/gfz54r

Andrew P. Witkin and Paul S. Heckbert. 1994. Using Particles to Sample and Control Implicit Surfaces. In *Annual Conference Series (Proceedings of SIGGRAPH)*. ACM, New York, NY, USA, 269–277. https://doi.org/10/bv24kc

Brian Wyvill, Andrew Guy, and Eric Galin. 1998. *The Blob Tree- Warping, Blending and Boolean Operations in an Implicit Surface Modeling System.* Technical Report. University of Calgary. https://doi.org/gfz57d

Brian Wyvill, Andrew Guy, and Eric Galin. 1999. Extending the CSG Tree: Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum* 18, 2 (1999), 149–158. https://doi.org/10/ffd743

Brian Wyvill, Craig McPheeters, and Geoff Wyvill. 1986a. Animating Soft Objects. *The Visual Computer* 2, 4 (Aug. 1986), 235–242. https://doi.org/10/ct7psx

Geoff Wyvill, Craig McPheeters, and Brian Wyvill. 1986b. Data Structure for Soft Objects. *The Visual Computer* 2, 4 (Aug. 1986), 227–234. https://doi.org/10/dndmwc

Geoff Wyvill and Andrew Trotman. 1990. Ray-Tracing Soft Objects. In *Proceedings of Computer Graphics International (CGI)*, Tat-Seng Chua and Tosiyasu L. Kunii (Eds.). Springer Japan, 469–476.