Geometric Texturing Using Level Sets

Anders Brodersen, Ken Museth, *Member*, *IEEE*,
Serban Porumbescu, and Brian Budge, *Student Member*, *IEEE*

Abstract—We present techniques for warping and blending (or subtracting) geometric textures onto surfaces represented by high-resolution level sets. The geometric texture itself can be represented either explicitly as a polygonal mesh or implicitly as a level set. Unlike previous approaches, we can produce topologically connected surfaces with smooth blending and low distortion. Specifically, we offer two different solutions to the problem of adding fine-scale geometric detail to surfaces. Both solutions assume a level set representation of the base surface, which is easily achieved by means of a mesh-to-level-set scan conversion. To facilitate our mapping, we parameterize the embedding space of the base level set surface using fast particle advection. We can then warp explicit texture meshes onto this surface at nearly interactive speeds or blend level set representations of the texture to produce high-quality surfaces with smooth transitions.

Index Terms—Geometric texture mapping, parameterization, implicit surfaces, volume texturing, geometric modeling.

1 Introduction

We present a novel 3D fine-scale explicit and implicit geometry mapping technique based on level sets, interpolation, and radial basis functions (Fig. 1). Our work is motivated by the need to easily model fine geometric detail in a convenient fashion. For years, the standard approaches to increase geometric complexity have primarily been 2D texture [1], bump [2], and displacement mapping [3]. These techniques, while capturing a wide range of geometric phenomena, are limited in the types of detail they can represent. Kajiya and Kay [4] realized this early on and introduced volumetric textures to represent more topologically complex structures. Recently, the focus has shifted toward more sophisticated volumetric and geometric texturing approaches in an effort to capture a wider range of complex geometric phenomena [5], [6], [7], [8].

Our contribution leverages the recent introduction of DT-Grid data structures and algorithms [9] and the large body of level set research to bridge the gap between existing volumetric and explicit geometric mapping techniques. This is achieved by providing a fast geometric mapping suitable for modeling and previewing, as well as an implicit mapping approach that complements existing explicit mapping techniques (for example, [8]) by generating closed smoothly blended surfaces. Our general approach uses an implicit level set representation of both the base surface and the texture geometry [10], [11], [12] (Fig. 2). This representation allows for robustness to topology changes during the mapping, flexibility when defining the blend of the base and texture geometry, and is amenable to high quality offset surface generation (see Fig. 15 for a comparison of implicit versus explicit offset surfaces). Additionally, level sets offer a large body of advanced numerical techniques for easily computing surface properties and performing arbitrary deformations. In fact, as has been shown in previous work [13], direct control of blended surface properties is easily achievable with level sets. This high degree of robustness and flexibility, however, comes at the price of increased computational complexity when compared to purely explicit approaches. To address this issue, we have also developed a fast *semiimplicit* technique that can conveniently be used for near real-time previewing. It combines an implicit level set representation of the base surface and an explicit polygonal representation of the textures.

We assume that we are given a base surface as a compact level set and a geometric texture either defined by a triangle mesh or as a compact level set surface. If required, conversion between triangle meshes and level sets can be performed using a fast scan-conversion technique [14] or marching cubes [15]. Given this geometry, our system works as follows:

- First, the user manually outlines a patch on the base level set, which defines the location of the geometric textures. Given such a patch outline, we then construct a parameterization of the space above the patch. This effectively creates the mapping needed to warp the texture into the space near our base level set surface. The process of defining the patch and the creation of the parameterization is described in Section 2.
- Section 3 then presents a simple procedure that maps the texture mesh onto the base level set at nearly interactive rates.
- Alternatively, to produce a single topologically connected surface with smooth blending between

- *A. Brodersen is with the Department of Computer Science, University of Aarhus, IT-Parken, Aabogade 34, DK-8200 Aarhus N, Denmark. E-mail: rip@daimi.au.dk.*
- *K. Museth is with Lindköping University, Sweden and the University of Aarhus, Denmark, Digital Domain, 300 Rose Avenue, Venice, CA 90292. E-mail: museth@acm.org.*
- *S. Porumbescu and B. Budge are with the University of California, Davis, 1 Shields Ave., Davis, CA 95616. E-mail: sdporumbescu@ucdavis.edu, budge@cs.ucdavis.edu.*

Manuscript received 22 Aug. 2006; revised 27 Jan. 2007; accepted 7 May 2007; published online 4 June 2007.
Recommended for acceptance by R. Fedkiw.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0134-0806.
Digital Object Identifier no. 10.1109/TVCG.2007.70408.

Fig. 1. Carving patterns into an irregular surface by subtracting a geometric texture using the proposed technique.

the texture and base, the user can utilize a higher quality implicit mapping. This is the topic of Section 4.

1.1 Contributions

The techniques presented in this paper include the following:

Implicit geometry mapping with smooth blending. We complement existing explicit geometry mapping techniques by using an implicit approach, which smoothly blends mapped geometry to create closed surfaces suitable for rendering and various surface property computation. We do this using compact level set representations of the base and texture surfaces. It is the first general texture space to shell space mapping technique utilizing implicits that we are aware of.

Fast semi-implicit geometry mapping. We also introduce a near real-time *semiimplicit* mapping approach that combines an implicit level set representation of the base surface with an explicit polygonal representation of mapped textures. This technique is useful as a preview tool (prior to implicit mapping) and as a stand alone method for mapping explicit geometry.

Flexible volumetric parameterization. We compute a low-distortion parameterization with a minimum of user interaction. Our parameterization is not dependent on prior surface texture coordinates. Instead, it is based on a local parameterization generated on the fly, using a simple and easy to use point and click interface. Furthermore, our parameterization is characterized by the distribution of a set of particles, but is independent of the algorithm used to distribute these particles. This means that the particles can be distributed in a number of different ways, allowing for a vast number of unique mapping effects. Finally, we include results from a simple free-form variation of our mapping technique where the texture warping is derived and controlled by a deformable spline curve.

1.2 Related Work

Our work builds on level set, implicit surface modeling, and volumetric and geometric texture research. A recent body of

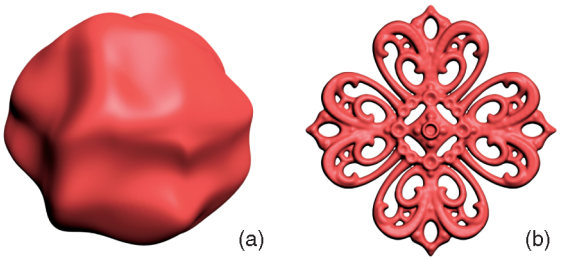


Fig. 2. (a) Base geometry and (b) texture geometry used to create Fig. 1.

work proposing various compact data structures and fast algorithms for level set models [9], [16], [17], [18] is critical to our work. We have chosen to base our texture mapping technique on the "Dynamic Tubular Grid" (DT-Grid) presented in [9]. This data structure has been shown to be very CPU and memory efficient and allows us to represent level set models of effective resolutions exceeding $1{,}000^3$ using less then 100 Mbytes.

Much effort has been put into deriving methods for adding textures to unparameterized 3D models, specifically implicit surfaces and level sets; these include vector field driven texture synthesis [19] and methods based on parameterizations of support surfaces of lower geometric complexity [20], [21], [22]. These methods generally lack flexibility and user control. Pedersen [23] presented an interactive method to create a parameterization of implicit surfaces by letting the user manually divide the surface into rectangular and triangular *texture patches*. This method has generally been considered state of the art since its publication in 1995. Recently, Schmidt et al. presented a local parameterization based on discrete exponential maps (DEM) [24], producing a simple yet powerful interface for texturing implicit surfaces, provided only that a local parameterization is required.

Kajiya and Kay introduced the notion of volumetric textures [4]. Their method utilizes volumetric data sampled on a regular grid and traces rays through a shell volume on a surface. Rays that intersect the shell are transformed to texture space and traced through the sampled data grid. Material properties were constrained across any region. Neyret extended volume textures, allowing the use of multiple different materials in a single region and objects of different types to be tiled onto a surface [25]. Wang et al. presented a generalization of displacement maps. For each location in a grid surrounding the base surface, a distance is computed to the geometric texture, called the mesostructure, for some discretization of all directions. Several other variables are precomputed for rendering, including BRDF information and local shadows [6]. Peng et al. [26] averaged distance field functions to generate offset surfaces. Then, 3D volumes are sliced into 2D textures, and the textures are applied to various levels of the offset surface. The technique allowed interactive rendering of the resultant volume. All of these techniques map geometry by first 3D scan converting models into a regular grid, which leads to data storage and aliasing related issues.

Fleischer et al. proposed to use a cellular-texturing technique to produce organic looking surface details [27].

Although producing impressive results, their modeling approach is not very intuitive due to a rather complicated underlying biologically motivated simulation engine. Bhat et al. demonstrated a volumetric extension of the image analogies technique [7]. This allowed them to tile a surface with semirepeatable patterns at high effective resolution. The patterns do not need to be height fields and can represent complex structures on the surface.

Recently, Shell Maps [8] generalized the notion of volumetric textures by mapping explicit geometry without converting models into regular grids. Shell maps are invertible mappings between texture space and shell space—the space near an object—that facilitate the transfer of explicit geometry, procedural functions, and scalar fields as fine scale detail near an object. The approach generates a correspondence between texture space and shell space via a tetrahedral tiling. Point location queries coupled with barycentric mappings between corresponding tetrahedra are used to transform objects between spaces. The technique is powerful, but the resultant mappings are only C^0 continuous at tetrahedral boundaries and can create artifacts like the one shown in Fig. 12b. Furthermore, the mapped geometry and the base mesh do not create a new closed mesh, which can be problematic for applying shaders over the entire surface. The level set approach presented in this paper complements the explicit geometry representation of Shell Maps by more naturally dealing with sharp discontinuities and changes in topology necessary to generate closed surfaces (when desired).

We present a novel technique for the mapping of geometry onto surfaces. Although sharing some conceptual similarities with other methods that map 2D textures (for example, images, bump, or displacement maps) and 3D textures (that is, volumetric and geometric) onto meshes there are some significant differences. Our technique can map explicit geometry but can also treat geometry implicitly, which allows us to create closed continuous meshes (topological 2-manifolds). This nice property allows us to define important surface properties like normals and curvatures on the resulting surface. The method requires no surface-wide parameterization, and our local parameterization only requires the user to select the region where they want to map their geometry.

1.3 Notation

As a prelude to a more detailed discussion of our techniques, we shall briefly introduce some terminology. In this paper, the term geometry is used interchangeably for both explicit meshes and implicit level sets. Assume we wish to map a geometric texture A onto a base surface B. We shall denote the explicit mesh representation of A as M_A and the implicit level set representation by ϕ_A. The geometric representation of B is always implicit and will therefore be denoted as ϕ_B. The embedding space of A (for example, defined from its bounding box) will be called *texture space*. The corresponding embedding space of A, after it has been mapped onto B, is called *patch space* (analogous to a portion of "shell space" in [8]). The semi-implicit texture mapping then simply works by defining a map of vertices of M_A from texture space to patch space. In contrast, the implicit texture mapping is based on a resampling of ϕ_A into patch space, which amounts to establishing a map from grid points in patch space to texture space. Thus, both techniques are based on defining a mapping between the two embedding spaces, but in different directions (see Fig. 3).

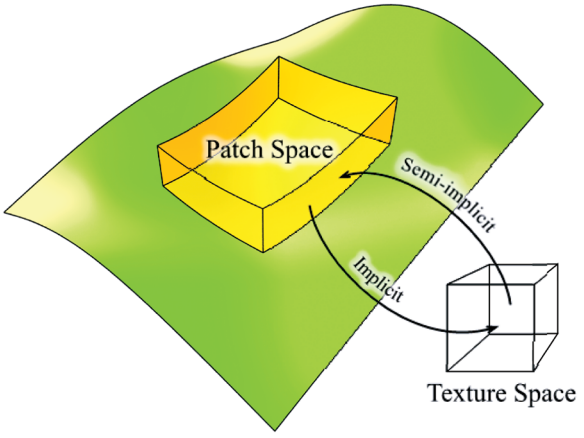


Fig. 3. The **semi-implicit** method uses a direct correspondence between grid points in patch space and texture space. For a given point $\mathbf{x}_t$, the corresponding eight surrounding points in texture space are found. Weights are computed from these eight points, and the weights are applied in a trilinear interpolation in patch space. The **implicit** method uses the correspondence between points in patch space and texture space to solve for the weights of a radial basis function. Patch space can then be sampled with $\mathbf{x}_p$s, finding corresponding points in texture space using the radial basis function. Finally, a trilinear interpolation on the texture volume is used to find the distance value.

2 Parameterizing Patch Space

Although the volumetric parameterization of texture space is assumed known (for example, $u = x$, $v = y$, and $w = z$), we have to derive the warped parameterization of the corresponding patch space. For this, we have developed a number of techniques, based on an initial u, v parameterization of a 2D patch of the base surface and using Lagrangian tracker particles to sweep out u, v, and w in the corresponding patch space. The specific distribution of these tracker particles is created in different ways thereby offering distinct features such as following the base surface faithfully or lowering distortion for the resulting 3D texture mapping. This flexibility is one of the strengths of our system. In the following, we describe the common base for our current particle distribution methods.

A common initial step for these mapping techniques is the definition and parameterization of a 2D patch on the base surface where the texture is to be applied. We define this patch as a simple control quadrilateral[1] on ϕ_B. Constrained interaction with the vertices, $V_i, i = 1 \ldots 4$, of this control quadrilateral is easily implemented since projections of V_i onto ϕ_B amounts to the closest point transform $V_i - \phi_B(V_i)\nabla\phi_B(V_i)$. This is a consequence of our requirement that the level set ϕ_B must be represented by a signed distance function. This control quadrilateral is parameterized using a technique similar to Pedersen's [23]. In short, approximate geodesics are first computed

1. Note that this is not a regular planar quadrilateral since the edges are constrained to lie on the base surface.

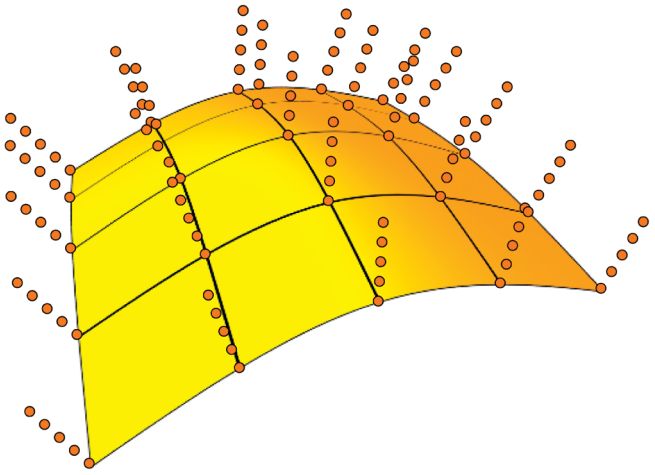


Fig. 4. The surface conforming parameterization propagates the (u, v) texture coordinates using a Lagrangian advection method. The particles roughly follow the normal direction, and the time of arrival is used as the third texture coordinate w.

between the vertices V_1 and V_2, V_2 and V_3, V_3 and V_4, and V_1 and V_4. These edges are then subdivided evenly with a resolution determined by the roughness of the surface[2] and assigned u, v coordinates. Next, u and v are swept into the interior of the quadrilateral by means of defining a 2D grid of isoparametric curves of approximate geodesics connecting the subdivided edges with each curve corresponding to a unique u or v value. At each of the grid points of this 2D isoparametric grid, we place a Lagrangian tracker particle, that is, an infinitely small and massless particle, each associated with a unique u, v, w coordinate. The u and v values are obtained from the two curves intersecting at that point, and the w value is set to zero. In the following, we refer to these Lagrangian tracker particles as patch particles or just particles. The position of the patch particles are then optimized to reduce texture distortion. This is achieved by means of a simple constrained mass-spring model [28], where particles on the boundary curves of the patch quadrilateral are fixed, and the remaining interior particles are restricted to lie on the base surface.

Surface conforming parameterization. Once the patch particles are generated on the base surface, we propagate them along the gradient field of ϕ_B until they reach a desired offset (that is, level of ϕ_B). The w texture coordinate for the advected particles is then defined to be 1. In the case of the implicit mapping described in Section 4, it is often necessary to have intermediate layers of particles with $0 < w < 1$. This is obtained by distributing a number of particles evenly on the line segment between each advected particle and its corresponding particle on the surface using linear interpolation to determine the w value. Fig. 4 illustrates the particle set distributed for a single patch using this method. Note that even though ϕ_B is defined as a signed distance function, two particles with the same w coordinate will generally not lie at the same distance away from B (unless $w = 0$). This is a consequence of the fact that the gradients are, strictly speaking, not defined at points that have more than one closest point transform to B since here, ϕ_B is only C^0. This occurs along the medial-axis of B and numerically manifests itself as $|\nabla\phi_B| \leq 1$ when using central finite differences to compute the gradient. This has the desired feature that although the advected particles might reach other particles, they will never cross paths.[3] As the particles generated by this method are generally not uniformly distributed in patch space, this can lead to significant distortion of the geometric texture. We note that depending on the application, this may or may not be a desirable feature.

By distributing the tracker particles, as outlined above, we end up with a mapping that essentially resembles shell mapping [8]. Consequently, this distribution scheme is hampered by most of the limitations of Shell Maps, in particular, the sensitivity of the mapping with respect to the curvature of the base surface (see Section 5). However, one of the main strengths of our method is the flexibility with respect to distributing the tracker particles. We next present two alternative particle distribution schemes that offer different and improved properties of the resulting geometric texture mapping.

Reduced distortion level set parameterization. The problem with the previous particle distribution method is the (implicit) dependence of the curvature of the base surface. As the tracker particles are advected away from the surface in a direction normal to the surface, small irregularities in the surface can cause severe distortion of the texture. This is due to the fact that particles will typically move closer together in concave regions and away from each other in convex regions. To mend this, we introduce a particle distribution scheme with a stronger focus on the vertices of the user specified control quadrilateral. With this method, these vertices are the only particles to be offset along the gradient field of ϕ_B. At regular intervals, derived from the desired offset height and the desired number of particle levels, a new level of particles is created from the four advected control vertices. We do this using the same technique as used for the particles on the surface, only this time, we embed it on the ϖth level set of ϕ_B, where ϖ is the (fictitious) time during the propagation. The particles at this level are assigned a w value ϖ divided by the desired offset height. The overall result is a uniform parameterization of each discrete level in the patch space, see Fig. 5, leading to geometric texture mappings with significantly less distortion. This method has an additional number of advantages over the first particle distribution method. First, as a new set of particles are generated at the individual levels, the number of particles at each level are independent. Thus, if the *surface area* of the patch changes with the distance to the base surface, we can adjust the number of particles generated at each level to maintain a desired particle density, thereby enabling a sufficient sampling of each level. Furthermore, we can optionally let the user specify the direction along which each control vertex is offset rather than forcing it to be in the normal direction. The effect of this is depicted in Fig. 6. By allowing the user to specify the offset direction, we add an extra level of control over the final result. This allows, for example, the user to control the distortion of a texture with a large offset in the w direction on a highly

2. As we assume ϕ_B is regularly sampled with $dx = dy = dz$, keeping the sample distance below dx guarantees a sufficient sampling. If, however, the surface is smooth, a lower sample rate is often sufficient.

3. Numerical round-off errors and inaccuracies in the finite difference potentially breaks this guarantee, although such particles are still guaranteed to remain close together.

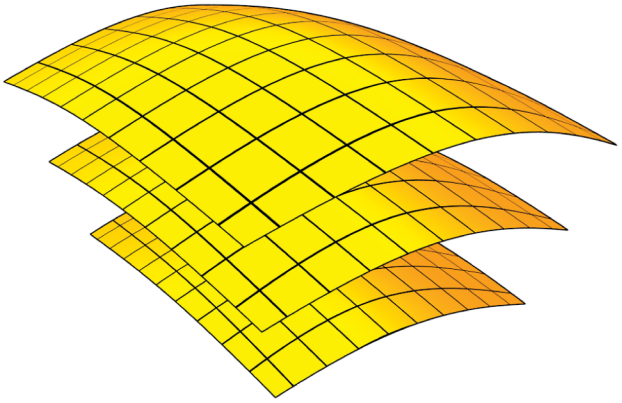


Fig. 5. The low-distortion parameterization can be thought of as uniform layers of an onion. The particles are advected as with the surface conforming parameterization, but then they are relaxed to give each level a uniform parameterization.

curved surface, as seen in Fig. 6. We have used this extra control in several examples in the following sections, most notably in Fig. 12a.

The difference between the (initial) surface conforming parametrization and the new low-distortion parameterization is illustrated in Fig. 8. This example clearly shows how the mapping based on individually advected particles (Fig. 8a) follows the local curvature of the base surface more closely than the mapping based on uniformly distributed particles (Fig. 8b), whereas the latter reduces the overall distortion of the mapped geometry.

Spline advection. The two particle distribution schemes outlined above both rely on the distance transform of the base surface (that is, the level set ϕ_B) to respectively propagate the particles in the patch space. This effectively means that texture information is propagated in a fixed direction away from the base surface. To add more flexibility, we have developed a third parameterization scheme where the particles are propagated along a spline curve originating at the center of the patch. It works as follows: As with the previous distribution schemes, we start by generating the particles on the base surface, assigning u, v-coordinates to each particle. The particles are then propagated in small steps in the direction defined by the spline curve. At each step, the particles are furthermore rotated around the current spline point to align with the tangent of the curve at that point, see

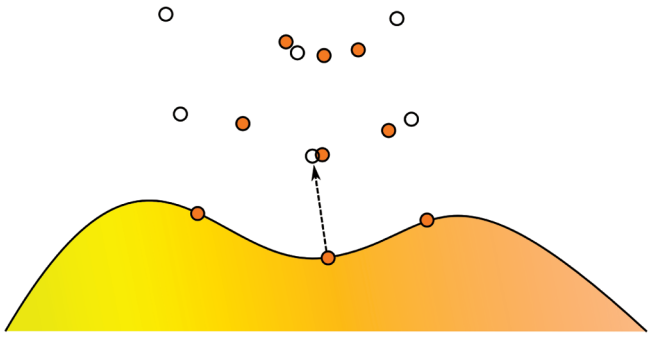


Fig. 6. Specifying a different direction for the particles to evolve along adds extra flexibility to the parameterization. The white particles are obtained by specifying a custom direction of evolution, parallel to the normal at the center point, at both control vertices. The orange particles correspond to the surface conforming distribution.

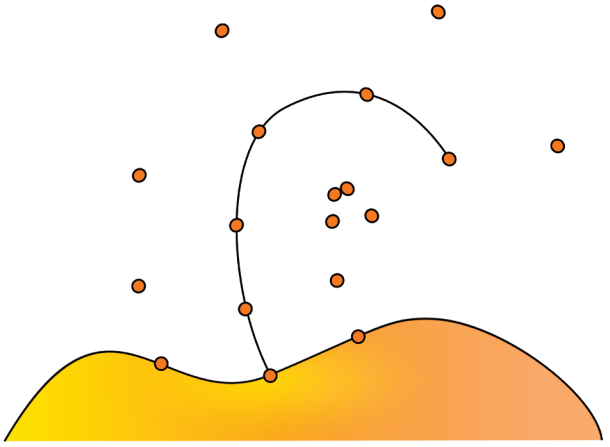


Fig. 7. Parameterization of a patch (simplified to 2D) using the spline advection parameterization. Note how the particles *move* faster or slower depending on the curvature of the spline curve.

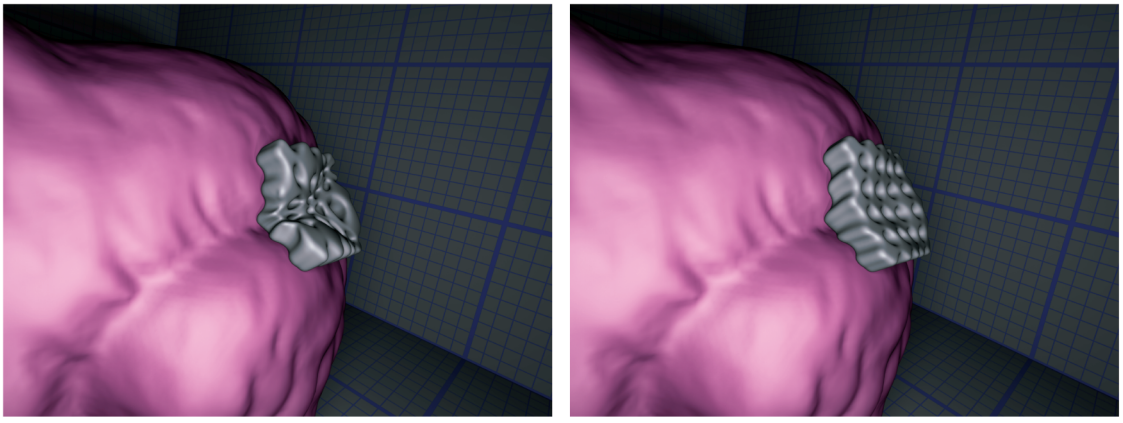
Fig. 7. The points are rotated by an angle corresponding to the angle between the tangent to the spline at the previous point and the tangent to the spline at the current point around the axis perpendicular to both tangents. As in the previous methods, copies of the particles are saved at regular intervals, and a w-coordinate, derived from the normalized distance traveled along the spline, is assigned to each particle. An example mapping generated with this technique is shown in Fig. 9.

We note that during the propagation of the particles along the spline curve, care must be taken to avoid particles crossing paths. This would potentially lead to nonmonotonic interpolations of the corresponding texture coordinates, which in turn result in inconsistent texture mappings. One possible solution to this problem is to treat the advancing particles as small spheres and apply *continuous collision detection* algorithms [29] to ensure that particles do not cross. Continuous collision detection algorithms, even though more difficult to implement, offer several advantages over their discrete counterparts. Most notable are their ability to compute the time of first contact versus the discrete approach of simply sampling an object's trajectory and reporting intersections (small fast-moving objects could pass through each other).

As a final remark we note that both the surface conforming and the low distortion parameterization assume that ϕ_B is defined throughout the patch space. Since we employ a very storage-efficient level set representation of ϕ_B, [9], distance information is only stored in a narrow tube of B. Hence, as a prelude to the parameterization methods outlined above, we first sweep out distances from this narrow tube to the remaining patch space (which is typically a very small subspace of the bounding volume of B). This has been implemented very efficiently using the *fast sweeping* method [30], which has linear time complexity in the number of voxels in the patch space.

3 Near Real-Time Semi-Implicit Mapping

We have developed a simple and efficient semiimplicit technique, which can be used as a "preview mode" for our implicit mapping to be described in the next section. The semiimplicit method maps an (explicit) polygonal mesh,



(a) (b)

Fig. 8. (a) Mapping a geometry texture to a bumpy part of the bunny using the surface conforming parameterization. (b) Using the low distortion parameterization.

M_A, onto the implicit base surface, ϕ_B, by warping the vertices of M_A in texture space into patch space using fast trilinear interpolation. The mesh connectivity is left unchanged. This technique, as well as the implicit technique, can be used in combination with any of the parameterization methods described in Section 2.

The semiimplicit mapping makes use of the fact that the patch particles form a *semiregular* 3D lattice in texture space—see Fig. 10a. By this, we mean that, in texture space, the particles are distributed into regularly spaced levels in the w direction. Each of these levels consists of a 2D regular grid of particles, but the number of particles need not be the same at all levels (see Fig. 10 for a 2D example). Since the *texture value* associated with each particle is given by their position in patch space, we can define a mapping $\Phi_{t\to p}(\mathbf{x}_t) = \mathbf{x}_p$ of a vertex $\mathbf{x}_t = (x_u, x_v, x_w) \in M_A$ as a trilinear interpolation of the particle texture values. As the number of particles may not be the same at each level in the patch space, we need to apply the interpolation in a specific order: We first interpolate at the two levels located immediately above and below the vertex in texture space, followed by an interpolation in between the levels. Fig. 10 illustrates this: First, the patch space position of the blue dots is obtained from interpolation along the green line segments. Next, we interpolate the values (patch space position) of the blue dots along the yellow line to get the patch space position of the vertex (red dot). Because each particle level form a regular 2D grid and the levels are uniformly spaced, finding the interpolants is a constant time operation. Thus, calculating

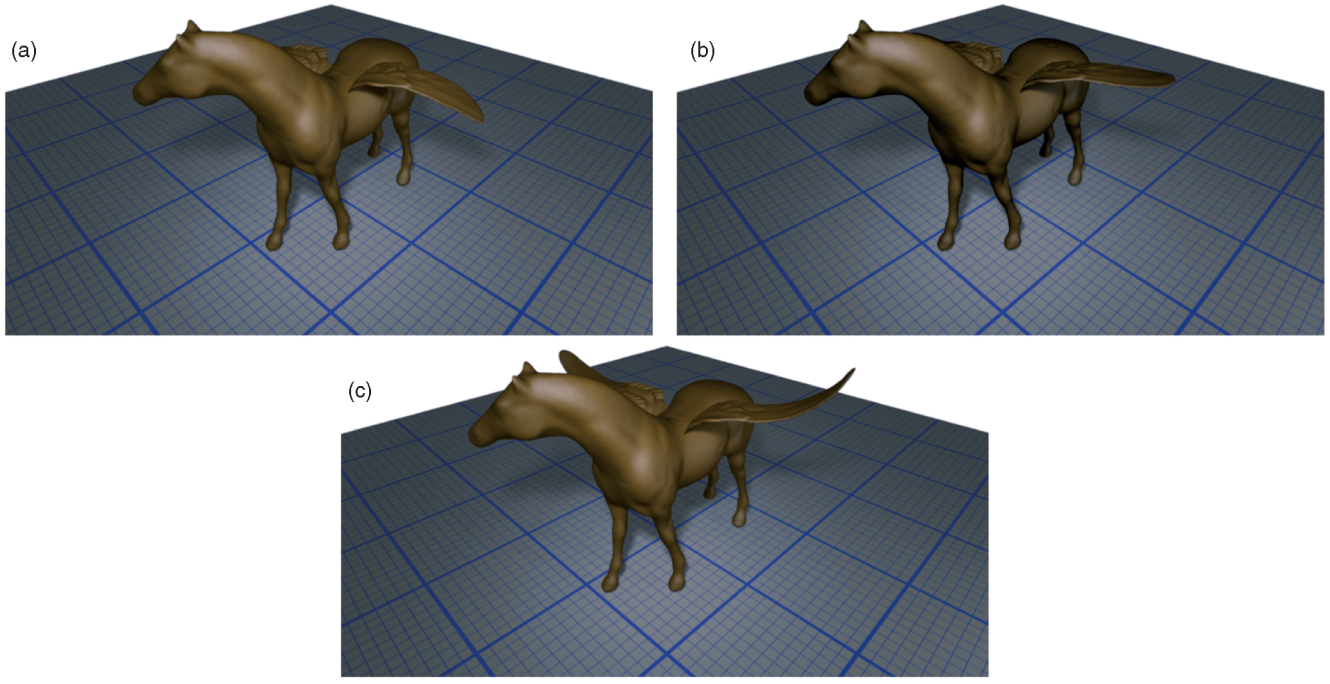


Fig. 9. The three images show a horse with wings mapped onto it in three different postures using the spline-based particle distribution scheme.

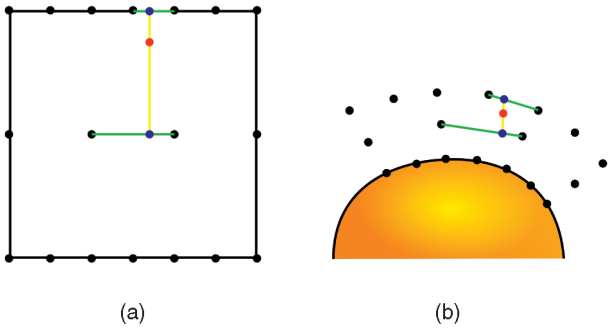


Fig. 10. Illustrating the semiimplicit mapping on a patch set with a different number of particles at each level. To get the position of a given vertex (red dot) in patch space (b) given its position in texture space (a), we first interpolate along the green lines to get the patch space position of the blue dots. These are then used to interpolate along the yellow line to get the patch space position of the texture space vertex.

the patch space position of a single vertex is also a constant time operation.

We briefly note that the proposed mapping is somewhat reminiscent of the free-form deformation technique presented by Sederberg et al. [31]. The main difference is that our scheme can handle semiregular samplings and is strictly bounded to the patch space. These properties are very important for our application and are not shared by the higher order interpolation proposed in [31]. Fig. 5 in [31] clearly illustrates that geometry is not bounded to the control polygon, which in our application would result in textures mappings that are not explicitly confined to the base surface.

4 High-Quality Implicit Mapping

The implicit mapping allows us to warp and subsequently blend level set representations of both the geometric texture and base surface. We use radial basis functions to perform our mapping. The algorithm is given as follows. First, we define a regular 3D grid, bounding the region of space spanned by the patch particles. We call this the *embedding volume*. The resolution of this grid is chosen to match the resolution of the grid on which the texture level set is sampled in texture space. Next, we define a mapping from the patch space into the texture space by means of radial basis function interpolation. This essentially allows us to resample our texture geometry in patch space. More specifically, for each grid point x_p in the embedding volume, we map it to texture space via the radial basis function, resulting in the point x_t. We then use the point x_t to perform an interpolation[4] on the texture volume, thereby getting the desired distance value. Once all points in the grid are assigned a distance value, the embedding volume will contain a warped instance of the texture geometry.

The method we use for our radial basis function is similar to that of Dinh et al. [32], which is a good candidate because of its robustness with respect to irregularities of the sample points. Furthermore, it adds flexibility due to the fact that it allows for both strict interpolation, as well as approximation, simply by varying a parameter (λ_i). For the sake of completeness, we will summarize this technique below.

Assume the patch particles have Cartesian coordinates $\{\mathbf{p}_i, i = 1 \dots n\}$ and texture coordinates $\{k_i, k = u, v, w, i = 1 \dots n\}$, as described in Section 2. Now, we wish to establish a mapping from Cartesian coordinates in patch space to texture coordinates in texture space, $\Phi_{p\to t}$. The key idea is to split the mapping into three independent mappings:

$$\Phi_{p\to t}(\mathbf{x}_p) = \mathbf{x}_t \Rightarrow \begin{array}{r} \Phi_{p\to t,u}(\mathbf{x}_p) = x_u \\ \Phi_{p\to t,v}(\mathbf{x}_p) = x_v \\ \Phi_{p\to t,w}(\mathbf{x}_p) = x_w \end{array}$$

with each of the texture mapping functions, $\Phi_{p\to t,k}$, expressed as a sum of weighted *radial basis functions*:

$$\Phi_{p\to t,k}(\mathbf{x}_p) = P_k(\mathbf{x_p}) + \sum_{i=1}^{n} \omega_{k,i}\varphi(|\mathbf{x}_p - \mathbf{p}_i|), \tag{1}$$

where $\varphi(\mathbf{x}_p)$ is a radially symmetric basis function; n is the number of basis functions; $\mathbf{p}_i$ is the center of the ith basis; $\omega_{k,i}$ are the weights for the ith basis for texture coordinate k; and $P_k(\mathbf{x}_p) = \rho_{k,0}x_x + \rho_{k,1}x_y + \rho_{k,2}x_z + \rho_{k,3}$ is a polynomial spanning the null space of the basis function. Similar to that in [32], we center a basis function at each patch point.

To find the weights, $\omega_{k,i}$, and polynomial coefficients, $\rho_{k,j} = \{\rho_{k,0}, \rho_{k,1}, \rho_{k,2}, \rho_{k,3}\}$ for each mapping, $k = \{u, v, w\}$, we apply (1) to each of the patch points. Since we already have assigned a k coordinate to each patch point, this leads to a linear system of $n + 4$ equations with $n + 4$ unknowns:

$$\begin{bmatrix} \varphi(|\mathbf{p}_1 - \mathbf{p}_1|) + \lambda_1 & \cdots & \varphi(|\mathbf{p}_1 - \mathbf{p}_n|) & \mathbf{p}_1 & 1 \\ \vdots & & \vdots & \vdots & \vdots \\ \varphi(|\mathbf{p}_n - \mathbf{p}_1|) & \cdots & \varphi(|\mathbf{p}_n - \mathbf{p}_n|) + \lambda_n & \mathbf{p}_n & 1 \\ \mathbf{p}_{1,x} & \cdots & \mathbf{p}_{n,x} & 0 & 0 \\ \mathbf{p}_{1,y} & \cdots & \mathbf{p}_{n,y} & 0 & 0 \\ \mathbf{p}_{1,z} & \cdots & \mathbf{p}_{n,z} & 0 & 0 \\ 1 & \cdots & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \omega_{k,1} \\ \vdots \\ \omega_{k,n} \\ \rho_{k,0} \\ \rho_{k,1} \\ \rho_{k,2} \\ \rho_{k,3} \end{bmatrix} = \begin{bmatrix} k_1 \\ \vdots \\ k_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{2}$$

After solving this linear system for each $k = \{u, v, w\}$, the resulting $\{\omega_{k,i}, \rho_{k,j}\}$ and are next backsubstituted into (1) to compute u, v, and w coordinates on the 3D grid in patch space. The resampled texture level set is then simply computed by interpolation in the texture space.

The λ values on the diagonal of the matrix in (2) allow us to control the smoothness of the mapping. As previously mentioned, each particle, $\mathbf{p}_i$ with position $\mathbf{x}_{p,i}$ maps to a specific position in the texture space $\mathbf{x}_{t,i}$. By adding the λ_i values to (2), we can relax this correspondence leading to the following inequality: $|\Phi_{p\to t}(\mathbf{x}_{p,i}) - \mathbf{x}_{t,i}| \leq \zeta_i$, where the constant ζ_i is deducted from λ_i. The larger λ_i is, the larger ζ_i will be. Also, if λ_i is zero, then so is ζ_i. As the ζ values increase, the interpolation between the sample values

4. We typically employ trilinear, and occasionally tricubic, interpolation, but essentially, any bounded interpolation scheme can be used.

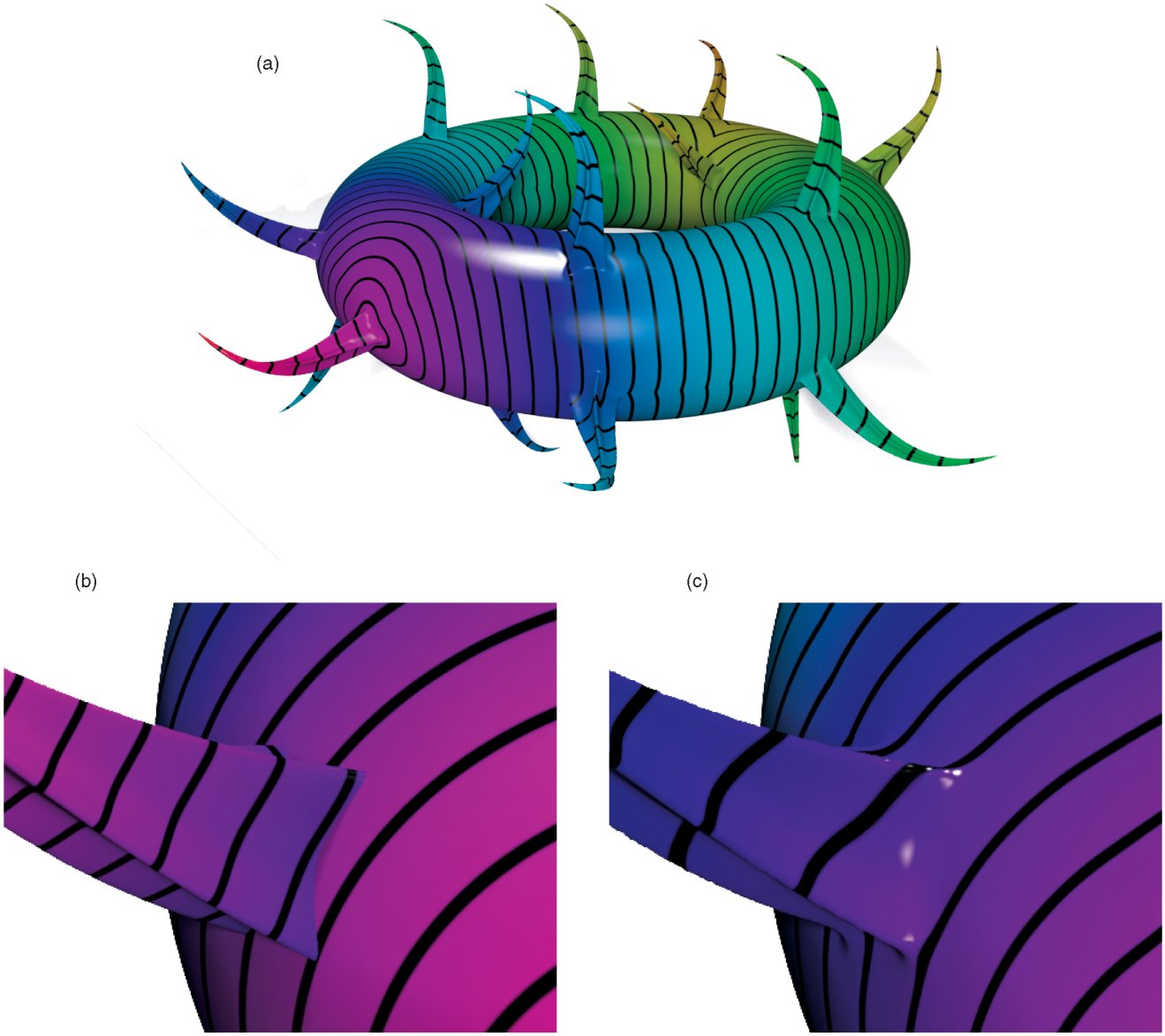


Fig. 11. Mapping a number of spikes onto a torus. The closeups of a single spike shows the difference between not applying the CSG union (b) and applying the CSG union, as well as the blending of the intersection (c). Notice the discontinuity in the shading in (b), which is a result of the spike and torus being separate geometries. Merging (and blending) the two surfaces resolves the issue (c).

becomes less restricted enabling a smoother interpolation, and thereby also a smoother mapping. For the results in this paper, we have typically used two different λ values. Particles on the interface (that is, particles with $w = 0$) are assigned small λ values to ensure that the mapping follows the interface closely. These values typically fall in the range 0.001 to 0.01. The remaining points are assigned a larger λ usually between 0.1 and 0.5 to ensure a smoother mapping away from the interface.

Since the implicit mapping uses level set representations for both the texture and the base geometry, we can easily produce a single topologically connected surface by merging and blending the two volumes. This can be achieved with Boolean (constructive solid geometry (CSG)) operations like union or difference of the two level sets. This, in turn, simply amounts to a min/max operation of the distance fields followed by a reinitialization in the resulting narrow band. However, the result of Boolean CSG operations typically create very visible C^0 discontinuities along the intersection seam. To further address this, we employ the techniques described in [13] that performs localized mean curvature-based smoothing in the vicinity of the intersection of the two level sets. This approach allows for direct user control of mean curvature and, thus, the smoothness, of the resulting volume. Both the merging/CSG union and the smoothing of the intersection are optional operators applied, if desired, once the mapping is completed. Due to numerical issues, we cannot guarantee that the base surface and the texture will match up exactly. Thus, to ensure a sufficient overlap between the two surfaces required to get a nice blending, we *push* the texture slightly downwards by adding a small offset to the w texture coordinate.

Fig. 11 shows a torus with several spikes mapped onto it using this technique. Figs. 11b and 11c shows a close up of the intersection of the torus and a single spike, one with the merging and blending performed, Fig. 11c, and one with-

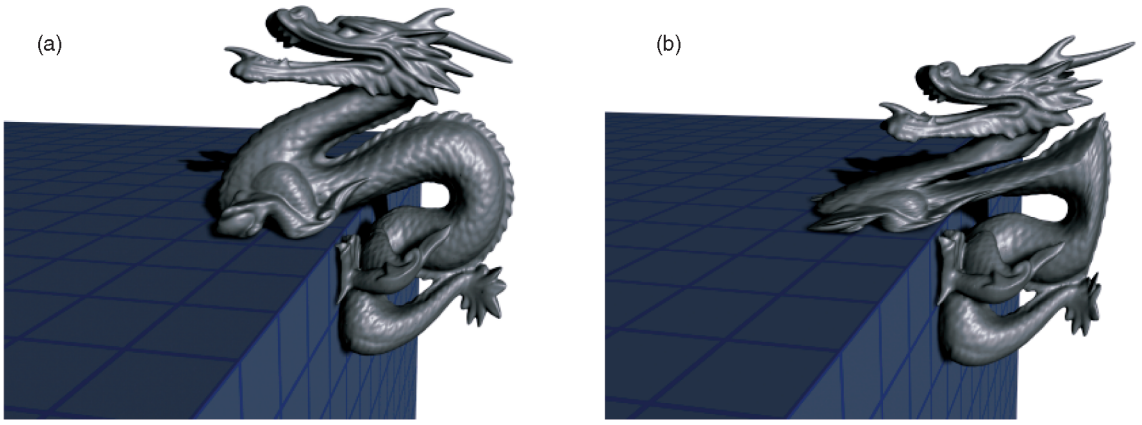


Fig. 12. Mapping a dragon onto a sharp corner using (a) our new geometric texture mapping technique (semi-implicit mapping) and (b) using the technique in [8]. Both mappings were done in less than one second.

out, Fig. 11b. The texture maps are shaded isocontours of a Euclidian distance field ψ, computed by solving the Eikonal equation, $|\nabla_s \psi| = 1$, where ∇_s denotes the gradient projected on the surface, and $\psi = 0$ at the lower left spike. Since the surfaces in Fig. 11b have not been merged to form a single topologically connected surface, the resulting procedural texture mapping is discontinuous along the intersections. An alternative approach is of course to use procedural volumetric textures defined in the embedding space of the surfaces like in [33]; however, such techniques severely limits the sizes and resolutions of the models.

One potential issue with using radial basis function interpolation as just described is that the mapping is no longer guaranteed to be one to one but could potentially be one to many. That is, two or more points in patch space may potentially map onto the same point in texture space. We have, however, not observed any artifacts related to this in practice.

5 Results and Applications

Fig. 12 shows an example of mapping a geometric texture onto an object with a sharp edge. Due to the underlying parameterization of *shell space*, which is based on an offset surface generated by offsetting the base mesh vertices in the direction of the vertex normals, the object mapped using the shell-mapping technique in [8], Fig. 12b is severely distorted. As our technique allows a guaranteed uniform distribution of the patch points, our mapping, Fig. 12a, guarantees a smooth mapping, even across such sharp edges. Although the distortion minimization technique presented in [34] can help reduce the distortion in the case of shell mapping, it cannot completely resolve the problem due to the linear interpolation in shell space. The only way to completely resolve this problem is to generate a smoother offset surface, which is exactly what our approach does. As for the performance of the two techniques, both mappings were done in roughly the same time, which is in less than one second.

One major problem with using regularly sampled implicit surfaces is the memory requirements of the 3D grid, which imposes a problematic limit at high and useful resolutions. This is the primary reason for using the DT-grid, which allows us to use significantly higher volume resolutions. The large dragon in Fig. 13 has an effective resolution of $512 \times 244 \times 350$ and all of the 12 "baby" dragons are made using the same resolution.

Although the two mapping schemes presented in Sections 3 and 4 produce almost visually identical results in many cases, they are in many ways very different methods offering a different set of features in addition to the obvious difference in the geometric representation of the texture. The most important feature of the semiimplicit method is its speed. Although the time complexity of the implicit method scales with the number of particles times the number of voxels in the embedding volume, the semiimplicit mapping is linear in the number of vertices on the texture geometry. The dragon in Fig. 12a contains more than 400,000 vertices and was mapped in less than a second using the semi-implicit method. Although the semiimplicit method is often capable of producing good results relatively fast, the implicit method offers some distinct benefits. Most importantly, since both the texture and base surface are represented using level sets, we can readily produce a simple topologically connected surface by means CSG operations—either prior to a mesh extraction or alternatively during direct ray casting. Furthermore, we can apply a smoothing operation (see [13]) on the intersection of

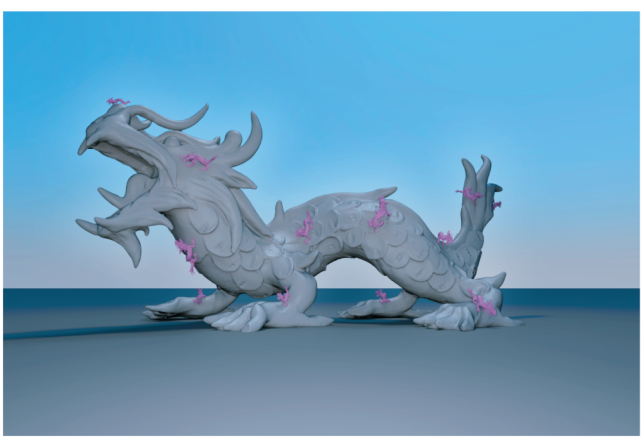


Fig. 13. Mapping a number of small dragons onto a mother dragon using the proposed technique.

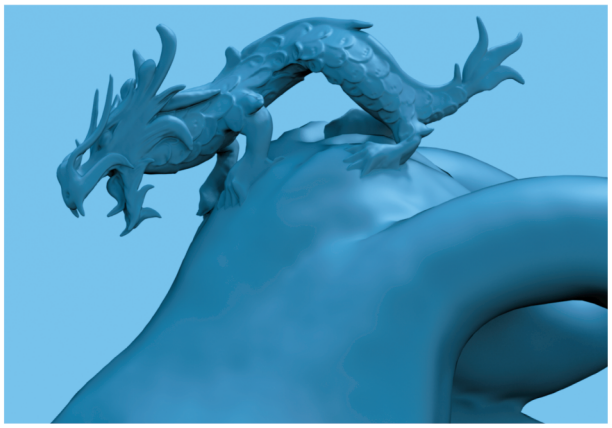
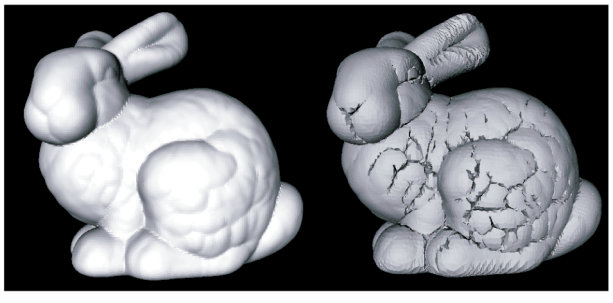


Fig. 14. Closeup of a single one of the baby dragons in Fig. 13. Notice the high level of detail and topological change that results from blending.

the base surface and the warped texture, if a smooth intersection with continuous normals is desired. Another advantage of the radial basis function interpolation is that it is significantly less sensitive to the distribution of the patch points. If the base surface has many high-frequency features, these features will directly influence the result of an explicit mapping. On the other hand, the implicit mapping allows for direct control of the smoothness through the parameters λ_i entering the linear system in (2). By increasing λ_i, the implicit mapping will retain the ability to produce a smooth mapping while still allowing for lower frequency features of the base surface. Also, even though the semiimplicit mapping is only C_0, the implicit mapping allows for multiple orders of continuity, although the exact order is determined by the chosen basis function. In our tests, we have seen the best results when using $f(r) = r$ as our basis function. Still, the implicit mapping is much slower than a semiimplicit scheme. Mapping a single model takes 20-30 seconds for the two latches in Fig. 16 using 280 particles and an embedding volume of four million voxels for the small latch and 660 particles and 5.6 million voxels for the larger. Mapping times vary between 4-5 minutes per baby dragon in Figs. 13 (and 14) using 3-400 particles and an embedding volume of 20-30 million voxels.



(a) (b)

Fig. 15. The offset on the left is a natural result of the level set's implicit representation. On the right, we see an explicit polygonal offset (note the degeneracies in concave regions) using the method proposed in Shell Maps [8].

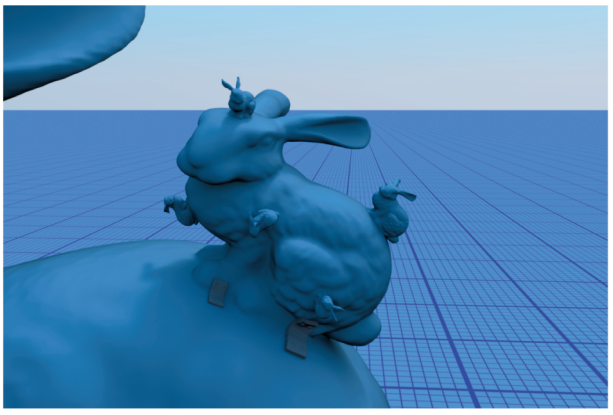


Fig. 16. Recursive mapping: mapping a bunny onto another bunny, which then again has several smaller bunnies mapped onto it. The middle bunny is "held onto" the bigger bunny using a couple of metal latches that is in fact texture-mapped cuboids.

Another benefit of our implicit approach is that we can easily map new objects onto previously mapped objects. Fig. 16 shows two latches and several bunnies mapped onto a base surface and a previously mapped bunny. To achieve a similar result, Shell Maps would have to fuse the two bunnies together, generate new u, v coordinates, and finally create a new offset surface.

Using the signed distance of the level set function for generating *offset surfaces* offers several advantages. First of all, the further we move away from the base surface, the smoother the offset surface becomes. This means that the influence of high-frequency details in the base geometry decreases away from the surface, resulting in smoother looking results, as shown in Fig. 12a. Previous approaches have employed explicit geometry representations, which can lead to problematic self-intersections of the dilated offset surfaces. Consequently, these methods have been limited to rather small offsets, which in turn only allows for the mapping of small geometric textures. This self-intersection problem is illustrated in Fig. 15, where two offset surfaces are generated from the bunny model using, respectively, level sets and the technique presented in Shell Maps [8]. It should be evident from this simple example that our current method is significantly more robust with surface offsets. The small bunnies in Fig. 16 is an example of mappings using larger offsets (although our method allows for even larger offsets).

6 Conclusions and Future Work

We have presented fast and flexible techniques for warping and blending (or subtracting) geometric details, in the form of a geometric texture, onto level set surfaces. These techniques are similar in nature to the shell-mapping technique, though we have eliminated some of the limitations of the shell-mapping approach. Our current approach is based on using implicit geometry, which makes it easy to merge the base and texture geometry into a single topologically connected object, as well as smoothing the intersection between the base and texture geometry guaranteeing a smooth surface with smooth normals. Furthermore, our mapping employs a flexible particle-based parameterization. As the parameterization is characterized by the distribution of the particles, we can change

the parameterization by changing the way the particles are distributed. To demonstrate this flexibility, we have presented three different methods for distributing the particles including a method that reduces the overall texture distortion.

Although the semiexplicit mapping proposed in this paper is very fast, the implicit mapping is rather slow. The problem is that the speed of the implicit mapping depends not only on the size of the volume it is being mapped into but also on the total number of particles defining the parameterization. We are currently considering a different approach to address this issue. One idea is to replace the current global radial basis functions with functions that have only local support. Another interesting approach would be to only resample the level set of the geometric texture in a local neighborhood of its surface. However, this idea is far from simple, and so far, we have not been able to devise a robust algorithm.

Another interesting idea for future work is to replace the 2D parametrization technique of Pedersen [23] with DEM in [24]. The latter approach seems more intuitive and simpler to use from an artist's point of view.[5]

ACKNOWLEDGMENTS

The authors would like to thank Michael Nielsen and other members of the Graphics Group at Linköping University for allowing us to use their software—in particular, the DT-Grid implementation. Additional thanks goes to Ola Nilsson for assistance with rendering and Louis Feng for help in generating illustrations. Finally, we would like to acknowledge our reviewers for many good comments that helped to improve our paper.

REFERENCES

[1] J.F. Blinn and M.E. Newell, "Texture and Reflection in Computer Generated Images," *ACM Comm.*, vol. 19, no. 10, pp. 542-547, 1976.
[2] J.F. Blinn, "Simulation of Wrinkled Surfaces," *Proc. ACM SIGGRAPH '78*, pp. 286-292, 1978.
[3] R.L. Cook, "Shade Trees," *Proc. ACM SIGGRAPH '84*, pp. 223-231, 1984.
[4] J.T. Kajiya and T.L. Kay, "Rendering Fur with Three Dimensional Textures," *Proc. ACM SIGGRAPH '89*, vol. 23, no. 3, pp. 271-280, July 1989.
[5] Y. Chen, X. Tong, J. Wang, S. Lin, B. Guo, and H.-Y. Shum, "Shell Texture Functions," *ACM Trans. Graphics*, vol. 23, no. 3, pp. 343-353, Aug. 2004.
[6] X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum, "Generalized Displacement Maps," *Proc. Eurographics Symp. Rendering*, pp. 227-233, 2004.
[7] P. Bhat, S. Ingram, and G. Turk, "Geometric Texture Synthesis by Example," *Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing (SGP '04)*, pp. 41-44, 2004.
[8] S.D. Porumbescu, B.C. Budge, L. Feng, and K.I. Joy, "Shell Maps," *Proc. ACM SIGGRAPH '05*, vol. 24, no. 3, pp. 626-633, 2005.
[9] M.B. Nielsen and K. Museth, "Dynamic Tubular Grid: An Efficient Data Structure and Algorithms for High Resolution Level Sets," *J. Scientific Computing*, vol. 26, no. 3, pp. 261-299, 2006.
[10] S. Osher and J.A. Sethian, "Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations," *J. Computational Physics*, vol. 79, pp. 12-49, 1988.
[11] J.A. Sethian, *Level Set Methods and Fast Marching Methods*, second ed. Cambridge Univ. Press, 1999.
[12] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002.
[13] K. Museth, D. Breen, R. Whitaker, and A. Barr, "Level Set Surface Editing Operators," *Proc. ACM SIGGRAPH '02 (ACM Trans. Graphics)*, vol. 21, no. 3, pp. 330-338, July 2002.
[14] S. Mauch, "Efficient Algorithms for Solving Static Hamilton-Jacobi Equations," PhD dissertation, California Inst. of Technology, 2003.
[15] W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163-168, July 1987.
[16] F. Losasso, F. Gibou, and R. Fedkiw, "Simulating Water and Smoke with an Octree Data Structure," *ACM Trans. Graphics*, vol. 23, no. 3, Aug. 2004.
[17] B. Houston, M. Nielsen, C. Batty, O. Nilsson, and K. Museth, "Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation," *ACM Trans. Graphics*, vol. 25, no. 1, pp. 1-24, 2006.
[18] S.F. Frisken, R.N. Perry, A.P. Rockwood, and T.R. Jones, "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics," *Proc. ACM SIGGRAPH '00*, pp. 249-254, 2000.
[19] G. Turk, "Texture Synthesis on Surfaces," *Proc. ACM SIGGRAPH '01*, pp. 347-354, 2001.
[20] M. Tarini, K. Hormann, P. Cignoni, and C. Montani, "PolycubeMaps," *ACM Trans. Graphics*, vol. 23, no. 3, pp. 853-860, 2004.
[21] R. Zonenschein, J. Gomes, L. Velho, L. de Figueiredo, M. Tigges, and B. Wyvill, "Texturing Composite Deformable Implicit Objects," *Proc. Int'l Symp. Computer Graphics, Image Processing, and Vision (SIBGRAPHI '98)*, p. 346, 1998.
[22] R. Zonenschein, J. Gomes, L. Velho, and L. de Figueiredo, "Controlling Texture Mapping onto Implicit Surfaces with Particle Systems," *Proc. Third Int'l Workshop Implicit Surfaces*, pp. 131-138, 1998.
[23] H.K. Pedersen, "Decorating Implicit Surfaces," *Proc. ACM SIGGRAPH '95*, pp. 291-300, 1995.
[24] R. Schmidt, C. Grimm, and B. Wyvill, "Interactive Decal Compositing with Discrete Exponential Maps," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 605-613, 2006.
[25] F. Neyret, "Modeling, Animating, and Rendering Complex Scenes Using Volumetric Textures," *IEEE Trans. Visualization and Computer Graphics*, vol. 4, no. 1, pp. 55-70, 1998.
[26] J. Peng, D. Kristjansson, and D. Zorin, "Interactive Modeling of Topologically Complex Geometric Detail," *ACM Trans. Graphics*, vol. 23, no. 3, pp. 635-643, 2004.
[27] K.W. Fleischer, D.H. Laidlaw, B.L. Currin, and A.H. Barr, "Cellular Texture Generation," *Proc. ACM SIGGRAPH '95*, pp. 239-248, 1995.
[28] X. Provot, "Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior," *Proc. Graphics Interface Conf. (GI '95)*, pp. 147-154, 1995.
[29] S. Hadap, D. Eberle, P. Volino, M.C. Lin, S. Redon, and C. Ericson, "Collision Detection and Proximity Queries," *Proc. ACM SIGGRAPH '04*, p. 15, 2004.
[30] H. Zhao, "Fast Sweeping Method for Eikonal Equations," *Math. of Computation*, vol. 74, pp. 603-627, 2004.
[31] T.W. Sederberg and S.R. Parry, "Free-Form Deformation of Solid Geometric Models," *Proc. ACM SIGGRAPH '86*, vol. 20, no. 4, pp. 151-160, 1986.
[32] H.Q. Dinh, G. Turk, and G. Slabaugh, "Reconstructing Surfaces by Volumetric Regularization Using Radial Basis Functions," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, no. 10, pp. 1358-1371, 2002.
[33] S. Worley, "A Cellular Texture Basis Function," *Proc. ACM SIGGRAPH '96*, pp. 291-294, 1996.
[34] K. Zhou, X. Huang, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum, "Mesh Quilting for Geometric Texture Synthesis," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 690-697, 2006.

5. For a single patch, our current approach requires the user to place four particles on the surface, whereas an approach based on DEM would require only two. Furthermore, changing the size or orientation of the patch requires us to change the position of all four particles, as opposed to only moving a single point around with the DEM approach.

Anders Brodersen received the MSc degree in computer science from the University of Aarhus, Denmark, in 2003 and is currently waiting to defend his PhD dissertation. He is currently a research assistant at the University of Aarhus, Denmark. His research interests focus on geometric modeling and real-time rendering. He is also the cofounder of a small software company (43D) specializing in architectural visualization, where he is in charge of maintaining and developing the 3D engine driving the company's visualization software.

Ken Museth received the MSc degree in physical chemistry and the PhD degree in computational quantum dynamics from the University of Copenhagen in 1994 and 1997, respectively. He is a full professor of computer graphics at Linköping University, Sweden, and an adjunct professor at Aarhus University, Denmark. From 1998 to 2003, he was a visiting faculty member in the Chemical Physics Department, then a research scientist in the Computer Science Department at the California Institute of Technology. He has also been a scientific consultant to the movie houses Digital Domain and Rhythm & Hues, as well as worked on mission design and visualization for NASA's Jet Propulsion Laboratory. In 2003, he was appointed a chair at Linköping University, where he is heading the Graphics Group. He has published more than 50 papers, a significant portion of which are on deforming geometry and level set methods. He is currently visiting Digital Domain to work on fluid effects for the feature movie "*Pirates of the Caribbean: At Worlds End*." He is a member of the IEEE.

Serban Porumbescu received the bachelor's degrees in computer science engineering and electrical engineering and the master's and PhD degrees in computer science from the University of California, Davis. He completed a stay as a visiting researcher at Sony Computer Entertainment of America, where he focused on real-time fluid simulations and dynamic ambient occlusion for the Playstation 3. Currently, he is working at Nvidia as a member of the OS X OpenGL driver team.

Brian Budge received the BS degree in mathematics from the University of Utah and is currently working toward the PhD degree in the Department of Computer Science, working at the Institute for Data Analysis and Visualization, University of California, Davis. His research interests involve computer-aided complexity modeling and physically based rendering. He is a student member of the IEEE.