Available online at www.sciencedirect.com

SCIENCE DIRECT

Computers & Graphics 28 (2004) 851–868

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Surface processing methods for point sets using finite elements

Ulrich Clarenz[a], Martin Rumpf[a], Alexandru Telea[a,b,*]

[a]Institut für Mathematik, Duisburg University, Lotharstrasse 65, 47048 Duisburg, Germany
[b]Department of Mathematics and Computer Science, Eindhoven University of Technology, Den Dolech 2, 5600 MB, Eindhoven, The Netherlands

Abstract

We present a framework for processing point-based surfaces via partial differential equations (PDEs). Our framework efficiently and effectively brings well-known PDE-based processing techniques to the field of point-based surfaces. At the core of our method is a finite element discretization of PDEs on point surfaces. This discretization is based on the local assembly of PDE-specific mass and stiffness matrices, using a local point coupling computation. Point couplings are computed using a local tangent plane construction and a local Delaunay triangulation of point neighborhoods. The definition of tangent planes relies on moment-based computation with proven scaling and stability properties. Once local stiffness matrices are obtained, we are able to easily assemble global matrices and efficiently solve the corresponding linear systems by standard iterative solvers. We demonstrate our framework by several types of PDE-based surface processing applications, such as segmentation, texture synthesis, bump mapping, and geometric fairing.
© 2004 Elsevier Ltd. All rights reserved.

Keywords: Point set rendering; Finite elements; Surface processing; Differential equations

1. Introduction

Surface processing tools and techniques are widespread in computer graphics, animation, medical imaging, computer aided modeling, and computer vision. Many surface processing operations can be described via partial differential equations (PDEs). Using PDEs to implement surface processing has a long history and several advantages, as compared to other more algorithmic surface processing techniques. First, PDEs describe concisely and naturally a large spectrum of transformations, such as deformations, smoothing, or denoising. Secondly, PDE-based approaches come with a solid mathematical basis that provides quantitative and qualitative results about the way they alter a given surface. Finally, many efficient and exact methods for PDE discretization and solving are readily available. Among the latter, the by far most used approach is the combination of finite element (FE) discretization and iterative numerical methods, which naturally matches the triangular mesh models ubiquitous in computer graphics.

Recently, point based representations have been proposed as an alternative to triangles for three-dimensional (3D) surfaces, with a number of advantages. No 'mesh', or connectivity information has to be

*Corresponding author. Department of Mathematics and Computer Science, Eindhoven University of Technology, Den Dolech 2, 5600 MB, Eindhoven, The Netherlands. Tel.: +31-40-247-5008; fax: +31-40-246-8508.

E-mail addresses: clarenz@math-uni-duisburg.de (U. Clarenz), rumpf@math-uni-duisburg.de (M. Rumpf), alext@win.tue.nl (A. Telea).

0097-8493/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2004.08.013

stored explicitly. This allows a simple and compact representation, ideal for fast rendering and editing. When combined with advanced rendering techniques such as splatting [1,2], point based surfaces can be superior to triangle meshes in terms of rendering quality and data storage flexibility.

Processing point-based surfaces via PDEs should add the modeling power of PDE representation to the flexibility of the point based model. However, defining and solving PDEs using finite elements on point based surfaces is not straightforward. The main problem here is that point-based surfaces are mesh-less, so there is no direct way to define the finite elements underlying the PDE space. We shall not consider the option of building a global mesh from the point set [3,4] here, as it undermines the fundamental philosophy and advantages of point based models. Moreover, global point cloud triangulations and PDEs on triangle meshes respectively have been already extensively treated.

Instead, we propose an alternative approach for finite element based PDEs on point surfaces. We proceed by constructing a number of local FE matrices that represent the surface properties over small point neighborhoods. These matrices are next assembled in a single matrix which allows PDE discretization and solving on the complete surface. We extend the results presented in [5] for surface segmentation, fairing, and texture synthesis by presenting new applications of our framework for bump mapping and point cloud triangulation.

We first review the basics of point based representations and point cloud triangulations (Section 2), some basic PDE problems on surfaces (Section 3), and the general finite element approach on triangular surfaces (Section 4). Next, we detail the difficulties of treating PDEs on point clouds (Section 5). Section 6 presents our construction of tangent spaces and local meshes. We use this basis to build finite elements on point surfaces, by assembling local and global FE matrices (Section 7). Using such matrices, we solve several PDEs on point surfaces, leading to segmentation and texture processing (Section 8.1), texture synthesis (Section 8.2), inpainting (Section 8.4), bump mapping (Section 8.3), surface fairing (Section 8.5), and point cloud triangulation (Section 8.6). Section 9 details our implementation decisions. Finally, Section 10 concludes the paper.

2. Related work

In the last years, a large number of papers related to point-based surfaces has emerged. We briefly outline those related to our work, without attempting a complete overview.

Point set methods have two main components: approximating a usually smooth surface from the point set, followed by rendering the approximation. Approximating surfaces from points can be done by many techniques. These mainly differ in the assumptions laid on the point set and the smoothness model. A quite simple approximation replaces the point set with a triangulated surface model, or triangle mesh. Several efficient triangulation methods for point clouds exist [3,4,6,7]. In many cases, such techniques can be seen as producing piecewise $\mathscr{C}^1$ approximations of the point cloud. Although efficient and reasonably simple to implement, such techniques may produce surfaces lacking the desired smoothness, as described in [8,9]. To alleviate this, smoothing operations can be applied to the obtained triangle mesh, such as iterative Laplacian smoothing [10], curvature flow fairing [11], or discrete variational fairing [12]. Alternatively, increased smoothness can be obtained by using higher local approximations, such as piecewise polynomials [13].

Rendering point surfaces follows the surface approximation assumptions. Different rendering primitives may be used, ranging from simple flat shaded planar discs, such as used by the QSplat system [14], up to elliptically weighted Gaussian splats [1] and differential points [15]. To achieve a smooth surface, one can use the moving least squares method [2,8], or blend disk primitives which are tangent to the surface [1]. More complex primitives encode more information on the vicinity of a rendered point, such as surface geometry, at additional rendering expense. Simple primitives render faster but may have limited quality, especially for nonuniformly sampled surfaces.

3. PDEs on surfaces

We start by defining basic notions of differential operators on surfaces. To this aim, we will use the concept of tangential gradients on embedded surfaces $\mathscr{M}$ in $\mathbb{R}^3$. The tangential gradient $\nabla_{\mathscr{M}} u$ for a function u defined in a neighborhood of $\mathscr{M}$, is given by

$$\nabla_{\mathscr{M}} u = \nabla_{\mathbb{R}^3} u - (n \cdot \nabla_{\mathbb{R}^3} u)\, n,$$

where $n : \mathscr{M} \to S^2$ is the normal mapping. The gradient in the ambient space is thus projected onto the surface's tangential space. The result coincides with the classical geometric gradient $\nabla_{\mathscr{M}} u$ for embedded surfaces. In the following, we denote the scalar product of two vectors $a, b \in \mathbb{R}^m$ by $a \cdot b$. The components of $\nabla_{\mathscr{M}} u$ are denoted by $\nabla_i u$.

For a vector field $v : \mathscr{M} \to \mathbb{R}^3$ of components $v = (v_1, v_2, v_3)$, we define its divergence using the components of the tangential gradient, i.e.,

$$\operatorname{div}_{\mathscr{M}} v := \nabla_i v_i.$$

Here and in the following, we use the Einstein summation convention. In this notation, the Laplace operator on surfaces is given by

$$\Delta_{\mathcal{M}} u = \nabla_i \nabla_i u.$$

For surfaces in $\mathbb{R}^3$, curvature may be expressed by the shape operator S which is—using tangential gradients—given by the 3×3-matrix $S = \underline{D}_{\mathcal{M}} n = \nabla_{\mathcal{M}} n$. This matrix operates as a symmetric endomorphism on the tangent spaces and may be diagonalized by the principal curvatures κ_1 and κ_2. The classical mean curvature is then given by $h = \mathrm{tr}\, \nabla_{\mathcal{M}} n = \kappa_1 + \kappa_2$ and we have the well known identity $\Delta_{\mathcal{M}} x = -h\, n$ where x is the position vector of the surface.

We can now formulate, on surfaces, the same type of problems as on Euclidean domains. A basic problem type is the boundary value problem:

For a subset $\Omega \subset \mathcal{M}$, a diffusion tensor A, and a right-hand side f we ask for a function $u : \bar{\Omega} \to \mathbb{R}$ which solves

$$-\mathrm{div}_{\mathcal{M}}(A \nabla_{\mathcal{M}} u) = f \tag{1}$$

on $\mathcal{M}$ and reaches values $u = u^{\partial}$ on the boundary $\partial\Omega$ of $\mathcal{M}$.

An example application for such an elliptic problem is the inpainting of a locally destroyed coloring of the surface. As a second problem type we consider reaction diffusion problems on $\mathcal{M}$:

Find a function $u : \mathbb{R}_0^+ \times \mathcal{M} \to \mathbb{R}$ with $u(0, \cdot) = u_0$ for some function u_0, such that

$$\partial_t u - \mathrm{div}_{\mathcal{M}}(A \nabla_{\mathcal{M}} u) = f, \tag{2}$$

where A is the diffusion tensor and f the source term.

Here u can be a scalar or a vector valued quantity, and $A = A[u]$ and $f = f[u]$ may depend nonlinearly on u. An example of such a problem is segmentation via diffusion of a "marker color" which stops at the surface's feature lines. Another example is the smoothing of a gray scale surface texture, which leads to a scalar diffusion problem, where A is the nonlinear anistropic diffusivity. In addition, we consider reaction diffusion systems for texture synthesis, such as introduced by Turk [16]. Given several components, or species, f encodes the coupling of the species' concentrations via a particular reaction. Reaction diffusion systems are a simple and effective way for synthesizing repetitive textures on surfaces. As the next application, if we consider $u = x$, where x is the position vector of the surface itself, we obtain a curvature motion problem. Indeed, if $A = 1$ and $f = 0$, it is well-known that Eq. (2) is equivalent to mean curvature motion, i.e. $\partial_t x = -h(x)\, n(x)$. Using a nonlinear diffusivity, we obtain a feature preserving fairing method. Different PDEs can be modelled similarly, if desired.

4. Reviewing finite elements on triangular surfaces

Before we consider solving these PDEs on point based surfaces, we briefly discuss the by now classical finite element discretization scheme, frequently used for the numerical treatment of PDEs on discrete triangular surfaces. Let $\mathcal{M}_h$ denote an approximating triangulation of $\mathcal{M}$, where h indicates the corresponding grid size. We define the space of piecewise affine, discrete functions $\mathcal{V}_h$ and consider the fully discrete weak formulation of the elliptic problem (1) in $\mathcal{V}_h$. We ask for a discrete function $U \in \mathcal{V}_h$ such that

$$\int_{\mathcal{M}_h} A \nabla_{\mathcal{M}_h} U \cdot \nabla_{\mathcal{M}_h} \Phi = \int_{\mathcal{M}_h} f\, \Phi, \tag{3}$$

for all discrete test functions $\Phi \in \mathcal{V}_h$. Functions U in $\mathcal{V}_h$ can be represented by nodal vectors $\bar{U}$ in $\mathbb{R}^n$, where $\bar{U} = (U_j)_{j=1,\dots,n}$ is a vector with components U_j. Let $\{\Phi_i\}_{i=1,\dots,n}$ be the usual nodal basis of $\mathcal{V}_h$ with $\Phi_i(x_j) = \delta_{ij}$ for all vertices x_j of the grid $\mathcal{M}_h$. We can express a discrete function U in terms of its nodal values $U_j = U(x_j)$ and get $U = U_j\, \Phi_j$. The discrete problem can be expressed in matrix vector notation by

$$L\bar{U} = M\bar{F},$$

where the mass matrix M and the stiffness matrix L are given by

$$M = \left(\int_{\mathcal{M}_h} \Phi_i \Phi_j \right)_{i,j}, \tag{4}$$

$$L = \left(\int_{\mathcal{M}_h} A \nabla \Phi_i \cdot \nabla \Phi_j \right)_{i,j}, \tag{5}$$

and $\bar{F} = (f(x_i))_i$ is the vector of nodal values $f(x_i)$ at vertices of the triangulation x_i. Hence, the discrete elliptic operator in matrix form turns out to be $M^{-1}L$. In case of the parabolic problem (2), we consider a time discretization with time step τ and have to find a sequence $\{U^k\}_{k=1,\dots,} \subset V^h$ of approximations to the continuous solution $(U^k(\cdot) \approx u(\tau k, \cdot))$ such that

$$\int_{\mathcal{M}_h} \frac{U^{k+1} - U^k}{\tau} \Phi + \int_{\mathcal{M}_h} A[U^k] \nabla_{\mathcal{M}_h} U^{k+1} \cdot \nabla_{\mathcal{M}_h} \Phi - f[U^k]\, \Phi = 0$$

for all $\Phi \in V^h$. Note that $\mathcal{M}_h = \mathcal{M}_h(k\tau)$ if the surface itself is evolving as e.g., in case of surface fairing applications. We obtain, for each time step of our problem, the system of linear equations

$$(M + \tau L[U^k]) \bar{U}^{k+1} = M(\bar{U}^k + \tau \bar{F}[U^k]),$$

where the stiffness matrix depends on the discrete solution, i.e.

$$L[U] := \left(\int_{\mathcal{M}_h} A[U] \nabla \Phi_i \cdot \nabla \Phi_j \right)_{i,j}.$$

Algorithmically, the integral expressions in (4) and (5) are split up into a sum over local contributions on triangles. The matrices and right-hand side vector are computed as follows. We initialize $L = 0$ and next do a traversal of all triangles $T \in \mathcal{M}_h$. On each T with nodes P^0, P^1, P^2, a corresponding local matrix $(l_{ij}(T))_{ij}$ is computed first, corresponding to all pairings of local nodal basis functions, and next added to the matching locations in the global matrix L. For every pair i,j we update $L_{\alpha(i),\alpha(j)} = L_{\alpha(i),\alpha(j)} + l_{ij}(T)$. Here $\alpha(i)$ is the global index of the node with local index i. We proceed similarly with the mass matrix M.

5. Differences for point based surfaces

One faces several difficulties when aiming to transfer the PDE discretization approach outlined above to point cloud surfaces. Conceptually, such surfaces are not described in terms of a two dimensional set in $\mathbb{R}^3$. In particular, there is no global mesh available and, as outlined in Section 1, it would conflict with the general paradigm of point based modeling to replace the usually huge unstructured point set by a standard mesh. In particular, one would stop working on the actual, usually noisy, arbitrarily sampled data, along with their statistical properties, and completely replace them by a mesh having other properties. The standard method for handling point surfaces is to extract a local approximate tangent space on the point cloud [2,4,8,13]. This tangent space is generally used just for computing point normals used e.g., in shading. We will use this idea to obtain proper discrete counterparts of the differential operators $\mathrm{div}_{\mathcal{M}}$ and ∇_M and a metric for the discrete integration over the surface. Hence, we proceed by constructing a local tangent space and consider the local projection of the point set onto this tangent space. We are then able to define stable coupling quantities between neighbor points, using a *strictly local* Delaunay meshing. The mentioned coupling quantities to be defined will turn out to be suitable discretizations of the off-diagonal entries in the global stiffness matrix we aim to recover. The local tangent spaces of different points usually do not coincide, which induces a loss of symmetry in our matrix. To remove this problem, we finalize the matrix construction by applying a suitable symmetrization. Finally, the diagonal entries of the stiffness matrix can be defined based on a requested invariance property. Indeed, the continuous differential operator $\mathrm{div}_{\mathcal{M}}(A\nabla_{\mathcal{M}}\cdot)$ applied to a constant function u should vanish. Hence, we require that $L\bar{U} = 0$ for $\bar{U} = (1,\ldots,1)$. We proceed similarly for the mass matrix and the right-hand side of the considered PDE. The complete approach is detailed in the following sections, leading to a stable and consistent approximation scheme for general PDEs.

6. Tangent spaces and local meshes

We proceed by defining, for every point x in the considered cloud, a *tangent space*. This space attempts to approximate the points in a small neighborhood N of x. The size of N should be chosen such that (a) it contains enough points for stable computation of a tangent space and (b) the radius of N is smaller than the feature size we want to be visible in the approximation. For (a), N can be efficiently computed using the k nearest neighbors of x, for given k. For (b), N can be defined as the ball $B_\varepsilon(x)$ of given radius ε centered at x. In practice, combinations of the two criteria give the best results. We prescribe a minimum number of neighbors k_{min}, to enforce the first requirement. If the k_{min}^{th} closest neighbor of x is closer than the prescribed minimal feature size ε, we consider all additional nearest neighbors in $B_\varepsilon(x)$. For relatively uniformly sampled surfaces, the first criterion is, by itself, sufficient to guarantee a stable tangent plane computation, if k is taken large enough, as discussed later in this section. For non-uniformly sampled surfaces, the second criterion guarantees that the neighborhood N has a minimal size ε. The choice of ε, in connection with the sampling rate, should provide a stable tangent plane computation. The combination of these two criteria have given good results for all point surfaces we considered. Obviously, highly non-uniformly sampled surfaces may exist for which the above criteria either yield unstable tangent planes or produce too much smoothing. However, most point surfaces encountered in practice are densely sampled, as they try to capture as many surface details as possible. Moreover, as explained later in this section, we use the tangent planes just as an instrument to compute local neighborhoods. For this purpose, we can tolerate some amount of orientation error in the plane computation.

Next, we use the *zero and first order moments* of N. Moments have several proven properties that allow us to robustly compute the tangent spaces as well as to distinguish between smooth and non-smooth surface parts, both as a function of the ball size ε. Robustness is clearly needed in the tangent plane computation. Distinguishing smooth from non-smooth surface areas is needed for our surface segmentation (Section 8.1) and fairing (Section 8.5) applications. In this section, we give the moment definitions and properties relevant for the tangent plane computation. Next, we discuss the concrete moment-based implementation of the tangent planes.

For a continuous surface $\mathcal{M}$, the zero moment is given by the local barycenter of $\mathcal{M}$ with respect to a Euclidean ball $B_\varepsilon(x)$ centered at x:

$$M_\varepsilon^0(x) = M_\varepsilon^0 := \fint_{B_\varepsilon \cap \mathcal{M}} x \, \mathrm{d}x. \tag{6}$$

The first order moment is then defined as

$$M_\varepsilon^1(x) := \fint_{B_\varepsilon \cap \mathcal{M}} (x - M_\varepsilon^0) \otimes (x - M_\varepsilon^0)\, \mathrm{d}x$$
$$= \fint_{B_\varepsilon \cap \mathcal{M}} (x \otimes x - M_\varepsilon^0 \otimes M_\varepsilon^0)\, \mathrm{d}x, \qquad (7)$$

where $y \otimes z := (y_i z_j)_{i,j=1,\dots,3}$. It turns out that the first moment approximates the matrix $\Pi_{\mathcal{T}_x \mathcal{M}}$ corresponding to the projection onto the tangent space $\mathcal{T}_x\mathcal{M}$. Indeed, in smooth surface regions we have:

$$M_\varepsilon^1(x) = 2c\varepsilon^2 \Pi_{\mathcal{T}_x\mathcal{M}} + o(\varepsilon^2),$$

where c is a constant that only depends on the dimension. For a proof, we refer to [17]. This shows that the eigenvectors of the first moment define an orthonormal basis of $\mathbb{R}^3$, where the surface normal belongs to the vanishing (smallest) eigenvalue. For a discussion of the non smooth case we refer to again [17]. Let $\lambda_0 > \lambda_1 > \lambda_2$ be the eigenvalues of the 3 by 3 symmetric matrix M_ε^1, then we consider the corresponding eigenvector e_2 as the normal on the approximate tangent plane, whereas e_1 and e_0 form a 2D coordinate system in the plane itself. Fig. 1a illustrates the above in two dimensions. The zero and first moment (Eqs. (6) and (7)) are computed numerically as sums over the sample points. Because of this, our tangent plane computation is very similar to the principal component analysis, or so-called 'surface variation', used in [2,8,13]. A more exact discretization of the moment integrals is presented in [17] for triangular meshes. We could use this discretization for point-based surfaces too, by using the local triangulations presented below in this section. However, this makes the computations slower. Essentially, the radius ε in both our moment-based and the surface variation approaches has the role of a filter size: The tangent plane ignores features significantly smaller than ε.

Once the tangent plane $\{x \in \mathbb{R}^3 \mid e_2 \cdot (x - x_i) = 0\}$ is defined for a point x_i, we project all neighbors x_i^j in the neighbor set N_i onto it, yielding the projected point set $N_i^p = \{X_i^j\}_{j=0,\dots,k}$ in the 2D coordinate system (e_0, e_1) (Fig. 1a). Here $X_i^j \in \mathbb{R}^2$ are the projections of $x_i^j \in \mathbb{R}^3$ onto the above 2D coordinate system. To simplify notation, we incorporate x_i as x_i^0 in the set of neighbors N_i^p. Next, we compute the Delaunay triangulation of the points N_i^p. This yields a triangle mesh $\mathcal{T}_i$ in the tangent plane. The triangulation is a strictly 2D process, confined to the frame (e_0, e_1). From the triangulation, we select the triangle fan $\mathcal{F}_i^p = \{T_i^p\}_i$ of projected triangles T_i^p around the projected seed point X_i^0 (Fig. 1b). Finally, we define the neighbor set $\mathcal{N}_i^p$ of x_i as being the points x whose projections X_i^j are used by the triangles in the fan $\mathcal{F}_i^p$. By $\mathcal{N}_i$ we denote the corresponding set of 3D points before projection on the tangent plane.

Note that the points $\mathcal{N}_i$ define a local 3D triangle fan $\mathcal{F}_i$ of the point set whose projection on the tangent plane is exactly the triangle fan $\mathcal{F}_i^p$ defined by the point set $\mathcal{N}_i^p$.

However, the above scheme has a problem. The Delaunay triangulation may produce triangles with too small angles which, when assembled in matrices discretizing PDEs, can cause inaccuracy and instability problems when solving these PDEs (Section 7). These problems are well known from discretizing PDEs on ill-conforming meshes. We prevent this as follows. If an angle smaller than a user given α_{min}, set in practice to around 25 degrees, appears in the triangulation, we remove one of its points from N^p and re-triangulate the remainder. The process is repeated until no ill-shaped triangles are created. In practice, this causes no visible slow-down. We tested a large number of noisy point sets of 30 000 up to a million points. The worst case encountered contained a few tens of such triangles per point set, which were successfully removed in three re-triangulation passes. Even though these cases are rare, their removal is essential to ensure robust convergence of PDE discretization schemes.

Our tangent plane construction has several desirable properties. First, the moment-based computation is a noise-robust way to define the tangent plane. Larger neighborhoods N act as stronger noise filters. It is important to note that our approach is *not* the same as producing a smoothed mesh. Indeed, the neighbors $\mathcal{N}_i$ of a point x_i are defined to be only the *immediate* neighbors of x_i in the Delaunay triangulation of the projected neighborhood N_i^p. As N_i increases due to increase of ε or k_{min}, the set $\mathcal{N}_i$ practically stays of constant size. In practice, $\mathcal{N}_i$ contains the average number of points in a conforming Delaunay triangulation, i.e. 4 up to 8..10 points, whereas the average size of N_i, for the point sets we worked with, ranged between 30 and 100 points. Secondly, the computation of (e_0, e_1, e_2) does *not* need to be very accurate. We use them just as a means of finding the neighbor set $\mathcal{N}_i$ out of the points in

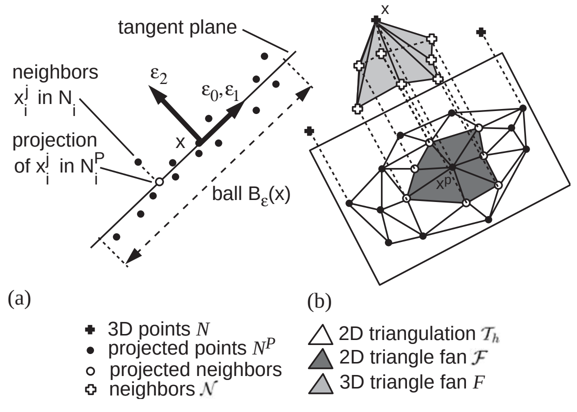


Fig. 1. Tangent plane (a) and local triangulation (b).

the point set. Even reasonably large orientation variations of e_0, e_1 cause no change in $\mathcal{N}_i$, as the Delaunay triangulation of $X^j_{i\,j}$ uses the same closest points. We do not produce new points, but just couple the existing ones. Finally, the removal of ill-shaped triangles ensures that our stiffness matrices (Section 7 and further) are well conditioned, a property which is not directly enforced by classical finite elements on arbitrary triangle meshes.

Our method of triangulating tangent plane projections for the k closest neighbors resembles the local triangulation proposed by Linsen et al. [4]. However, we consider large k values (30 up to 100). Linsen et al. use $k = 6$, which should lead to considerably less stable tangent planes. Our triangulation quality check enforces minimal angles α_{min}, whereas [4] does not guarantee this. A very similar method to the triangulation proposed in this paper is presented by Floater and Reimers, in the context of triangulating unorganized point sets [18].

Fig. 2 (left) shows the bunny model in which we chose three points on the left ear. The points, their neighborhoods N for $k = 60$, and lines to their neighbors $\mathcal{N}_i$ are shown in red, yellow, respectively green in the detail image Fig. 2 (right). Note that, although each N_i is large (60 points), the corresponding $\mathcal{N}_i$ has 6 neighbors. Moreover, the neighbor set $\mathcal{N}_i$ stays the *same* for $k \in [20..60]$, which outlines the stability of our method.

7. Assembling the finite element matrices

We have shown how to construct, for every x_i in a point set, the neighbor set $\mathcal{N}_i$. In this section, we show how to build the matrices needed for solving PDEs on point surfaces. Given the local 3D triangle fan $\mathcal{F}_i = \{T_l\}_l$, we define the preliminary matrix entry $\tilde{L}_{ij}$ as

$$\tilde{L}_{ij} = \sum_l \mathrm{A}\, \nabla_{T_l} \Phi_i \cdot \nabla_{T_l} \Phi_j \, |T_l|, \qquad (8)$$

where Φ_j are the affine linear basis functions on the triangles of F defined by $\Phi_k(x^j_i) = \delta_{kj}$ for all k and j. A is the application-dependent discrete diffusivity term. Here the gradient ∇_{T_l} is the gradient on the affine triangle T_l.

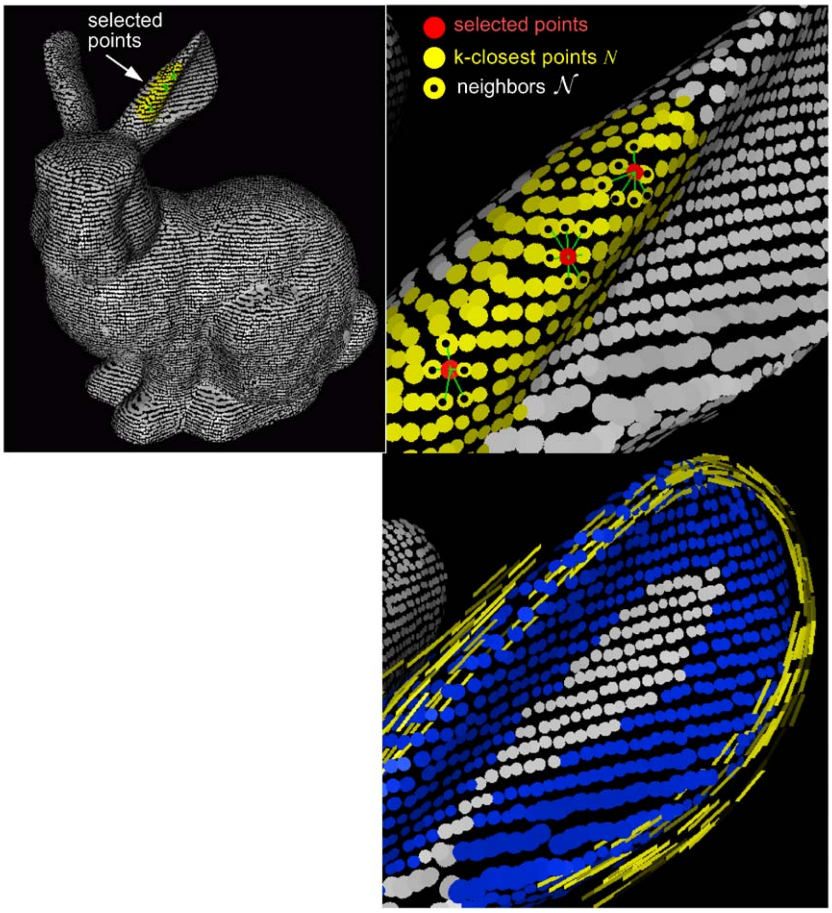


Fig. 2. Point set (left). Three selected points with neighbor sets N_i and $\mathcal{N}_i$ (right). Major eigenvectors along edges (bottom).

We could alternatively define $\tilde{L}_{ij}$ by integrating *in the tangent plane* only, i.e., on the triangles of $\mathcal{F}_i^p$, instead of $\mathcal{F}_i$. The right choice is application dependent. If we process (e.g., denoise) the surface itself, it is incorrect to use projected quantities, as they do not take into account the spatial orientation of the points. Indeed, this would couple points in flat regions as strongly as points in curved regions. As already outlined, if we compute on the 3D triangle fan, we use the tangent planes just as a help for the triangulation and obtaining the neighbor relations $\mathcal{N}^p$, and perform all other computations in 3-space. When processing a fixed and very noisy point cloud, the smoothing induced by the tangent space construction may be desirable. In that case, one would replace the 3D triangle fan $\mathcal{F}_i$ by its projection $\mathcal{F}_i^p$.

The quantity $\tilde{L}_{ij}$ describes the coupling of point i with all its neighbors j, from the point of view of i. Clearly, $\tilde{L}_{ji}$ is not necessarily equal to $\tilde{L}_{ij}$, as the neighbor computations of i and j are strictly speaking independent (Section 6). Moreover, we must still define a point's self-coupling, i.e. the matrix's diagonal entries. To produce a complete, symmetric 'stiffness' matrix L, we now define

$$L_{ij} = \frac{1}{2}(\tilde{L}_{ij} + \tilde{L}_{ji}), \tag{9}$$

for $i \neq j$ and for the diagonal entries

$$L_{ii} = -\sum_{x_j \in \mathcal{N}(x_i)} L_{ij}. \tag{10}$$

The latter ensures—as already mentioned in Section 4—the desirable property that $L(1,\dots,1)^{\mathrm{T}} = 0$. The matrix L has now the same properties as the classical stiffness matrix on a triangular mesh. However, L is *not* produced via a global triangulation, but via our local, on-the-fly triangulation.

Finally, for the mass matrix M, we consider a diagonal, lumped mass matrix, and set

$$M_{ii} = \frac{1}{3}\sum_l |T_l|. \tag{11}$$

Here, as for the stiffness matrix, we can either consider triangles in the 3D fan $\mathcal{F}_i$ or alternatively their 2D projections in the projected fan $\mathcal{F}_i^p$.

8. Applications

8.1. Surface segmentation

Nonlinear diffusion methods are well known in image processing applications [19,20]. In these applications, one solves Eq. (2), where u is the scalar gray value or vector-valued image color. Time plays the role of a scaling parameter: $u(t=0)$ is the initial image, and $\{u(t)\}_{t>0}$ is a family of progressively smoothed images. Appropriate choices for the diffusivity A and source term f yield different diffusion types. For example, $A = 1$ and $f = 0$ gives the well known heat equation with its isotropic smoothing effect. A better choice for image processing is to set A small in areas where we want to keep image details and large in areas where we desire strong smoothing. Finally, we can enforce the diffusion direction to follow the feature lines (Fig. 3).

As a more challenging application, we consider the segmentation of regions on a surface $\mathcal{M}$ which are bounded by sharp edges. For this, we use a diffusion process where we limit diffusion across and close to edges and have it large in smooth areas. For this, we can set $A = \mathcal{C}_\varepsilon$, where

$$\mathcal{C}_\varepsilon = G\left(\frac{||M_\varepsilon^0(x) - x||\lambda_2(M_\varepsilon^1(x))}{\varepsilon\lambda_0(M_\varepsilon^1(x))}\right),$$

with $G(s) = (\alpha + \beta s^2)^{-1}$ with suitably chosen $\alpha, \beta > 0$. In all our applications, we have fixed $\alpha = 0.01$ and $\beta = 100$. The function G causes $\mathcal{C}_\varepsilon$ to be much larger in relatively smooth surface areas than close to edges, thus makes the surface classification easier.

Fig. 4 shows the classifier $\mathcal{C}_\varepsilon$ for the bunny model, for different values of ε. Specifically, ε was implicitly determined as the distance from the current point to the kth closest point, for different values of k. Using a blue-to-red (rainbow) colormap for $\mathcal{C}_\varepsilon$, smooth regions appear red, whereas edges appear blue. Moreover, surface 'features' are detected at different scales, the scale being given by the value of ε. These results are very similar to the multiscale features presented by Pauly et al. for point sets [21]. The main difference is that we use the 'enhancement' function G to clearly separate, by several orders of magnitude of $\mathcal{C}_\varepsilon$, smooth areas from edges. For our applications (e.g., segmentation), this strong separation is essential.

This is the surface classifier presented in [17] which is small in the vicinity of edges on the surface and almost 1 in smooth surface regions. However, the above choice for A may not stop diffusion *completely* close to edges, which is what we need for segmentation. To this end, we set $A = H(\mathcal{C}_\varepsilon)$ and $f = K(u)$, where

$$H(u) = \begin{cases} 0; & u > \gamma, \\ \alpha(u-\gamma)^q; & u \leqslant \gamma, \end{cases}$$

$$K(u) = \begin{cases} 0; & u > 1, \\ \alpha(1-u)^q; & u \leqslant 1, \end{cases}$$

for suitably chosen $0 < q \ll 1$, $\alpha > 0$, and $\gamma > 0$. In practice, a good choice is $q = 0.5$, $\alpha = 1$, and $\gamma = 0.05$. Given that $\mathcal{C}_\varepsilon$ ranges from very small close to edges to 1 in flat areas (Section 6), our choice for γ, and subsequently for H, ensures that diffusion is zero close to edges and strong in smooth surface areas. We set the initial condition $u_0 = 0$

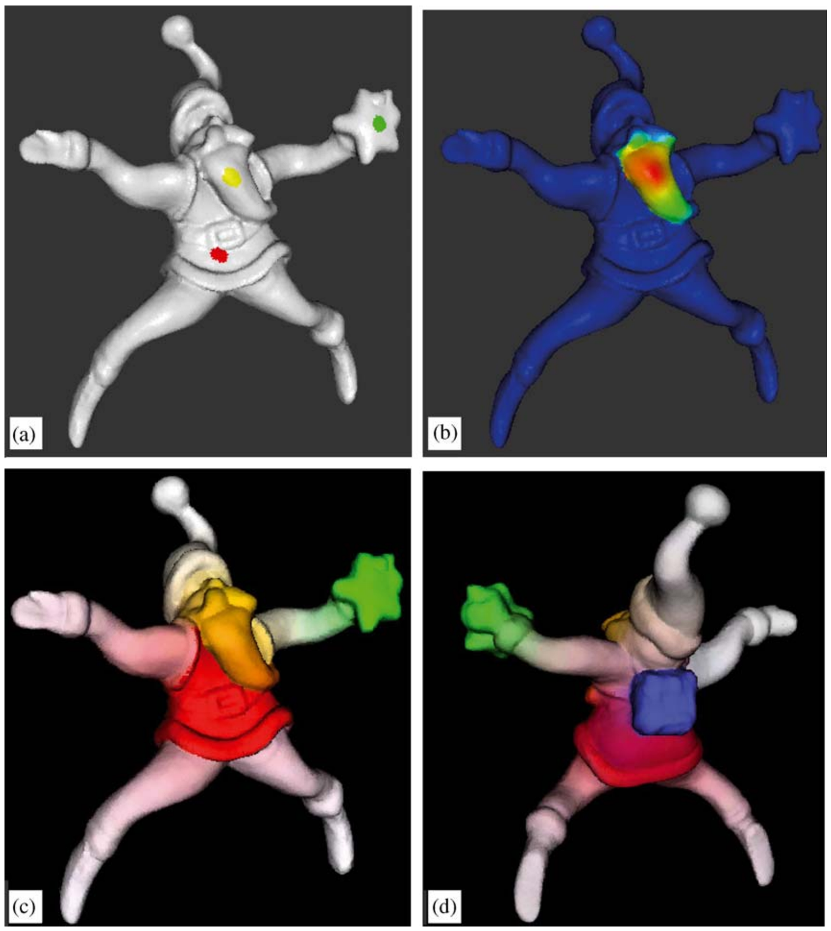


Fig. 3. Seed points (a), diffused signal u for yellow seed after 5 iterations (b), and two views of the obtained segments (c, d) for a surface having 75 781 points.

over the whole surface $\mathcal{M}$, except for a small hand-picked seed area within the region to be segmented, where we set $u_0 = 1$. The diffusion process stops spreading the seed intensity and stop at the surrounding edges, due to the choice of A. Furthermore, the right-hand side f serves as a contrast enhancement, which pushes u to the value 1 at any position which has a positive u value. Figs. 3 and 5 illustrate the segmentation process on two different point set surfaces, where we used different colors for every region and corresponding seed. The colors' saturations correspond to the diffused signal u. In Fig. 3 we show the use of $A = \mathcal{C}_\varepsilon$, which causes a very small amount of diffusion to leak out of the segmented regions. This is visible as the light red and green tints on the model's arms that come from the respective red and green segmented regions. Exact segments can be easily obtained by e.g., by using an upper threshold on the signal u. A better choice is shown in Fig. 5 where we use the second option $A = H(\mathcal{C}_\varepsilon)$. Here, the obtained segments are clearly separated by white areas. These areas correspond to high curvature regions, where the function H has zero values which completely block diffusion.

One issue is how to perform the seed point generation automatically. Several possibilities exist. For example, we can detect the 'flattest point' (by using the maximum of the classifier C_ε). This point is seeded, and its corresponding surface segment is computed via diffusion. We can repeat the process to get further segments, by eliminating the already segmented points from the detection step. This method does not require more diffusion steps as compared to the manual seed point placement.

Computing the diffusion-based segmentation is efficient, as the matrix A needs to be assembled just once. On a Pentium IV 1.8 MHz machine, one diffusion iteration takes 0.3 s for the 75 781 points model in Fig. 3 and 0.57 s for the 121 723 points model in Fig. 10.

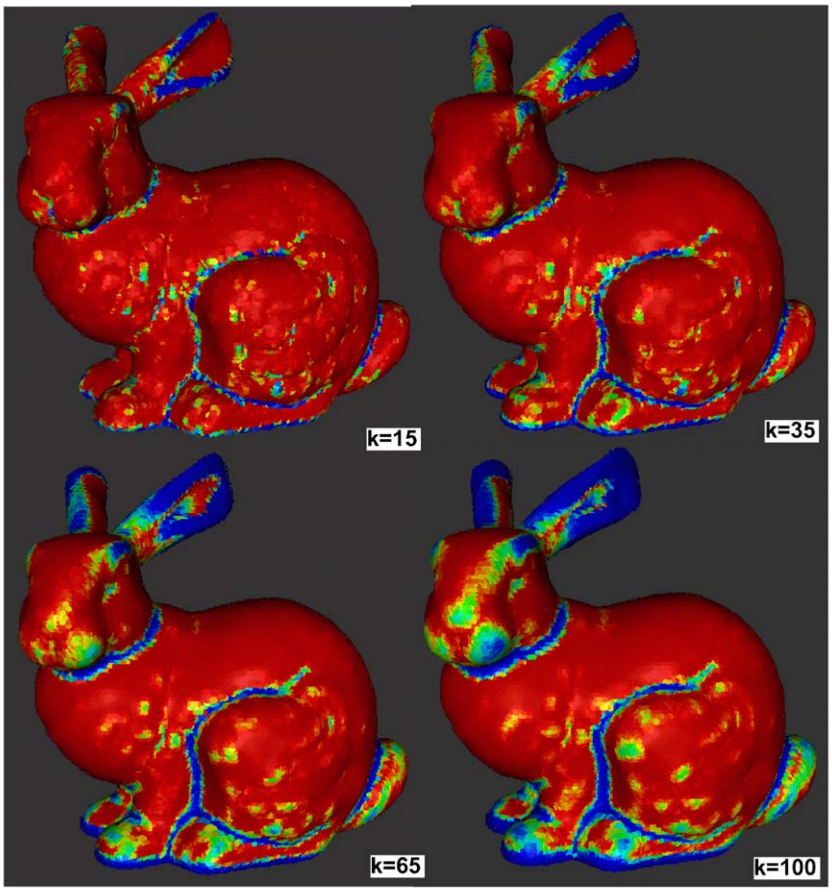


Fig. 4. Classifier for different *k*-closest point values.

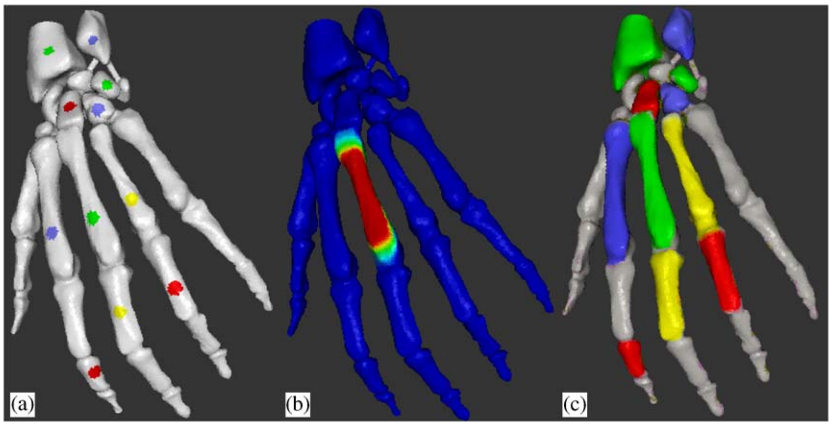


Fig. 5. Seed points (a), diffused signal for a seed after 15 iterations (b), and obtained segments (c) for a surface having 65 500 points.

Segmentation typically needs 10 to 20 iterations. The above performance figures could probably be improved by a factor of 3 by using a more efficiently coded linear solver.

8.2. Texture synthesis

We describe now the use of PDEs to generate textures on point surfaces, using the reaction diffusion method

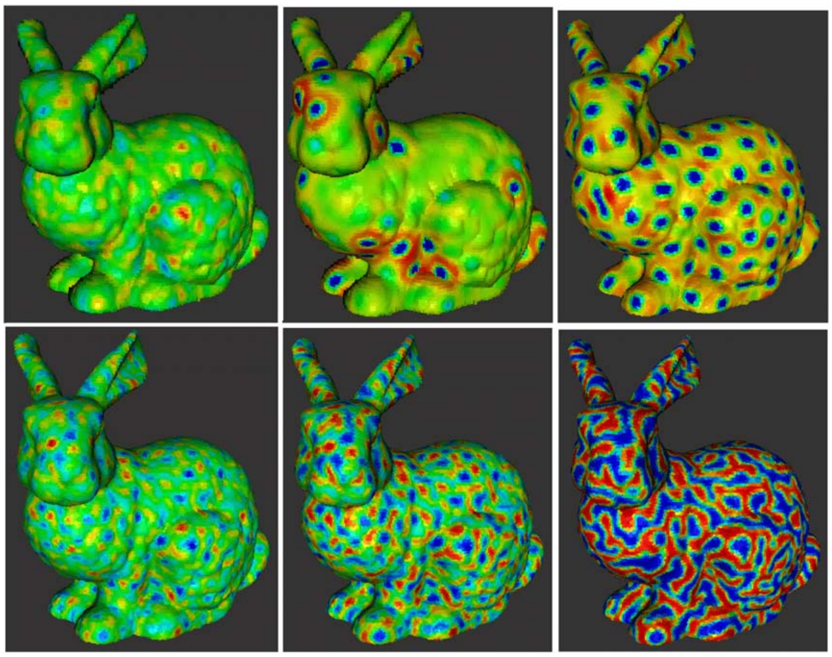


Fig. 6. Top row: spot texture synthesis. Bottom row: stripes texture synthesis.

presented by Turk in [16]. This method uses two 'chemical species' concentration functions a and b that diffuse and react, i.e., build up or annihilate each other, on a given surface. The process is described by

$$\frac{\partial a}{\partial t} = F_a(a,b) + D_a \Delta a, \quad (12)$$

$$\frac{\partial b}{\partial t} = F_b(a,b) + D_b \Delta b, \quad (13)$$

where F_a and F_b are the creation rates and D_a and D_b are the diffusivities of the species a and b, respectively. The system is initialized with constant a and b values biased by a small random perturbation. After several iterations, regular patterns appear (cf. Fig. 6, top row). By using a five species system, stripe-like patterns can be generated (cf. Fig. 6, bottom row). We have used exactly the same PDEs and parameter settings as the original work by Turk [16].

We discuss first the synthesis of spots and stripes textures. Fig. 6 shows these types after 100, 600, and 1700 iterations for the spots (top row) and 100, 300, and 600 iterations for the stripes (bottom row). Here, we visualize the concentration (a or b) result of the reaction diffusion via a blue-to-red colormap. The patterns and the number of iterations needed are practically identical with the ones produced by [16].

Next, we synthesize a zebra-like pattern, by disabling the initial random perturbation and setting the initial concentration to a given value v. We extend Turk's method by forcing the zebra pattern to follow the surface edges by relating v to the moment-based classifier value. For this, we select all points where $\mathscr{C}_\varepsilon$ is closer to its minimum value than 10% of its range. This delivers points on and close to surface edges. We next set v to 1 on these points and 0 on the remainder and proceed with the texture synthesis. The species start diffusing from the surface edges (red regions in Fig. 7a) into the smooth areas (blue regions in Fig. 7a). Fig. 7a–c shows three instants of the zebra pattern formation. A second application is shown in Fig. 8. Here, the thresholded classifier selection allows us to easily select the 'bumpy features' of the dinosaur model, such as the fingertips, eye, and back ridge. The selected points are shown drawn in yellow in Fig. 8. Fig. 8b–d depict three instants of the zebra pattern formation, showing how the zebra stripes propagate from the seed set. These results are very similar to the zebra patterned animal presented in [16], where the seed selection was done, however, manually. On a Pentium IV 1.8 GHz machine, for the bunny dataset, the spot formation took around 30 s, the stripes 8 s, and the zebra pattern 18 s.

8.3. Bump mapping

A second application of reaction–diffusion equation is to generate bump mapped surfaces. For this, we solve the same set of Eqs. (12) and (13) as for texture generation. However, we use now the gradient ∇a of the computed species concentration a to bias the points' normal vectors $\mathbf{n}$:

$$\mathbf{n}_{bump} = \mathbf{n} - K\nabla a. \quad (14)$$

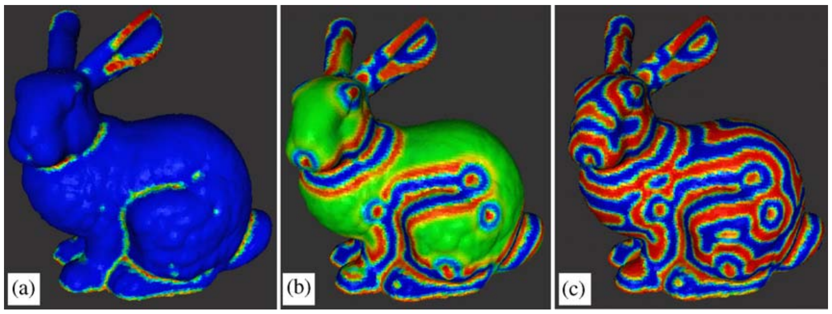


Fig. 7. Aligned zebra patterns after 10 iterations (a), 100 iterations (b), and 1300 iterations (c).

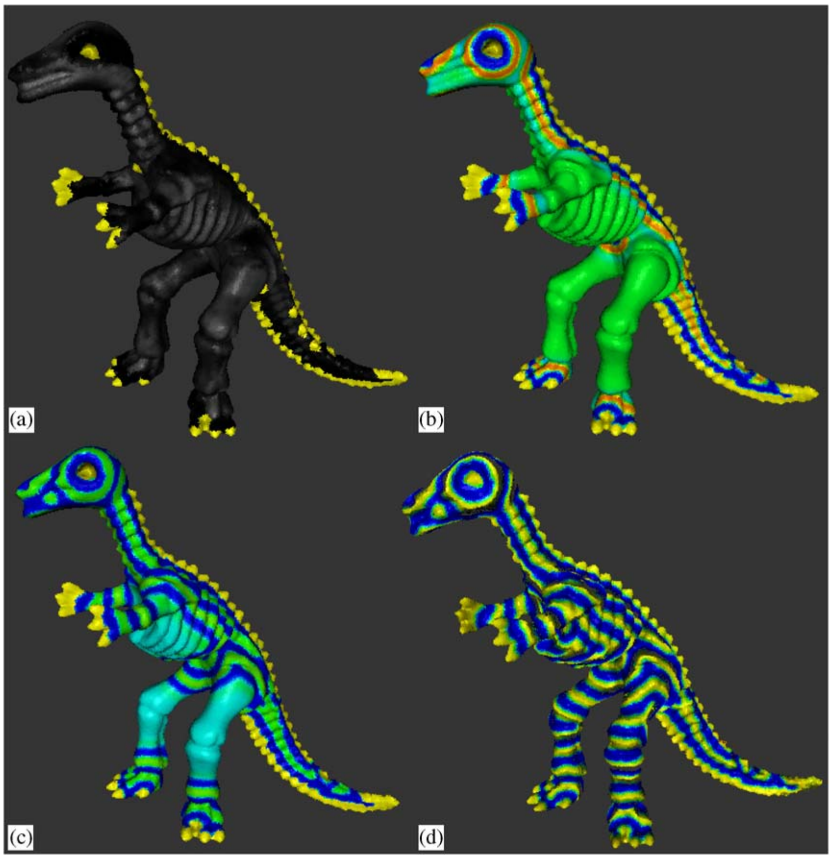


Fig. 8. Constructing zebra patterns. Selected seeds (a), patterns after 20 iterations (b), 100 iterations (c), and 900 iterations (d).

The bias factor K controls the amount of bump mapping. Good effects are obtained for $|K|$ values between 0.1 and 1. Rendering the point set using the biased normals $\mathbf{n}_{bump}$ produces the effect of a bump mapped point surface. Positive K values produce convex bumps, whereas negative values yield concave surface indentations.

Computing the gradient ∇a of the reaction–diffusion species a can be done in several ways. The first method is to consider, for every point x_i in the point set, its triangle fan $\mathscr{F}_i = \{T_l\}_l$, which is constructed during the stiffness matrix assembly process described in Section 7. Next, we compute the gradient $\nabla_{T_l} a$ on every triangle T_l as usually, i.e. by projecting the normal to the graph of a on T_l, and set ∇a by averaging all $\nabla_{T_l} a$ for all triangles T_l. This is the classical finite element setting, and is the method we have used in our implementation. A second method is described by Turk in [16]. Essentially, Turk's method involves interpolating a between the sample points x_i using 3D radial basis functions:

$$a(x) = \frac{\sum_{q \in B_\varepsilon(x)} a(q) w(|x-q|/d)}{\sum_{q \in B_\varepsilon(x)} w(|x-q|/d)}. \tag{15}$$

Only those points q are considered which are in the neighborhood $B_\varepsilon(x)$ of the point x. The radial basis functions w are defined as

$$w(r) = \begin{cases} 2r^3 - 3r^2 + 1; & 0 < r < 1, \\ 0; & r > 1. \end{cases}$$

Finally, the value d is taken to be the double of the average inter-point distance. With this scheme, ∇a is numerically computed by forward differences from the interpolation (15).

Fig. 9 shows several bump mapping examples. The bunny (Fig. 9a) has been bump mapped with convex bumps ($K = 0.5$) created from a spot texture (shown in Fig. 6, top row). For the statue head (Fig. 9b), we used concave bumps ($K = -0.3$) created from a similar spot texture. Finally, the object in Fig. 9c has been bump mapped by using an edge-following zebra texture. This produces bump structures that are parallel with the object's sharp edges (Fig. 9d).

Taking larger d values and larger neighborhoods B_ε acts like a low pass filter on ∇a, and thus produces softer bump mapping, at a higher computation cost. At the other extreme, the smallest neighborhood B_ε is actually $\mathcal{N}_i$, the neighbors of x_i used by its triangle fan $\mathcal{F}_i$. In this case, the normal estimation method using radial bases becomes practically identical with the one using linear finite elements.

An important question to answer is whether the above radial basis interpolation could be used to define a completely mesh-free PDE discretization approach for point surfaces. In principle, the answer is yes. However, two problems appear in practice. First, the above scheme does not provide a 'partition of unity', i.e. does not guarantee that the sum of all basis functions for all points x_i is identically one overall on the surface. Second, there is no efficient way to compute the integrands of the stiffness and mass matrices (Eqs. (4) and (5), Section 4) in case of radial basis functions. In contrast, evaluating these integrands reduces to simple sums over triangle fans (Eq. (8)) for the affine linear basis functions our approach uses.

8.4. Inpainting textures

Inpainting, originally an artist's work, is the process of repairing local damages in an image, or texture, by

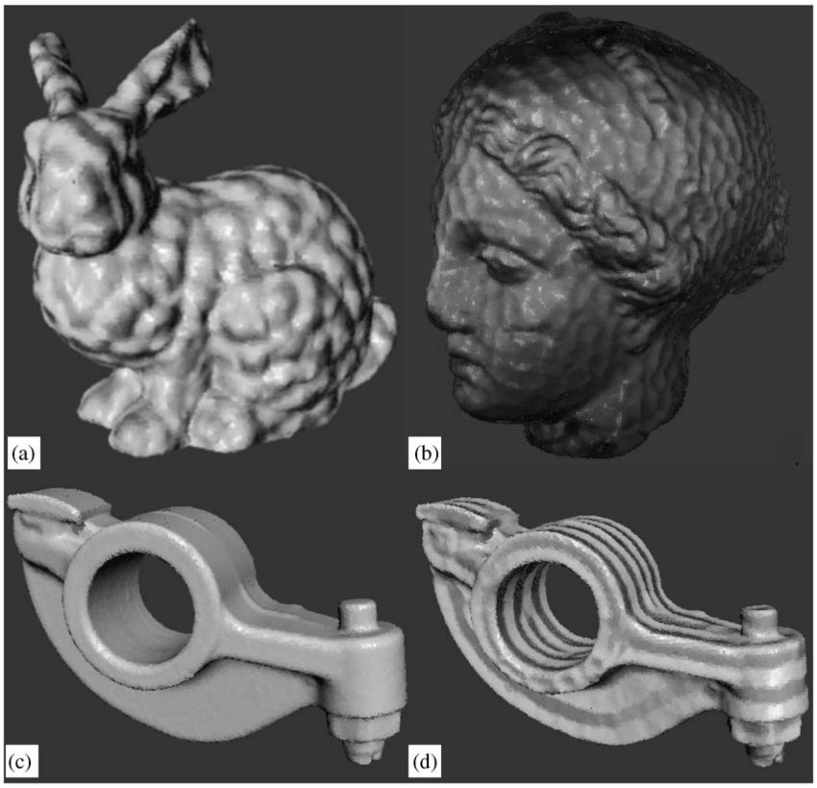


Fig. 9. Bump mapping examples.

using the image colors outside of, and close to, the damaged area to fill in, or 'paint in', the defect itself. Several inpainting methods exist which essentially try to restore the damaged area so that various properties (statistic data, gradient information) of the valid image are extrapolated in the damaged area in a natural way [22,23]. We demonstrate here a simple linear inpainting method which allows repairing the color texture on a damaged region $\mathscr{D}$ of a point set surface $\mathscr{M}$, by extending the texture on the boundary $\partial\mathscr{D}$ of $\mathscr{D}$ into the interior of $\mathscr{D}$. For this, we consider the boundary value problem (1) with $A = \mathscr{C}_\varepsilon$. This diffusivity choice prefers rather independent texture expansion on both sides of an edge. Hence, we avoid texture smearing across edges. Fig. 10 shows the inpainting of a model of 121 723 points. First, we created several defects by painting on the model. Such defects are shown in yellow in Fig. 10c,e. The texture was set to black in the defects area. Using the anisotropic diffusivity A (low close to creases, high in flat areas) for inpainting diminishes texture smearing close to creases: The model's black hair color and facial color are kept separate (Fig. 10c,d). The same happens with the leg's skin color and white shoe color (Fig. 10e,f). However, the black trouser color tends to diffuse on the left leg (Fig. 10b), as this region is flat. If desired, this can be prevented by using an anisotropy tensor A that incorporates color gradient information. We reserve this application, as well as using a more sophisticated inpainting model, for future work.

8.5. Fairing of point based surfaces

The next application of our framework for PDEs on point based surfaces is surface fairing using anisotropic geometric diffusion. Here, geometrical surface noise is smoothed out, whereas features such as edges are preserved or possibly even enhanced [11,24]. This is especially useful for point surfaces acquired via noisy scanning. Given an initial compact embedded manifold $\mathscr{M}_0$ in $\mathbb{R}^3$, we compute a family of faired manifolds $\{\mathscr{M}(t)\}_{t\in\mathbb{R}_0^+}$, with corresponding coordinate mappings $x(t)$. The time t describes the fairing process and $x(t)$ are given by solving the system of anisotropic evolution equations:

$$\partial_t x - \operatorname{div}_{\mathscr{M}}(A\nabla_{\mathscr{M}} x) = 0. \qquad (16)$$

We start with the initial condition $\mathscr{M}(0) = \mathscr{M}_0$. We define the tensor A such that we have strong diffusion along surface features and weak diffusion across them. As before, we use a moment-based classifier. When computing it, we also obtain a basis w^1, w^2 in the tangent plane $\mathscr{T}_x\mathscr{M}$, defined by the major and medium eigenvectors of the first order moment (Section 6). In this basis, the tensor A is defined as

$$A = \begin{pmatrix} 1 & 0 \\ 0 & \mathscr{C}_\varepsilon \end{pmatrix}.$$

Since $\mathscr{C}_\varepsilon$ is high in smooth regions and low close to edges and corners, Eq. (16) smooths the surface by keeping the features. Due to the anisotropy A, we enforce a signal enhancement in the direction of the eigenvector w^1. In the direction of w^2, the diffusion is proportional with the classifier $\mathscr{C}_\varepsilon$, i.e., strong in smooth areas and weak close to edges, which is exactly what we desire. For more details, we refer to [25], which describes this application, but without detailing the actual PDE discretization we consider here. Fig. 11 shows several results, all obtained with a few tens of diffusion iterations. The important surface edges, such as the bunny's ear edges, body-hip contact line, the transversal femur cut, and the chiselled letters 'A' and 'X', are preserved. Small 'noise' details, such as the bunny's skin ripples, bone irregularities, and stone graininess, are removed. Remark, also, that the point model of the carved stone (Fig. 11 right) exhibits a small (black) hole at the extremity of the bottom right ending of the letter 'A'. Part of the point model, this hole does not cause problems to the fairing process, which underlines the stability of the proposed approach.

8.6. Point set triangulation

As a last application, we construct a triangle mesh from a given point set. For this, we recall the process of local tangent plane and triangle fan construction (Section 6) where, for each point x_i in the point cloud, a triangle fan $\mathscr{F}_i = \{T_l\}_l$ is constructed. If a triangle T_l in the fan has the points with indexes (i, j, k) as vertices, and $i<j$, we keep T_l in our triangulation, otherwise we skip it. This ensures that, if the same triangle is generated as part of the local triangle fans of two different points i and j, it will be added only once to the triangulation. Finally, we render the resulting triangle set using the point set normals as vertex normals. Fig. 12(a, c) show two such triangulations. For visual comparison, Fig. 12d shows the point set rendering of the dragon rendered as triangulation in Fig. 12c, which look practically identical. For better insight in the triangulation quality, Fig. 12d shows a close-up on the right paw of the dinosaur triangulation from Fig. 12a.

It is important to stress that, given the purely local nature of our triangle fan construction, this method is *not* guaranteed to produce a perfect triangle mesh from the point cloud, in which e.g., there are no holes, every edge is shared by exactly two triangles, and there are no overlapping triangles. However, just as in the triangulation method of Linsen et al. [4], such problems occur very seldomly, e.g., tens of occurrences for a point cloud

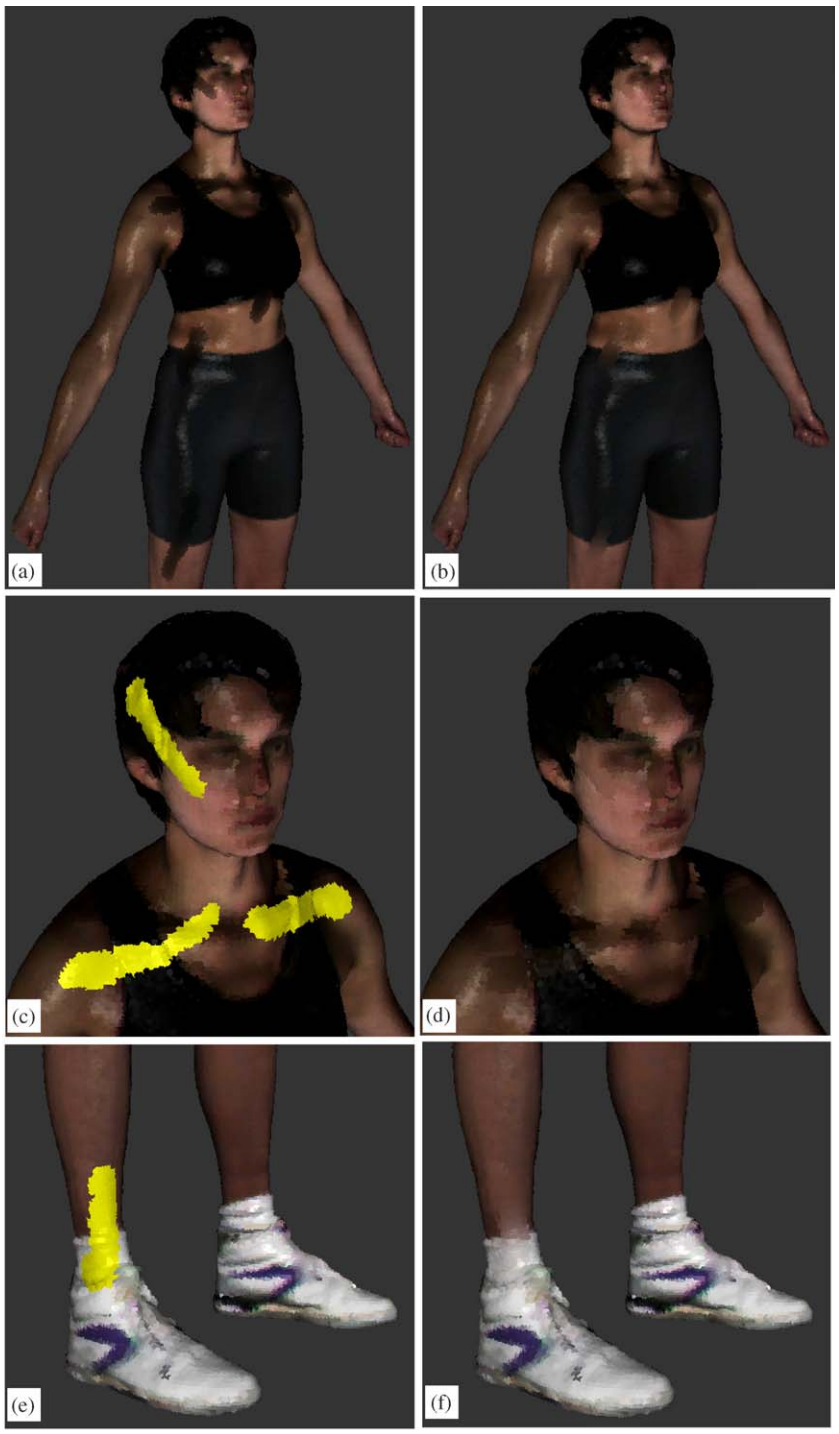


Fig. 10. Texture inpainting after 1 step (a) and 30 steps (b). Details: defects (yellow) (c, e) and their inpainting (d, f).

of 100 000 points. We have verified the above statement experimentally, by visual inspection of the rendered meshes. Since these problems occur seldomly, and their extend is purely local (pairs of points shared by more than two triangles, typically by three), believe that an extra pass can be devised that would remove them and create a manifold mesh. The computational cost associated with the triangulation is practically identical

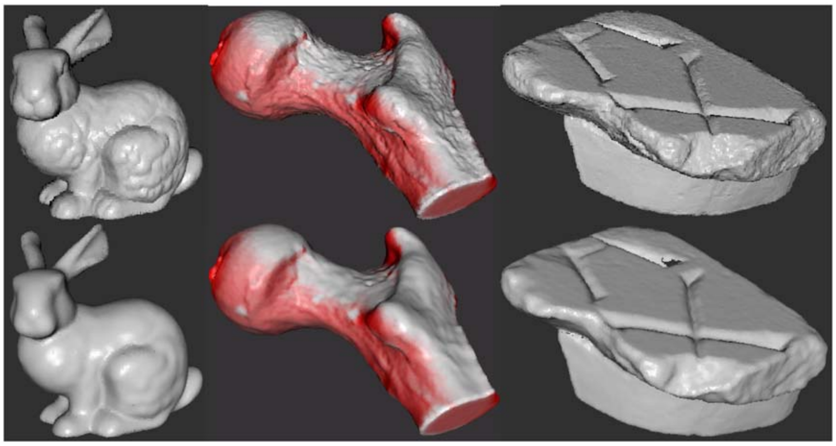


Fig. 11. Top row: initial surfaces. Bottom row: surfaces faired via diffusion.

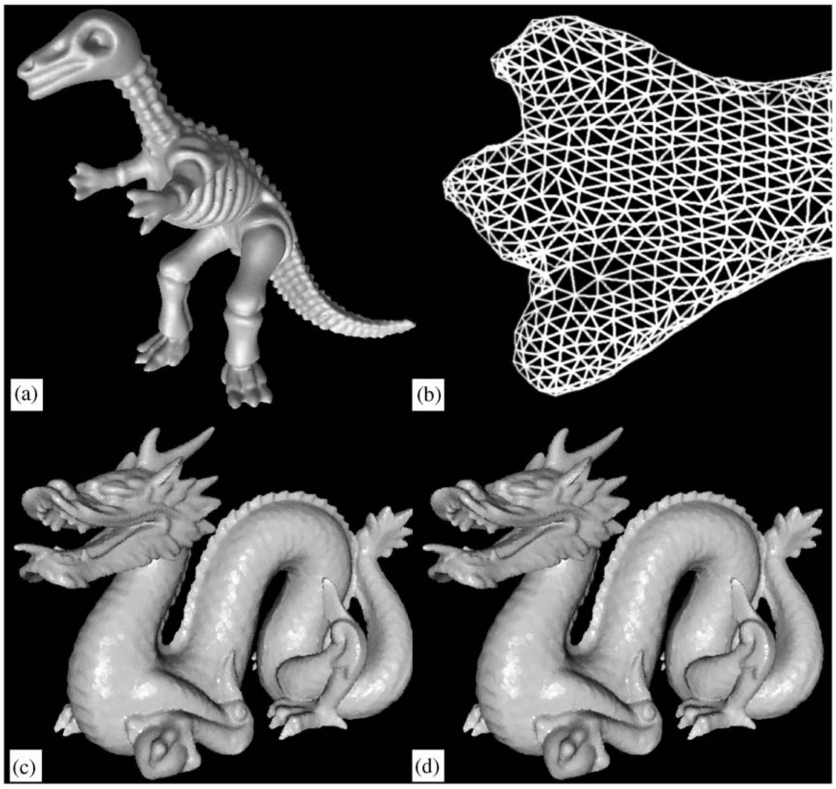


Fig. 12. Point set triangulation.

to the matrix assembly cost (more on this in Section 9). We present these triangulation results with the main aim of bringing further evidence to our claim that the purely local construction of finite element matrices (Section 7), based on the same local point couplings as the triangulation, produces robust results.

9. Implementation

Several aspects are essential for an efficient implementation. One of the costliest computations in the whole process is the nearest neighbor search used for the classifier and tangent plane computation (Section 6). We accelerate this search using the Kd and/or Bd trees provided by the ANN package [26], also used by the pointshop 3D point rendering system [27]. However, ANNs standard Kd and Bd tree implementations treat the (usually very numerous) points in the point cloud independently: searching the neighbors of every point implies, in a worst case, a full leaf-to-root search tree traversal. In many point sets, the points are not completely randomly distributed. Points geometrically close to each other come close to each other in the point vector too. One of the reason of this is the coherence inherent to the 3D scanning process by which many point models have been acquired. We use this to accelerate the neighbor search, as follows. We do not try to return the *exact* k closest points, but k points contained within a small given radius ε from a given point. These points are kept in a cache. If the cache is empty, we fill it by executing the standard k closest neighbor search. If the cache is not empty, it contains search results for the *previous* point, so we retain those k' points closer than ε from the *current* point. The cache miss, i.e. the remaining, usually few, $k-k'$ points are found by the usual tree search. This acceleration pays off as function of k. Indeed, as k increases, neighborhoods of close points will largely overlap. For k equal to 20, 50, and 100 closest points, we got a speedup factor of 2.62, 3.92, respectively 5.46 as compared to the standard tree search. This speedup was consistently observed for point sets between 100 000 and one million points. On our Pentium IV 1.8 GHz machine, for the point set and four k-closest-points values in Fig. 2, we need 0.4, 0.6, 1.05, and 1.83 s, respectively for the nearest neighbor search, tangent plane, and classifier computations.

For the Delaunay triangulation (Section 6), we used the triangle software [28] which provides efficient and robust checking and enforcing of various quality norms on the produced triangles, such as minimal angles. This is important for the conditioning of the assembled matrices (Section 7). We store the stiffness and mass matrices used to discretize the PDEs in a compressed row format that retains only the nonzero elements for each row. Given that there are as many nonzeros per row as points in a triangle fan (Section 7), i.e. usually less than 10, this scheme accounts for massive memory savings as compared to a full matrix storage. Finally, we solve the linear systems given by the above matrices using standard iterative techniques, such as conjugate gradient.

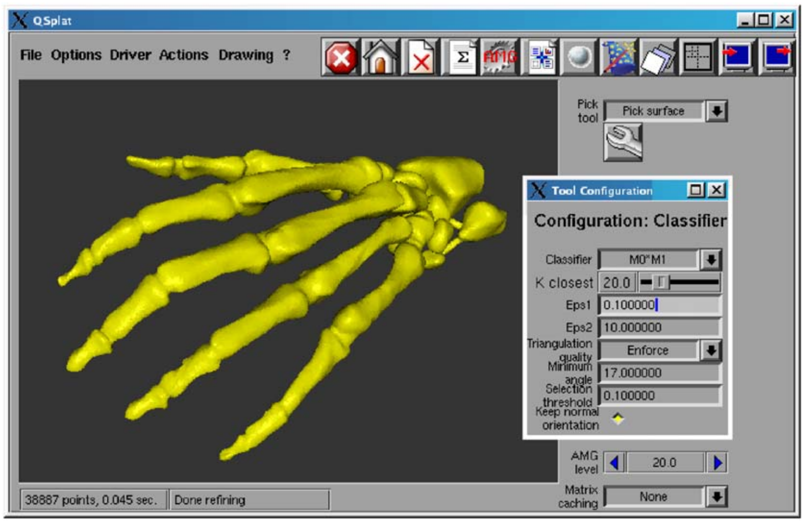
We built our system, called QSplat++, based on the QSplat rendering software [14] which uses a bounding sphere hierarchy to quickly and progressively render very large point sets. We perform all our moment, tangent plane, and PDE solving computations on the finest hierarchy level, i.e. the real points themselves. If desired, the color, normal, and position results can be propagated upwards in the hierarchy, so that we immediately benefit from QSplat's efficient rendering for very large models. We could also perform all our PDE computations on *coarser* hierarchy levels, e.g., to trade off accuracy for speed.

One of the strengths of point-based methods is that they can be easily restructured dynamically, i.e. allow point insertion and removal at various stages of the modelling process. For example, during a fairing process, one may want to remove points in shrunk surface regions, and insert points respectively in dilating regions. This implies recomputing both the point neighborhoods and the stiffness matrix. In our implementation of the fairing, we need to recompute the point normals, neighborhoods, classifier, and stiffness matrix, every few (2..3) smoothing steps [25]. Consequently, point insertion and removal are supported by default. However, there are applications where one would like to perform *local* point insertion or removal, and not have to reassemble the complete matrix. This can be achieved by computing the ε-neighborhoods of the inserted and removed points, and updating only the matrix entries that correspond to points in these neighborhoods.

All techniques described in this paper (texture synthesis, segmentation, fairing, inpainting, bump mapping, and triangulation), as well as a number of other standard painting and editing tools for point sets have been implemented using a modular, plugin-like architecture, similar to pointshop 3D [27]. Fig. 13 shows a snapshot from the user interface of QSplat++.

10. Conclusions

The main aim of the presented framework is to carry over the surface processing capabilities of finite element PDE methods, well proven for mesh based surfaces, to point based surfaces. Our framework can be seen as a two-scale approach. On the fine scale, we build local point couplings by using Delaunay triangulations of point projections on local tangent planes. The local couplings define fine-scale finite elements. It is only on this scale that the actual interpretation of the data as a function is clear and straightforward. On the next scale, we consider the different tangent spaces of different points, and average the first-scale FE models of these points to obtain the 'global' stiffness matrix (Section 7). To interpret data as a function on the second scale, one can average the function values on first-scale local triangles and interpret them as function values on interpolated points, where point interpolation is done by averaging point interpolations from the fine scale. We use the local tangent planes solely as a means of computing the point couplings. Thus, our approach differs from other methods on point clouds, such as [2,4,8,9,13]. Let us note that, given different surface approximations, like any produced by the afore cited



Pick tool: Pick surface

Configuration: Classifier

Classifier: M0*M1
K closest: 20.0
Eps1: 0.100000
Eps2: 10.000000
Triangulation quality: Enforce
Minimum angle: 17.000000
Selection threshold: 0.100000
Keep normal orientation

AMG level: 20.0
Matrix caching: None

38887 points, 0.045 sec. Done refining

Fig. 13. QSplat + + tool interface.

methods, we could easily extend our matrix assembly process to such surfaces, by reimplementing Eqs. (8) and (11) on this approximation.

Running our PDEs on the same surfaces represented as triangle meshes and point sets respectively, with the same parameter settings, produced virtually identical results. Let us emphasize that we avoid building a global surface representation. Our only global object is the stiffness matrix describing the PDE to solve. Assembling this sparse global matrix allows computing the point couplings only once. If desired, however, we could *completely* avoid assembling this matrix, e.g., by using iterative methods needing only one matrix row at a time, which is computed on the fly. Such approaches are well known e.g., in the field of progressive radiosity. In this case, when computing e.g., the matrix row i, one would proceed as follows: Compute all entries $\tilde{L}_{ij}$ for row i, then compute, for all neighbors j of point i, the entries $\tilde{L}_{ji}$, and finally symmetrize to yield the entries L_{ij} of row i. As compared to assembling the complete global matrix L, this would double the number of computations. However, this would allow storing only a single matrix row or column at a time, thus would allow processing huge point sets by trading speed for storage space.

Our framework can be extended in several directions. First, more types of PDEs could be solved, such as flow problems, by merely adapting the matrix assembly step. Secondly, one could extend the approach sketched in Section 8.6 to build consistent global triangulations from point clouds. Finally, multiresolution schemes on point surfaces can be built to accelerate the PDE solving to target interactive applications.

References

[1] Zwicker M, Pfister H, Van Baar J, Gross M. Surface splatting. In: Proceedings of ACM SIGGRAPH; 2001. p. 267–75.

[2] Pauly M. Point primitives for interactive modeling and processing of 3D geometry. Dissertation, Department of Computer Science, ETH Zürich, 2003.

[3] Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Surface reconstruction from unorganized points. In: Proceedings of ACM SIGGRAPH; 1992. p. 71–8.

[4] Linsen L, Prautzsch H. Global versus local triangulations. In: Proceedings of Eurographics (short presentations); 2001. p. 71–8.

[5] Clarenz U, Rumpf M, Telea A. Finite elements on point-based surfaces. In: Proceedings of Symposium on Point-Based Graphics (SPBG). The Eurographics Association. 2004. p. 201–11.

[6] Gopi M, Krishnan S, Silva CT. Surface reconstruction based on lower dimensional localized delaunay triangulation. In: Proceedings of Eurographics, vol. 19(3), 2000.

[7] Boissonnat JD. Geometric structures for three-dimensional shape representation. ACM Transactions on Graphics 1984;3(4):266–86.

[8] Alexa M, Behr J, Cohen-Or D, Fleishman S, Levin D, Silva C. Point set surfaces. In: Proceedings of IEEE Visualization; 2001. p. 21–8.

[9] Adamson A, Alexa M. Approximating and intersecting surfaces from points. In: Proceedings of Eurographics Symposium on Geometry Processing; 2003. p. 89–97.

[10] Taubin G. A signal processing approach to fair surface design. In: Proceedings of ACM SIGGRAPH; 1995. p. 351–8.

[11] Desbrun M, Meyer M, Schroeder P, Barr A. Implicit fairing of irregular meshes using diffusion and curvature

flow. In: Proceedings of ACM SIGGRAPH; 1999. p. 317–24.

[12] Kobbelt L. Discrete fairing. In: Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces; 1997. p. 101–31.

[13] Xie H, Wang J, Hua J, Qin H, Kaufman A. Piecewise c^1 continuous surface reconstruction of noisy point cloud via local implicit quadric regression. In: Proceedings of IEEE Visualization; 2003. p. 198–206.

[14] Rusinkiewicz S, Levoy M. QSplat: a multiresolution point rendering system for large meshes. In: Proceedings of ACM SIGGRAPH; 2000. p. 343–52.

[15] Varshney A, Kalaiah A. Differential point rendering. In: Proceedings of 12th Eurographics Workshop on Rendering; 2001. p. 139–50.

[16] Turk G. Generating textures on arbitrary surfaces using reaction–diffusion. Computer Graphics, SIGGRAPH'91 Proceedings, vol. 25(4). 1991. p. 289–98.

[17] Clarenz U, Rumpf M, Telea A. Robust feature detection and local classification for surfaces based on moment analysis. IEEE TVCG. 2004;10(5):516–24.

[18] Floater M, Reimers M. Meshless parameterization and surface reconstruction. Computer Aided Geometric Design 2001;18:77–92.

[19] Kimmel R. Intrinsic scale space for images on surfaces: the geodesic curvature flow. Graphical Models and Image Processing 1997;59(5):365–72.

[20] Preusser T, Rumpf M. Anisotropic nonlinear diffusion in flow visualization. In: Proceedings of IEEE Visualization; 1999.

[21] Pauly M, Keiser R, Gross M. Multi-scale feature extraction on point-sampled surfaces. In: Proceedings of Eurographics, vol. 22(3). 2003. p. 121–30.

[22] Chan T, Shen J. Mathematical models for local deterministic inpainting. In: Technical Report CAM-00-11, Image Processing Group, UCLA, 2000.

[23] Bertalmio M, Sapiro G, Caselles V, Ballester C. Image inpainting. In: Proceedings of ACM SIGGRAPH; 2000. p. 417–24.

[24] Clarenz U, Diewald U, Rumpf M. Nonlinear anisotropic diffusion in surface processing. In: Proceedings of IEEE Visualization; 2000. p. 397–405.

[25] Clarenz U, Rumpf M, Telea A. Fairing of point based surfaces. In: Proceedings of Computer Graphics International (CGI). 2004, IEEE CS Press, Los Alamitos, CA, ISBN 0-7695-2171-1, pp. 600–4.

[26] Mount DM. Ann: a library for approximate nearest neighbor searching, www.cs.umd.edu/~mount/ANN.

[27] Zwicker M, Pauly M, Knoll O, Gross M. Pointshop 3d: an interactive system for point-based surface editing. In: Proceedings of ACM SIGGRAPH; 2002. p. 322–9.

[28] Shewchuk JR. Triangle: engineering a 2d quality mesh generator and delaunay triangulator. In: First Workshop of Applied Computational Geometry. New York: ACM Press; 1996. p. 124–33.