Access to surface properties up to order two for visualization algorithms

Helwig Hauser[1], Thomas Theußl[2], and Eduard Gröller[2]

[1] VRVis Research Center, http://www.VRVis.at/
[2] Institute of Computer Graphics and Algorithms, Vienna University of Technology, http://www.cg.tuwien.ac.at/home/

Summary. Elaborated visualization techniques which are based on surfaces often are independent from the origin of the surface data. Nevertheless, many of the previously presented visualization methods were developed for a specific type of surface, although principally applicable to generic surfaces. In this paper we discuss a model for a general access to surface properties up to order two, i.e., surface-point locations, normals, and curvature properties, (almost) regardless of the origin of the surface. Surface types and access algorithms are compared and summarized. At the end of this paper we shortly present an implementation of this model.

Key words: visualization, surfaces, surface properties.

1 Introduction

Surfaces are important geometric primitives for 3D visualization [20]. Useful techniques are available to render surfaces of various kind. For instance, scalar data volumes ($\mathbf{R}^3 \rightarrow \mathbf{R}$) from medical applications are represented using iso-surfaces [17, 19]. Three-dimensional vector fields ($\mathbf{R}^3 \rightarrow \mathbf{R}^3$) from flow analysis are visualized by the use of stream surfaces [12, 28, 16].

In the past years, advanced visualization techniques based on surfaces were proposed which use semi-transparency and local curvature properties to enhance the perceptability of surfaces in 3D. Gerstner, for example, demonstrated the use of multiple, semi-transparent iso-surfaces for visualizing very large datasets [6]. Interrante et al. [15, 13, 14] show how curvature-based techniques enhance the use of surfaces for the visualization of volumetric data. Surface curvature also plays an important role in surface design [5], surface fairing [26, 11], surface trimming [10], surface evaluation and analysis [2, 25], and surface visualization [4].

In this paper we develop an access model to surface properties up to degree two, (almost) regardless of the origin of the surface. Several algorithms which are necessary to deal with different types of surfaces are discussed. A C++ implementation called SMURF – short for SMart sURFace model – of such an abstract access layer between advanced visualization algorithms and surfaces of various origin (see Fig. 1) is described to demonstrate the ease-of-use of this approach.

One advantage of specifying a generic interface like SMURF is that visualization techniques are easily ported from one application to another. Algorithms like modulating the opacity of the surface according to its curvature properties are not bound to one application,

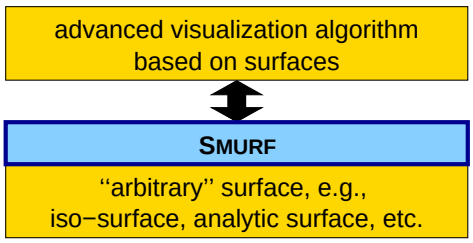

Fig. 1. SMURF is a generic interface between surface-based visualization and surface implementation.

but can be re-used for other surfaces as well. A similar approach in the area of mesh access is described by Rumpf et al. [22].

The remainder of this paper is organized as follows. First we give an overview of several surface types apparent in visualization (Sect. 2). We then discuss the access to surface properties up to degree two in terms of the previously mentioned surface types (Sect. 3). This section includes a review of algorithms which are necessary for accessing different surface types. An implementation of this model (SMURF) is presented in Sect. 4. Some results of SMURF applications are discussed in Sect. 5.

2 Surface Types

In the following we describe seven types of surfaces which are often used in computer graphics and visualization. Surfaces can be defined implicitly, for example, as an iso-surface of scalar volume data, or explicitly, i.e., analytically. Using SMURF the following surface types can be dealt with:

Implicitly defined iso-surfaces for discrete scalar data volumes (in the following case 1) – scalar data values $f_{\mathrm{samp}}(\mathbf{x}_i)$ are given at certain discrete locations $\mathbf{x}_i$ in 3D, e.g., on a regular grid or as scattered data. A certain interpolant $f(\mathbf{x})$ of these values f_{samp} is considered to implicitly define an iso-surface $\mathbf{s}$ (corresponding to a certain iso-value $f_{\mathbf{s}}$): $\mathbf{s} = \{\, \mathbf{x} \mid f(\mathbf{x}) = f_{\mathbf{s}} \,\}$.

Implicitly defined iso-surfaces for analytic scalar data in 3D (case 2) – a scalar function $f(\mathbf{x})$ is given (as a "black box"), which can be evaluated at arbitrary locations $\mathbf{x}$ in 3D. A scalar continuum over 3D is assumed to be the application of the function to all points. A certain iso-value $f_{\mathbf{s}}$ specifies the iso-surface $\mathbf{s} = \{\, \mathbf{x} \mid f(\mathbf{x}) = f_{\mathbf{s}} \,\}$.

Implicitly defined stream surfaces for discrete vector fields (case 3) – vectorial data $\mathbf{v}_{\mathrm{samp}}(\mathbf{x}_i)$ is given at certain discrete locations $\mathbf{x}_i$, for example, on a curvilinear grid. For a specific initial line segment or curve $\mathbf{s}_0(u)$ the corresponding stream surface $\mathbf{s}(u,t)$ is implicitly defined as $\mathbf{s}_0(u) + \int_0^t \mathbf{v}(\mathbf{s}(u,\tau))\,\mathrm{d}\tau$ where $\mathbf{v}(\mathbf{x})$ is an interpolant of the discrete values $\mathbf{v}_{\mathrm{samp}}(\mathbf{x}_i)$.

Implicitly defined stream surfaces for analytically specified dynamical systems (case 4) – a vectorial function $\mathbf{v}(\mathbf{x})$ is given (as a "black box") to be evaluated at arbitrary lo-

cations $\mathbf{x}$ in 3D. A vectorial continuum over 3D is assumed as the application of the function to all points. A certain initial line segment or curve $\mathbf{s}_0(u)$ is implicitly integrated to define the stream surface $\mathbf{s}(u,t) = \mathbf{s}_0(u) + \int_0^t \mathbf{v}(\mathbf{s}(u,\tau))\,\mathrm{d}\tau$.

Explicitly defined parametric surfaces (case 5) – such a surface is defined by a parametric function $\mathbf{s} : \mathbf{R}^2 \to \mathbf{R}^3$.

Explicitly defined surfaces given in implicit form (case 6) – an equation $f(\mathbf{x}) = 0$ defines a surface in 3D (note that this case is similar to case 2).

Explicitly defined discrete surface approximations (case 7) – a mesh, i.e., a set of polygons is used to explicitly specify an approximation of a smooth surface.

There are other surface types as well, for example, explicitly expressing one coordinate in terms of the others, $\mathbf{s}(x,y) = (\,x\;\; y\;\; z(x,y)\,)^{\mathrm{T}}$. Usually they can be either transformed into one of the above mentioned cases, or appear rather rarely. Therefore, they are not considered separately in this paper.

3 Access to Surface Properties

Algorithms used for visualization of volumetric data can be broadly separated into two groups:

Image space techniques are usually based on *ray casting*. The data is intersected with a viewing ray, which is defined by an eye point and a viewing direction, to locate visible surface locations.

Object space techniques project the data onto the image plane to render the surface. In this case often incremental *surface curve traversal* is used to loop over the surface object.

Elaborated surface visualization methods usually are based on surface properties up to the order of two, i.e., the calculation of surface-point locations, surface normals, and surface curvature properties. The access to surface properties, i.e., the evaluation of these properties for certain points of the surface, involves a number of algorithms [8] which are dependent on the type of the surface. Examples are function reconstruction and gradient approximation. In the following we briefly summarize the most common approaches for the surface types described in Sect. 2.

3.1 Surface-point location

When rendering a surface, the most important information about the data is where surface points lie. Based on the knowledge of surface point locations, higher-order properties, such as surface normals or curvature properties, can be computed.

Since a generic interface should be usable for image space and object space techniques, SMURF supports both *ray casting* and incremental *surface curve traversal*. In the following part we firstly discuss ray casting with respect to surfaces of various types.

Ray casting. The intersection of a certain ray (given by a view-point $\mathbf{eye}$ and a viewing direction $\mathbf{dir}$) and the surface to be shown yields a sorted list of surface-points (hit list). Usually just the first entry in the hit list is investigated as the (one and only) visible intersection. Also, advanced visualization algorithms also use semi-transparency of surfaces, thus requiring the computation of successive intersections. Therefore, a hit list of intersections should

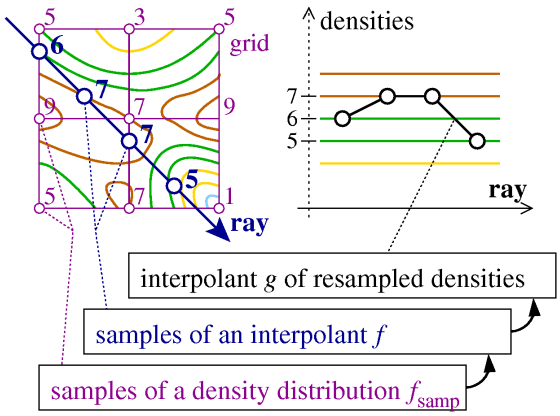

Fig. 2. 2D example of root finding along a ray for iso-surface ray tracing – often two-fold reconstruction is used!

be returned by the "ray casting" surface interface (see Sect. 4). Depending on the type of surface, the intersection calculation is done differently:

Analytic solution (cases 5, 6, and 7) – in the case of an explicitly specified surface, the intersection between a ray and a surface usually can be expressed and sometimes also computed analytically. However, in the usual case the evaluation of this intersection expression has to be done using numerical methods like root finding (see below).

In the case of a parametric surface (case 5) the following two equations have to be solved in terms of u and v ($\mathbf{n}_1$ and $\mathbf{n}_2$ are two orthogonal vectors which are both normal to the viewing ray):

$$\mathbf{s}(u,v) \cdot \mathbf{n}_1 = \mathbf{eye} \cdot \mathbf{n}_1$$
$$\mathbf{s}(u,v) \cdot \mathbf{n}_2 = \mathbf{eye} \cdot \mathbf{n}_2$$
$$\text{where } \mathbf{n}_i \cdot \mathbf{dir} = 0 \text{ and } \mathbf{n}_i \cdot \mathbf{n}_j = \delta_{ij}$$

Depending on the complexity of $\mathbf{s}$, solving the above equations is usually not possible in closed form. There are numerical techniques to compute the list of intersections in terms of u and v [7].

In the implicit case (case 6) the following equation has to be solved in terms of λ:

$$f(\mathbf{eye} + \lambda\,\mathbf{dir}) = 0$$

Again, often numerical methods are required to solve the above equation.

In the polygonal case (case 7) theoretically all polygons have to be intersected with the ray to evaluate the hit list. However, spatial coherence can be exploited by using special data structures to speed up the intersection process [1, 24, 9]. Other simple but effective enhancements like back-face culling are available as well.

Root finding (cases 1 and 2) – considering a ray being cast into a density volume, e.g., a continuum that interpolates discrete data values (case 1) or the application of f to the entire domain (case 2), implicitly yields a scalar density function at all points of the ray. This

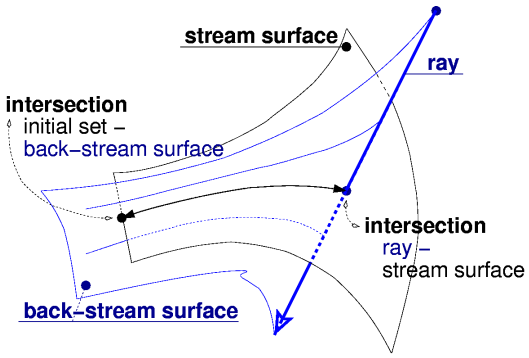

Fig. 3. Lazy evaluation ray casting of stream surfaces by the use of a "back-stream surface".

function can be sampled along the ray to search for intersections with the iso-surface. An interpolant, for instance, linear interpolation, is used to approximate the function along the ray.

Figure 2 illustrates these steps in a 2D example. Discrete density values $f_{\mathrm{samp}}(\mathbf{x}_i)$ are samples of a particular density distribution (arranged, for example, on a regular grid). One typical ray casting approach is to resample an interpolant f, often a tri-linear interpolation, at certain locations along the ray, i.e., $f(\mathbf{r}_i)$ with $\mathbf{r}_i = \mathbf{eye} + i\,\Delta\mathbf{dir}$. For the identification of the ray / iso-surface intersections $\mathbf{p}_k$ an interpolant $g(\mathbf{x})$ along the ray is taken (e.g., linear interpolation) with $g(\mathbf{r}_i) = f(\mathbf{r}_i)$. Equation $g(\mathbf{p}_k) = g(\mathbf{eye} + \lambda_k\,\mathbf{dir}) = f_{\mathbf{s}}$ is solved in terms of λ_k.

Note, that the use of a separate interpolant g yields a double reconstruction of the original function f. Instead, g can also be defined to be the projection of interpolant f onto the ray, which actually would be the more accurate solution. Unfortunately, this approach is rather complex as already the projection of a tri-linear function f induces the interpolant g to be a cubic function in terms of λ_k [23].

Stream surface intersection (cases 3 and 4) – the most demanding problem within the task of locating surface-points is stream surface intersection. This is mainly due to the fact that stream surfaces are implicitly defined through an additionally required integration step of the underlying vectorial data. In flow visualization often pre-computed, i.e., pre-integrated, stream surfaces are used. Numerical techniques, like Euler or Runge-Kutta integration, are used to step-by-step generate a polygonal approximation of the stream surface, which afterwards is visualized using standard mesh rendering methods (compare to case 7).

Another approach exploits the reversibility of flow integration: instead of explicitly generating the stream surface itself, a "back-stream surface" is computed, considering the ray as an initial condition and performing flow integration backwards in time. Any intersection of this "back-stream surface" and the original initial set directly corresponds to an intersection of the investigated stream surface and the ray via a stream line (cf. Fig. 3).

This approach is useful, for example, when lazy evaluation is used (see Sect. 4). Unfortunately this approach is rather expensive when many intersections should be computed. On the other hand, this duality (Fig. 3) can be exploited to increase numerical stability of the intersection computations – divergent flows are more accurately integrated backwards.

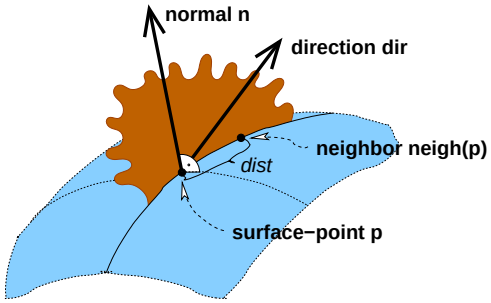

Fig. 4. Surface-curve traversal for iterative object-order surface rendering.

Surface-curve traversal. In addition to ray casting, incremental surface-curve traversal was chosen as a complementary SMURF strategy to access surface-points. Starting with an initial surface-point $\mathbf{p}$, a neighboring location separated by a specific distance $dist$ is searched in a certain direction $\mathbf{dir}$. Surface-point $\mathbf{p}$, surface normal $\mathbf{n}$, and direction $\mathbf{dir}$ define a plane which intersects the surface in a certain surface-curve. Out of both points on the curve which are $dist$ (in terms of curve length) away from $\mathbf{p}$ the one which is mostly aligned with $\mathbf{dir}$ is considered to be the searched location. See Fig. 4 for an illustration of this procedure.

Iso-surfaces (cases 1 and 2) – for implicitly defined iso-surfaces a numerical approximation is used to loop over the surface-curve. First, a tangent vector $\mathbf{t}$ in surface point $\mathbf{p}$ is defined on the basis of the surface normal $\mathbf{n}(\mathbf{p})$ and direction $\mathbf{dir}$ by $\mathbf{t} = \mathbf{n}(\mathbf{p}) \times \mathbf{dir} \times \mathbf{n}(\mathbf{p})$. Then, from a point $\mathbf{q} = \mathbf{p} + dist\,\mathbf{t}$ a local ray casting step is performed in direction $\pm\mathbf{n}(\mathbf{p})$. If no unique solution is found within a small distance from $\mathbf{q}$, a sequence of smaller, iterative steps is performed instead of one with distance $dist$.

Stream surfaces (cases 3 and 4) – stepping along a stream surface is usually done either along the flow, i.e., in direction $\mathbf{v}(\mathbf{p})$, or across the flow, i.e., along time lines. A step along the flow equals the integration of the underlying flow data from point $\mathbf{p}$ for a certain distance: $\mathbf{s_p}(t) = \mathbf{p} + \int_0^t \mathbf{v}(\mathbf{s_p}(\tau))\,\mathrm{d}\tau$. A neighbor $\mathbf{neigh}(\mathbf{p}, \mathbf{v}(\mathbf{p}), dist)$ of point $\mathbf{p}$ therefore is computed as $\mathbf{s_p}(d)$ such that the curve-length between $\mathbf{p}$ and $\mathbf{s_p}(d)$ is $dist$. A surface neighbor of point $\mathbf{p} = \mathbf{s}(u_\mathbf{p}, t_\mathbf{p})$ along a time line in the stream surface is defined as $\mathbf{s}(u_\mathbf{p} \pm \Delta u, t_\mathbf{p})$ such that the length of the time line segment inbetween both points is $dist$. Usually, Δu not really approximates $dist$ in a sufficiently accurate way, so an iterative approach is required like bisectioning.

Explicit surfaces (cases 5, 6, and 7) – in the case of a parametric surface (case 5), the tangent vector $\mathbf{t} = \mathbf{n} \times \mathbf{dir} \times \mathbf{n}$, which corresponds to direction $\mathbf{dir}$, is first decomposed into parameters $u_\mathbf{t}$ and $v_\mathbf{t}$ such that $\mathbf{t} = u_\mathbf{t}\mathbf{s}_u + v_\mathbf{t}\mathbf{s}_v$ ($\mathbf{s}_\omega = \partial\mathbf{s}/\partial\omega$). Then, surface point $\mathbf{s}(u_\mathbf{p} + \frac{dist}{|\mathbf{t}|}u_\mathbf{t}, v_\mathbf{p} + \frac{dist}{|\mathbf{t}|}v_\mathbf{t})$ can be used as a first approximation in an iterative procedure for locating the surface neighbor. In case 6, the procedure for case 2 is used. In case 7, the intersection line between the polygon on which point $\mathbf{p}$ lies and the plane which is spanned by $\mathbf{n}(\mathbf{p})$ and $\mathbf{dir}$, is computed. If the distance of $\mathbf{p}$ and the polygon edge (in the direction of the intersection line) is larger than $dist$, then the respective neighbor of $\mathbf{p}$ lies on the same polygon. Otherwise, the neighbor polygon has to be searched across the polgon edge. There, the procedure repeats with the intersection of polygon and plane, etc.

3.2 Surface normal computation

Various computer graphics algorithms use surface normals, e.g., for shading or back-face culling. The acquisition of a normal corresponding to a certain surface-point again depends on the type of surface:

Analytic solution (cases 5 and 6) – if the surface is given explicitly, usually the surface normal at a certain point can be computed analytically. In the parametric case (case 5) the cross-product $\partial\mathbf{s}/\partial u\,|_{\mathbf{p}} \times \partial\mathbf{s}/\partial v\,|_{\mathbf{p}}$ of two tangent vectors yields a (not yet normalized) surface normal at point $\mathbf{p}$. Of course, this is only possible if both tangents are not collinear. In the implicit case (case 6) the gradient $\nabla f|_{\mathbf{p}}$ is a surface normal of the iso-surface through $\mathbf{p}$ (not normalized).

Gradient reconstruction from densities (cases 1 and 2) – assuming a function $f(\mathbf{x})$ which can be evaluated at arbitrary points $\mathbf{x}$ (f is either the "black box", case 2, or an interpolant, case 1), surface normals (not normalized) can be computed using central differences, for example:

$$\mathbf{n}(\mathbf{p}) = \nabla f|_{\mathbf{p}} \approx \tfrac{1}{2} \begin{pmatrix} f(\mathbf{x}+\mathbf{e}_1) - f(\mathbf{x}-\mathbf{e}_1) \\ f(\mathbf{x}+\mathbf{e}_2) - f(\mathbf{x}-\mathbf{e}_2) \\ f(\mathbf{x}+\mathbf{e}_3) - f(\mathbf{x}-\mathbf{e}_3) \end{pmatrix}$$
$$\mathbf{e}_i = (\delta_{1i}\ \delta_{2i}\ \delta_{3i})^{\mathrm{T}}$$

Higher-order approximations of the gradient are possible as well. In general, an arbitrarily complex derivative filter can be applied for gradient reconstruction [3, 18].

Note, that in case 2 central differences easily are evaluated at arbitrary surface locations, whereas in case 1 (when dealing with data samples on a regular grid) normals are usually approximated at grid locations and then interpolated within cells, for example, using tri-linear interpolation.

Normal reconstruction from polygons (case 7) – a standard procedure for reconstructing normals within polygons is used for Phong shading [21]: at the vertices of a polygon a weighted sum of all the normals of adjacent polygons is computed. These vertex normals are interpolated within the polygon to approximate the normals of the surface which is approximated by the polygons.

Stream surface normals (cases 3 and 4) – In the case of a pre-computed stream surface in the form of a set of polygons, again techniques for case 7 can be used. In the other case, the (not yet normalized) surface normal $\mathbf{n}(\mathbf{p})$ can be computed as the cross-product of two tangent vectors:

$$\mathbf{n}(\mathbf{p}) = \mathbf{n}(\mathbf{s}(u_{\mathbf{p}}, t_{\mathbf{p}})) = \mathbf{v}(\mathbf{p}) \times (\mathbf{s}(u_{\mathbf{p}}+\Delta u, t_{\mathbf{p}}) - \mathbf{s}(u_{\mathbf{p}}-\Delta u, t_{\mathbf{p}}))$$

One tangent, $\mathbf{v}(\mathbf{p})$, equals the vectorial data in the surface point whereas the other tangent is numerically approximated as (central) difference from neighboring stream lines in the stream surface.

3.3 Surface curvature

Second-order surface properties, i.e., curvature information, is used to enhance surface-based visualization. Shape and location of a surface can be better perceived, for example, if curvature directed strokes are applied to the surface [14].

Surface curvature usually is expressed in several terms, e.g., Gaussian or mean curvature. Both curvature properties depend on a surface-curvature definition [5] which is dependent on

a specific tangent direction. Principal directions are those tangent directions which yield either maximum or minimum curvature.

Curvature calculation for parametric surfaces (case 5) – The first and second fundamental coefficents (with the usual abbreviations) of a parametric surface $\mathbf{s}(u,v)$ are defined as

$$E = \mathbf{s}_u \cdot \mathbf{s}_u,\ \ F = \mathbf{s}_u \cdot \mathbf{s}_v,\ \ G = \mathbf{s}_v \cdot \mathbf{s}_v$$
$$L = \mathbf{s}_{uu} \cdot \mathbf{n},\ M = \mathbf{s}_{uv} \cdot \mathbf{n},\ N = \mathbf{s}_{vv} \cdot \mathbf{n}$$

with $\mathbf{n} = (\mathbf{s}_u \times \mathbf{s}_v) \,/\, |\mathbf{s}_u \times \mathbf{s}_v|$ being the unit normal vector and $\mathbf{s}_u = \partial\mathbf{s}/\partial u$, $\mathbf{s}_v = \partial\mathbf{s}/\partial v$. The normal curvature in tangent direction $u' : v'$ is

$$\kappa = -\frac{Lu'^2 + 2Mu'v' + Nv'^2}{Eu'^2 + 2Fu'v' + Gv'^2}$$

For κ being extremal it must satisfy the equation [5]

$$\det \begin{bmatrix} \kappa E - L & \kappa F - M \\ \kappa F - M & \kappa G - N \end{bmatrix} = 0$$

The extreme values κ_1 and κ_2 are the principal curvatures of the surface at $\mathbf{x}$ and

$$\begin{aligned} \kappa_1\kappa_2 &= (LN - M^2)/(EG - F^2) \\ \kappa_1 + \kappa_2 &= (NE - 2MF + LG)/(EG - F^2) \end{aligned}$$

are the Gaussian and mean curvature, respectively.

Curvature calculation for implicit surfaces (case 6) – In a surface-point $\mathbf{p}$ of interest we consider $\mathbf{n}(\mathbf{p})$ to be a unit normal of the plane which is tangent to the surface through $\mathbf{p}$, i.e., $\mathbf{n}(\mathbf{p}) = \nabla f|_{\mathbf{p}} \,/\, \left|\nabla f|_{\mathbf{p}}\right|$. Assuming $\mathbf{e}_1$ to be an arbitrary vector of unit length contained in the tangent plane, we construct a local Frenet frame:

$$\begin{gathered} \boldsymbol{\Phi} = \left(\, \mathbf{e}_1 \ \mathbf{e}_2 \ \mathbf{n}(\mathbf{p}) \,\right) \\ \mathbf{e}_1 \cdot \mathbf{n}(\mathbf{p}) = 0 \\ \mathbf{e}_2 = \mathbf{n}(\mathbf{p}) \times \mathbf{e}_1 \end{gathered}$$

Searching for the principal curvature of the surface through $\mathbf{p}$, we have to investigate the changes of $\mathbf{n}(\mathbf{x})$ near $\mathbf{p}$ with respect to changes of $\mathbf{x}$ within the tangent plane, i.e., $\mathbf{x} = \mathbf{p} + r\mathbf{e}_\varphi$ with $\mathbf{e}_\varphi$ being a unit length vector orthogonal to $\mathbf{n}(\mathbf{p})$, i.e., lying in the tangent plane.

Direction $\mathbf{e}_\varphi$, where $\nabla\mathbf{n}|_{\mathbf{p}} \cdot \mathbf{e}_\varphi$, i.e., the directional derivative of $\mathbf{n}$ near $\mathbf{p}$ into direction $\mathbf{e}_\varphi$, is greatest in terms of length (called $\mathbf{e}_{\bar{\varphi}}$ in the following), is then the first principal direction of the surface through $\mathbf{p}$. The second principal direction is orthogonal to both $\mathbf{e}_{\bar{\varphi}}$ and $\mathbf{n}(\mathbf{p})$. The related curvatures are the lengths of the directional derivatives along the principal directions.

As derivation is a linear operator, the directional derivative of $\mathbf{n}$ into some direction $\mathbf{e}_\varphi = \cos\varphi\ \mathbf{e}_1 + \sin\varphi\ \mathbf{e}_2$ can be written in terms of the directional derivative of $\mathbf{n}$ into directions $\mathbf{e}_1$ and $\mathbf{e}_2$:

$$\nabla\mathbf{n}|_{\mathbf{p}} \cdot \mathbf{e}_\varphi = \left(\, \nabla\mathbf{n}|_{\mathbf{p}} \cdot \mathbf{e}_1 \ \ \nabla\mathbf{n}|_{\mathbf{p}} \cdot \mathbf{e}_2 \,\right) \cdot \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix}$$

Since $\nabla\mathbf{n}|_{\mathbf{p}} \cdot \mathbf{e}_\varphi$ is orthogonal to $\mathbf{n}(\mathbf{p})$ also, we can express it in terms of $\mathbf{e}_1$ and $\mathbf{e}_2$ by the

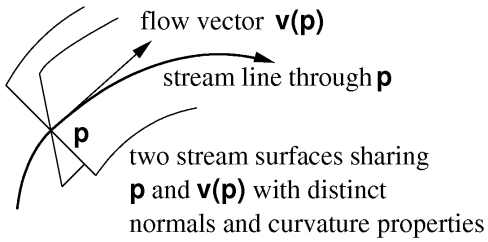

Fig. 5. The definition of a stream surface through a point $\mathbf{p}$ is ambiguous – depending on the seeding structure, a variety of different stream surfaces are possible. Therefore, normals and curvature properties of stream surfaces are of limited use for the visualization of flow data.

use of decomposition with $(x\ y)^{\mathrm{T}} = (\cos\varphi\ \sin\varphi)^{\mathrm{T}}$:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} \mathbf{e}_1^{\mathrm{T}} \\ \mathbf{e}_2^{\mathrm{T}} \end{pmatrix} \cdot \begin{pmatrix} \nabla\mathbf{n}|_{\mathbf{p}} \cdot \mathbf{e}_1 \\ \nabla\mathbf{n}|_{\mathbf{p}} \cdot \mathbf{e}_2 \end{pmatrix}^{\mathrm{T}}}_{\mathbf{A} = (\omega_{ij}),\ \omega_{ij} = \mathbf{e}_i \cdot \nabla\mathbf{n}|_{\mathbf{p}} \cdot \mathbf{e}_j} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Searching for the greatest eigenvector $(x_{\bar{\varphi}}\ y_{\bar{\varphi}})^{\mathrm{T}}$ of matrix $\mathbf{A}$, directly yields the corresponding first principal direction via $\mathbf{e}_{\bar{\varphi}} = (\mathbf{e}_1\ \mathbf{e}_2) \cdot (x_{\bar{\varphi}}\ y_{\bar{\varphi}})^{\mathrm{T}}$.

Curvature reconstruction from densities (cases 1 and 2) – to reconstruct curvature properties of iso-surfaces obtained from scalar volume data essentially the same procedure as for implicit surfaces can be used. A function $f(\mathbf{x})$ is assumed, which can be evaluated at arbitrary points $\mathbf{x}$ (see Sect. 2), as well as a function $\mathbf{n}(\mathbf{p}) = \nabla f|_{\mathbf{p}} \,/\, \left| \nabla f|_{\mathbf{p}} \right|$ which yields the unit normal at an arbitrary point $\mathbf{p}$ (see Sect. 3.2). Again the eigenvalue decomposition of matrix $\mathbf{A} = (\mathbf{e}_i \cdot \nabla\mathbf{n}|_{\mathbf{p}} \cdot \mathbf{e}_j)_{ij}$ in terms of a local Frenet frame gives the searched curvature properties.

Stream surface curvature (cases 3 and 4) – in the case of a pre-computed stream surface techniques described for case 7 (see below) are used. In the case of a stream surface on demand curvature properties could be derived by investigating the changes of a normal with respect to changes within the tangent plane. It must be noted here that stream surface curvature is rarly used for visualization since it easily might be misinterpreted as a property of the underlying vector field. Figure 5 illustrates why stream surface curvature – even stream surface normals – usually lack importance in visualization. Both properties heavily depend on the choice of the initial condition.

Curvature reconstruction from polygons (case 7) – To reconstruct curvature properties from polygons, one obvious procedure would be to construct an interpolant and calculate analytically the curvature of the interpolant. Todd and McLeod [27], however, report that this approach yields in general completely unsatisfying results.

Therefore, they propose to approximate the Dupin indicatrix from the vertices of the polygons, by exploiting Meusniers theorem [5], estimating normal curvatures in particular directions (which requires to estimate the normal, for example, with the approach used in Phong shading, as described in Sect. 3.2) and finally fitting a central conic to that data.

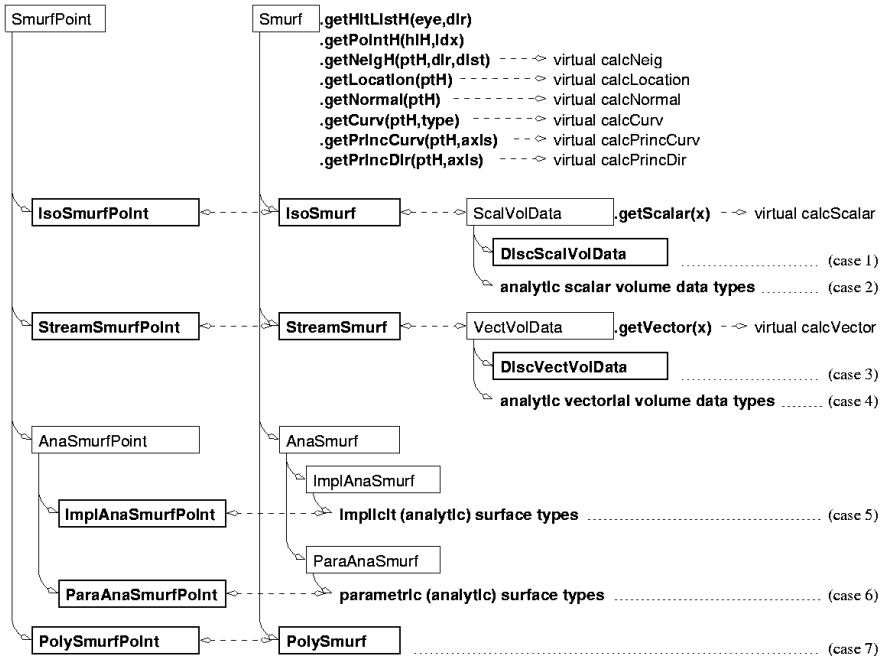

Fig. 6. SMURF class hierarchy and interface.

4 SMURF Classes

After having identified the different surface types (Sect. 2) and access schemes (Sect. 3) the integration in a C++ class hierarchy called SMURF (see Fig. 6) is straight-forward. An abstract base class provides all common properties of the different surface types and the access interface to surface properties as virtual functions. Sub-classes, corresponding to the surface types, are derived from this abstract base class and redefine the access schemes accordingly.

To distinguish discrete scalar volume data-sets (case 1) from analytic scalar volume functions (case 2) a class `ScalVolData` hides the interface. Therefore, class `IsoSmurf` can treat these two cases the same way. The same holds for discrete vector fields (case 3) and analytically specified dynamical systems (case 4) via class `VectVolData`. This means, that all the seven cases of surface types which were presented in Sect. 2 are mapped to four sub-classes of SMURF, i.e., ISOSMURF, STREAMSMURF, ANASMURF, and POLYSMURF. See Fig. 6 for the relations between these classes.

An important concept in the implementation is the one of lazy evaluation, i.e., computing not more than necessary at a certain point in time. For example, ray casting can be terminated after finding the desired intersection point. Further surface properties are evaluated on demand, and stored for future use. Therefore, a class hierarchy `SmurfPoint` is introduced which mirrors the `Smurf` class hierarchy and serves as memory element for the specific surface types, i.e., it stores all relevant information already computed which can be re-used. Again, Fig. 6 illustrates the relations between these classes.

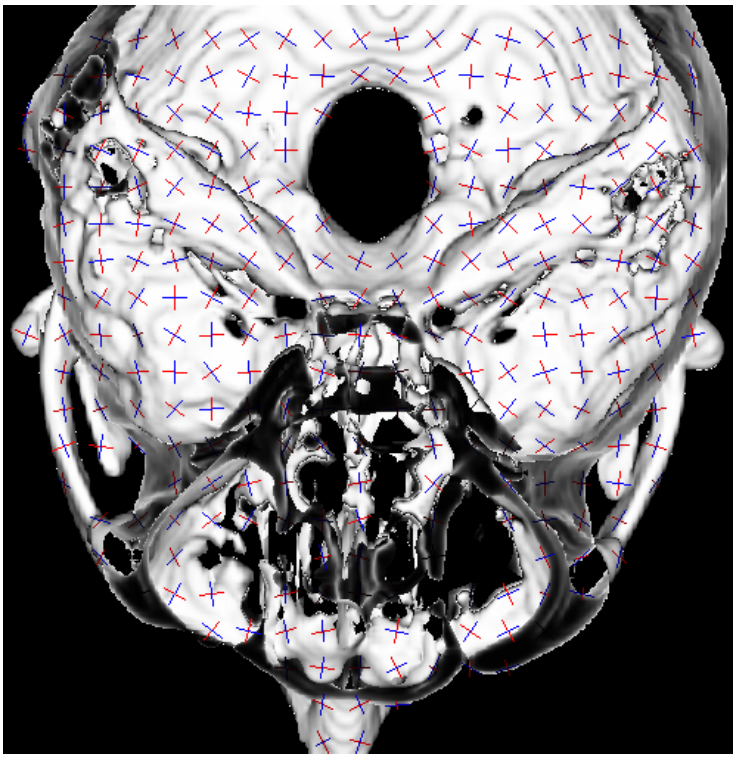

Fig. 7. Iso-surface computed for ten slices of a human head scan with curvature crosses.

5 Results, Applications

By hiding the intrinsic differences between the surface types identified in Sect. 2 SMURF supports the user with the following tasks:

Implementation of advanced visualization techniques – SMURF eases this task by providing an interface for obtaining surface properties independently of the surface type. Figure 7 shows an iso-surface with crosses aligned to the principal directions (similar to Beck et al. [2]). Figure 8 was generated using the code depicted in Fig. 9 – any other surface type could be rendered using the same code by just changing the very first line.

Comparison of algorithms – for instance, reconstruction schemes can be easily compared by sampling an analytic function and applying a visualization algorithm to both the scalar data volume and the analytic function. In Fig. 10 this concept was used to compare linear and cubic interpolation with respect to function reconstruction and the computation of the Gaussian curvature with the corresponding analytic function (a quadratic in 3D). Linear reconstruction of densities is clearly seen in Fig. 10(a), whereas there is no perceivable difference between Fig. 10(b) and (c). Comparing the curvature plot (color was used to visualize the Gaussian curvature), subtle differences can be obtained even between Fig. 10(b) and (c).

Figure 11 compares the quality of curvature reconstruction (by calculating principal curvature lines of a cylinder) depending on linear and cubic density reconstruction. In Fig. 11(a) small errors accumulated during numerical integration of the curvature lines are visible.

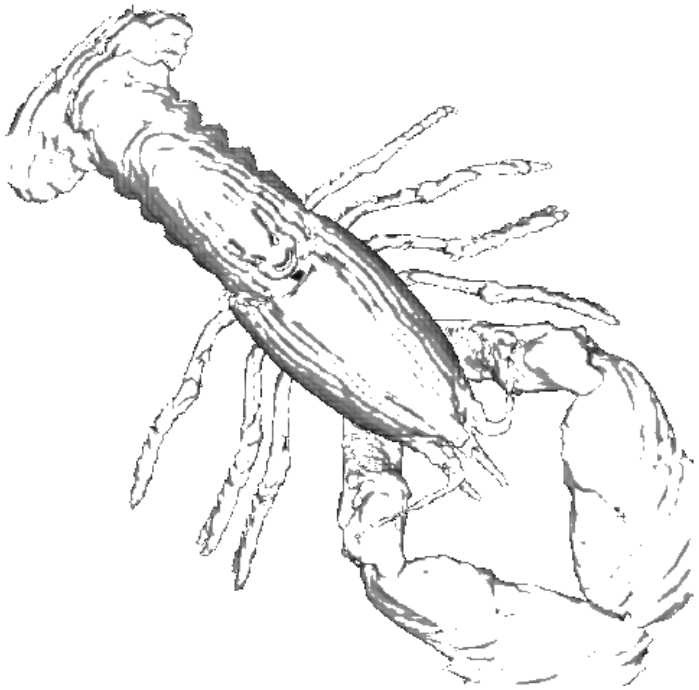

Fig. 8. Contour display of a lobster CT scan – see Fig. 9 for the SMURF-code used to render this image.

```
Smurf *pSmurf = new IsoSmurf("lobster.dat",threshold);

for (p=pFirstPixel(); p!=NULL; p=pNextPixel())
{
  VEC3 dir               = normalize(*p-eye);
  SmurfHitListHandle HLH = pSmurf->getHitListH(eye,dir);
  SmurfPointHandle   PH  = pSmurf->getPointH(HLH,0);
  VEC3 normal            = pSmurf->getNormal(PH);

  if (-dir*normal < 0.6)
    p->set(1-(-dir*normal));
  else
    p->set(0);
}
```

Fig. 9. Code used for drawing lobster contours depicted in Fig. 8.

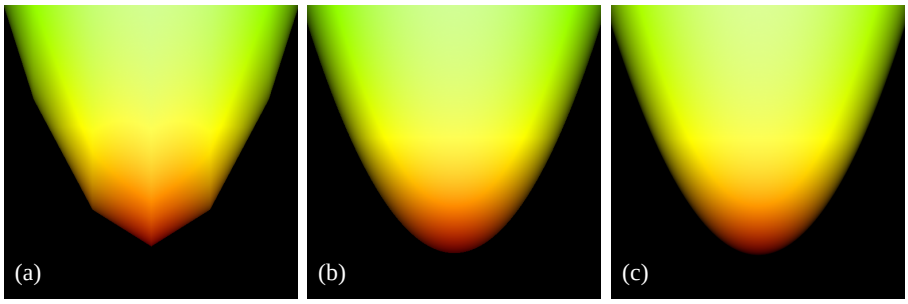

Fig. 10. Gaussian curvature plot using linear (a) and cubic (b) function reconstruction vs. analytic computation (c).

6 Future Work

One obvious disadvantage of a general scheme like SMURF is that it principally suffers from inefficiency. Performance can be improved by including, e.g., intelligent caching strategies and implementation short-cuts. Furthermore, it should be possible to exploit ray-to-ray coherence of visualization algorithms. Thus, again, caching and addressing of external data must be allowed by the scheme.

Another idea is to extend the SMURF concept to a 'set of SMURFs' class with a similar interface. Consecutive intersections along a ray are reported in correct order from different surfaces, e.g., stacked iso-surfaces or multiple stream surfaces. Even surfaces of different type could be easily combined using this concept, for example, patient data together with objects from (virtual) surgery planning.

7 Conclusions

Our general purpose surface interface SMURF allows to easily re-use elaborated visualization techniques that are based on surfaces in 3D. Examples are curvature-directed strokes or plotting curvature lines, with surfaces originating from various applications like iso-surfaces from medical applications or stream surfaces from flow visualization. Surface properties up to the order of two, i.e., curvature information, are available to the user in a transparent way. Ray casting as well as incremental surface-curve traversal are provided as surface access strategies. Thus, advanced surface visualization techniques can be developed without having to care about specific algorithms for calculating particular surface properties. Their portability to other surface types is another advantage of the SMURF concept.

For the realization of this concept we first identified the most often used surface types and compared various algorithms for accessing surface properties in general before we unified them in a unique interface. As most visualization applications have to deal with sampled data, analytic evaluation is discussed as well as various reconstruction schemes. The usefulness of this approach is demonstrated by several results we obtained with our actual implementation.

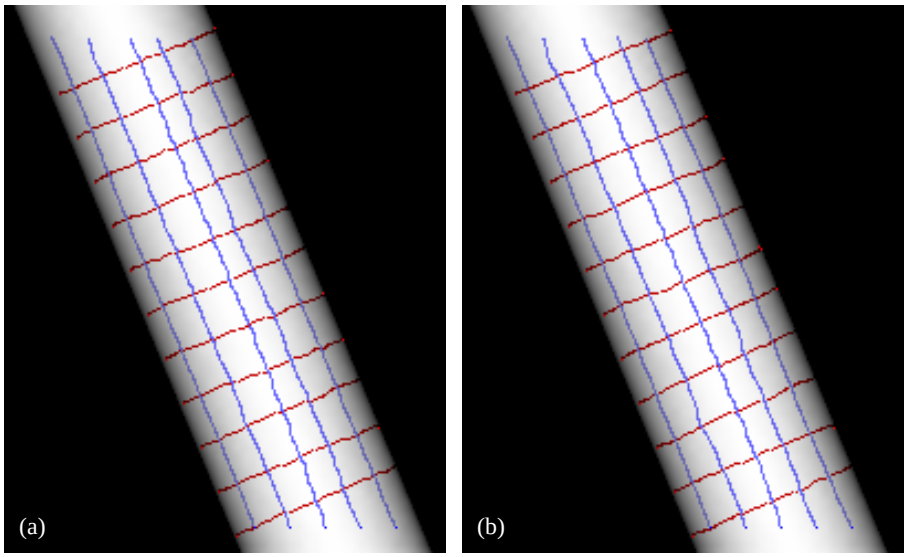

Fig. 11. Quality of curvature calculations depending on the function reconstruction scheme – linear (a) vs. cubic (b) reconstruction.

ACKNOWLEDGMENTS

The work presented in this publication has been supported by V^{is}M^{ed}. Please refer to `http://www.vismed.at/` for further information on this project. Parts of this work have been done in the VRVis Research Center (`http://www.VRVis.at/`) in Vienna, Austria, which is funded by an Austrian research program called K plus.

References

1. James Arvo and David Kirk. A survey of ray tracing acceleration techniques. In Andrew Glassner, editor, *An introduction to ray tracing*, pages 201–262. Academic Press, 1989.
2. James Beck, Rida Farouki, and John Hinds. Surface analysis methods. *IEEE Computer Graphics and Applications*, 6(12):18–36, 1986.
3. Mark Bentum, Barthold Lichtenbelt, and Tom Malzbender. Frequency Analysis of Gradient Estimators in Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):242–254, 1996.
4. John Dill. An application of color graphics to the display of surface curvature. *Computer Graphics*, 15(3):153–161, August 1981.
5. Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Morgan Kaufmann, 4th edition, 1997.
6. Thomas Gerstner. Multiresolution extraction and rendering of transparent isosurfaces. *Computers and Graphics*, 26(2):219–228, April 2002.
7. Andrew Glassner, editor. *An Introduction to Ray Tracing*. Acad. Press, 1989.

8. Hans Hagen, Stefanie Hahmann, Thomas Schreiber, Yasuo Nakajima, Burkard Wordenweber, and Petra Hollemann-Grundstedt. Surface interrogation algorithms. *IEEE Computer Graphics and Applications*, 12(5):53–60, Sept. 1992.
9. Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University, Praha, Czech Republic, April 2001. Available from http://www.cgg.cvut.cz/˜havran/phdthesis.html.
10. Josef Hoschek and Franz-Josef Schneider. Spline conversion for trimmed rational Bézier- and B-spline surfaces. *Computer Aided Design*, 22(9):580–590, 1990.
11. Andreas Hubeli and Markus Gross. Fairing of non-manifolds for visualization. In *Proceedings IEEE Visualization*, pages 407–414, 2000.
12. Jeff Hultquist. Constructing stream surfaces in steady 3D vector fields. In *Proceedings IEEE Visualization*, pages 171–177, 1992.
13. Victoria Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. *Computer Graphics*, 31(Annual Conference Series):109–116, August 1997.
14. Victoria Interrante, Henry Fuchs, and Stephen Pizer. Conveying the 3D shape of smoothly curving transparent surfaces via texture. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):98–117, 1997.
15. Victoria Interrante, Henry Fuchs, and Steven Pizer. Illustrating transparent surfaces with curvature-directed strokes. In *Proceedings IEEE Visualization*, pages 211–218, 1996.
16. Helwig Löffelmann, Lukas Mroz, Eduard Gröller, and Werner Purgathofer. Stream arrows: Enhancing the use of streamsurfaces for the visualization of dynamical systems. *The Visual Computer*, 13:359–369, 1997.
17. William Lorensen and Harvey Cline. Marching cubes: A high resolution 3D surface construction algoritm. *Computer Graphics*, 21(4):163–168, July 1987.
18. Torsten Möller, Raghu Machiraju, Klaus Müller, and Roni Yagel. Evaluation and Design of Filters Using a Taylor Series Expansion. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):184–199, 1997.
19. Gregory Nielson and Bernd Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. In *Proceedings IEEE Visualization*, pages 83–91, 1991.
20. Gregory Nielson, Bruce Shriver, and Lawrence Rosenblum. *Visualization in Scientific Computing*. IEEE Computer Society Press, 1990.
21. Bui-Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.
22. Martin Rumpf, Alfred Schmidt, and Kunibert Siebert. Functions defining arbitrary meshes - A flexible interface between numerical data and visualization. *Computer Graphics Forum*, 15(2):129–142, 1996.
23. Georgios Sakas, Marcus Grimm, and Alexandros Savopoulos. Optimized maximum intensity projection (MIP). In *Proceedings* EUROGRAPHICS *Rendering Workshop*, pages 51–63, 1995.
24. Hanan Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, 1989.
25. Lee Seidenberg, Robert Jerad, and John Magewick. Surface curvature analysis using color. In *Proceedings IEEE Visualization*, pages 260–267, 1992.
26. Gabriel Taubin. A signal processing approach to fair surface design. *Computer Graphics*, 29(Annual Conference Series):351–358, November 1995.
27. Philip Todd and Robin McLeod. Numerical estimation of the curvature of surfaces. *Computer-Aided Design*, 18(1):33–37, January 1986.
28. Jarke van Wijk. Implicit stream surfaces. In *Proceedings IEEE Visualization*, pages 245–252, 1993.