# Smart surface interrogation for advanced visualization techniques

**Helwig Hauser**[1] **, Thomas Theußl, Andreas König, and Eduard Gröller**

Institute of Computer Graphics, Vienna University of Technology,
Karlsplatz 13/186, A-1040 Vienna, Austria, `http://www.cg.tuwien.ac.at/home/`
`mailto:{helwig|theussl|koenig|groeller}@cg.tuwien.ac.at`

## ABSTRACT

Highly elaborated visualization techniques that are based on surfaces often are independent from the origin of the surface data. Nevertheless, most of the recently presented advanced visualization methods were developed for a specific type of surface although principally applicable to generic surfaces. In this paper we discuss a unified surface interrogation model which provides generic access to surface properties up to degree two, i.e., surface-point locations, normals, and curvature properties, (almost) regardless of the origin of the surface. Surface types and interrogation algorithms are compared and summarized. At the end of this paper we present an object-oriented implementation of this model, called SMURF.

**Keywords:** visualization, surfaces, surface properties

## 1 INTRODUCTION

Surfaces are important geometric primitives for 3D visualization [14]. 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 [12]. Three-dimensional vector fields ($\mathbf{R}^3 \rightarrow \mathbf{R}^3$) from flow analysis are visualized by the use of stream surfaces [8, 11].

Recently, advanced visualization techniques based on surfaces were proposed which use semi-transparency and local curvature properties to enhance the perceptability of surfaces in 3D. Interrante et al. [9, 10] 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, surface fairing, surface trimming, surface evaluation and analysis, and surface visualization [2, 4, 5, 19].

In this paper we develop a unified access model to surface properties up to degree two, (almost) regardless of the origin of the surface. Various 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 interrogation layer between advanced visualization algorithms and surfaces of various origin (see Fig. 1) is described to demonstrate the ease-of-use of this approach.

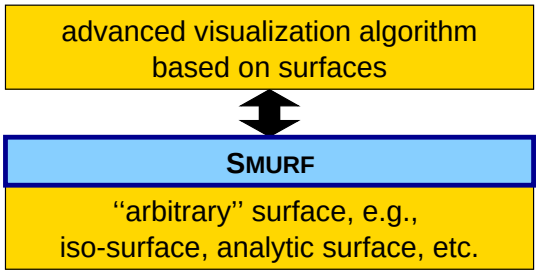



Figure 1: SMURF is a generic interface between surface-based visualization and surface implementation.

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, but can be

[1] previous name: Helwig Löffelmann

re-used for other surfaces as well. A similar approach in the area of mesh access is described by Rumpf et al. [16].

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

## 2 SURFACE TYPES

In the following we describe some of the most important types of surface 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:

CASE 1: **implicitly defined iso-surfaces for discrete scalar volume data** – scalar data values $f_{\text{samp}}(\mathbf{x}_i)$ 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 $f_{\text{samp}}(\mathbf{x}_i)$ is considered to implicitly define an iso-surface $\mathbf{s}$ corresponding to a certain iso-value $f_{\mathbf{s}}$: $\mathbf{s} = \{\, \mathbf{x} \mid f(\mathbf{x}) = f_{\mathbf{s}} \,\}$.

CASE 2: **implicitly defined iso-surfaces for analytic scalar volume functions** – 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 as the application of the function to all points. A certain iso-value $f_{\mathbf{s}}$ specifies the iso-surface $\mathbf{s} = \{\, \mathbf{x} \mid f(\mathbf{x}) = f_{\mathbf{s}} \,\}$.

CASE 3: **implicitly defined stream surfaces for discrete vector fields** – vectorial data $\mathbf{v}_{\text{samp}}(\mathbf{x}_i)$ 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 the integral set $\mathbf{s}_0(u) + \int_0^t \mathbf{v}(\mathbf{s}(u,\tau))\,\mathrm{d}\tau - \mathbf{v}(\mathbf{x})$ is an interpolant of the discrete values $\mathbf{v}_{\text{samp}}(\mathbf{x}_i)$.

CASE 4: **implicitly defined stream surfaces for analytically specified dynamical systems** – a vectorial function $\mathbf{v}(\mathbf{x})$ is given (as a "black box") to be evaluated at arbitrary locations $\mathbf{x}$ 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 $\mathbf{s}(u,t) = \mathbf{s}_0(u) + \int_0^t \mathbf{v}(\mathbf{s}(u,\tau))\,\mathrm{d}\tau$.

CASE 5: **explicitly defined parametric surfaces** – such a surface is defined by a parametric function $\mathbf{s} : \mathbf{R}^2 \to \mathbf{R}^3$.

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

CASE 7: **explicitly defined discrete surface approximations** – a set of polygons or a mesh 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 – $\mathbf{s}(x,y) = (\ x \quad y \quad z(x,y)\ )^{\mathrm{T}}$. 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 SURFACE INTERROGATION

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

**Image space** techniques, which are usually based on *ray casting*, i.e., the data is intersected with a viewing ray, which is defined by an eye point and a pixel on the image plane, to locate visible surface locations.

**Object space** techniques, which project the data onto the image plane and perform compositing. 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. Surface interrogation, i.e., the evaluation of these properties for certain points of the surface, involves a number of algorithms [7] 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

The location of a point on a surface is not only necessary for projection methods but also for ray casting since all surface properties, like normals and curvature properties, are dependent on the location of a surface point.

Since a generic surface interrogation 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 **eye** and a viewing direction **dir**) and the surface 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. Sometimes, advanced visualization algorithms also use semi-transparency of surfaces, thus requiring the computation of successive intersections also. Therefore, a hit list (featuring lazy evaluation) of intersections should 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 – or even computed – analytically. Usually, 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 vectors orthogonal to each other, while both being normal to the viewing ray):

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

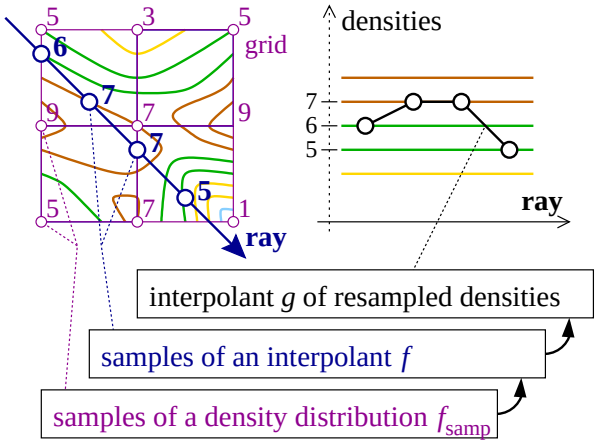



Figure 2: 2D example of root finding along a ray for iso-surface ray tracing – often two-fold reconstruction is used!

Depending on the complexity of $\mathbf{s}$, solving the above equations usually is not possible in closed form. There are numerical techniques to compute the list of intersections in terms of $u$ and $v$ [6].

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

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

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. Spatial coherence can be exploited using special data structures for storing the polygons to speed up the intersection process [1, 18]. 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 function can be sampled along the ray to search for intersections with the iso-surface. Usually, an interpolant is used to approximate this function along the ray, for instance, linear interpolation.

Fig. 2 illustrates these steps in 2D. Discrete density values $f_{\text{samp}}(\mathbf{x}_i)$ are samples (arranged, for example, on a regular grid) of a particular density distribution. One typical ray

casting approach is to resample an interpolant $f$, e.g., a tri-linear or a tri-cubic interpolant, at certain locations along the ray, i.e., $f(\mathbf{r}_i)$ – $\mathbf{r}_i = \mathbf{eye} + i \cdot \Delta\,\mathbf{dir}$. For the identification of the ray / iso-surface intersections $\mathbf{p}_k$ usually an interpolant $g(\mathbf{x})$ along the ray is assumed ($g(\mathbf{r}_i) = f(\mathbf{r}_i)$), and equation $g(\mathbf{p}_k) = g(\mathbf{eye} + \lambda_k \cdot \mathbf{dir}) = f_\mathbf{s}$ is solved in terms of $\lambda_k$.

Note, that the use of a separate interpolant $g$ is a second reconstruction step 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 usually rather complex as the projection of a tri-linear function $f$, for example, induces the interpolant $g$ to be a cubic function in terms of $\lambda_k$ [17].

**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.

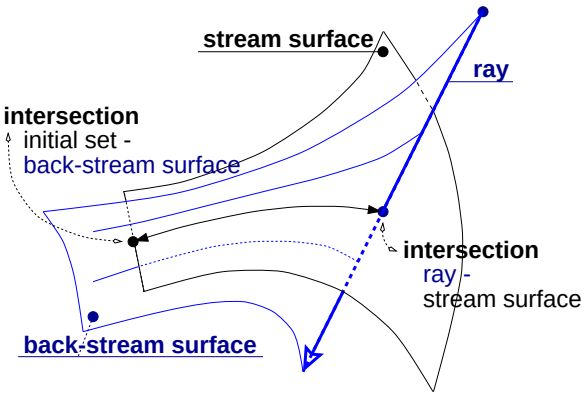



Figure 3: lazy-evaluation ray casting of stream surfaces by the use of a "back-stream surface".

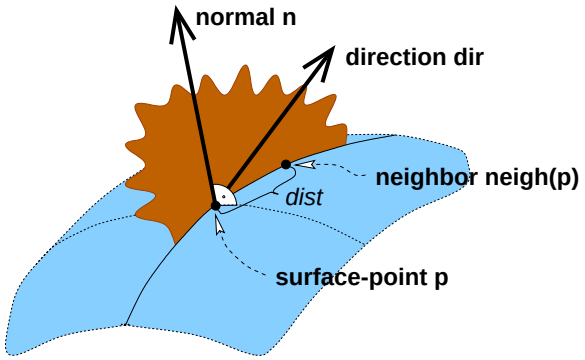



Figure 4: surface-curve traversal for iterative object-order surface rendering.

### 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 **dir**. Surface-point $\mathbf{p}$, surface normal $\mathbf{n}$, and direction **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 **dir** is considered to be the searched location. See Fig. 4 for an illustration of this procedure.

## 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:

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

Higher-order approximations of the gradient are possible as well. In general, an arbitrarily complex derivative filter can be applied for gradient reconstruction [3, 13].

**Normal reconstruction from polygons** (case 7) – a standard procedure for reconstructing normals within polygons is used for Phong shading [15]: 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 directly approximating the stream surface by a set of polygons, again techniques for case 7 can be used. In the other case (using the duality shown in Fig. 3) the surface normal can be built as the cross-product of two tangent vectors. One is equal to the vectorial data value at the point of interest. The other can be approximated by investigating neighboring stream lines.

### 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 [10].

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

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

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 $\kappa$ 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 $\kappa_1$ and $\kappa_2$ 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 Frenét frame:

$$\Phi = \begin{pmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{n}(\mathbf{p}) \end{pmatrix}$$
$$\mathbf{e}_1 \cdot \mathbf{n}(\mathbf{p}) = 0, \ \mathbf{e}_2 = \mathbf{n}(\mathbf{p}) \times \mathbf{e}_1$$

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 $\mathbf{e}_{\bar{\varphi}}$, where $\nabla\mathbf{n}|_\mathbf{p}\cdot\mathbf{e}_{\bar{\varphi}}$, i.e., the directional derivative of $\mathbf{n}$ near $\mathbf{p}$ into direction $\mathbf{e}_{\bar{\varphi}}$, is greatest (in terms of length), 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 lengthes 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 $\mathbf{e}_2$:

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

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

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

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 $\mathbf{e}_{\bar{\varphi}} = (\ \mathbf{e}_1 \quad \mathbf{e}_2\ )\cdot(\ x_{\bar{\varphi}} \quad y_{\bar{\varphi}}\ )^\mathrm{T}$.

**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 Sec. 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 Sec. 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 Frenét 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 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. Fig. 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.

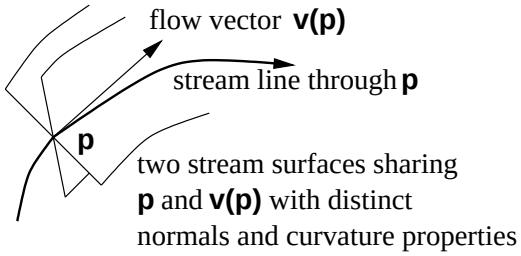



Figure 5: why normal and curvature properties of stream surfaces do not tell a lot about the underlying flow.

**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 [20], 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 Sec. 3.2) and finally fitting a central conic to that data.

## 4 SMURF CLASSES

After having identified the different surface types (Sect. 2) and surface interrogation 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 various surface types and the interrogation interface as virtual functions. Various sub-classes, corresponding to the surface types, are derived from this abstract base class and redefine the interrogation schemes accordingly.

To distinguish discrete scalar volume data-sets (case 1) from analytic scalar volume functions

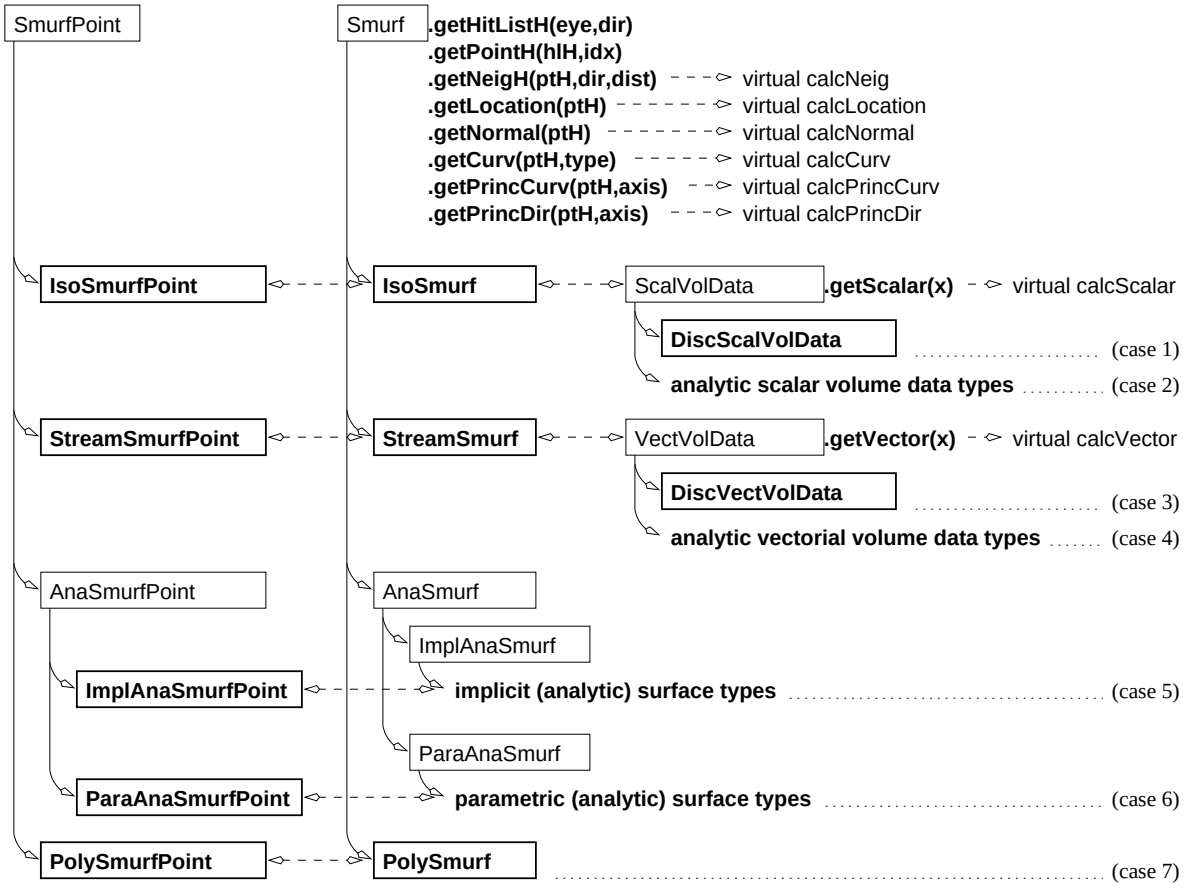



Figure 6: SMURF class hierarchy and interface.

(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 various 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 reused. Again, Fig. 6 illustrates the relations between these classes.

## 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. Fig. 7 shows an iso-surface with crosses aligned to the principal directions (similar to Beck et al. [2]). Fig. 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 computation of Gaussian curvature with the corresponding analytic function. Linear reconstruction of densities is clearly seen in Fig. 10(a), whereas there is

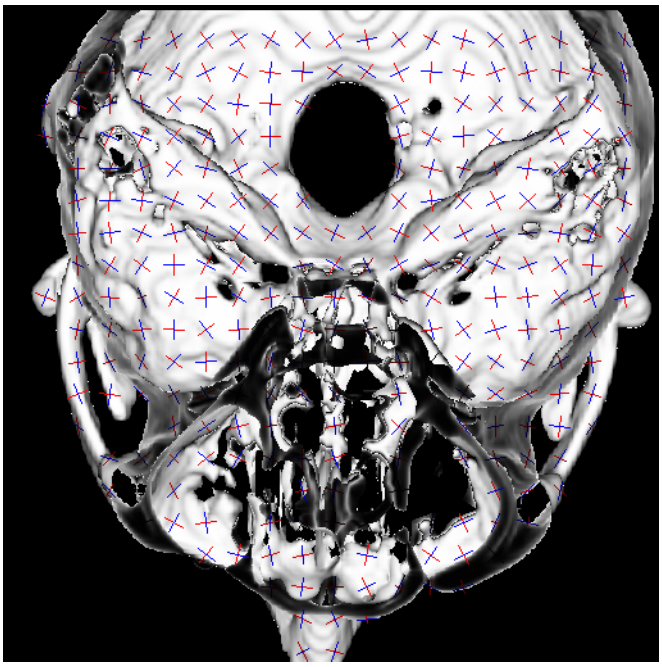


Figure 7: iso-surface computed for ten slices, scanned from a human head with curvature crosses.

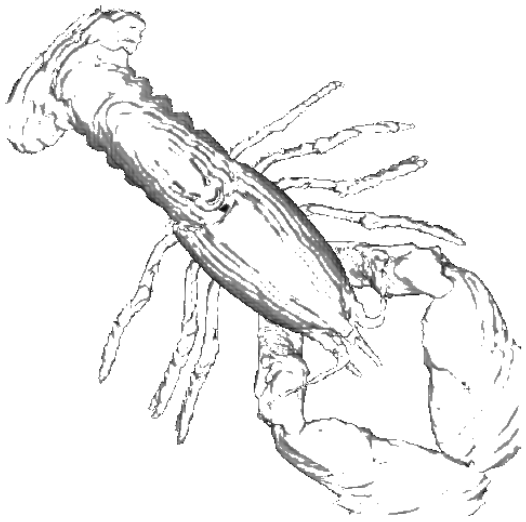


Figure 8: contour display of lobster CT scan – see also Fig. 9.

no perceivable difference between Fig. 10(b) and (c). Comparing the curvature plot, subtle differences can be obtained even between Fig. 10(b) and (c).

Fig. 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 clearly visible. For color plates, please refer to URL `http://www.cg.tuwien.ac.at/research/vis/misc/Smurf/`.

## 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 highly 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 algorithm for calculating particular surface properties. Their portability to other surface types is another advantage of the SMURF concept.

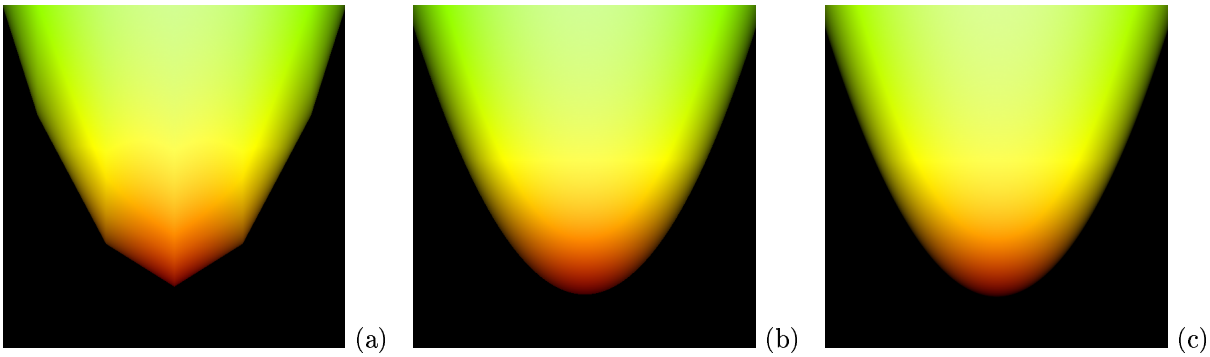
For the realization of this concept we first identified the most often used surface types and compared various surface interrogation algorithms necessary to unify them under a unique interface. As most visualization applications have to deal with sampled data, analytic evaluation is discussed as well as various reconstruction schemes. The usefulness of this approach is demonstrated by several results we obtained

```
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);
}
```

Figure 9: code used for drawing lobster contours depicted in Fig. 8.



(a) (b) (c)

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

with our actual implementation.

## 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.

## REFERENCES

[1] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In A. Glassner, editor, *An introduction to ray tracing*, pages 201–262. Academic Press, 1989.

[2] J. Beck, R. Farouki, and J. Hinds. Surface analysis methods. *IEEE Computer Graphics and Applications*, 6(12):18–36, 1986.

[3] M. Bentum, B. Lichtenbelt, and T. Malzbender. Frequency Analysis of Gradient Estimators in Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):242–254, 1996.

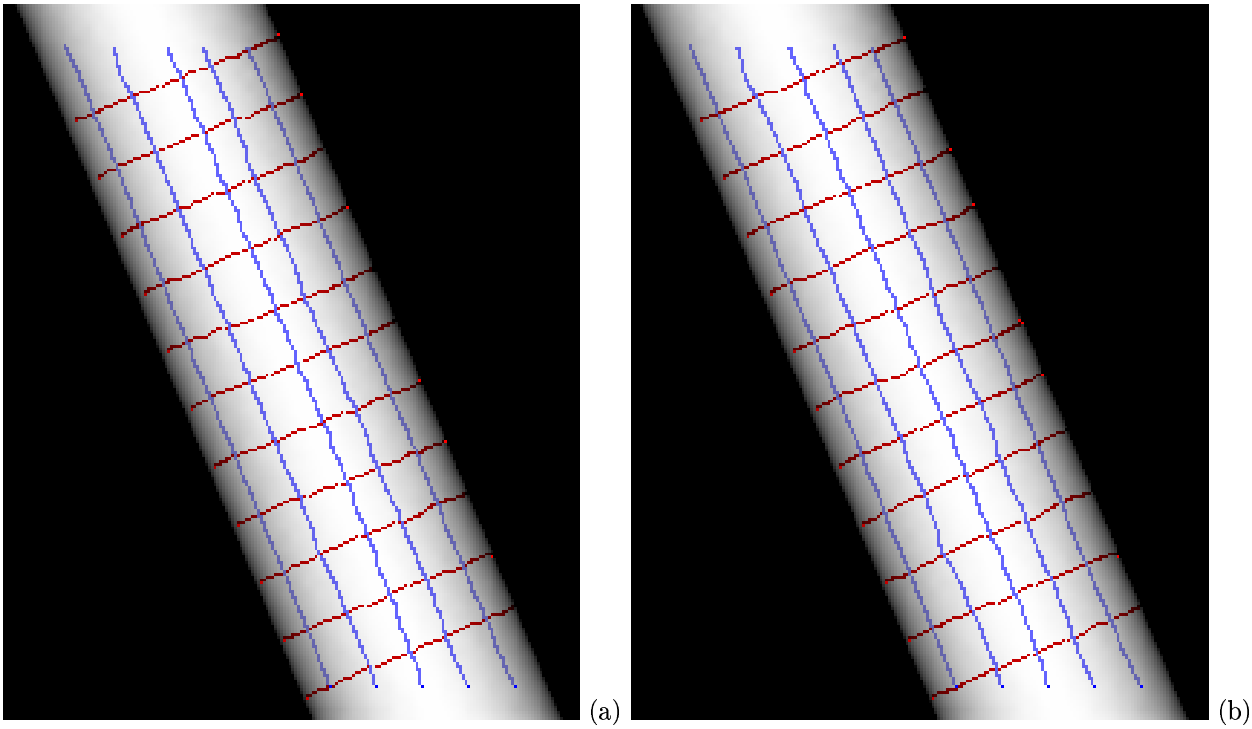
[4] J. Dill. An application of color graphics to the display of surface curvature. *Proceedings SIGGRAPH*, 15(3):153–161, 1981.

[5] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 3$^{rd}$ edition, 1993.

[6] A. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press, 1987.

[7] H. Hagen, S. Hahmann, T. Schreiber, Y. Nakajima, B. Wordenweber, and P. Hollemann-Grundstedt. Surface interrogation algorithms. *IEEE Computer Graphics and Applications*, 12(5):53–60, 1992.

[8] J. Hultquist. Constructing stream surfaces in steady 3D vector fields. In *Proceedings IEEE Visualization*, pages 171–177, 1992.



(a) (b)

Figure 11: quality of curvature calculations depending on the function reconstruction scheme – linear (a) vs. cubic (b) reconstruction


[9] V. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *Proceedings SIGGRAPH*, pages 109–116, 1997.

[10] V. Interrante, H. Fuchs, and S. Pizer. Conveying the 3D shape of smoothly curving transparent surfaces via texture. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):98–117, 1997.

[11] H. Löffelmann, L. Mroz, E. Gröller, and W. Purgathofer. Stream arrows: Enhancing the use of streamsurfaces for the visualization of dynamical systems. *The Visual Computer*, 13:359–369, 1997.

[12] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Proceedings SIGGRAPH*, 21(4):163–169, 1987.

[13] T. Möller, R. Machiraju, K. Müller, and R. Yagel. Evaluation and Design of Filters Using a Taylor Series Expansion. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):184–199, 1997.

[14] G. Nielson and B. Shriver. *Visualization in Scientific Computing*. IEEE Computer Society Press, 1990.

[15] B.-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[16] M. Rumpf, A. Schmidt, and K. Siebert. Functions defining arbitrary meshes - A flexible interface between numerical data and visualization. *Computer Graphics Forum*, 15(2):129–142, 1996.

[17] G. Sakas, M. Grimm, and A. Savopoulos. Optimized maximum intensity projection (MIP). In *Proceedings* EUROGRAPHICS *Rendering Workshop*, pages 51–63, 1995.

[18] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, 1990.

[19] L. Seidenberg, R. Jerad, and J. Magewick. Surface curvature analysis using color. In *Proceedings IEEE Visualization*, pages 260–267, 1992.

[20] P. Todd and R. McLeod. Numerical estimation of the curvature of surfaces. *Computer-Aided Design*, 18(1):33–37, January 1986.