Decal-maps: Real-Time Layering of Decals on Surfaces for Multivariate Visualization

Allan Rocha, *Student Member, IEEE*, Usman Alim, *Member, IEEE*, Julio Daniel Silva, and Mario Costa Sousa

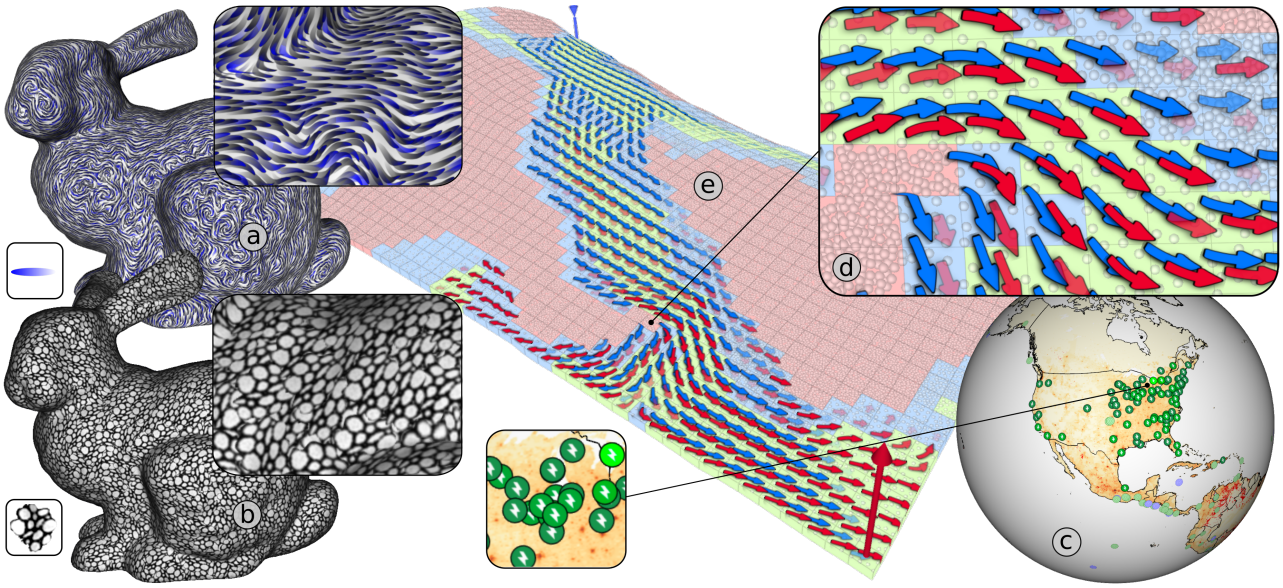

Fig. 1. (a) Vector field visualization using decal-maps, notice how the brush strokes' decals deform to represent vectors. (b) We cover the bunny with decals. (c) Illustrative visualization of nuclear power plants in the United States. (d) Illustrative visualization of six attributes on top of a petroleum reservoir model, namely, rock type, porosity, and directions and magnitudes for oil and water flow. (e) Illustrative visualization of a water-oil simulation time-step, on top of a reservoir model.

Abstract— We introduce the use of *decals* for multivariate visualization design. Decals are visual representations that are used for communication; for example, a pattern, a text, a glyph, or a symbol, transferred from a 2D-image to a surface upon contact. By creating what we define as *decal-maps*, we can design a set of images or patterns that represent one or more data attributes. We place decals on the surface considering the data pertaining to the locations we choose. We propose a (texture mapping) local parametrization that allows placing decals on arbitrary surfaces interactively, even when dealing with a high number of decals. Moreover, we extend the concept of layering to allow the co-visualization of an increased number of attributes on arbitrary surfaces. By combining decal-maps, colormaps and a layered visualization, we aim to facilitate and encourage the creative process of designing multivariate visualizations. Finally, we demonstrate the general applicability of our technique by providing examples of its use in a variety of contexts.

Index Terms—Multivariate, Visualization, Real-time, Decal, Surface, Layering, Design

1 Introduction

Technological advances have led to an increase in data acquisition in several areas of science and applications. In these domains, experts explore multivariate tridimensional data to understand certain phenomena. For example, doctors explore hemodynamic attributes (e.g. wall shear stress and the inflow jet) coming from cerebral aneurysm data [13]; engineers analyse flow attributes (e.g. velocity, vorticity) computed from flow simulations [3]; and geologists explore geological properties (e.g. porosity, permeability) coming from 3D reservoir models [39]. Therefore, multivariate visualizations are essential to facilitate the process of data exploration. Moreover, tridimensional data is commonly explored in terms of surfaces due to its complexity. In the previous examples, the surfaces can encode one or more attributes that need to be interpreted and correlated for decision making. However, *what* to visualize and *how* to visualize multiple attributes in a single view [12, 38] are still key visualization challenges to address.

Layering techniques [16] try to visualize multiple attributes combining glyphs, colors and textures [59]. Analogically, the concept of layering refers to the way that an artist creates oil paintings on a 2D canvas. In visualization, layering techniques overlay data attributes mapped to a visual representation (e.g. glyphs, colors, textures) onto a plane or screen. They explore the empty space between objects and the transparency between layers. For example, 2D flow visualizations combine arrow glyphs and color to represent magnitude and direction of a vector field [3]. Another example is geo-visualization where we commonly see glyphs, colors, lines, and text overlaid on maps (e.g. Google Maps). In 2D, layering techniques have been successfully explored in information and scientific visualization [12, 16, 27, 59].

On surfaces however, layering techniques have received less attention. Most of the extant techniques combine a texture-based approach, such as line integral convolution (LIC) [5] with colormaps [17, 23]. Additionally, the number of co-visualized attributes on surfaces is reduced when compared to 2D layered visualizations. For example, Kirby *et al.* [19] design a 2D layered visualization with six to nine

• *Allan Rocha, Usman Alim, Julio Daniel Silva, and Mario Costa Sousa are with the University of Calgary.*
E-mails: {acarocha, ualim, jd.silva, smcosta}@ucalgary.ca

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

flow attributes, whereas on surfaces, the number is limited to two or three. On surfaces, we need to consider more complex variables such as curvature, depth and occlusion. Therefore, the layering of glyphs, textures, colors and other visual variables on arbitrary surfaces is not straightforward. Nevertheless, since surfaces are two-dimensional objects, a natural extension of the work of Kirby et al. [19] is to create multivariate layered illustrative visualizations on surfaces.

In this work, we introduce the concept of *decals* for multivariate visualization design. A decal is a pattern or image printed on a material (e.g. plastic or paper) to be transferred to another surface upon contact (a process called *decalcomania*) [32]. Decals are designed and used for several forms of expression and communication. They can encode information in the form of glyphs, textures, text or other visual signs. In computer graphics, decals are commonly used for their aesthetic value; artists place and edit decals on 3D models to create characters and scenarios for games and animations [32, 45].

Decals can represent multivariate data via a *decal-map*: a set of patterns or images designed to represent one or more data attributes. Decal-maps can be 1D (array of images) or 2D (matrix of images). For example, a set of arrows of different sizes and colors is a case of a 1D decal-map and a matrix of images representing variations of a tensor is a 2D decal-map. By designing decal-maps, we can create complex glyphs (e.g. Chernoff Faces [7]) encoding data and semantic information. We can also use elements such as texture patterns, shapes or normal maps for this purpose. Additionally, we can explore visual variables such as position (e.g. using importance sampling) and size to represent data [40]. It is by combining layering techniques and decal-maps that we design multivariate visualizations. For example, Fig. 1(d) shows the visualization design of six geological attributes; it combines decal positions, decal-maps and a pastel colormap in a layered fashion. Thus, decal-maps provide a new way of visualizing data by giving more freedom to design glyphs and patterns to represent multivariate surface data.

In order to place decals on surfaces, we need a texture mapping technique with the following three requirements. 1) It should be independent of the surface parametrization (texture coordinates) since many scientific datasets are not represented parametrically; 2) it should not rely on topological information from the underlying mesh such as geodesic distance because this is expensive to compute and depends on the mesh type; and 3) it should be efficient as we need a high number of decals to represent data attributes in several locations of the surface. These requirements are stringent and are not met by many established decal placement techniques such as path-based methods [32, 37] and exponential maps [45]. Deferred decals [20] and implicit decals [10] are an exception and can render a high number of decals interactively via local texture parametrizations. Inspired by these techniques, we propose a simple technique called *layered decals* that is specially designed for multivariate surface visualizations. Our technique relies on the observation that the intersection between a solid sphere and a surface is a disk. We denote this intersection as a *sphere mask*. Using sphere masking, we can compute a local texture parametrization and map decals that follow the surface geometry. Our technique is flexible and can handle overlapping decals as well as multiple layers of decals.

The main contributions of this paper are:

- Decals as a form of representation for visualization design, from observing the use of 2D-glyphs and textures in scientific and information visualization.
- The concept of *decal-map* to represent and visualize multivariate data on surfaces.
- A real-time technique to place decals on surfaces using a local (texture mapping) parametrization.
- Inspired by Kirby *et al.* [19], we extend the concept of layering to allow the co-visualization of an increased number of attributes on surfaces.
- A multivariate illustrative visualization design for geological data to visualize, for the first time in the domain, six attributes simultaneously.
- An abstract framework applicable to a variety of contexts.

We provide concrete examples for geological and geographic datasets in order to validate our technique and demonstrate how multivariate visualizations on surfaces can be designed (Sec. 5). It is important to mention that our work was envisioned by researchers such as Crawfis and Allison [9], Laidlaw *et al.* [22], Kirby *et al.* [18], and Taylor [50] many years ago. Our goal is to bring back the concept introduced by them, showing that it can be used to create multivariate illustrative visualizations on surfaces.

2 Related Work

We believe that visualizations applied to planes also provide useful guidelines to design layers for surfaces. Therefore, we present three main topics related to our work: glyph-based visualization, layering techniques, and decals on surfaces.

2.1 Glyph-Based Visualization on Surfaces

Design and Guidelines: Glyphs are widely used in visualization to represent multivariate data [1, 58]. Ropinski *et al.* [40] propose a taxonomy that classifies glyph-based techniques applied to spatial medical data. They distinguish between *pre-attentive* (visual elements that catch our attention quickly) and *attentive* (elements that need focused attention due to existent distractors) glyph properties. Following a broader approach, Borgo *et al.* [1] survey the state of the art of glyph-based visualization techniques. Their work takes a holistic overview of these visualizations in terms of concepts and theory, design guidelines, algorithms and techniques, and applications. They also discuss the historical use of glyphs as visual signs for communication.

Since 2D glyphs can be used as decals (on surfaces), we argue that the guidelines introduced by Ropinski *et al.* [40] and Borgo *et al.* [1] are very relevant to the problem of designing decal-maps for surfaces.

Two Dimensional Glyphs on Surfaces: Some works explore image-space approaches to visualizing glyphs that encode scientific data. Peng and Laramee present an image-space glyph placement technique to visualize boundary flow [34]. Their technique projects a vector field to the R,G,B, and alpha color channels in image-space. Using arrow glyphs, they visualize direction and magnitude (arrow color) of a vector field defined on complex adaptive meshes from computational fluid dynamics (CFD). Later, Peng *et al.* used this image-space approach to develop a robust and automatic clustering algorithm that combines the properties of the underlying vector field and mesh into a unified error-driven representation [33]. For tensor data, Chen *et al.* propose an image-space technique to visualize asymmetric tensor fields using evenly-spaced hyperstreamlines and elliptical glyphs [31]. While these image-space techniques are definitely useful, they can lead to flickering, popping and other perceptual artifacts during interaction such as the so-called 'shower-door effect'; the animated model appears as if it is seen through a textured glass door [26]. These problems are avoided by rendering the glyphs only when the user stops interacting with the model. In our case, these perceptual issues do not occur since we apply decals to surfaces in object space.

In object-space, the problem of using 2D glyphs to encode data on surfaces is relatively less explored. Sanyal *et al.* use glyphs, ribbons and spaghetti plots to visualize ensemble uncertainty in weather data [41]. Pelt *et al.* design and place four 2D glyphs (disk, flower, web and time variation strips) on a surface extracted from aneurysm data to explore wall shear stress [56]. The design of these glyphs is inspired by information visualization. Using these glyphs, they implement a details-on-demand approach that places glyphs on the surface of the aneurysm model depending on the level of zoom during data exploration. In cardiac visualization, Termeer displays a 2D glyph on the surface of the epicardium to represent perfusion data [51].

For the placement of 2D glyphs in object-space, a common technique (such as the one used by Pelt *et al.* [56]) is to create an oriented quad on the surface and generate texture coordinates to map or design a desired glyph representation. However, this approach has its limitations. The quads can be clipped by the surface depending on factors such as the size and number of glyphs, and the curvature of the surface. An attempt to avoid the clipping process is to offset the glyphs from the surface, which can lead to the glyphs flying in space (Sec 4.6). In

contrast, our approach avoids these problems by mapping decals so that they follow the curvature of the surface.

2.2 Layering Techniques on Planes and Surfaces

Layering refers to a multi-pass rendering technique where each data field is mapped to a visual representation [4]. Following an artistic analogy, this technique is similar to the way artists paint, for example, background as first layer, large objects as second layer and details as third layer. Using proper visual encoding and concepts of painting, Kirby *et al.* [19] were able to display 6–9 data values from a 2D flow dataset. In biological science, Laidlaw *et al.* apply layering techniques to visualize diffusion tensor images of the mouse spinal cord [22]. This technique also follows a painting metaphor and layers up to four attributes on a 2D slice of the volume data. More recently, Schroeder *et al.* present a sketch-based interface to create multivariate visualizations based on concepts of art and brushing [46]. In short, these works emphasize that art can give good insight into creating visualizations of multiple attributes. In addition, it can also increase the aesthetics of the visualization [15, 21, 49, 53].

Crawfis and Alisson present a novel approach to visualize multiple attributes on surfaces using texture mapping and color [9]. Later, Taylor examines several approaches for the representation of multiple attributes using visual variables such as color, texture, and density [50]. Another example of layering can be found in cardiac visualization. Termeer creates a multivariate visualization on the epicardium surface encoding three quantities: functional data (wall thickening) using a color coding, perfusion data using 2D glyphs of varying sizes, and scar displayed using a striped pattern overlay [51].

Many layering techniques have been applied to flow visualization [3, 23, 25]. These techniques either combine multiple 2D vector fields based on texture and color [54, 55], or display 3D vector fields on layered surfaces [6]. They explore techniques such as LIC [5] to represent the direction of a vector field overlaid with another attribute represented by a colormap. Khlebnikov *et al.* introduce a procedural texture synthesis technique based on random-phase Gabor noise [17]. They can layer three attributes where the first is modulated by the texture frequency of the gabor noise, the second by the texture orientation and the third is color coded.

Many of the designs and techniques applied on surfaces can be implemented or combined with decals. Our goal is not to replace any of these techniques but to complement them by providing a new way of designing multivariate visualizations.

2.3 Decals on Surfaces

In computer graphics, decals are scene elements, mostly represented as small tiles, which are individually placed on a surface via a local parametrization of the surface [10, 45]. They are routinely used by artists to incorporate details in 3D scenes and characters. Perdersen was one of the pioneer researchers to create a decalling interface for this purpose [32]. However, Perdersen's system had several practical limitations such as the requirement of a global parametrization of the 3D object [32]. Later, other works have proposed techniques to apply decals locally on surfaces. Lefebvre *et al.* apply decals using a technique based on hardware-accelerated octree textures and planar projection [24]. This technique can apply decals on arbitrary surfaces. However, Schmidt et al. argue that this introduces significant distortions on curved surfaces (due to planar projection) and does not easily support complex editing operations [45]. Schmidt *et al.* propose a decalling interface based on the *exponential maps* [11, 45]. Exponential maps [45] provide a robust technique to place decals on surfaces since they compute a local parametrization based on the geodesic distance. More recently, Schäfer *et al.* present a technique that demonstrates an efficient employment of decals by modifying and subdividing 3D surfaces using an efficient GPU memory management scheme [42]. However, these techniques depend on the surface representation and are not feasible when a very large number of decals needs to be positioned and displayed interactively [10].

Deferred decals [20] is another technique that has become popular in games due to its efficiency and simplicity. It consists of converting the scene's world-space position stored in the geometry buffer (G-buffer) to 'decal space' [20] which is defined by the decal bounding box. Texture coordinates are obtained by simply projecting the decal orthogonally onto the surface. This technique works well for planar and low curvature surfaces. However, for other cases such as thin surfaces (e.g. thin tube), areas of high curvature and at sharp edges, it may introduce stretching or smearing artifacts due to the orthogonal projection [20, 45]. Moreover, the decals do not properly wrap around the surface. Stretching and smearing are alleviated by clipping or fading out the decals, a solution that is not suitable for multivariate data visualization. Surface wrapping is improved by changing the texture coordinates based on the surface normals. *Volume decals* attempt to address some of these problems by using a volumetric decal that is sampled by the surface [35]. However, this approach has a higher memory consumption and is only suitable for 2D images that can be easily extended to 3D.

Following a different approach, de Groot *et al.* propose a local parametrization known as *implicit decals* [10]. This technique is based on the assumption that for small decals, fine distortion control is not required. The local texture parametrization is given by an isotropic spherical field function centered locally at a surface position. Using polar coordinates, the authors calculate an angular coordinate and a radial coordinate based on the Euclidean distance. Their implicit method uses an octree representation to avoid the inclusion of all decals in the calculation of the final texture coordinates [10]. Their results illustrate that the Euclidean distance is sufficient in approximating the surface locally, thus avoiding the far more expensive computation of the geodesic distance [10]. However, Euclidean distance may not be enough in areas of high curvature, introducing undesired distortions.

Our technique is inspired by both deferred decals [20] and implicit decals [10] but has some important differences. Comparing with deferred decals, instead of a box whose orientation depends on the underlying mesh, we place an orientation independent sphere on the surface centered at the location where the decal is to be placed. However, the sphere itself does not provide the area on the surface where the decal should be placed; this is given by the part of the surface that lies inside the sphere (sphere masking (Sec. 3.1)). We efficiently compute this area and texture coordinates using the G-buffer in a multi-pass approach (Sec. 4). The main advantage of our approach over deferred decals is that it does not compute texture coordinates via an orthogonal projection but instead, it computes them using sphere masking which can easily accommodate various approximations to the geodesic distance. This also differentiates our method from implicit decals which only considers the Euclidean distance [10]. Furthermore, when compared with implicit decals, our technique does not require auxiliary data structures such as octrees or 3D textures and computes texture coordinates only for visible decals. It is intentionally designed to leverage the GPU pipeline for the purpose of creating layered visualizations of multivariate data. Inspired by this, we denote our technique as *layered decals*. To the best of our knowledge, we are unaware of any other works that use decals for data visualization on surfaces.

3 Decal Parametrization

In this section, we describe the parametrization that allows us to place decals on surfaces. In order to use *decal-maps* to visualize multiple surface attributes, we design our technique focusing on the layering process. We remark that this parametrization does not rely on expensive computations for rendering or for texture mapping the decals.

3.1 Sphere Masking and Local Parametrization

We begin with a surface on which we want to place decals. Let us denote this surface as M. The first problem we need to tackle is the one of building a parametrization of this surface that would allow the placement of decals, *one at a time* in an efficient manner. Since decals are small, such a parametrization need not be global. Therefore, for each decal, we build a local parametrization on M that is defined only inside a Euclidean ball containing the decal.

Let us assume that the intersection of the surface M and a ball B_c, centered at the point c in M, is a disc $D_c := B_c \cap M$. The disc D_c may be understood as a patch of M over which a decal is to be placed. We refer the reader to Fig. 2 to illustrate the discussion that follows.

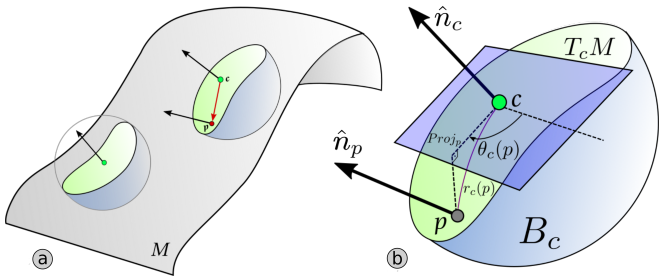

Fig. 2. Sphere masking approach: (a) Patch $D_c = B_c \cap M$, shown in green. (b) Polar coordinate system on patch D_c. Angular coordinate $\theta_c(p)$: angle between the projection of p on the tangent plane T_cM and a reference vector. Radial coordinate $r_c(p)$: approximates the geodesic distance from c to p.

Let us consider the tangent plane of surface M at point c, which we denote as T_cM. On such a tangent plane, we can define a Cartesian coordinate system given by a previously chosen orthogonal basis $\{\hat{u}_c, \hat{v}_c\}$, for which any point $x\hat{u}_c + y\hat{v}_c$ in T_cM can be mapped to a pixel $\{x,y\}$ in the texture. The orthogonal basis $\{\hat{u}_c, \hat{v}_c\}$ can be either a fixed *a priori* choice (*e.g.* randomly chosen) or computed from the data as explained in Sec. 3.4.

The problem of texture mapping then becomes the problem of matching points p in the patch $D_c \subset M$ to pixels in the decal. Since we are assuming that the patch $D_c = B_c \cap M$ is a disc, we can choose a radial coordinate system in D_c. Any point p in the patch D_c has the coordinates $p = (r_c(p), \theta_c(p))$. Point p is then mapped to the pixel $(x(p), y(p))$ in T_cM for which $r_c(p) = \sqrt{(x(p)^2 + y(p)^2)}$, and $\theta_c(p) = \arctan(x(p), y(p))$. Since D_c is a disc, these equations always have a solution $\{x(p), y(p)\}$.

The angle $\theta_c(p)$ can be obtained by simply projecting p onto T_cM and computing the angle of this projection with a preselected vector in T_cM, *e.g.* $\hat{u}_c$. The radius $r_c(p)$, however, is given by the geodesic distance between c and p in M, which may be expensive to compute in the general case. Because decals are supposed to be small compared to the surface M, we can use approximations of the geodesic distance to estimate $r_c(p)$. We discuss three such approximations in the next section. In Sec. 4, we show how to use the graphics pipeline to perform these computations efficiently.

3.2 Estimating the Radial Coordinate

Any good approach to estimating the radial coordinate would have to consider the problem of minimizing texture stretching, given an allowed bound for its computational cost. However, since decals are small in comparison with the original surface, fine distortion control may not be required as discussed in de Groot *et al.* [10].

It must be stressed that estimating the radial coordinate efficiently and accurately is a problem well outside the scope of the present work. We here present estimates that are either straightforward (Euclidean distance) or already available in the literature. We provide in Sec. 3.3 a simple, yet objective, approach to compare such estimates. We remark that our decal-mapping implementation is independent of how one estimates the radial coordinate, thus allowing our algorithm to remain useful as the significant topic of approximating the geodesic distance on surfaces evolves.

Euclidean Distance: A simple way to estimate the radial coordinate $r_c(p)$, for a point p in the patch D_c, is to approximate it by the Euclidean distance $r_c(p) \approx \|p-c\|$. This approximation can provide good results, as seen in the work of de Groot *et al.* [10], yet it suffers from texture stretching in the general case. At its core, the Euclidean distance uses solely the position of point p to estimate the radial coordinate $r_c(p)$. Additional information from M must be used to mitigate texture stretching.

Cosine Interpolation: Bowers *et al.* [2] introduce cosine interpolation as a means to efficiently approximate the geodesic distance on a surface, which they use to discard samples during the execution of their Poisson sampling technique. We have used this distance to reduce texture stretching caused by the Euclidean distance.

Cosine interpolation uses the surface normals at c and p, $\hat{n}(c)$ and $\hat{n}(p)$ respectively, to approximate the geodesic distance. Writing the cosine of the angle between $\hat{n}(c)$ and the unit vector pointing from c to p as $q_c = \hat{n}(c)\cdot(p-c)/\|p-c\|$, as well as writing the cosine of the angle between $\hat{n}(p)$ and the aforementioned unit vector as $q_p = \hat{n}(p)\cdot(p-c)/\|p-c\|$, one can linearly interpolate between these two values by writing $q(t) = (1-t)q_c + tq_p$. The approximation is given by:

$$r_c(p) \approx \int_0^1 \frac{\|p-c\|}{\sqrt{1-q(t)^2}}dt = \frac{\arcsin(q_c) - \arcsin(q_p)}{q_c - q_p}\|p-c\|, \quad (1)$$

and $\|p-c\|/\sqrt{1-q_c^2}$, for $q_c = q_p$. This formula has two remarkable features [2]: 1) when c and p are coplanar, it becomes the Euclidean distance; and 2) when c and p lie on a sphere with radius R, it yields the exact geodesic distance $2R\arcsin(\|p-c\|/2R)$.

The Isophotic Distance: Pottman *et al.* [36] envisioned a distance that is based on the Gaussian map of a surface M, *i.e.*, the map that associates to any point p in M its normal $\hat{n}(p)$. Given two points p_1 and p_2 in M, the *pure isophotic distance* between p_1 and p_2 is the angle between $\hat{n}(p_1)$ and $\hat{n}(p_2)$. As Pottman *et al.* observe, this distance is not a metric. They then proceed to build a metric from the *pure isophotic distance* by combining it with the Euclidean metric, which finally yields the *isophotic metric*.

Inspired by Pottman *et al.*, Geng *et al.* propose a weighted distance field to locally approximate the geodesic distance during their Poisson sampling computation [14], which they still denote as an isophotic distance and write as:

$$r_c(p) \approx \|p-c\|\left(1 + (1 - \hat{n}(p)\cdot\hat{n}(c))^b\right), \quad (2)$$

where the parameter b is used to adjust the influence of the normals on the distance. In our tests, the range $4 \leq b \leq 6$ yielded the best results.

3.3 Estimating the Approximations' Error

In order to compare the quality of approximations for the radial coordinate, we use the fact that any smooth surface can be locally approximated by an *osculating paraboloid* (Fig. 3(a)), see do Carmo [11]. Thus, we can reduce the problem of comparing decals over small patches of M to decals over paraboloids that approximate M well enough in such patches. Moreover, in paraboloids, the radial coordinate can be computed *exactly*. Therefore, we can provide objective evidence to validate the usefulness of the estimates we previously described for decal mapping, at least in the case of small decals. We can also use paraboloids as a synthetic benchmark for mapping large decals, because, no matter how big the decal may be, we will always be able to compute the radial coordinate exactly. Details are provided in the next subsection.

A comprehensive discussion on the effect of the approximations' distortion on decals up to the scale of the surface is well outside the scope of this work. However, this framework to evaluate the approximations' error is useful because: 1) it is based on a set of surfaces that are simple, yet not trivial as the plane or the sphere; 2) it is easy to implement; and 3) it is general, in light of the fact that surfaces can always be locally approximated by osculating paraboloids.

Computing the Exact Radial Coordinate on Paraboloids: Here we fix $c = (0,0,0)$, and the normal $\hat{n}(c) = (0,0,1)$. The curvatures of the paraboloid at c are given as k_1, k_2. For simplicity, we fix the principal directions of curvature as $\hat{t}_1 = (1,0,0)$ and $\hat{t}_2 = (0,1,0)$. Thus, for any point $p = (p_1, p_2, p_3)$ in this paraboloid, the radial coordinate is simply the length of the parametric curve:

$$\gamma(t) = \left(p_1 t, p_2 t, \tfrac{1}{2}(k_1(p_1 t)^2 + k_2(p_2 t)^2)\right), \quad (3)$$

which connects the center c to the point p. The radial coordinate for point p is given as $r_c(p) = \int_0^1 \|\gamma(t)\|dt$, which yields the formula:

$$r_c(p) = \tfrac{1}{2}\left(\sqrt{A+B} + \left(A/\sqrt{B}\right)\operatorname{arcsinh}\left(\sqrt{B/A}\right)\right), \quad (4)$$

in which $A = p_1^2 + p_2^2$ and $B = (k_1 p_1^2 + k_2 p_2^2)^2$.

Comparison: In our average case, neither the curvatures have significant sizes, nor are they several orders of magnitude distinct. Quantitatively, we allowed the principal curvatures to vary in the $[-63, 64]$ range in our tests. In these conditions, all distances we used yielded

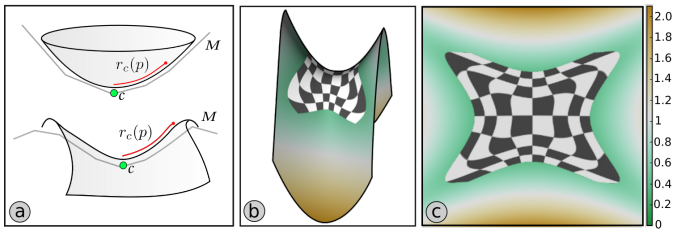

Fig. 3. (a) (top) Elliptic paraboloid and (bottom) hyperbolic paraboloid; the paraboloid adapts to the curvature of the surface. (b) Checkerboard texture mapped to a hyperbolic paraboloid. (c) Top view of (b).

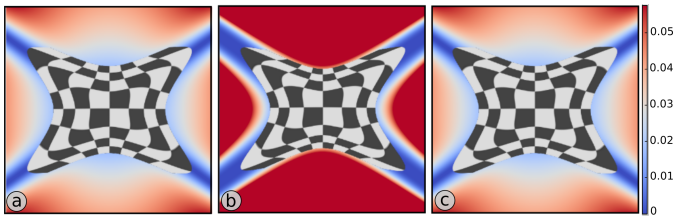

Fig. 4. Checkerboard decal mapped into a hyperbolic paraboloid ($k_1 = -8$, $k_2 = 16$). Colors: blue represents zero error, red represents maximum error for the Euclidean distance. (a) Euclidean distance, max$\|error\| = 0.06$; (b) cosine distance, max$\|error\| = 0.7$; and (c) isophotic distance, max$\|error\| = 0.06$.

good qualitative results when compared to the geodesic distance of the paraboloids (4). Figure 3(c) displays the top view of texture mapping a checkerboard to a hyperbolic paraboloid ($k_1 = -8$ and $k_2 = 16$). Notice the necessity of using the geodesic distance of the paraboloid as a base case to compare the other distances, since (even for these mild curvatures) texture stretching is significant. The colormap in Fig. 3(b and c) depicts geodesic distance.

Quantitatively, both the Euclidean and the isophotic distances behaved similarly with errors no greater than 10^{-1}, but usually an order of magnitude smaller. The cosine approximation deviated the most from the geodesic distance on paraboloids with high curvatures, yet, remarkably, its results were still qualitatively similar to the Euclidean and the isophotic distances. By inspecting the error function, instead of its maximum, one can observe that the cosine distance's error spreads out uniformly as points get away from the center, which contributes to its good qualitative behavior. For paraboloids with small curvatures, the cosine distance yielded the best results. Fig. 4 illustrates the same checkerboard mapped into the paraboloid, now with the error compared to the geodesic distance. The qualitative differences of those mappings to the geodesic map (Fig. 3(c)) are subtle.

3.4 Allowing the Angular Coordinate to Vary with Data

When the data includes a vector field, we may use the normalized value of this vector field at the center c as a basis vector for T_cM, *e.g* we may choose this vector in place of $\hat{u}_c$ (see Sec. 3.1). If $\widehat{V}$ denotes the normalized vector field, we have $\hat{u}_c = \widehat{V}(c)$. This choice orients the decal in the direction of the vector field.

We can further improve the sense of flow by incorporating information about the local variation of the vector field around c. To accomplish this, given a point p in the patch D_c, we compute the angle coordinate $\theta_c(p)$ in a slightly different manner from what we described in Sec. 3.1. We first compute the angle between $\hat{u}_c$ and the projection of p onto T_cM, which we denote as $\theta_c^0(p)$. This angle is exactly what we used as the angular coordinate for p in Sec. 3.1. We then compute the angle between $\hat{u}_c$ and the projection onto T_cM of the the vector $\widehat{V}(p)$; we denote this angle as $\theta_c^{\widehat{V}}(p)$. The angle coordinate of p is finally given as:

$$\theta_c(p) = \theta_c^{\widehat{V}}(p) - \theta_c^0(p). \quad (5)$$

Fig. 1 (b and d) shows the decals deformed by the vector field. These images were rendered using the isophotic and Euclidean distances respectively. The decals follow the vector field direction improving the sense of flow.

4 Layered Decals

We now present our framework to map and visualize decals on surfaces. Our algorithm is independent of the surface representation and allows a high number of decals to be rendered at interactive frame rates. Fig. 5 provides an overview of our implementation. We explore the graphics pipeline using OpenGL and GLSL. Our implementation to create a surface layer consists of a multi-pass approach which is divided into three main steps: 1) compute the *geometry buffer* (G-buffer); 2) compute the *sphere masking buffer* (SM-buffer); and 3) apply the decals on the surface (*decal mapping*). In the next three sections, we detail each one of these steps.

4.1 Surface G-buffer

In the first pass (Fig. 5, 1st pass), we send the surface geometry and attributes to the GPU, and render it to a framebuffer object (FBO) with three attachments (G-buffer). In the fragment shader, we output the surface vertices, depth, normals and colors to this framebuffer. After the depth test, the G-buffer contains the attributes of the visible faces of the model stored in pixel space (RGBA).

4.2 Sphere Masking Computation

In the second pass, we compute the intersections between the spheres and the surface and store them in a buffer that we call the *SM-buffer*. Computing these intersections analytically is impractical but we can use the GPU to do this efficiently. Before computing the intersections, we need to distribute samples over the surface at the locations where we want to place the decals. This distribution depends on factors such as data resolution, attribute variation, or even user interest. Therefore, depending on the application, this distribution can be given by a uniform or data driven sampling strategy [40]. In our work, we consider uniform and Poisson sampling [8].

The generated samples are sent to the GPU along with a coarse sphere mesh. We employ *instanced rendering* [48] to efficiently render a sphere at each sample location. With face culling turned off, we test the depth of the sphere fragments against the depth of the surface fragments (from the G-buffer). If a sphere fragment is above the surface, we discard it (surface clipping). For the remaining fragments of the sphere, we create a mask by checking if they belong to the front-faces or back-faces (Fig. 5, 2nd pass). After the depth test is performed, the final image contains the visible front and back faces of the clipped sphere rendered to a framebuffer. The visible back-face fragments correspond to the intersection between the sphere and the surface (sphere masking). This approach is fast and allows us to compute a high number of intersections in parallel, limited only by the number of spheres that can be rendered using instanced rendering.

4.3 Decal Mapping

In the third pass, we render the decals on the surface. To do so, we draw a screen quad and input the G-buffer and the SM-buffer in the fragment shader to compute the local parametrization. We only consider the fragments that are marked in the SM-buffer as back faces (Fig. 5, 3rd pass). For each of these fragments, we access the sphere ID stored in the pixel channel during the previous pass. With this ID, we access the center, normal and other attributes stored as texture buffer objects (TBO) [48]. Hence, using these sphere attributes as well as the mesh attributes from the G-Buffer, we build the local coordinate system at the center of the sphere and compute the parameters $r_c(p)$ and $\theta_c(p)$ following the approach introduced in Sec. 3. Finally, using the surface data attributes, we access the decal-map and map the corresponding decal to the surface. It should be stressed that, unlike implicit decals [10], we compute the local parametrization in a deferred fashion only for visible decals.

4.4 Sphere Overlapping and Layered Rendering

The overlapping between visual elements can be useful for visualization. For example, a slight overlap between decals can convey a sense of connectivity [59] (e.g overlapped arrow glyphs can increase the sense of continuity; or overlapping circles the sense of proximity). In a multiscale visualization, we may want to increase the decal sizes to generate a clustered visualization in order to emphasize parts of the

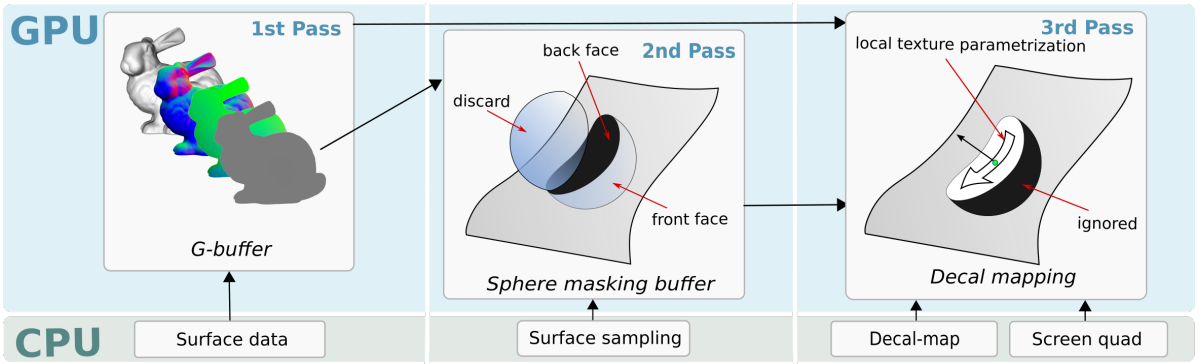

Fig. 5. Implementation overview: (1) The geometry buffer (G-Buffer) is computed; (2) A sphere distribution is generated over the surface. Sphere fragments above the surface geometry are clipped and the remaining labeled as front and back faces. In image-space, the back faces occluded by the front faces represent the intersection between the surface and the sphere creating the sphere masking buffer (SM-Buffer); (3) Using the G-buffer and the SM-buffer, we compute the local parametrization and perform decal mapping.

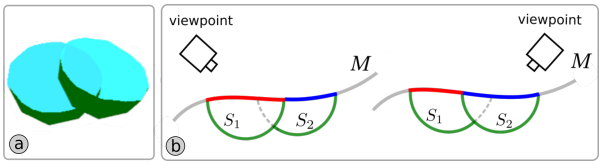

Fig. 6. (a) In image-space, the visible front faces "cut" the visible back faces representing the sphere intersections with the surface. (b) Decals (red and blue) seen from different points of view. From one point of view, the fragments of the front faces of S_2 (dashed curve) are seen inside of S_1; from the second point of view, otherwise.

data during an overview; or reduce the decal sizes during a zoom-in interaction to visualize their specific locations. In order to achieve this, we need to adapt our GPU algorithm to handle sphere intersections. The number of intersections depends on the radius of the sphere as well as other factors such as data resolution and local feature size. In our implementation, sphere radius is a user controlled parameter.

In our sphere masking approach, we use the back and front faces of the spheres to obtain the intersection between the spheres and the surface. However, when two or more spheres are intersecting each other, this approach leads to some artifacts. Consider two intersecting spheres S_1 and S_2 as shown in Fig. 6. Depending on the point of view, the front faces of one sphere overlap the back faces of the other sphere in screen space (Fig. 6(a)). We could discard the front face fragments of S_1 inside of S_2 or vice-versa. However, we cannot discard the front face fragments that are outside of the intersection since they are used for sphere masking – the intersection with the surface is given by the overlap between the front and back faces in screen space. Note that in this situation, we have two conditions for the front face fragments in the same shader. This is however a difficult problem to solve using fragment-based operations only. Even if we were able to solve it, a change of view point during the interaction would switch the order between the spheres and consequently the decals (Fig. 6(b)).

To solve this problem, we need to consider the order between the overlapping spheres. A solution is to render the sphere S_1 to FBO_1 and the sphere S_2 to a FBO_2 and compose the final image combining both buffers. In order to implement this idea, we first detect the intersections between the spheres for a chosen radius. This is done in the CPU in a pre-processing stage. We formulate the problem as follows. Given a set $\mathbb{S}$ of overlapping spheres, find N non-overlapping subsets $\mathbb{L}_k$ such that $\mathbb{S} = \bigcup_{k=1}^{N} \mathbb{L}_k$, and for any X and $Y \in \mathbb{L}_k$, X does not intersect Y. Each subset L_k thus represents a sub-layer of non-overlapping spheres. We use a simple collision strategy (Algorithm 1) to obtain the subsets $\mathbb{L}_k$. If there are n spheres, this algorithm is $O(n^2)$ in the *worst-case* since it has to conduct $n(n-1)/2$ collision tests. In general, it can achieve $O(n \log n)$ using *broad-phase algorithms* [30]. However, in our collision scenario (e.g. centroid based comparison and non-deformable objects) spatial hashing collision strategies [52] can solve the case in $O(n)$. See also a parallel implementation [2, 30].

After the non-overlapping subsets are determined, we label each

Algorithm 1 Splitting $\mathbb{S}$ into N non-overlapping sets

```
1:  i = 0
2:  while S is not empty do
3:      i ← i + 1, L_i = ∅
4:      remove first sphere from S and add it to L_i
5:      for all remaining spheres S in S do
6:          if S does not intersect any of the spheres in L_i then
7:              remove S from S and add it to L_i
8:          end if
9:      end for
10: end while
11: N ← i
```

	No. decals	No. sub-layers	FPS
random uniform sampling	50K	26	19
	100K	41	11
	150K	58	7.5
	200K	72	5.9

Table 1. Performance results for the Stanford bunny model using a screen resolution of 1280x1024 (laptop Intel®i7 with a GeForce GTX 960M 2G) and sphere radius of 0.003.

sphere with its corresponding subset ID. We send all the spheres with their respective IDs to the GPU to be rendered to a target framebuffer. However, in order to render all the spheres at once (instead of a multi-pass approach), we use as a render target a *layered framebuffer object* (LFBO) [48]. The LFBO is a set of framebuffers containing their own depth and color attachments. The number of framebuffers is controlled by the number of sub-layers N. Each sphere is assigned to its corresponding sub-layer during the rendering. This avoids the sphere overlapping problems described previously since each sub-layer performs its own depth test operations and there are no intersections per sub-layer. In order to map decals to each sub-layer, we send the LFBO as a 2D texture array to the fragment shader in the third pass and iterate over each sub-layer to apply decal mapping as outlined in Sec. 4.3. The resulting fragments are composited to produce the final image. Similar to volume rendering, if the accumulated opacity becomes one during compositing, we do not need to iterate over the other sub-layers.

In order to handle multiple layers, we simply repeat this procedure, rendering each of the layers and using conventional OpenGL blend operations to compose the final image.

4.5 Performance

Our implementation is mainly impacted by two variables: the number of decals and the number of sub-layers; the latter being more critical. In order to test the performance of our approach in the context of multivariate visualization (i.e. practical cases using sampling strategies), we generated a high number of decals uniformly distributed on the surface of the bunny model using a Monte Carlo sampling strategy [8]. We chose a fixed sphere radius so that the number of sub-layers increases progressively with the number of decals. Table 1 summarizes the performance of our current implementation. We obtain interactive frame rates even when displaying a high number of decals organized

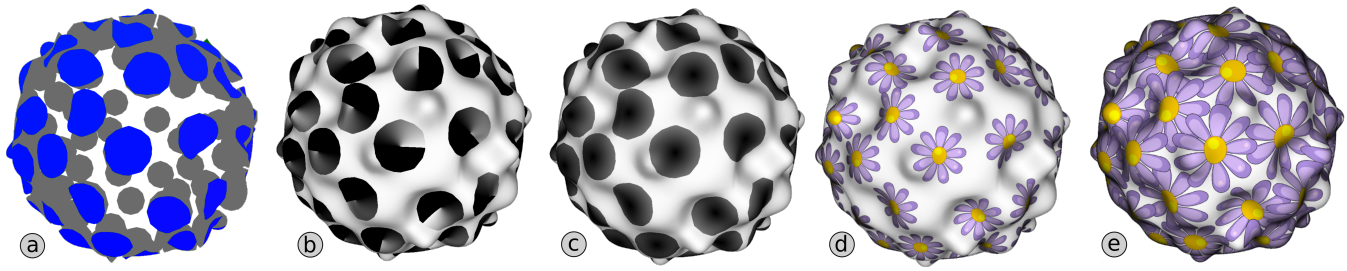

Fig. 7. Steps of our implementation: (a) Sphere masking; (b) Angular coordinate; (c) Radial coordinate; (d) Decal mapping; (e) Decal overlapping.

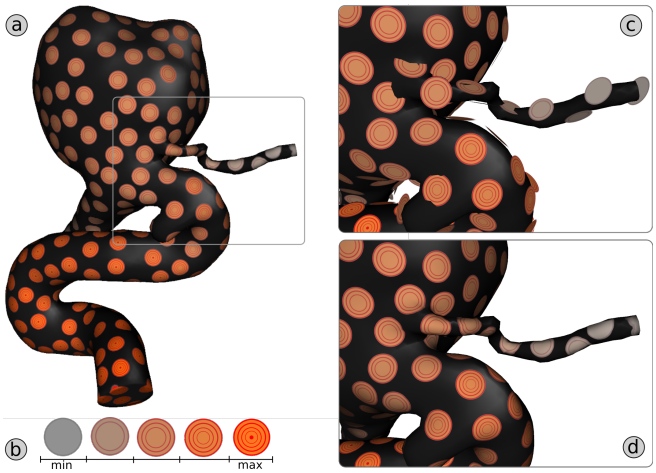

Fig. 8. (a) Aneurysm data; (b) pressure decal-map; (c) quad-based 2D-glyph visualization inspired by Pelt *et al.* [56]; (d) our approach.

in several layers. Thus, in case of a Poisson sampling (low number of sub-layers), we reckon our algorithm will perform considerably better, resulting in real-time interactions.

4.6 Qualitative Comparison

We compare our technique with the standard quad-based approach to place 2D glyphs on surfaces. Pelt *et al.* adopt this strategy to explore wall shear stress (WSS) in aneurysm datasets [56]. Inspired by their design, we place 2D glyphs on an isosurface extracted from a dataset [57]. We implement their quad-based approach and compare it with our decal-based approach. Like their approach, we distribute quads uniformly (in our case using Poisson sampling [8]) where each quad is oriented based on the surface normal at its center. We also generate each quad in the geometry shader and clip the fragments that are not part of the circular glyph in the fragment shader.

For simplicity, instead of the WSS, we visualize the pressure on the aneurysm wall. We use a decal-map composed of five circular glyphs (similar to Pelt *et al.*'s zoom level 1 [56] and Sanyal *et al.*'s circular glyphs [41]) with an ordered circular pattern and different levels of saturation to indicate the pressure magnitude (Fig. 8(b)). While the quad-based approach works well for planes and surface areas of low curvature, it can introduce severe artifacts otherwise. Fig. 8(c) illustrates artifacts such as clipping and flying glyphs which make interpretation difficult. Our technique adapts well to areas of high curvature of the aneurysm and does not suffer from artifacts caused by the depth test and glyph orientation (Fig. 8(a and d)). Moreover, it works even for the thin vessel of the aneurysm (Fig. 8 (d)), where techniques such as deferred decals are susceptible to fail. Lastly, we can apply proper lighting to our decals by accessing the normal map from the G-buffer.

In this section we have presented *an abstract framework* to place decals on arbitrary surfaces as well as a generic approach to handle overlapping decals. In summary, Fig. 7 illustrates each step of our technique applied to a *bumpy sphere*.

5 Results and Discussion

We explore the design space associated with multivariate layered illustrative visualizations using decal-maps and colormaps. We justify our design choices based on previous designs, visual perception studies, traditional illustrations and discussions with domain experts (two reservoir engineers and two geologists). We visualize multivariate data from two domains: *geographic data* and *geological data*.

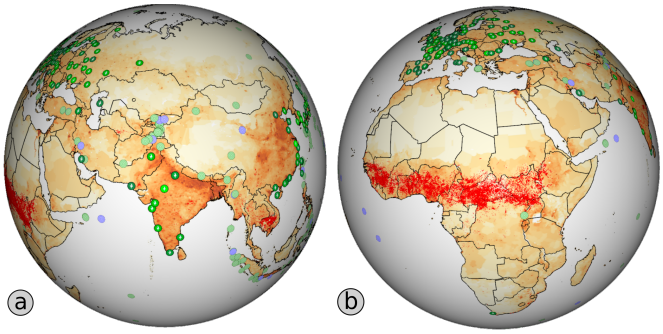

Fig. 9. (a) Areas of high population density in India (sequential orange colormap). (b) Areas of fire detection (red dots) in Africa.

geographic data	visual mapping
population density	sequential colormap (one hue)
earthquake location	earthquake decal placement
earthquake magnitude	earthquake decal-map
NP location	NP decal placement
number of nuclear reactors	color saturation
fire detection	point placement

Table 2. Mapping geographic data attributes to visual representations.

5.1 Multivariate Geographic Data Visualization

Geographic data consists of information associated with spatial positions on the Earth given in latitude/longitude coordinates. Some examples are socioeconomic data (e.g. population density), hazard data (e.g. earthquakes) and physical data measurements (e.g. temperature). These datasets are explored by geographic information systems (GIS) experts to understand patterns and data relationships. GIS apply the concept of layering to 2D maps, encoding multiple attributes using glyphs, colormaps, lines, points and other visual elements supported by scientific and information visualization research [27, 59].

An example of a multivariate geographic visualization is the *NASA SEDAC Hazards Mapper* (NSHM) [47] which illustrates the data attributes *population density* [29], *earthquakes* [29], *nuclear plants (NP)* [29] and *fire* [28] distributed on the Earth map in a layered fashion. Inspired by this visualization, we map these data attributes and their respective representations to the spherical surface of the Earth. This is a good fit since the cosine distance approximation (Sec. 3.2) is exact for the sphere.

Visualization Design We use a design similar to NSHM [47], a summary is shown in Table 2. We organize the attributes in four layers: a base layer, earthquake layer, nuclear plant layer and fire layer. In the following, we present a justification of our design choices.

Base layer: We use a sequential colormap to represent population density. This choice is suitable since population density is quantitative and ordinal data. This visualization emphasizes areas of high (e.g. India in Fig. 9(a)) and low (e.g. Sahara desert in Fig. 9(b)) population density (trends). Since this is a base layer which covers large areas, we use light tones and secondary colors such as orange [58].

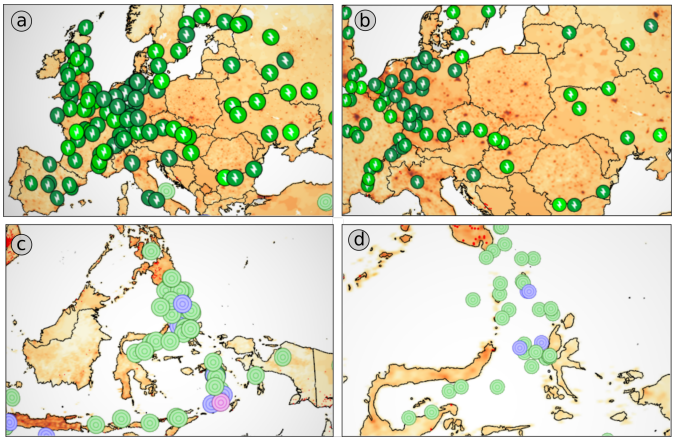

Fig. 10. (a and b) Multivariate geo visualization combining nuclear power plants (green decals), population density (sequential orange colomap) and earthquakes displayed at two levels of zoom. Number of reactors for each nuclear plant is visualized using saturation. (c) Earthquake decals of several magnitudes clustered in the Indonesia area; (d) a closer view showing the locations more precisely.

Fig. 11. (a) Earthquake decal-map; (b) Nuclear plant decal-map.

Earthquake layer: The earthquake data consists of a distribution of earthquakes and their magnitude on the Richter scale. The distribution shows locations (visual variable position) where earthquakes are recurrent. For magnitude, we use a decal composed of concentric circles and four main colors [47]. We create a decal-map using these four glyphs as illustrated in Fig. 11(a). In NSHM [47], the size of the earthquake glyphs is larger during an overview, resulting in overlapping glyphs, and smaller in a close view to provide details about the specific locations of the earthquakes (details-on-demand). Our decal mapping technique is easily able to accommodate this. We first choose a maximum radius for the spheres (decal size). Using this radius, we process the earthquake distribution to obtain the sub-layers as described in Sec. 4.4 and control the decal size based on the level of zoom. Fig. 10(c and d) illustrates two levels of zoom for the earthquake layer. Observe that this layer has several overlapping decals since earthquakes are recurrent at specific locations. This generates a high number of sub-layers (32 in our case using an earthquake data with frequency ≥ 4 within a month). However, since there are few decals, the overall performance is not impacted.

Nuclear plant layer: The NP data contains information about the number of reactors at various locations where NPs are present. Similar to earthquakes, the visual variable position is used to visualize NP locations. In NSHM [47], a nuclear power plant is represented as a green glyph containing a lightning bolt in the middle (Fig. 11(b)). We use the same glyph as a decal. Additionally, we visualize the number of reactors (ordinal/quantitative data) by changing the color saturation; the range is quantized to five bins. Therefore, we obtain an overview of areas with high and low number of reactors. Moreover, we visualize the NP decals using the same *details-on-demand* approach applied to the earthquake layer (Fig. 10).

Fire layer: We visualize fire detection data captured during a period of 7 days in Jan. 2016. From this data, we observe that fire detection is common in forest areas such as the African Savanna (Fig. 9(b)). Like NSHM [47], we visualize the fire detection distribution using red dots. The point size is slightly increased depending on the zoom level.

5.2 Multivariate Geological Data Visualization

In the oil and gas domain, geological reservoir models are represented as irregular or regular hexahedral grids embedding several geological properties that describe the reservoir. The properties can be static or dynamic [60]. Reservoir engineers explore these properties as parameters for better prediction of oil recovery. They aim to study the spatial configurations of properties to check data correlations and identify connected areas. The goal is to propose optimal reservoir development strategies and to better predict dynamic reservoir performance.

geological data	visual mapping
rock type	pastel colormap
porosity	proximity between decals
oil flow direction	red arrow decal
oil flow magnitude	decal transparency and size
water flow direction	blue arrow decal
water flow magnitude	decal transparency and size

Table 3. Mapping geological data attributes to visual representations.

However, despite the nature of this multivariate problem, most commercial visualization systems (e.g. Eclipse® [43] and Petrel ® [44]) in this domain only rely on color-based mono-visualizations of geological properties without considering aspects such as data type. This limits visualizing multiple properties within the same reservoir context. Moreover, based on our discussions with domain experts, this visualization approach forces the viewer to frequently toggle between different properties for supporting data exploration tasks.

Motivated by the need to visualize multiple attributes, we introduce for the first time in this domain, a multivariate illustrative visualization of a reservoir simulation model combining six attributes: *rock type*, *porosity*, *oil flow direction and magnitude*, and *water flow direction and magnitude*. We visualize these attributes on the surface of the hexahedral grid. Even though reservoir grids may have degenerate cells (to embed geological features such as faults [60]), our technique works well since it does not depend on the surface representation.

For our example, we use the *Zmap* simulation reservoir model (Fig. 1(e)). Since this model is relatively flat, we use the Euclidean approximation for the radial coordinate. The data consists of a black oil simulation conducted by a domain expert. The main goal of this simulation is to analyze the process of oil recovery. During the simulation, water is injected at a location of the reservoir (using an injection well) to push the oil to another location from where it is to be extracted (using a production well) [60]. Oil migration depends on other properties such as porosity, rock type and permeability, reinforcing the need to provide an integrated multivariate visualization.

Visualization Design Table 3 shows a summary of our visualization design. Many of the design ideas come from initial discussions with domain experts, literature review as well as inspiration from geological illustrations. It is important to note that this is an initial design. An in-depth domain problem characterization, task requirements research and evaluation are required. However, we consider these to be out of the scope of this paper.

Rock type layer: Within reservoir models, rock type is represented as a set of indices. Thus, the visual variable suitable for rock type is the one used to represent *categorical data* (no intrinsic ordering) [59]. Since each reservoir usually has just a few different rock types, we use a pastel colormap to represent each rock type (Fig. 12). Pastel colormaps have been applied to categorize areas in 2D maps with minimum visual interference between layered objects such as lines (e.g. lines used to represent rivers) [59].

Porosity layer: Porosity is a volumetric quantity expressed as a percentage that measures the capacity of rocks to store fluids [60]. Naturally, visual variables such as value and position can be used for indicating high and low values of porosity. In our case, we adopt position; highly clustered areas indicate low porosity and vice versa. Our inspiration for this choice comes from traditional illustrations where porosity is represented as a set of rock grains packed in a certain configuration (Fig. 13(b)). In our design, we vary the distribution of decals to convey porosity. For the porosity decal, we use the normal map of a sphere combined with the color of the rock type. It gives the impression that the porosity 'grains' are inside the reservoir model (Fig. 12). Moreover, it decreases the visual interference with other layers.

Oil flow layer: Oil flow is visualized using a red arrow decal-map (Fig. 14(a)). This creates a high contrast with the rock type and porosity background. Red is also a conventional choice for visualizing oil

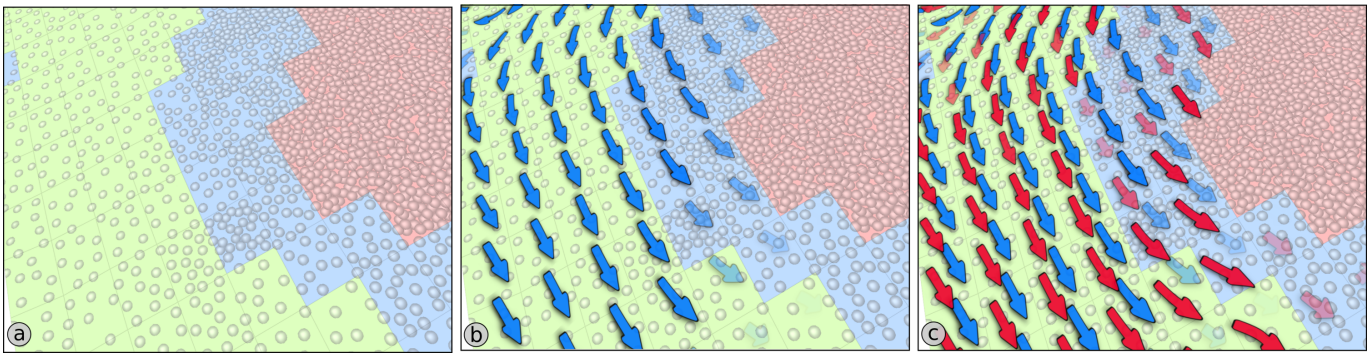

Fig. 12. Layering on the surface of a reservoir model combininig rock type, porosity, and oil and water flow.

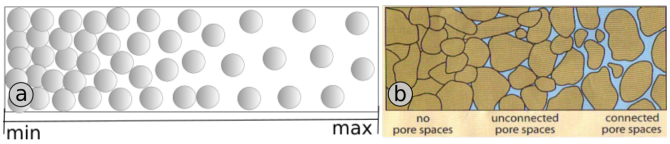

Fig. 13. (a) Porosity visual mapping; (b) geological illustration of porosity.

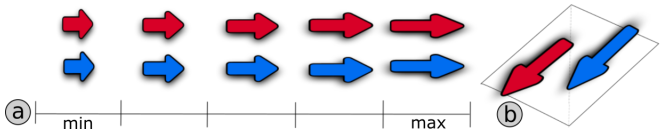

Fig. 14. (a) Water and oil decal-maps; (b) decal placement.

(domain expert feedback). In our design, we orient the arrow decal to follow the oil flow direction using the approach introduced in Sec. 3.4, whereas the magnitude is used to control the size and transparency of the decal. We use both size and transparency due to the fact that the size of the decals is altered due to perspective projection. Design guidelines recommend the use of orthogonal projection, if possible, when using size as a visual variable [59]. Since we do not want to compromise the 3D visualization, we decided to reinforce the oil flow magnitude by using transparency as a secondary visual variable. Other variables such as saturation or value can also be considered. We use transparency because it also reduces visual clutter by emphasizing areas of the model where the magnitude of the oil flow is high; this can be useful for analyzing the process of oil recovery.

Water flow layer: For water flow, we use a design similar to the oil flow layer. We use a blue arrow (domain expert feedback) decal-map to visualize water flow direction (Fig. 14(a)) as well as size and transparency to encode water flow magnitude.

Decal placement: We need to consider decal placement strategies for porosity, oil and water flow. For the porosity distribution, we map the porosity values (generally between 0-0.3 (30%)) to proximity using an importance driven Poisson sampling technique [8]. We slightly overlap the 'grains' to increase the sense of connectivity [59]. For the placement of the oil and water flow decals, we use the quad faces of the reservoir grid (Fig. 14(b)). We place a decal at the center of each triangle of the quad face. This helps decrease decal overlapping between the layers. To increase the sense of flow and connectivity, we slightly overlap the arrow decals with each other.

Observations: Fig. 12(c) illustrates our multivariate illustrative visualization combining rock type, porosity, oil, and water flow measurements. Flow characteristics depend on porosity. The green areas of the model are part of a channel with high porosity (Fig. 1(e)); this gives rise to a higher flow rate. In blue areas, the porosity is lower making oil and water flow less; light red areas are impenetrable.

When designing a layered visualization, we need to consider visual cues that help us segment each layer independently so that we can understand each property clearly. A designer has to consider variables such as color, contrast, depth cues, and the concept of integral and separable dimensions [59]. In Fig. 12(a), the shading variation caused by the normal map used to represent porosity gives rise to the *corn-sweet effect* [59]: water flow arrows appear to be over the porosity layer (Fig. 12(b)). Additionally, we applied a soft blurred shadow to the flow decals (Fig. 14). This is similar to an *unsharpen mask* or *halo* effect which helps separate objects from the background [59]. Our layering design is further aided by the use of light colors, a normal texture (shading) and primary solid colors (arrows).

Last but not least, our visualization represents the data in an illustrative fashion. It is intuitive to visualize porosity using grains, flow using arrows and rock type via color. A quick glance of the visualization can tell a lot about the phenomenon thus facilitating the interaction between professionals from different backgrounds.

6 Conclusion and Future Work

We introduced the use of *decals* and *decal-maps* as a new way of representing multivariate data on surfaces. To map decals on surfaces, we proposed a real-time technique that computes a local parametrization approximating the geodesic distance locally without relying on expensive computations. In order to obtain a better local geodesic approximation, we discussed and contrasted three distances. We provided an efficient and simple implementation of our technique, taking advantage of the graphics pipeline. We demonstrated the applicability and usefulness of our technique via case studies from two different domains. Another example of the broad applicability of our technique is shown in Fig. 1(b) where we precomputed the overlapping sub-layers and rendered the decals in real-time to generate a texture pattern covering the entire surface of the bunny model such as lapped textures [37].

We also introduced a way of deforming decals locally based on the angular coordinate of our local parametrization. This can be useful in applications such as flow visualization. An example is shown in Fig. 1(a). This figure illustrates a synthetic vector field generated over the surface of the bunny model. We deformed brush stroke decals based on the underlying vector field and distributed decals densely over the surface to achieve an effect similar to LIC.

Exploring the design space is critical in multivariate visualizations [46] and we hope that our work will facilitate this exploration. The following are some of the lines of investigation that our work paves the way for. (1) A further detailed investigation of the radial coordinate in order to find better local surface approximations of the geodesic distance. (2) A detailed investigation of deformation based on the angular coordinate. (3) The study of heuristics and methods to obtain an automatic initial radius size from the surface. (4) Investigation of approaches for creating multiscale multivariate visualizations using decal-maps on surfaces. (5) The use of interaction techniques such as lenses to explore layering. (6) Investigation of techniques to interact with decals such as click and zoom to provide additional information about the data. (7) Application of decal-mapping to other contexts such as non-photorealistic rendering and texture coverage.

Acknowledgments

We wish to thank the anonymous reviewers for their constructive comments, Hamidreza Hamdi for providing the reservoir data, and Andrew Owens and Sowmya Somanath for their valuable feedback. This research was supported in part by the NSERC/ AITF/ FCMG IRC program in Scalable Reservoir Visualization.

REFERENCES

[1] R. Borgo, J. Kehrer, D. H. Chung, E. Maguire, R. S. Laramee, H. Hauser, M. Ward, and M. Chen. Glyph-based visualization: Foundations, design guidelines, techniques and applications. *Eurographics STARs*, pages 39–63, 2013.

[2] J. Bowers, R. Wang, L.-Y. Wei, and D. Maletz. Parallel poisson disk sampling with spectrum analysis on surfaces. In *ACM TOG*, volume 29, page 166. ACM, 2010.

[3] A. Brambilla, R. Carnecky, R. Peikert, I. Viola, and H. Hauser. Illustrative flow visualization: State of the art, trends and challenges. In *EuroGraphics STARs*, pages 75–94, 2012.

[4] S. Bruckner, P. Rautek, I. Viola, M. Roberts, M. C. Sousa, and M. E. Gröller. Hybrid visibility compositing and masking for illustrative rendering. *Computer & Graphics*, 34(4):361–369, 2010.

[5] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proc. of SIGGRAPH*, pages 263–270. ACM, 1993.

[6] R. Carnecky, B. Schindler, R. Fuchs, and R. Peikert. Multi-layer illustrative dense flow visualization. In *CGF*, pages 895–904, 2012.

[7] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68(342):361–368, 1973.

[8] M. Corsini, P. Cignoni, and R. Scopigno. Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE TVCG*, 18(6):914–924, 2012.

[9] R. A. Crawfis and M. J. Allison. A scientific visualization synthesizer. In *Proc. of Vis.*, pages 262–267. IEEE, 1991.

[10] E. de Groot, B. Wyvill, L. Barthe, A. Nasri, and P. Lalonde. Implicit decals: Interactive editing of repetitive patterns on surfaces. *CGF*, 33(1):141–151, 2014.

[11] M. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.

[12] R. Fuchs and H. Hauser. Visualization of multi-variate scientific data. In *CGF*, pages 1670–1690, 2009.

[13] R. Gasteiger, M. Neugebauer, O. Beuing, and B. Preim. The flowlens: A focus-and-context visualization approach for exploration of blood flow in cerebral aneurysms. *IEEE TVCG*, 17(12):2183–2192, 2011.

[14] B. Geng, H. Zhang, H. Wang, and G. Wang. Approximate poisson disk sampling on mesh. *Science China Inf. Sciences*, 56(9):1–12, 2011.

[15] C. G. Healey. Formalizing artistic techniques and scientific visualization for painted renditions of complex information spaces. In *IJCAI*, pages 371–376, 2001.

[16] J. Kehrer and H. Hauser. Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE TVCG*, 19(3):495–513, 2013.

[17] R. Khlebnikov, B. Kainz, M. Steinberger, M. Streit, and D. Schmalstieg. Procedural texture synthesis for zoom-independent visualization of multivariate data. In *CGF*, pages 1355–1364, 2012.

[18] R. M. Kirby, D. F. Keefe, and D. H. Laidlaw. Painting and visualization. In *Visualization Handbook*, pages 873–891. Academic Press, 2004.

[19] R. M. Kirby, H. Marmanis, and D. H. Laidlaw. Visualizing multivalued data from 2d incompressible flows using concepts from painting. In *Proc. of Vis.*, pages 333–340. IEEE, 1999.

[20] J. Krassnigg. A deferred decal rendering technique. In E. Lengyel, editor, *Game Engine Gems 1*, pages 271–280. Jones and Bartlett, 2010.

[21] D. Laidlaw. Loose, artistic textures for visualization. *IEEE CG&A*, 21(2):6–9, 2001.

[22] D. H. Laidlaw, E. T. Ahrens, D. Kremers, M. J. Avalos, R. E. Jacobs, and C. Readhead. Visualizing diffusion tensor images of the mouse spinal cord. In *Proc. of Vis.*, pages 127–134. IEEE, 1998.

[23] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. In *CGF*, pages 203–221, 2004.

[24] S. Lefebvre, S. Hornus, and F. Neyret. Texture sprites: Texture elements splatted on surfaces. In *Proc. of I3D*, pages 163–170. ACM, 2005.

[25] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. In *CGF*, pages 1807–1829, 2010.

[26] B. J. Meier. Painterly rendering for animation. In *Proc. of SIGGRAPH*, pages 477–484. ACM, 1996.

[27] T. Munzner. *Visualization Analysis and Design*. CRC Press, 2014.

[28] NASA. Active fire data. `http://tinyurl.com/j3sv4pl`, 2016. [Online; accessed 25-March-2016].

[29] NASA. Ny: Nasa sedac. `http://sedac.ciesin.columbia.edu/`, 2016. [Online; accessed 25-March-2016].

[30] H. Nguyen. *GPU Gems 3: Broad-Phase Collision Detection with CUDA*. Addison-Wesley Professional, 2007.

[31] D. Palke, Z. Lin, G. Chen, H. Yeh, P. Vincent, R. Laramee, and E. Zhang. Asymmetric tensor field visualization for surfaces. *IEEE TVCG*, 17(12):1979–1988, 2011.

[32] H. K. Pedersen. Decorating implicit surfaces. In *Proc. of SIGGRAPH*, pages 291–300. ACM, 1995.

[33] Z. Peng, E. Grundy, R. S. Laramee, G. Chen, and N. Croft. Mesh-driven vector field clustering and visualization: An image-based approach. *IEEE TVCG*, 18(2):283–298, 2012.

[34] Z. Peng and R. S. Laramee. Vector glyphs for surfaces: A fast and simple glyph placement algorithm for adaptive resolution meshes. In *VMV*, pages 61–70, 2008.

[35] E. Persson. Volume decals. In *GPU Pro 2*, pages 115–120. A. K. Peters, Ltd., 2011.

[36] H. Pottmann, T. Steiner, M. Hofer, C. Haider, and A. Hanbury. *The isophotic metric and its application to feature sensitive morphology on surfaces*. Springer, 2004.

[37] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. In *Proc. of SIGGRAPH*, pages 465–470, 2000.

[38] P. Rautek, S. Bruckner, M. E. Gröller, and I. Viola. Illustrative visualization: New technology or useless tautology? *SIGGRAPH Comput. Graph.*, 42(3), 2008.

[39] P. Ringrose and M. Bentley. *Reservoir model design*. Springer, 2015.

[40] T. Ropinski, S. Oeltze, and B. Preim. Survey of glyph-based visualization techniques for spatial multivariate medical data. *Computer & Graphics*, 35(2):392–401, 2011.

[41] J. Sanyal, S. Zhang, J. Dyer, A. Mercer, P. Amburn, and R. J. Moorhead. Noodles: A tool for visualization of numerical weather model ensemble uncertainty. *IEEE TVCG*, 16(6):1421–1430, 2010.

[42] H. Schäfer, B. Keinert, M. Nießner, and M. Stamminger. Local painting and deformation of meshes on the gpu. *CGF*, 2014.

[43] Schlumberger. Eclipse industry reference reservoir simulator, 2014.

[44] Schlumberger. Petrel e&p software platform, 2014.

[45] R. Schmidt, C. Grimm, and B. Wyvill. Interactive decal compositing with discrete exponential maps. *ACM TOG*, 25(3):605–613, 2006.

[46] D. Schroeder and D. F. Keefe. Visualization-by-sketching: An artist's interface for creating multivariate time-varying data visualizations. *IEEE TVCG*, 22(1):877–885, 2016.

[47] N. N. SEDAC. Sedac hazard mapper. `http://tinyurl.com/ht8rw7b`, 2016. [Online; accessed 25-March-2016].

[48] G. Sellers, R. S. Wright, and N. Haemel. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley, 2013.

[49] L. G. Tateosian, C. G. Healey, and J. T. Enns. Engaging viewers through nonphotorealistic visualizations. In *Proc. of NPAR*, pages 93–102. ACM, 2007.

[50] R. Taylor. Visualizing multiple fields on the same surface. *IEEE CG&A*, 22(3):6–10, 2002.

[51] M. Termeer. *Comprehensive Visualization of Cardiac MRI Data*. PhD thesis, Vienna University of Technology, 2009.

[52] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross. Optimized spatial hashing for collision detection of deformable objects. In *VMV*, volume 3, pages 47–54, 2003.

[53] M. Tory and T. Möller. Human factors in visualization research. *IEEE TVCG*, 10(1):72–84, 2004.

[54] T. Urness, V. Interrante, E. Longmire, I. Marusic, S. O'Neill, and T. W. Jones. Strategies for the visualization of multiple 2d vector fields. *IEEE CG&A*, 26(4):74–82, 2006.

[55] T. Urness, V. Interrante, I. Marusic, E. Longmire, and B. Ganapathisubramani. Effectively visualizing multi-valued flow data using color and texture. In *Proc. of Vis.*, page 16. IEEE, 2003.

[56] R. van Pelt, R. Gasteiger, K. Lawonn, M. Meuschke, and B. Preim. Comparative blood flow visualization for cerebral aneurysm treatment assessment. *CGF*, 33(3):131–140, 2014.

[57] VisItUsers. Aneurysm data. `http://tinyurl.com/jcdmdlv`, 2016. [Online; accessed 10-June-2016].

[58] M. O. Ward. Multivariate data glyphs: Principles and practice. In *Handbook of Data Visualization*, pages 179–198. Springer, 2008.

[59] C. Ware. *Information visualization: perception for design*. Elsevier, 2012.

[60] P. William C. Lyons and B. Gary J Plisga. *Standard Handbook of Petroleum and Natural Gas Engineering*. Elsevier Science, 2011.