Decal-Lenses: Interactive Lenses on Surfaces for Multivariate Visualization

Allan Rocha, Julio Daniel Silva, Usman R. Alim, Sheelagh Carpendale, and Mario Costa Sousa

Abstract—We present *decal-lenses*, a new interaction technique that extends the concept of magic lenses to augment and manage multivariate visualizations on arbitrary surfaces. Our object-space lenses follow the surface geometry and allow the user to change the point of view during data exploration while maintaining a spatial reference to positions where one or more lenses were placed. Each lens delimits specific regions of the surface where one or more attributes can be selected or combined. Similar to 2D lenses, the user interacts with our lenses in real-time, switching between different attributes within the lens context. The user can also visualize the surface data representations from the point of view of each lens by using *local cameras*. To place lenses on surfaces of intricate geometry, such as the human brain, we introduce the concept of *support surfaces* for designing interaction techniques. Support surfaces provide a way to place and interact with the lenses while avoiding holes and occluded regions during data exploration. We further extend decal-lenses to arbitrary regions using *brushing* and *lassoing* operations. We discuss the applicability of our technique and present several examples where our lenses can be useful to create a customized exploration of multivariate data on surfaces.

Index Terms—Focus+context, lenses, interaction, design, multivariate, visualization, surfaces, decal

1 Introduction

In several domains, experts commonly deal with the exploration of multivariate scientific data to understand a phenomenon of interest [1], [2]. However, interpreting and correlating multiple attributes are difficult and time-consuming tasks to accomplish. Multivariate visualization has supported data exploration by providing techniques allowing the user to visually abstract and interact with these datasets and their respective representations [1], [3].

Visualizing multivariate 3D data across a surface (or iso-surface) is a challenging process. Unlike 2D multivariate data, exploring these datasets involves dealing with curvature and occlusion. Such factors hinder encoding and interacting with data using 2D visual representations. Common approaches to dealing with the visualization of more than one attribute on surfaces include the use of colormaps, glyphs and texture-based techniques [4] such as Line Integral Convolution (LIC) [5]. Inspired by concepts from Information Visualization, there have been some efforts to facilitate the visualization of multiple attributes on surfaces, for example using 2D-glyphs combined with detail-on-demand approaches [6], [7]. Rocha et al. drew upon concepts from painting [8] and extended the concept of layering to surfaces thus introducing decals and decal-maps for multivariate visualization design [9]. Their technique allows creating a layered visualization of multiple attributes on arbitrary surfaces by combining decals, decal-maps and colormaps [9], [10], [11]. These efforts signify the possible direction in which the visualization of multiple attributes on surfaces can evolve.

As in the 2D multivariate visualizations of spatial or non-spatial data, the visualization of multiple attributes layered on surfaces can inevitably lead to clutter even if proper design choices are made. Visualizing all available data is not a solution as it can mislead and confuse the user during data interpretation [3], [12], [13]. Instead, in a realistic context and depending on the task, the user starts with an overview of the data and subsequently moves to locations of interest. This method is an established concept in visualization introduced by Shneiderman as the *visual information seeking mantra*: "Overview first, zoom and filter, then details-on-demand" [14]. Interaction techniques play an important role in the realization of this 'mantra' and facilitate *on-demand* data exploration.

Lenses are frequently employed for on-demand data exploration and have also been applied to surfaces [12]. They belong to the focus+context family of techniques and were introduced by Bier et al. [15] as an interaction method or modality that modifies the visualization locally within the lens' interior. Lenses can be classified regarding their dimensionality as 2D, 2.5D or 3D [12], [16]. In the context of 3D data visualization, 2D lenses are defined in image/screen space (typically as circle-shaped lenses), 2.5D lenses as 2D objects that are placed in object space (e.g., circle-shaped lenses immersed in 3D), and 3D lenses as volumetric shapes that are placed in object space (e.g., sphere-shaped lenses). As noted by Gasteiger et al. [12], 2D lenses lack spatial correlation between the 2D screen space position of the lens and the 3D dataset. This aspect can obfuscate the process of multivariate data exploration since several locations of the data need to be analyzed and correlated. An attempt

• *The authors are with the Department of Computer Science, University of Calgary, Calgary, AB T2N 1N4, Canada.*
E-mail: {acarocha, jd.silva, ualim, sheelagh, smcosta}@ucalgary.ca.

Manuscript received 18 Nov. 2017; revised 23 May 2018; accepted 9 June 2018. Date of publication 26 June 2018; date of current version 28 June 2019. (Corresponding author: Allan Rocha.)
Recommended for acceptance by I. Hotz.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2018.2850781

to improve the spatial correlation with the 3D dataset is to create 2.5D and 3D lenses, where the movement and rotation are extended to 3D. However, these lenses require an increased interaction effort for proper placement and alignment within the 3D dataset. Moreover, useful operations such as compositing (multiple overlapping lenses) cannot be adequately implemented [16].

In this work, we propose a *new category* of lenses that adapt to the surface geometry to facilitate on-demand multivariate data exploration on surfaces. We propose *decal-lenses*, interactive lenses that follow the surface curvature. Decal-lenses extend the concept of magic lenses [15] to arbitrary surfaces and are inspired by the idea of applying decals to surfaces [9], [17]. Each lens delimits a specific region of the surface where one or more attributes can be filtered or combined using one or more lenses. Our object-space lenses allow the user to change the point of view during data exploration while maintaining spatial reference. Similar to 2D lenses, the user interacts with our lenses in real-time, freely moving them on the surface and switching between different attributes within the context of the lenses. Using *local cameras*, surface data representations can be visualized from the point of view of each lens. It allows for data accessibility even when the selected area is occluded.

By combining multiple decal-lenses, users can create *lens-regions* of arbitrary shapes. This is achieved due to the simplicity of our technique that is highly amenable to *composition*, e.g., by using *blending operations*. These regions inherit the characteristics of decal-lenses such as attribute selection and are interactively created using *brushing* and *lassoing* operations.

Complex surfaces having cavities, holes, loops or folds (e.g., brain model), complicate the use of object-space lenses (e.g., during placement, dragging and orientation). For instance, a lens may fall into surface cavities during interaction or be difficult to orient (e.g., 2.5D lenses) due to areas of high curvature. To allow a broader applicability of lenses in visualization, we introduce the concept of a *support surface* for designing interaction techniques. A support surface consists of an envelope surface that serves as a base for lens placement, navigation, and interaction. This simple concept can allow a lens to move over holes and cavities while keeping the spatial reference related to the surface area of exploration.

Our lens technique is simple, flexible and designed for real-time multivariate data visualization on surfaces. The main contributions of this paper are:

- *Decal-lenses*, object-space lenses that adapt to the geometry of a surface and allow the user to change the point of view during data exploration while maintaining spatial reference to where one or more lenses are placed.
- Five design goals based on a broad literature review that can be used as guidance for designing interaction techniques for multivariate visualization of spatial data.
- A GPU implementation that facilitates the placement, composition and user-interaction of multiple decal-lenses on arbitrary surfaces.
- The concept of *local cameras* to allow for data exploration from different points of view even when selected contextual areas are occluded.
- *Lens-regions* of arbitrary shapes that can be created *on-the-fly* by combining multiple decal-lenses using brushing and lassoing operations.
- The concept of *support surfaces* to extend the use of lenses for data exploration on surfaces with complex geometry.

It is important to mention that in the recent survey on interactive lenses for visualization [16], the authors emphasize the need to develop techniques allowing simple lens placement, interaction, and parametrization, as well as offering the flexibly of combining lenses to create new lens functionalities on-the-fly, and also facilitating reuse of the lenses [16]. In this paper, we address these issues by introducing a flexible lens technique and a set of concepts that can be applied in a variety of demonstration examples in the context of both information and scientific visualization.

2 Related Work

In this section, we review related work that proposes lenses for 3D data. For a more extensive review of lenses, we refer the reader to the survey by Tominski et al. [16]. We categorize lenses according to their dimensionality (2D, 2.5D and 3D), and also review related 'lens-like' techniques used for illustrative visualization purposes.

2.1 Lenses for 3D Data Visualization

The concept of magic lenses was introduced by Bier et al. as a new paradigm for data interaction and exploration [15]. Since then, lenses have been used extensively within the field of data visualization [16]. Most of the techniques apply the lens concept in screen space (2D lenses) to 2D multivariate datasets, for instance, in geo-visualization [18]. For 3D datasets, the use of 2D screen-space lenses has also been useful for data exploration [15], [19], [20]. However, most 2D lenses are designed for magnification [16], [21] and applied to geo-spatial data [20] or volume rendering [19]. In our work, we do not focus on designing lenses for data magnification. Instead, we are interested in lenses that allow a focused exploration of multivariate data on surfaces. Magnification is only one of the concepts discussed by Bier et al. [15]; here we are interested in other concepts such as filtering, selection, and composition (magic lens filters [15]).

In scientific visualization, 2D lenses that apply the concept of magic lens filters have also been proposed [16]. For example, motivated by the need to visualize and correlate several hemodynamic blood-flow attributes for decision making, Gasteiger et al. proposed the *FlowLens*, a lens designed to explore cerebral aneurysm data [12]. The lenses are placed and manipulated by an expert on the screen for a focused data exploration. However, Gasteiger et al. suggested that even though 2D lenses are simple to position and implement, and intuitive to use, they suffer from a lack of spatial correlation between the 2D position of the lens and the 3D dataset. In contrast, while maintaining simplicity, our technique alleviates the spatial correlation problem since the lenses are placed directly in object space.

2.5D lenses can be used to deal with the lack of spatial correlation. As in the work of Gasteiger et al. [12], these 2D lenses are placed in 3D by aligning and orienting them according to the area of interest. Fuhrmann and Gröller also proposed an

efficient implementation of 2.5D magic lenses applied to 3D flow data [22]. Their lenses are designed to augment the information density of 3D flow in a focus+context fashion. Even though these lenses address the problem of spatial correlation, they introduce new problems such as placement and orientation [12]. Moreover, the flat nature of the 2D lens makes it difficult to visualize from different points of view even with slight view changes; this can also be problematic during data exploration in collaborative environments. Last but not least, these lenses do not support extensions such as brushing and lassoing. In contrast, our orientation independent *decal-lenses* wrap to the surface and avoid such problems altogether. Our lenses are placed using a one-click approach and can easily be dragged over the surface. Moreover, our lenses allow the user to brush and lasso over the surface, compositing and creating *lens-regions of arbitrary shapes*.

Another lens-based approach to explore 3D data is to use *3D magic lenses* [23], [24], and it was first introduced by Cignoni et al. as *MagicSpheres* [23]. This metaphor consists of placing a sphere in 3D and changing the data visualization within the spherical volume. This volumetric lens was further classified by Ropinski and Hinrichs as either a *camera lens* which is fixed relative to the camera or a *scene lens* which can be positioned anywhere in 3D [25]. Another extension of magic lenses to 3D was proposed by Viega et al. [24]. They used a clipping-plane based approach to define 3D volumes such as boxes as lenses. Wang et al. also proposed volumetric lenses for magnifying and filtering volumetric data [26]. These 3D lens techniques are usually applied to 3D volume data and exploit the graphics hardware (fragment operations) using a multi-pass approach with multiple volume in/out comparisons [22], [24], [27]. When applied to surface data, 3D lenses may discard parts of the model that are independent of the point of view. Moreover, composition of 3D lenses is challenging and difficult to address. Differently from 2D lenses, the order of overlapping of multiple 3D lenses cannot be easily guaranteed [24], [28], [29]. Therefore, to the best of our knowledge, such lens techniques (like 2.5D lenses) also do not allow for more general operations such as brushing and lassoing.

2.2 Illustrative Techniques

Even though not formally defined as lenses, several 'lens-like' illustrative techniques have been introduced with a similar purpose [16]. These techniques support high-level abstraction–i.e., focusing on what to render instead of how to render [30],–with a set of interactive tools allowing the user to isolate regions of interest, highlighting important features of the data while preserving the overall context [31]. Examples include cutaways [32], [33] and ghost-views [31], [34]. These visual abstraction techniques are part of the *smart visibility* family of techniques that have mainly been applied to scientific data following the concept of importance driven visualization [31] and inspired by traditional scientific illustration techniques [35]. These techniques enable the visualization of multiple superimposed layers of information, focusing on highlighting salient visual information in the resulting image while preserving the context [31], [36]. The most relevant information is emphasized by *local modifications of visual representations* or *changes in spatial arrangement* (e.g., exploded views [37]).

Smart visibility techniques bear a resemblance to our work as decal-lenses can be used to provide cutaways and ghost-views. *ClearView*, for example, is an illustrative technique that creates ghost-views for focus+context and data-driven visualization of surface and volumetric data [38]. However, we do not focus on accentuating occluded information which makes decal-lenses fundamentally different. The purpose of decal-lenses is to *augment* the visualization of multivariate data thereby *reducing or managing visual clutter*. Moreover, smart visibility techniques are commonly view-dependent [39] whereas we are interested in view-independent local-and-controlled modifications of visual representations on arbitrary surfaces–each representation refers to a data attribute which is modulated by the lens context. These local-and-controlled modifications can allow for data comparison and correlation during exploration of multiple attributes (e.g., by superimposing multiple lenses).

Our work is inspired by previous illustrative visualization techniques. We do not, however, propose another 2D or 3D magic lens. Instead, we introduce a novel lens concept whereby lenses adapt to the surface geometry. These *deformed* lenses—their borders in particular—provide important depth cues about the surface geometry which are missing in 3D lenses. Moreover, the border is important when visualizing geometrically nested datasets (e.g., flow within a vessel) since it embeds curvature information [40]. Our lenses are designed specially for multivariate layered visualizations such as Decal-Maps [9] and the FlowLens [12], and take advantage of the GPU pipeline for a simple and efficient implementation.

3 Design Goals

In our literature review, we have highlighted the limitations of using various lens techniques to visualize multivariate data on surfaces. We also covered some illustrative techniques as well as concepts such as smart visibility that we are inspired by. Based on our review, we present the following design goals (DGs) to serve as guidelines for our proposed decal-lenses.

DG1. Consider Spatial Correlation. We argue that for multivariate visualization of surface data using lenses, spatial correlation is necessary. 2D lenses lack spatial correlation between the image-space position and the 3D dataset [12], [41]. Using a 2D lens to compare attributes leads to cognitive overload; the user is often required to re-position and adjust lenses when comparing different regions of a surface.

DG2. Facilitate Placement. Supporting positioning of lenses in a way that minimizes user effort is desirable [12], [41]. 2.5D lenses address spatial correlation but are difficult to position and orient in object-space according to the curvature of the surface.

DG3. Support Scalability. Lenses should be designed so as to combine the advantages of both image space and object space to help overcome problems of scalability and interaction. In the literature, most of the techniques are either limited to one lens or do not support the interaction between multiple lenses [29]. For example, the *join* operation [16] in 2.5D and 3D lenses is difficult to implement due to orientation problems (2.5D) or the generation of non-convex regions (3D) [24], [29]. In the case of 2D lenses, the placement in screen space avoids such problems. However, using multiple lenses on the screen may lead to confusion while

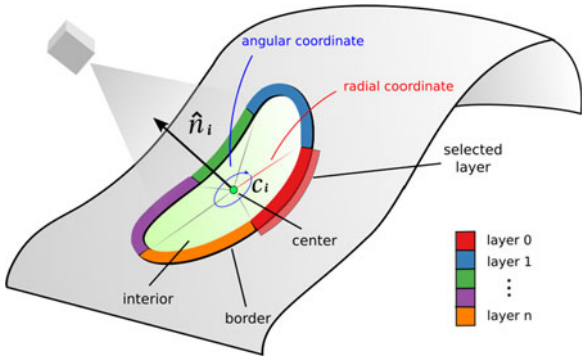

Fig. 1. Conceptual illustration of our approach. The Lens interior filters attributes represented as layers. The border displays each attribute represented as a color. To design the border, we take advantage of the local polar coordinate parametrization of the lens. A local camera can also be associated with the lens.

exploring multivariate data; this is further aggravated by the lack of spatial correlation. Moreover, 2D lenses are static on the screen during interactions such as zoom and pan, and are not re-sized proportionally to the surface.

DG4. Provide fluid interaction by (a) *using smooth animated transitions between states*; (b) *providing immediate visual feedback on interaction*; and (c) *minimizing indirection in the interface*. These guidelines were proposed by Elmqvist et al. [42] along with the idea of *fluid interaction* which draws from several sources such as the concept of flow and natural interaction. Since our lenses are meant to be interactive, we wish to incorporate these guidelines in the design process.

DG5. Consider Depth Disambiguation Cues. Spatial depth cues lead to a better spatial correlation between the lenses and the surface. Elmqvist et al. [41] created a taxonomy for 3D occlusion management techniques for visualization. Lenses are one of these techniques. In their taxonomy, the authors identified several dimensions related to the design space of occlusion management techniques. Among these dimensions, *depth cues* (DCs) are one of the most important. For 3D Magic Lenses [24], DC is classified as low. The authors also provided design patterns such as *multiple viewports* and *volumetric probes* that can be used to address the lack of depth cues. Our lens design is inspired by these ideas in order to provide a high level options of DCs.

4 Decal-Lenses Concept

Our lenses are patches of 2D manifolds built to attach smoothly to non-flat surfaces. To the best of our knowledge, such lenses have not been previously defined, and they do not fit in the previously discussed classification [16] of 2D lenses, 2.5D lenses, and 3D lenses.

Metaphorically, our lenses resemble 2D decals drawn over a surface; we, therefore, denote them as *decal-lenses*. However, decals and decal-lenses are fundamentally different; decals are textures stamped onto surfaces (e.g., to encode data attributes [9]), whereas decal-lenses are surface regions (patches) designed to allow for a wider range of uses (focus +context) and interactions. Some examples are user-defined placement (decal-lenses must be amenable to drag and drop operations at the user's will), multi-lens composition (when different lenses are superimposed, their output must be either filtered or combined), and lens interaction (the user may interact with the lenses to change their properties). We formally define decal-lenses in Section 4.1, describe the principles of lens selection and composition in Section 4.2, and explore lens properties in Section 4.3.

4.1 Definition

We refer the reader to Fig. 1 to illustrate the concepts we now describe. We denote the surface on which we will place the lenses as M. The user first picks a point c on the surface, which is used as the center of the lens about to be placed (*DG1*, *DG2*). Given the selected point, a normal vector $\hat{n}$ is computed (by averaging neighboring normal vectors); this normal vector is later used to position a local camera upon a user's request. We denote as B_c, a ball centered at c. The intersection of the surface M and B_c defines a patch $P_c := B_c \cap M$. In the context of decals, Rocha et al. denote this intersection as *sphere masking* [9] and assume that this patch is a disk since decals are small and require a local parametrization for texture mapping. In our case, the patch P_c may be understood as the region of M that will contain the lens and may not be a disk. The size of the lens is defined by the user-defined radius of the ball B_c.

To parametrize P_c, we can choose a radial coordinate system. Any point p in the patch P_c has the coordinates $p = (\rho_c(p), \theta_c(p))$. The angular coordinate $\theta_c(p)$ is simply computed by using a reference vector in the tangent space of the surface, which is an arbitrary choice. The radial coordinate $\rho_c(p)$ is given by an approximation of the geodesic distance of the surface patch. Here, we use the cosine approximation [9] for sphere-like surfaces and the euclidean distance of the 3D space for more complex geometries.

The border of the lens is designed as the ring obtained from a pre-defined range of the radial parametrization. By dividing the angular coordinate, the lens' border can be divided into as many pieces as there are properties to select (*DG4*(b)(c)). These pieces are ordered according to the reference vector in the tangent space. The active property has its respective part of the lens' border made wider by properly scaling the radial coordinate of the local parametrization (*DG4*(a)(b)). The radial coordinate is also used to blend the contents of superimposed lenses (Section 4.2).

We remark that, unlike the case of *decals* [9], the problem of parametrization distortion is less significant for *decal-lenses*, since we do not use this local parametrization to render what is inside the lens—the parametrization is only used for the lens' borders (*DG5*) and composition of multiple lenses through blending operations. Moreover, by superimposing and blending lenses, one can still cover large surface areas with high curvatures without any distortion penalty (*DG3*) by exploiting the ability to individually choose the lens' sizes (e.g., by brushing with small lenses).

Here the lens patch P_c may not be a disk but both sphere masking and the radial coordinate approximation would work in a more general case since their definitions are based on properties of the ambient (euclidean 3D) space instead of the surface. The lens border subdivision may suffer distortions, but the solution is simple: a color wheel can be drawn near the lens center (where the distortion is less since the patch P_c reduces to a disk) instead of at the lens border. One may argue that it would occlude information, however interference is minimal since the wheel only appears during attribute selection (*DG4*).

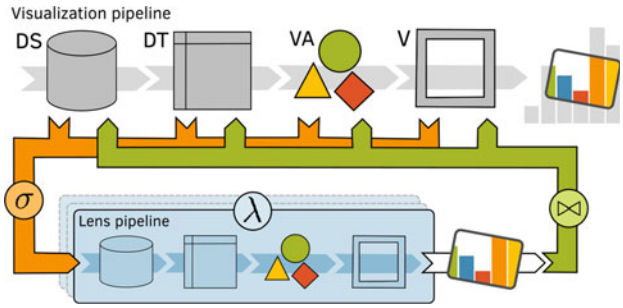

Fig. 2. Conceptual model of lenses (adapted from [16]).

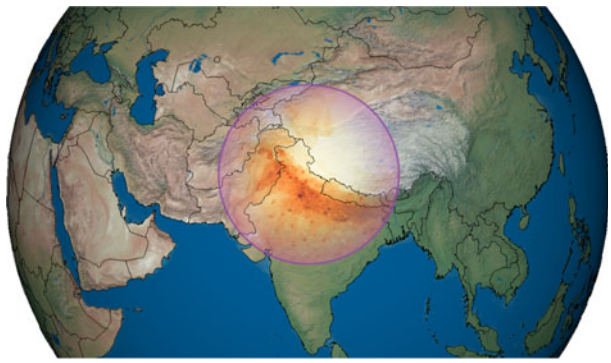

Fig. 3. Join ⋈ stage applied in V. A decal-lens displays population density on the Earth's surface using a light-to-dark orange colormap. The blending function $g(p)$ provides a smooth transition between lens content and the underlying visualization. The border color (purple) refers to the attribute selected by the user and is also combined with the underlying visualization for a better context and transition.

4.2 Conceptual Model of Decal-Lenses

Now we connect the decal-lens definition to the *conceptual model of lenses* introduced by Tominski et al. [16]. As envisioned by the authors, our goal is to use their taxonomy to provide a richer description of decal-lenses in visualization.

Fig. 2 illustrates the conceptual model of lenses introduced by Tominski et al. [16]. The top part of this model refers to the visualization pipeline [43], which describes the flow of data before visualization. First, data is obtained from a *data source* (DS) in a specific format (e.g., MRI data acquisition). Second, this data is processed and transformed into *data tables* (DT) (e.g., a grid), which are built to make searching and manipulating specific data instances easier. Third, the process of visual mapping transforms and encodes data tables into *visual abstractions* (VA) that combine marks (e.g., glyphs, decals) and visual variables (e.g., color, size). View transformations (e.g., scaling, projection) then modify the visual abstractions, which are finally rendered to a *visualization view* (V) (e.g., desktop screen, tabletop).

Lens techniques can interact with the visualization pipeline in any stage of data representation. This interaction depends on the lens technique and how the *selection* (denoted as σ) stage is conducted. For decal-lenses, we can use the enclosing sphere to apply the *selection* σ to any of the data stages (i.e., by simply verifying if a representation is inside the sphere) (*DG3*, *DG4*).

For example, a decal-lens positioned on the surface can select data samples at any vertex of the surface mesh (DT); select glyphs or decals (VA) drawn over the surface; or even select pixels during the rasterization process (V). As shown in Fig 2-bottom, the *selection* σ works as input for the *lens pipeline*, which performs a set of operations defined by a *lens function* λ. An image filter (e.g., contrast enhancement) is an example of a lens function λ applied in V.

Still following Fig. 2-bottom, after the *lens function* λ is performed, the result needs to be integrated back into the visualization pipeline (Fig. 2-right). This integration is known as the *join* $\bowtie$ stage [16]. In the last example, the outcome of an image filter (*lens function* λ) applied to selected pixels in the visualization view (V) can be combined with the base visualization through *blending operations*. The join $\bowtie$ can be performed in any stage of the visualization pipeline [16], however in the case of decal-lenses, we have chosen to perform the join $\bowtie$ in the visualization view using blending operations. Moreover, it is a common approach adopted by 2D lens techniques [16].

Blending is an essential component of combining the lens effect with the base visualization. Lens techniques commonly apply a fall-off blending function to visualize multivariate data (e.g., FlowLens [12]). This improves focus and context (avoiding sharp transitions) and also conforms to the see-through metaphor. In our case, the blending function $g(p)$ is based on the geodesic approximation, and (in the notation of our definition) is given by: $g(p) = 1 - \min(\rho_c(p)/R, 1)^{\alpha}$, where $\rho_c(p)$ is the geodesic approximation at a point p, R is the radius of the ball B_c ($\rho_c(p)/R \in [0,1]$ for the euclidean distance approximation), and α is a parameter that controls the decay function, i.e., how fast the radial function $g(p)$ decays with the distance. An example of this blending is depicted in Fig 3. In our visualization scenarios, α takes values between 3 and 6 based on our observations during the visualization design.

It is important to highlight that our lenses must be amenable to a simple implementation of multi-lens operations, i.e., when different lenses are superimposed, their output must be either filtered or combined. This is intrinsic to the definition of magic filter lenses as proposed by Bier [15]. In this work, we focus on sum (composition) and subtraction operations in the visualization view V. These two operations are essential for multivariate visualization and allow us to combine multiple lenses in a customized way (Section 6), which is essential for scalability (*DG3*).

4.3 Additional Properties

Key geometric properties of lenses include *shape*, *position*, *size*, and *orientation* [16]. These properties define the lens appearance as well as the part of the data where selection σ takes place. A decal-lens has a circular shape and is independent of orientation (*DG2*). Position and size of the lens are simply controlled by the center and radius of the ball B_c (*DG4*). Moreover, the simplicity of this approach does not limit the use of decal-lenses in more general cases; lens-regions of arbitrary shapes can be created by composing multiple lenses (*DG3*). Operations defined over a single lens (e.g., attribute selection) are then extended to multiple lenses. In Section 5.2, we describe how to perform operations over multiple decal-lenses.

However, unlike the case of a single lens, manipulating lens-regions on surfaces requires more complex operations since shape, size, position, and orientation (lens properties) are now functions of multiple lenses. For example, moving a lens-region requires the transport of multiple lenses over the surface while preserving the shape and orientation of

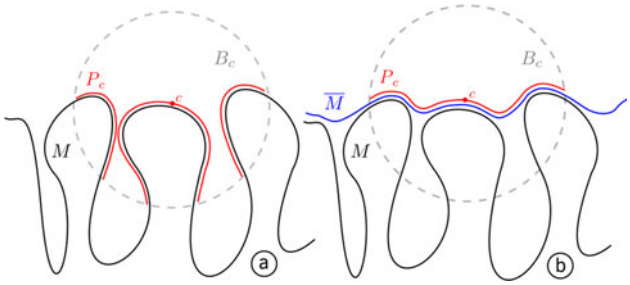

Fig. 4. 2D illustration of the support surface concept. (a) A case where the intersection P_c is not a disk due to complex surface M and the lens size. (b) Support surface is build to serve as a support for the lens placement.

the region. Investigating such issues seems to be interesting, but we consider them to be out of the scope of this paper. Moreover, we argue that these types of lens-region operations (e.g., dragging) may not always make sense: a user might create a lens-region based on a specific task and surface data and this choice would not work for general cases (e.g., other parts of the surface). Moreover, in our approach, the user can always edit or re-create a lens-region using quick operations such as brushing.

5 Extensions & Related Concepts

5.1 Local Cameras

Fig. 1 illustrates our concept of local cameras. Each lens has an attached camera which is automatically positioned by using the average normal direction around the center. Apart from its simplicity, we believe that this concept, when combined with our lenses, can be used for *data comparison and correlation* from different points of view (*DG5*)—even when some of the areas of the surface are not visible—by showing a close-up of the layered attributes on the surface. The application of local cameras is possible due to the advantage of having lenses in object space (*DG1*).

5.2 Operations Over Multiple Decal-Lenses

Decal-lenses can easily be abstracted to create arbitrary shaped lenses (*DG3*). We describe two such abstractions here: *brushing*, to create lenses by painting regions one wishes to have as lenses, and *lassoing*, to create lenses by encircling regions one wishes to have as lenses. Both brushing and lassoing build on the fact that decal-lenses are *amenable to composition*; the output lens is defined by merging the decal-lenses that are created by each procedure.

When *brushing*, a user paints a region on the surface that is meant to become a lens. During painting, a set of points are sampled to serve as centers for decal-lenses that are combined to constitute the *brushed* lens-region. When *lassoing*, a user draws a closed curve around a region that is meant to become a lens. Using a sampling strategy, the closed curve is filled with samples that are used as centers for the decal-lenses that constitute the new *lassoed* lens region. In Section 6.5 we provide an implementation to create brushed and lassoed lens-regions.

5.3 Support Surfaces

Consider now that the surface M consists of occluded regions and regions of high curvature (e.g., brain folds). These regions make the placement and navigation of lenses difficult. In the case of decal-lenses, which follow the surface geometry, such regions may cause undesired discontinuities on the lens patch. Fig. 4a illustrates this scenario in 2D. However, we also argue that such discontinuities provide essential depth cues (e.g., the lens border) (*DG5*) and can be of user interest as we discuss later in Section 6.6.

Let us now define a surface $\overline{M}$ that approximates M such that M is inside the region delimited by $\overline{M}$. Let us assume that $\overline{M}$ is smooth and devoid of the aforementioned regions thus making it suitable for lens placement, navigation and interaction (Fig. 4b). This surface $\overline{M}$ is defined as the *support surface* of M. The design of support surfaces for lens interaction depends on the surface of interest as well as the user's intention. In a simple case, a support surface $\overline{M}$ could be defined as the bounding box or bounding sphere of M. However, a user may wish to use a surface $\overline{M}$ that closely matches M. For example, if the surface M comes from the human brain, its bounding sphere (or ellipsoid) may provide a satisfactory approximation; whereas for an aneurysm vessel, this choice may be inappropriate.

Even though the idea of support surfaces has not previously been explored in the context of designing interaction techniques such as lenses, our work is inspired by other approaches that are based on similar concepts. Some examples are *outer envelopes* [44], *bounding proxies* [45] and *text scaffolds* [46]. Other techniques could be used to generate support surfaces, and a thorough literature review of such methods is out of the scope of this paper. Here, we highlight that our goal is to introduce this concept in the context of interaction techniques by providing an example of its use. In Section 6.6, we provide a simple approach to compute a support surface from a brain dataset and illustrate its use in combination with our lens technique.

6 Implementation

In this section, we describe the GPU-based implementation of our technique. Our work is inspired by the Decal-Maps approach and implementation [9]. In this paper however, we do not propose a technique that needs *attribute layers* with a high number of decals. Here, our implementation places *one lens* at a time. This allows us to modify the Decal-Maps pipeline [9] to create each decal-lens in one single pass.

We explain our implementation via walkthrough scenarios using different datasets. First, in Section 6.3 we demonstrate the use of decal-lenses for *geographic data visualization*, a common scenario in Information and Scientific Visualization. Second, in Section 6.5 we describe the implementation of *brushing* and *lassoing* operations by providing examples of their use on *an aneurysm surface* containing vessel regions of high curvature. Last but not least, in Section 6.6 we use *a brain surface*, consisting of challenging surface regions generated by the brain folds, to illustrate the concept of *support surfaces*. We believe these *examples* illustrate the potential use of our technique for focus+context multivariate visualization.

We implement our approach using OpenGL and GLSL. Here, we present an overview of our *generic pipeline* to implement decal-lenses on arbitrary surfaces. We refer the reader to Fig. 5 to illustrate the discussion that follows.

6.1 Visual Encoding and Multi-Pass Approach

First, all attributes (data and geometry) are read from a surface model. Depending on the type of representation,

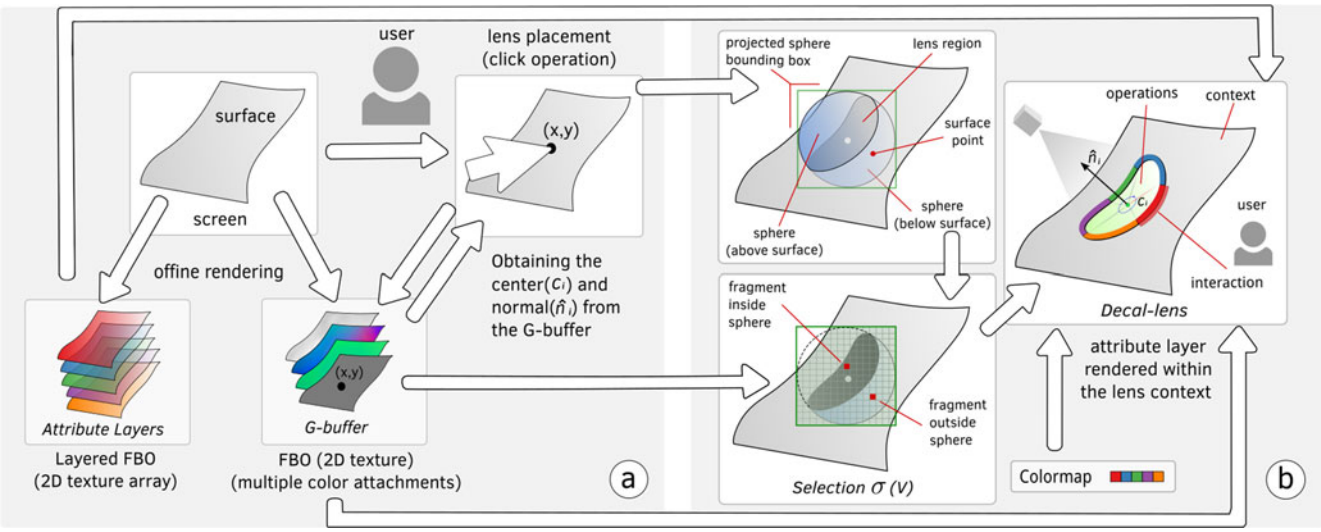

Fig. 5. Implementation workflow: (a) The geometry buffer (G-Buffer) and Attribute Layers (A-Layers) are computed; (b) A sphere is placed centered at the surface using one-click interaction. The G-Buffer and lens center are used to verify if a point of the surface is inside the lens sphere. This test is performed only for the fragments limited by the bounding box of the projected sphere. The points inside the sphere define the decal-lens region where shape and border are designed. A-Layers are mapped to the lens interior based on a chosen filtering operation. A-Layers can also be mapped to the lens exterior (context). A categorical colormap refers to the different data attribute representations. By picking one color, the user switches back-and-forth between different attributes representations, which are mapped to the lens interior.

attributes are represented as colormaps, decals or decal-maps. All these steps happen in a pre-processing stage before the actual rendering process. We extensively employ the process of multi-pass rendering both to render multiple layered attributes on surfaces and to create our decal-lenses. Thanks to the increasing computational capabilities of GPUs, multiple continuous passes of rendering are a common feature in real-time applications, e.g., games.

6.2 Rendering and Layered Representation

After the visual encoding step, we start the process of rendering (Fig. 5a). To place lenses on surfaces, we first build two view-dependent buffers in an off-screen step: an attribute layer buffer (AL-buffer) and the geometry buffer (G-buffer). The AL-buffer consists of a layered framebuffer object (LFBO), which in our case is a buffer composed of an array of framebuffer objects (FBOs) represented as 2D textures [47]. Each FBO has a color attachment and a depth attachment associated with it. The number of 2D textures is equal to the number of attribute layers. Applying a multi-pass process (in the draw call), we render each attribute and its respective visual representation to the AL-buffer. This is accomplished in a single pass by using the built-in variable `gl_Layer` in the geometry shader [47]. Each attribute layer is assigned a unique `id` which directs the render call to the corresponding 2D texture. Consequently, we obtain the AL-buffer, which contains all rendered attributes.

After this off-screen rendering step, all the attributes are available in image space as 2D textures. Each one of these images can be organized and rendered in any order; we simply iterate over the layers in a prescribed order and render each one separately. Since these layers are independent of each other, they can be turned on or off at the user's will.

6.3 Decal-Lenses

To explain our decal-lens implementation, we use a multivariate Earth dataset as an example. Such data is traditionally visualized and interpreted as layers of 2D maps using geographic information systems (GIS) (e.g., atmospheric data, satellite images, and socioeconomic data). Here, we visualize such layers on a *Digital Earth model* to avoid distortions created by projecting the Earth surface into the plane, which can lead to erroneous interpretation of length and area measurements [48]. However, the spherical nature of the Earth has inherent challenges related to modeling and visualization. In the survey of digital Earth [48], the authors emphasize the need for creative visualization of geospatial data sets. Here, we demonstrate the use of decal-lenses for multivariate data exploration on the Earth surface.

For our visualization, we consider the following attributes: (raster data) *vegetation density*, *temperature* and *population density*; and (point data) *nuclear plants* and *earthquakes*. Vegetation density is represented from a satellite image; whereas temperature and population density are represented using colormaps; cool-to-warm diverging and light-to-dark orange respectively. Using decal-maps, we represent nuclear plants and earthquakes similarly to Rocha et al. [9].

6.3.1 Lens Placement and Construction

We explain this process via the following walkthrough scenario (Fig. 5 (a-right and b)). A user wishes to inspect a surface with three data attributes. Two are represented as colormaps and one is represented as a decal-map. First, the attribute representations are rendered to the AL-buffer in an off-screen step. The user then enables two of these layers (a colormap and decal-map) over the surface. Since the other attribute cannot be layered using a colormap (as it would occlude the information), the user decides to use lenses to augment the visualization. The user then inspects the surface data seeking areas of interest. Once the region is identified, the user places a lens over the surface by performing a simple click operation (Fig. 5a (top-right)). The

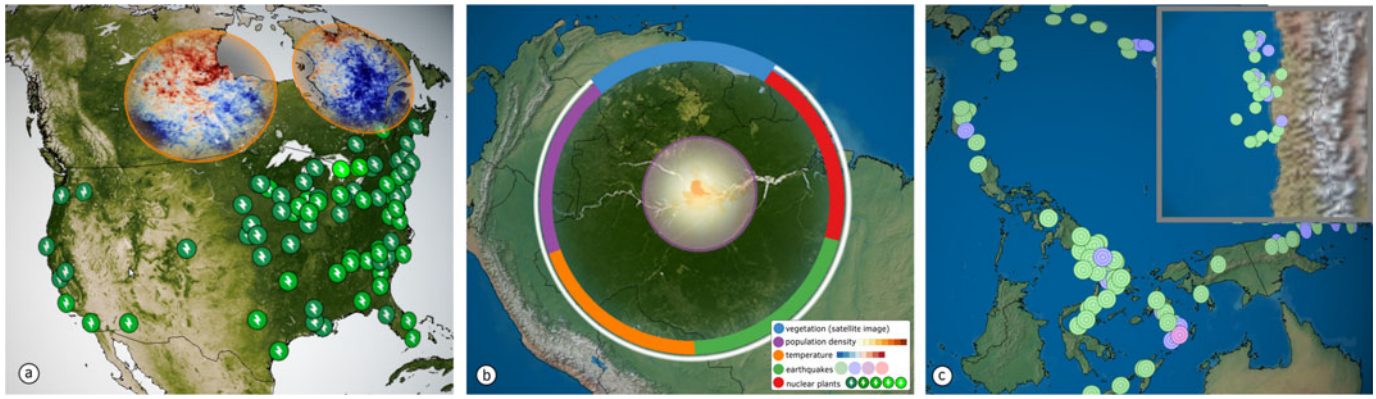

Fig. 6. Multivariate geo-visualization. (a) Two layers displaying vegetation density and nuclear plants in the USA and two decal-lenses display temperature variation in Canada. (b) A decal-lens is used as *context* to display vegetation density and another as *focus* illustrating population density in the capital state area (Manaus) along the Amazon river; the color wheel represents five attributes. (c) Comparison between earthquakes in the Oceania surroundings and earthquakes in Chile (where a decal-lens was placed) using a local camera.

clicked pixel (x, y) is used to obtain—in object-space—a position c (the lens center) and its associated normal vector $\hat{n}$ (used to place a local camera) from the G-buffer. Using the lens' center c and its radius R, we determine the projected lens' bounding box in screen space (Fig. 5b (top-left)). We then perform a *scissor test* [47] to only perform fragment tests and texture lookups within the projected lens' bounding box in screen space. For each pixel within the bounding box, we access the corresponding surface position p in object-space from the G-Buffer, and determine whether p lies inside the sphere at c. Fig. 5b (bottom-left) illustrates this process. The set of all valid p inside the sphere corresponds to the lens region in screen-space, where the attribute is then mapped.

Fig. 6a illustrates the aforementioned scenario. Two layers, vegetation density and nuclear plants, are displayed in the USA. Decal-lenses highlight the temperature variation in different areas. The color border of the lenses (orange) corresponds to the attribute category.

6.3.2 Interaction, Widgets, Visual Feedback and Blending

The attribute layers stored in the AL-buffer are fed as input to each lens (Fig. 5b-right). To switch between attributes, the user first selects a lens using a right-click operation. A position on the G-buffer is obtained from the projected pixel (corresponding to the user selection) on the surface. We then determine the center of the lens that is closest to the selected point. Since we have few lenses, this is efficiently accomplished via a quadratic-time algorithm, yet even for many lenses this approach would still work due to the quick selection test. After the lens is selected, the user can perform a variety of interaction operations such as enabling the selection wheel, hovering over different attributes along the wheel, resizing the lens, changing its order relative to the other lenses, dragging over the surface. As dictated by the user interaction, the lens can combine or filter particular attributes within the lens interior.

To draw the lens border and widgets, and to provide visual feedback during interaction, we use the local parametrization described in Section 1. We access the lens center and for each surface position in the lens interior (obtained from the G-Buffer), we compute a geodesic distance approximation. By delimiting a range of the radial coordinate, we design the lens border and wheel.

For the wheel, we also use the angular coordinate to divide the total lens arc into subsets depending on the number of attributes. Using a categorical colormap, we assign a color to each one of these sub-arcs, where each color corresponds to an attribute. The user can then hover over each one of these sub-arcs using the mouse. Based on the chosen color, the selected attribute is used to filter and display the corresponding layer within the lens interior. We provide a visual cue for the current attribute by increasing the thickness of the selected sub-arc. The border of the lens is also changed to the color of the selected attribute. Finally, the selected attribute is rendered over the surface where conventional blending operations are performed.

Fig. 6b illustrates the design of the lens border and wheel using the angular and radial coordinates, which can be exactly obtained in the case of the sphere [9]. A decal-lens is used as *context* to display the vegetation in the Amazon area and another one as *focus* to display population density. We can note the high population density in the city of Manaus (capital city of Amazonas) along the Amazon river.

6.4 Local Cameras

To implement local cameras, we need to average the normals in a neighborhood of the lens' center. We assume that during lens placement the user aligns the surface to be roughly parallel to the view plane. In our approach, accessing neighbors in screen space is similar to accessing neighbors in object space (i.e., on the surface). We, therefore, sample screen space positions and consequently access the neighboring normals around the lens center in object space (from the G-buffer). Finally, we average nearby normal directions and place the camera (drawn using a view-aligned quad) along this direction at a distance defined by the user.

Local cameras provide a strategy for overview-detail visualization as well as for data comparison. Fig. 6c illustrates the use of local cameras to compare earthquakes' location and magnitude in Oceania and Chile.

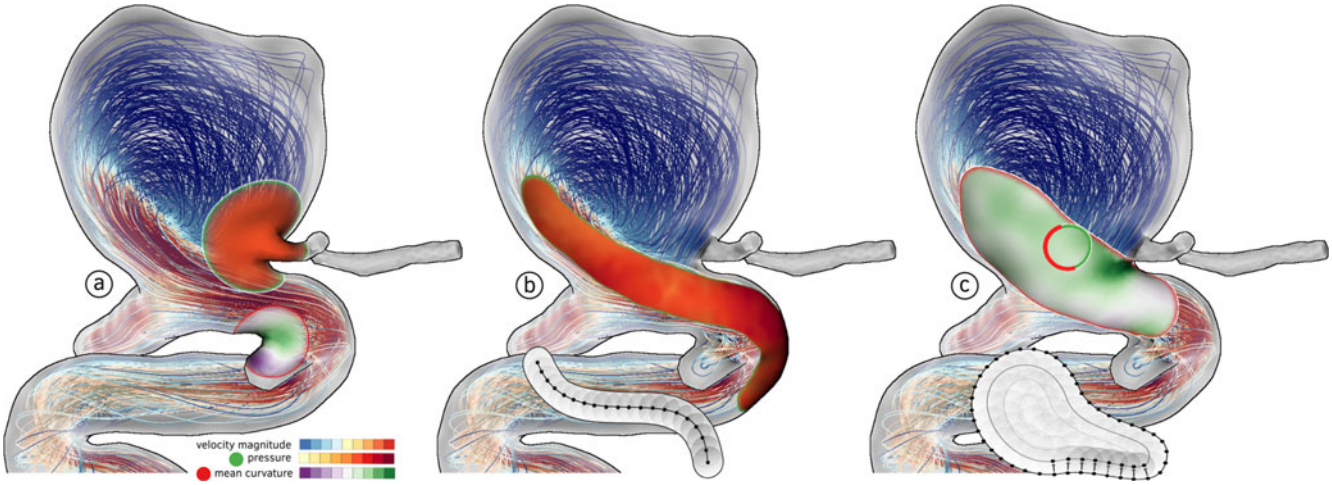

Fig. 7. Multivariate visualization of hemodynamic attributes. (a) Vessel and flow visualization inspired by Gasteiger et al. [49]. Velocity is represented using a cool-to-warm diverging colormap. Two decal-lenses display pressure, yellow-to-red colormap, and mean-curvature using a green-to-purple diverging colormap. (b) Brushing operation combining multiple decal-lenses displaying pressure. (c) Lassoing operation combining multiple decal-lenses displaying mean-curvature. A decal-lens is selected within the lassoed lens-region to be used for attribute selection.

6.5 Operations Over Multiple lenses

We now describe the implementation of *brushing* and *lassoing* operations, which require multiple decal-lenses. To demonstrate our technique, we use a publicly available aneurysm dataset [50]. Inspired by previous work [12], [49], [51], we visualize the following attributes: velocity, pressure and mean curvature of the aneurysm vessel. For velocity inside the vessel, we trace streamlines and visualize them using line rendering. We represent the velocity magnitude using a cool-to-warm diverging colormap. To preserve the flow context, we render the vessel of the aneurysm with Fresnel reflection shading similarly to Gasteiger et al. [49]. In this approach, the opacity o assigned to the vessel wall is given by $o = 1 - |v \cdot n|^r$, where n is the surface normal, v is the view vector, and $r \geq 0$ is the edge fall-off parameter [49], [52]. We obtained good results using values of r between 1 and 1.5. Following Gasteiger et al., we also visualize the contours of the surface to enhance surface shape and provide depth cues. Our contours are computed in the fragment shader using an image space *Sobel-operator* over the depth buffer. Fig. 7a illustrates our aneurysm visualization following this approach.

6.5.1 Arbitrary Lens-Regions

Let us suppose a domain expert user intends to analyze pressure and mean curvature on the aneurysm vessel for assessing the risk of rupture [12], [51]. The user can place and interact with decal-lenses over the surface for this purpose. Fig. 7a illustrates two decal-lenses over the aneurysm surface displaying pressure and mean curvature using purple-to-green and cool-to-warm colormaps respectively. We can notice how decal-lenses adapt to the surface geometry even in areas of high curvature. The user later decides to extend the lens effect to create *lens-regions* on customized surface areas of interest. To design such regions, we employ brushing and lassoing operations which are implemented as follows.

(1) *Brushing*. To define a brushed lens-region, the user enables the brushing mode (e.g., by keyboard interaction) and moves the mouse over the region of interest, which in this case covers part of the sac, the neck, and the parent vasculature of the aneurysm [49]. During the mouse motion, sampled surface points are obtained from the G-buffer. For each surface point, we place a decal-lens and compute the fall-off function introduced in Section 4.2. We place decal-lenses incrementally to provide a quick visual feedback to the user while brushing (following a painting metaphor, Fig. 7b-bottom). To avoid oversampling during interaction, we only sample consecutive points that satisfy a minimal distance from each other in object space. In our case, this minimal distance is a function of the lens radius, which is user defined. Moreover, this variable controls how continuous the border of the brush is, whereas the fall-off function controls how smooth it is. Fig. 7b illustrates the brushed region displaying pressure over the aneurysm surface.

(2) *Lassoing*. To define lassoed lenses, the user enables the lassoing mode and draws a curve in a counterclockwise orientation over the region of interest. A closed curve in screen space is then generated by linking the last and the first point. Now, we need to consider a sampling strategy within the lassoed region. For this purpose, we implement a propagation algorithm. First we interpolate the curve points using a cubic natural spline and compute normals from the interpolant, which is given by the counterclockwise orientation and the reference vector (0, 0, -1). We then duplicate the vertices and move the copies inwards by normal displacement (Fig. 7b-bottom). This displacement is controlled by a radius defined in screen space. For each iteration, we check for collisions between samples within this control radius. In the case of a collision, we remove the sample. The propagation stops when the current curve is empty. We observe that the number of internal curves is a function of the ratio between the bounding box diagonal and the control radius, which yields a stable stop condition. Having a sampling distribution defined in screen space, we use the G-buffer to push it back to object space. Fig. 7c illustrates a lassoed region displaying mean curvature over the aneurysm surface.

Discussion. Both the brushing and lassoing operations depend on sampling strategies and whether the sampling is performed in screen space or object space. In the case of brushing, our implementation does not re-sample the curve

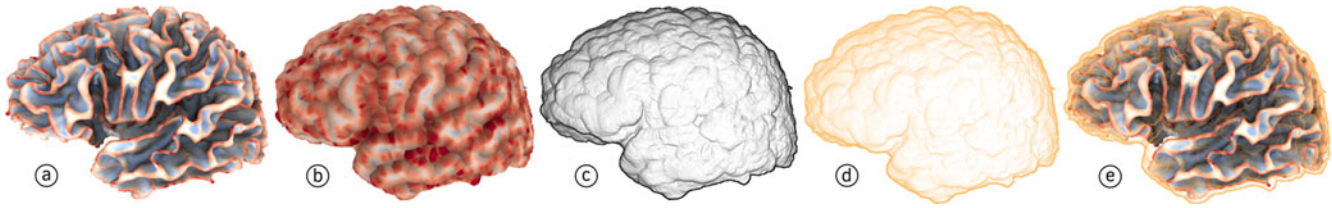

Fig. 8. Support surface implementation overview. (a) isosurface of a brain's left hemisphere; (b) surface inflation; (c) smooth outer surface; (d) outer surface with Fresnel shading; (e) brain support surface.

points during mouse motion (in object space) and does not prevent the user from brushing over the same region several times. These problems are inherent to sketch-based techniques and their remedy is out of the scope of this paper. In the case of lassoing, for simplicity we adopted a strategy that works well but has some limitations that are inherent to screen space approaches. For example, our sampling depends on the distance from the surface to the screen. We adopt a smaller screen radius when the surface is far from the screen, where we have less precision and resolution, and a large radius otherwise. Moreover, depending on the distance from the surface during lassoing, the generated lassoed region on the surface can be bigger or smaller then the lassoed area in screen space. We intend to study better approaches to generate such lassoed regions in the future.

6.5.2 *Interaction, Widgets, Visual Feedback*

Now we describe how to draw the border, widgets or other visual representations of lens-regions. Notice that, since these regions are defined over multiple lenses and we don't have a global parametrization of the whole region, it is not intuitive how to render such representations. Nevertheless, this can be accomplished in a simple way, as we now explain.

Multiple overlapped decal-lenses generate an opacity field that decreases from the medial axis of the region to the border, according to the exponent α that controls the fall-off function. In other words, overlapped lenses lead to opacity 1 whereas the transparency increases towards the border–where overlapping does not exist. Considering this opacity field, we first render the region to a FBO and store the opacity variation in pixel space. In a second pass, we draw a screen quad and access the blended values in the FBO as well as the AL-buffer. Now we have all the information required to draw the region border (i.e., by selecting opacity ranges close to 0 similar to isolines) and to map attributes similarly to the decal-lens approach (Figs. 7b and 7c).

To draw the wheel, we do not use the border of the lens-region, since the regions can vary in shape. Moreover, spliting the wheel in regions of arbitrary shape is not a straightforward task. To address this issue, we adopt a simple approach which is to activate the wheel of the decal-lens situated within the lens-region that is closest to the point given by the user click during the lens-region selection. This is achieved by a simple lens search within the region. This avoids indirection and confusion during interaction (e.g., search for the color around the border) and allows for a quick visual feedback. However, the selected decal-lens may be occluded by other lenses due to the brushing or lassoing process. Therefore, to allow the visibility of the current decal-lens and wheel, we move the selected lens to the top of the other ones by simply moving it to the end of the decal-lens list that composes the region. Finally, we draw the wheel to allow attribute selection in the region (Fig. 7c).

6.6 Support Surfaces

We use a brain data [53], [54] to demonstrate the concept of *support surfaces* that we here implement. This data is an isosurface of a brain's left hemisphere containing (Fig. 8a), containing several occluded and high curvature regions, which are generated by the brain folds. On the surface, we visualize the geometric attributes of mean and Gaussian curvature (commonly analyzed in brain studies [55]) using diverging cool-to-warm colormaps. The curvatures were estimated using Rusinkiewicz's method [56]. Our goal here is to provide a concrete example of how the concept of support surfaces can be used for designing interactions.

For our example, we consider the following scenario. A user chooses to visualize the mean curvature for the overall brain surface, and the Gaussian curvature using a large decal-lens with the goal of analyzing both attributes in an integrated way. The user wants to obtain an overview of the data by moving the lens over the surface. To allow a smooth interaction (following Section 5.3), we need to compute a brain support surface. We divide our approach into two main steps: (1) identify occluded regions of the brain surface; (2) close these regions (mathematical morphology). To accomplish this goal, we combine ideas from both Cohen et al.'s (envelope surfaces) and Cipriano et al. [46] (text scaffold) techniques as follows.

To address (1), we compute an object-space ambient occlusion factor (o) for each vertex (v) of the surface using the method proposed by Sarlete and Klein [57]. In our implementation we use 512 light sources and a *transform feedback* implementation [47] for performance. To address (2), we offset each vertex of the mesh in its normal direction ($\hat{n}$) similar to Cohen et al.'s method [44]. Inspired by Cipriano et al.'s method [46], we calculate the offset for each vertex based on the occlusion factor obtained from (1). Finally, the displaced vertices v are given by $v = v + k\hat{n}(1 - o^{\beta})$, where β is an exponent that controls the influence of the occlusion in the displacement and k is a normalization scale factor that depends on the order of magnitude of the vertices (in our case $\beta = 2$ and $k = 0.05$ since our surface is normalized between 0 and 1). Following this equation, vertices in non-occluded regions yield occlusion values close to 1 and occlusion values close to 0 otherwise. This idea forces highly occluded regions to close due to the inflation in the normal direction. Fig. 8b illustrates the inflated brain surface after our approach. For simplicity our implementation does not treat surface collisions.

Now, we consider the subsequent steps: (3) surface smoothing to avoid small discontinuities on the generated surface; (4) a visual feedback of the support surface (that the

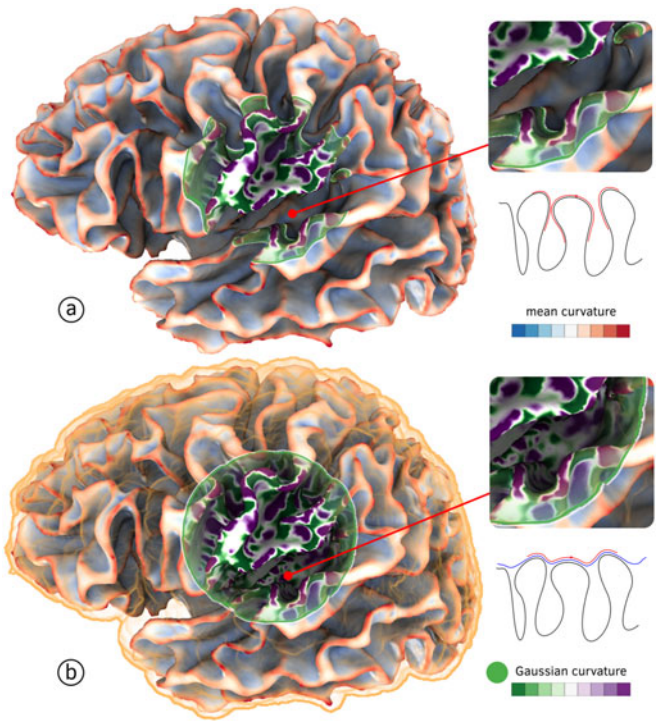

Fig. 9. (a) A decal-lens displaying Gaussian curvature over the brain surface; the lens patch provides depth cues of the brain surface such as depth and curvature. (b) *See-through window* metaphor through a decal-lens placed on the support surface; the orange Fresnel shading provides cues about the support domain with minimal visual interference.

user can turn on and off) to guide the user during navigation and interaction. To accomplish (3) we first render the support surface to a FBO, creating its respective G-buffer. We then deploy an image smoothing Gaussian filter (5x5) to the G-buffer (depth buffer) in a second pass in order to smooth the surface geometry in screen-space similarly to [58]. We then compute positions and normals from the smoothed buffer by accessing neighbor fragments (Fig. 8c). To address (4), transparency is a good candidate to provide the user a visual cue of the support surface with minimal interference (which depend on the design). Since we are only interested in the outermost part of the support surface and we want to incorporate surface shading, we address (4) by calculating Fresnel reflection shading, similarly to the aneurysm vessel (in this case $r = 0.5$), directly from the G-Buffer (where only the visible geometry is stored) right after applying the Gaussian filter (same pass). We color the support surface with a light tone of orange to minimize the contrast interference between the brain surface and the diverging colormaps (Fig. 8d). We compose the final visualization by using conventional OpenGL blending operations (Fig. 8e).

Figs. 9a and 9b illustrate a comparison between a decal-lens placed directly on the brain surface and the support surface. The good approximation of the support surface allows to keep the spatial reference of interest during data exploration, whereas the direct placement insert discontinuities in the lens patch, Fig. 9a. The Fresnel shading provides a depth cue of the support surface location as displayed in Fig. 9b. The user can obtain an overview of the Gaussian curvature of the surface through the lens interior following a *see-through window* metaphor.

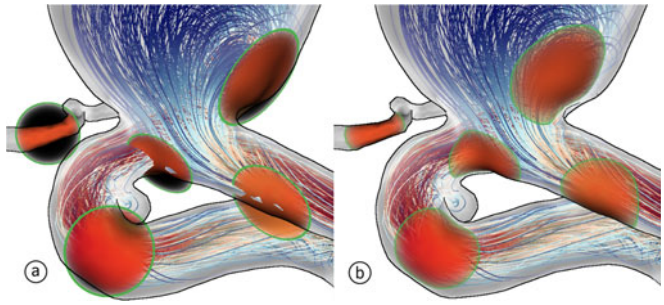

Fig. 10. Qualitative evaluation of lenses displaying pressure on an aneurysm surface: (a) 2.5D lenses; (b) our approach.

However, it is important to notice that in Fig. 9b the depth cue of curvature and how deep is the surface is decreased, once all the information projected in the lens patch is visualized. On the other hand, Fig. 9a the lens border highlights well the shape of the surface, and the lack of continuity in the lens patch also provides important depth cues related to the surface cavities. The use of each one of these approaches will depend on the application and the goal of the visualization. Investigating such aspects seem a promising future research direction.

7 Evaluation

In this section, we provide two evaluations of our technique. First, we consider a qualitative comparison between our decal-lenses and 2.5D lenses, such as FlowLens [12]. Second, we provide a performance evaluation to investigate how our technique scales with a large number of lenses.

7.1 Qualitative Comparison

We implement a 2.5D lens using a surface aligned quad. Similar to our local cameras approach, each 2.5D lens is aligned based on the average normal of the point clicked by the user. Fig. 10a illustrates 2.5D lenses displaying pressure on the vessel surface of an aneurysm. Depending on the surface curvature, the user may need to adjust each 2.5D lens' orientation and position locally for data exploration (not satisfying *DG2*). Moreover, dragging over the surface can lead to flickering and lack of smooth transitions during interaction due to re-orientation of the lens (not satisfying *DG4*). Depending on size, the lens may be clipped by the surface if depth test is enabled. This can be solved by ignoring the depth test (e.g., by rendering first to a FBO). However, it can lead to perceptual issues such as the lens flying over the surface or the lens showing when it should be occluded (not satisfying *DG5*). Last but not least, operating over multiple 2.5D lenses is challenging and remains an open issue (not satisfying *DG3*). One may argue that 2.5D lenses do not need to be placed on the vessel surface (thus avoiding clipping problems). However, embedding a 2.5D lens in 3D would further exacerbate the problem of orientation and positioning, as well as violate *DG1*.

In contrast, Fig. 10b illustrates decal-lenses placed at the same locations. Decal-lenses wrap around the surface during interaction (*DG1*, *DG2*, *DG4*) thereby isolating the region of interest, providing local depth information and allowing for proper lens shading (*DG5*), which is later blended with the color border (attribute) for context.

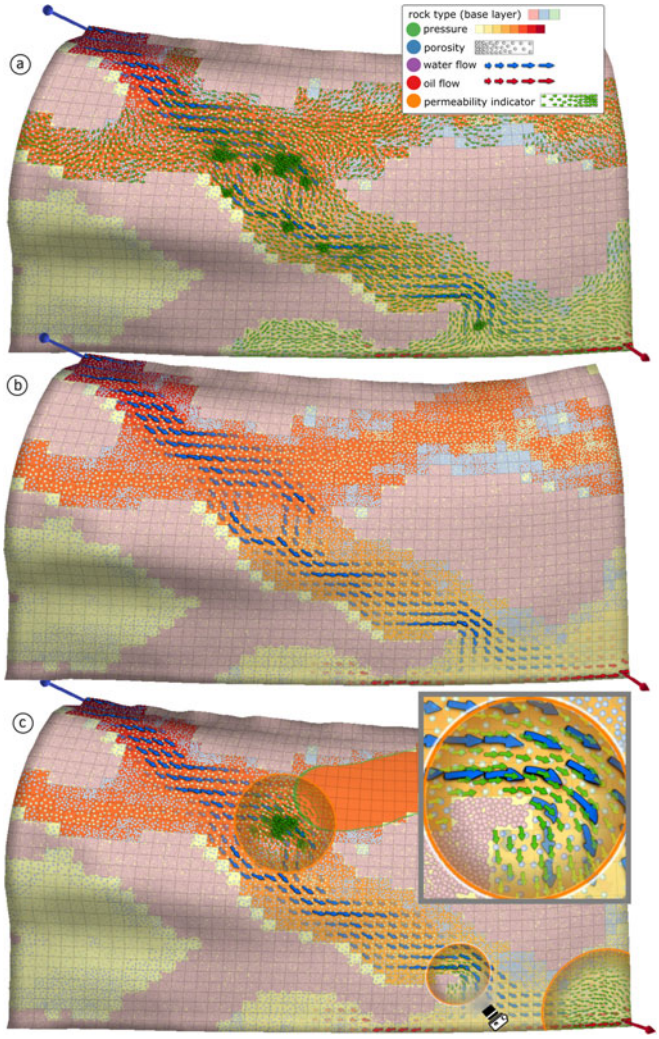

Fig. 11. Black oil simulation for oil recovery. (a) Pressure is represented by a heat-map, combined with porosity (clustered grain decals), rock type (grains painted with pastel colors), water (blue) and oil (red) flow rates (arrows), and a *permeability indicator* (green arrows). (b) Without the permeability indicator, it becomes clear that the flow is confined to a channel, however, the flow rates' variation throughout the channel cannot be easily interpreted. (c) Lenses with the k_{ind} layer providing details to better interpret the simulation's results. Local camera detail: notice how the flow velocity increases in the narrower section of the channel.

Moreover, multiple lenses can be blended and combined (*DG3*) as discussed in Section 6.5.1.

7.2 Performance

In practical cases, only few lenses are simultaneously used during data exploration [16]. However, for generating lens-regions, a large number of lenses may be required. Since our current implementation is applied to the visualization view (V), we do not consider the size of the dataset or the number of attribute layers being rendered.

For our experiment, we generated a set of lens centers uniformly distributed over the surface of the unit sphere using a Monte Carlo sampling strategy [59]. Since a sphere is isotropic, the rendering time is not affected by the depth test. The number of lenses was chosen to cover the whole surface (5000 lenses with a radius of 0.05). During rendering, we rotate the camera around the model and compute the average framerate. Our implementation (no fine optimizations) obtains interactive framerates (10 fps) using a resolution of 1280×1024 on an Intel® i7 *laptop* with a GeForce GTX 960 M 2GGPU. This emphasizes that our technique is scalable and allows for interaction on commodity hardware even with a high number of lenses.

It is important to mention that our implementation can be further improved. First, we render each lens individually in a serial fashion. Therefore, GPU parallelism is mostly used for pixel operations (decal-lens region). This expedites lens operations (e.g., brushing, lassoing) as well as interactions (e.g., edit, erase, dragging) but introduces an extra overhead due to the number of draw calls. One future direction is to cluster lenses by attribute and render all of them at once. Better sampling strategies can also be used to reduce the number of lenses used during brushing and lassoing operations.

8 A Detailed Example

We now present a detailed scenario where decal-lenses can be applied to augment multivariate visualization, e.g., by managing clutter through *on-demand* data exploration. The scenario consists of an oil recovery simulation, whose properties are defined on the surface of a 3D geological model (Fig. 11). Our design and visualization were conducted in collaboration with a domain expert in fluid dynamics and a reservoir engineer.

8.1 Fluid Flow and Petro-Properties in Oil Recovery

We focus on visualizing the result of a two-phase black-oil simulation [60] that models oil recovery through water injection. The geometric model is a hexahedral grid, and each grid block holds one value for each attribute. We have flow attributes (encoding the fluid dynamics), and petro-physical attributes (encoding geological properties of the porous medium). A significant difficulty is how to address the interrelation of these two types of attributes.

In this simulation the water/oil rates ($q_{[oil,water]}$) are given by Darcy's law: $\mu_{[oil,water]} q_{[oil,water]} = -k_{rock} k_{[oil,water]} \nabla P$, $q = q_{oil} + q_{water}$ where k_{rock} is the rock permeability tensor, ∇P is the gradient of pressure, $\mu_{[oil,water]}$ are oil/water viscosities, q is the total fluid velocity, and $k_{[oil,water]}$ are the oil/water relative permeabilities that model how the presence of each phase adversely interferes with the flow. Thus, permeability, pressure gradient, and the fluids' viscosities can all have significant impact on fluid flow, as, e.g., either a slow varying pressure filed, or a low permeability rock, or a high viscosity fluid would all lead to the perception of reduced flow in the simulation. Identifying which factor is the most important at places of interest (such as producer wells) is highly desirable. Such interrelation between fluid and rock attributes motivates our current multivariate visualization.

Previously, Rocha et al. [9] proposed a multivariate visualization of this simulation by applying the concept of layering on surfaces and following a perceptual-based design inspired by traditional illustration (for more details we refer the reader to [9]). Their multivariate visualization considers the following attributes: porosity (scalar), oil rate (vector), water rate (vector), and rock type (categorical data), which are sufficient to describe the basic aspects of the fluid flow. Here we extend this design to include two new attributes: the reservoir *pressure* (scalar, flow attribute), and an *average permeability indicator*, which highlights the influence of the

rock permeability (tensor, petrophysical attribute) in the oil/water rates (vectors, flow attributes). Moreover, we remark that oil/water rates are not independent quantities, they are related by the total fluid velocity (water cannot move where oil exists without pushing it first, and *vice versa*). Thus, unlike the visualization by Rocha et al. [9], we use the total fluid velocity to normalize water/oil rates.

Pressure is a continuous quantity that is commonly visualized using colormaps. To create a visual representation to express the change of pressure across the whole reservoir surface, without interfering with the visual representation of the other layers, we use a light-to-dark, orange colormap. Since porosity is strongly correlated with rock type, by painting porosity grains with the rock type pastel tones we can overlay porosity with pressure and still convey the rock type information. Fig. 11b illustrates this scenario.

Rock Permeability is a tensor that indicates the ability of a medium to support fluid flow. For simplicity, geologists and reservoir engineers represent permeability as a diagonal 3×3 matrix [11]. We *average* permeability as $k_{ind} = (k_{xx}^2 + k_{yy}^2 + k_{zz}^2)^{1/2}$, which we represent on the reservoir surface by clustering decals with an importance sampling strategy [59], where clustering means a higher k_{ind} value. Each decal is depicted by a small arrow with the normalized Darcy velocity direction. This k_{ind} layer conveys the local relevance of permeability to the flow, as well as, reinforcing flow direction everywhere.

Fig. 11a displays all the attributes encoded in layers on the surface of the reservoir model. As we can notice, this visualization suffers from clutter due to the overlay. By removing the k_{ind} layer (Fig. 11b), we can better visualize the flow pattern as well as the pressure variation from the injector to the producer well. Fig. 11 also illustrates how the oil/water rate normalization represents the flow more faithfully, e.g., the lens in the middle of the model (Fig. 11c), which is also depicted in the local camera detail, shows a region in which flow is significantly stronger than what is observed on its surroundings. The reason is conservation of mass: this lens is placed exactly above the narrower portion of the channel, which forces the fluid to move faster in the same way as air moves faster through doors and windows when flowing in a large room. The use of a decal-lens combined with a local camera in this context allows us to focus on a region of interest, accessing details of the visual representations, without losing the overall context.

Now, we focus on analyzing which factor is locally driving the fluid flow by comparing clustered areas in the k_{ind} layer with the flow arrow decals. However, to avoid clutter, this comparison is made by using decal-lenses, which allow us to select and quickly switch between layers. The user can drag the lens over the reservoir surface and select the k_{ind} attribute using the lens wheel. We return our attention to Fig. 11b, and notice that there is a clear pattern of fluid flowing through a channel. Inspecting the porosity's grain color, we observe that the channel is composed of two rock types (green and blue) that possibly extend to the top right part of the reservoir. Based on this analysis, a decal-lens is placed on the top-left of the model to depict how the permeability interacts with the flow near an injector well (Fig. 11c). To the right of this lens there is no flow, even though the permeability is high, which is a consequence of the pressure being almost constant, as highlighted by the lens-region brushed on the right of the decal-lens to emphasize the pressure variation by removing unnecessary information. Downstream this channel, near the producer well, the flow seems to reduce significantly. We then place another decal-lens close to the producer well. By looking at the k_{ind}, we can verify that the permeability in this region is still high, which implies that the reduction in the flow is due to the interaction between the two phases, and the fact that the oil viscosity (much higher than the water viscosity) is greatly impacting the flow (because fluid in this region is predominantly oil, depicted as red arrows).

9 Conclusions and Future Work

We proposed a *new lens technique* for multivariate visualization and layered visualization in general. Motivated by five design goals, we formalized *decal-lenses*–object-space lenses that follow the surface curvature and allow for multiple uses and interactions. We connected decal-lenses to the conceptual model of lenses [16], describing their properties, how they can interact with the visualization pipeline during the *selection* σ and *join* $\bowtie$ stages as well as operate over data through *lens functions* λ. Our GPU implementation allows users to place and interact with lenses *on the surface* to create focus+context customized visualizations while avoiding the interaction effort of re-positioning or re-orienting the lens during data exploration. We also combined decal-lenses with *local cameras* for data comparison and correlation.

Due to its simplicity and flexibility, our approach allows for designing *lens-regions* of arbitrary shapes by combining multiple decal-lenses. This addresses one of the main issues in lens techniques which is the need to combine lenses to create new lens functions *on-the-fly* [16]. In our approach, lens-regions can be easily created using *brushing* and *lassoing* operations. Moreover, we introduced the concept of *support surface* for designing interaction techniques. Exploring this concept, a *see-through window* metaphor can be used for data exploration. For each of these concepts, we provided detailed examples in a variety of contexts. We hope that these examples provide insights of how decal-lenses can be used to facilitate data exploration in the future.

For future work we cite some noteworthy avenues. Our decal-lenses implementation operates on the view stage of the visualization pipeline, but the ideas introduced with decal-lenses are not limited to the view stage themselves. For example, one could use decal-lenses for analytics, by selecting the data in the data tables stage. Moreover, this idea could also be explored in the context of data manipulation as pointed out by Tominski *et. al* survey [16]. Up until now, decal-lenses only modify what is inside the lens, however, one could have used decal-lenses to modify the context, if required by a specific design. We also would like to explore further sampling strategies to generate lens-regions. The concept of support surfaces also warranties more research, in particular we envision their use for data representation in objects that are not surfaces, such as streamlines. Last, but not least, decal-lenses can be used to correlate multivariate data defined on distinct level-sets of a single scalar function evolving in time, such as a moving fluid front.

There has been growing effort on bringing concepts from information visualization to scientific visualization. On

surfaces for example, we cite the use of 2D glyphs and the details-on-demand paradigm for medical visualization [6], [7], and the extension and application of the concept of layering using decal-maps [9], [10], [11]. Our work builds on these concepts by introducing to scientific visualization an interaction technique inspired by the magic lenses concept that is well established in information visualization. However, many more techniques that implement concepts for overview-detail, data comparison and correlation, spatial to abstract data integration and interaction techniques for data manipulation are still either incipient or not available in the scientific visualization community. We argue that exploring these ideas born in the information visualization community in the eyes of scientific visualization can provide a fruitful avenue of research in the years to come.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments and feedback. This research was supported in part by the NSERC/AITF/FCMG Industrial Research Chair Program in Scalable Reservoir Visualization and by Discovery Grants from NSERC.

References

[1] R. Fuchs and H. Hauser, "Visualization of multi-variate scientific data," *Comput. Graph. Forum*, vol. 28, pp. 1670–1690, 2009.
[2] J. Kehrer and H. Hauser, "Visualization and visual analysis of multifaceted scientific data: A survey," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 3, pp. 495–513, Mar. 2013.
[3] T. Munzner, *Visualization Analysis and Design*. Boca Raton, FL, USA: CRC Press, 2014.
[4] 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," *Comput. Graph. Forum*, vol. 23, pp. 203–221, 2004.
[5] B. Cabral and L. C. Leedom, "Imaging vector fields using line integral convolution," in *Proc. 20th Annu. Conf. Comput. Graph. Interactive Techn.*, 1993, pp. 263–270.
[6] R. van Pelt, R. Gasteiger, K. Lawonn, M. Meuschke, and B. Preim, "Comparative blood flow visualization for cerebral aneurysm treatment assessment," in *Comput. Graph. Forum*, vol. 33, pp. 131–140, 2014.
[7] M. Meuschke, S. Voß, O. Beuing, B. Preim, and K. Lawonn, "Glyph-based comparative stress tensor visualization in cerebral aneurysms," *Comput. Graph. Forum*, vol. 36, pp. 99–108, 2017.
[8] R. M. Kirby, H. Marmanis, and D. H. Laidlaw, "Visualizing multivalued data from 2D incompressible flows using concepts from painting," in *Proc. Conf. Vis.*, 1999, pp. 333–340.
[9] A. Rocha, U. Alim, J. D. Silva, and M. C. Sousa, "Decal-maps: Real-time layering of decals on surfaces for multivariate visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 821–830, Jan. 2017.
[10] A. Rocha, J. D. Silva, U. Alim, and M. C. Sousa, "Multivariate visualization of oceanography data using decals," in *Proc. Workshop Vis. Environ. Sci.*, 2017, pp. 31–35.
[11] A. Rocha, R. C. R. Mota, H. Hamdi, U. R. Alim, and M. C. Sousa, "Illustrative multivariate visualization for geological modelling," *Comput. Graph. Forum*, vol. 37, no. 3, pp. 465–477, 2018.
[12] 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 Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2183–2192, Dec. 2011.
[13] C. Ware, *Information Visualization: Perception for Design*. New York, NY, USA: Elsevier, 2012.
[14] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proc. IEEE Symp. Visual Languages*, 1996, pp. 336–343.
[15] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose, "Toolglass and magic lenses: The see-through interface," in *Proc. 20th Annu. Conf. Comput. Graph. Interactive Techn.*, 1993, pp. 73–80.
[16] C. Tominski, S. Gladisch, U. Kister, R. Dachselt, and H. Schumann, "Interactive lenses for visualization: An extended survey," *Comput. Graph. Forum*, vol. 36, pp. 173–200, 2017.
[17] E. Groot, B. Wyvill, L. Barthe, A. Nasri, and P. Lalonde, "Implicit decals: Interactive editing of repetitive patterns on surfaces," *Comput. Graph. Forum*, vol. 33, pp. 141–151, 2014.
[18] M. Sarkar, S. S. Snibbe, O. J. Tversky, and S. P. Reiss, "Stretching the rubber sheet: A metaphor for viewing large layouts on small screens," in *Proc. ACM Symp. User Interface Softw. Technol.*, 1993, pp. 81–91.
[19] E. LaMar, B. Hamann, and K. I. Joy, "A magnification lens for interactive volume visualization," in *Proc. 9th Pacific Conf. Comput. Graph. Appl.*, 2001, pp. 223–232.
[20] C. Pindat, E. Pietriga, O. Chapuis, and C. Puech, "JellyLens: Content-aware adaptive lenses," in *Proc. 25th Annu. ACM Symp. User Interface Softw. Technol.*, 2012, pp. 261–270.
[21] M. Carpendale and C. Montagnese, "A framework for unifying presentation space," in *Proc. 14th Annu. ACM Symp. User Interface Softw. Technol.*, 2001, pp. 61–70.
[22] A. Fuhrmann and E. Gröller, "Real-time techniques for 3D flow visualization," in *Proc. Conf. Vis.*, 1998, pp. 305–312.
[23] P. Cignoni, C. Montani, and R. Scopigno, "MagicSphere: An insight tool for 3D data visualization," *Comput. Graph. Forum*, vol. 13, pp. 317–328, 1994.
[24] J. Viega, M. J. Conway, G. Williams, and R. Pausch, "3D magic lenses," in *Proc. 9th Annu. ACM Symp. User Interface Softw. Technol.*, 1996, pp. 51–58.
[25] T. Ropinski and K. Hinrichs, "Real-time rendering of 3D magic lenses having arbitrary convex shapes," *J. WSCG*, vol. 12, no. 1-3, pp. 379–389, 2004.
[26] L. Wang, Y. Zhao, K. Mueller, and A. Kaufman, "The magic volume lens: An interactive focus+ context technique for volume rendering," in *Proc. Conf. Vis.*, 2005, pp. 367–374.
[27] C. Kirmizibayrak, M. Wakid, Y. Yim, D. Hristov, and J. K. Hahn, "Interactive focus+ context medical data exploration and editing," *Comput. Animation Virtual Worlds*, vol. 25, pp. 129–141, 2014.
[28] J. Plate, T. Holtkaemper, and B. Froehlich, "A flexible multivolume shader framework for arbitrarily intersecting multiresolution datasets," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1584–1591, Nov. 2007.
[29] C. W. Borst, J.-P. Tiesel, and C. M. Best, "Real-time rendering method and performance evaluation of composable 3D lenses for interactive VR," *IEEE Trans. Vis. Comput. Graph.*, vol. 16, no. 3, pp. 394–410, May/Jun. 2010.
[30] P. Rautek, S. Bruckner, E. Gröller, and I. Viola, "Illustrative visualization: New technology or useless tautology?" *ACM SIGGRAPH Comput. Graph.*, vol. 42, pp. 4:1–4:8, Aug. 2008.
[31] I. Viola, "Importance-driven expressive visualization," PhD thesis, Inst. Comput. Graph. Algorithms, Vienna Univ. Technol., Wien, 2005.
[32] J. Diepstraten, D. Weiskopf, and T. Ertl, "Interactive cutaway illustrations," *Comput. Graph. Forum*, vol. 22, pp. 523–532, 2003.
[33] W. Li, L. Ritter, M. Agrawala, B. Curless, and D. Salesin, "Interactive cutaway illustrations of complex 3D models," *ACM Trans. Graph.*, vol. 26, no. 3, 2007, Art. no. 31.
[34] S. Bruckner, M. E. Gröller, K. Mueller, B. Preim, and D. Silver, "Illustrative focus+ context approaches in interactive volume visualization," *Dagstuhl Follow-Ups*, vol. 1, pp. 136–162, 2010.
[35] E. Hodges and G. of Natural Science Illustrators (U.S.), *The Guild Handbook of Scientific Illustration*. Hoboken, NJ, USA: Wiley, 2003.
[36] S. Bruckner, P. Rautek, I. Viola, M. Roberts, M. C. Sousa, and M. E. Gröller, "Hybrid visibility compositing and masking for illustrative rendering," *Comput. Graph.*, vol. 34, no. 4, pp. 361–369, 2010.
[37] S. Bruckner and M. E. Groller, "Exploded views for volume data," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 1077–1084, Sep./Oct. 2006.
[38] J. Kruger, J. Schneider, and R. Westermann, "ClearView: An interactive context preserving hotspot visualization technique," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 941–948, Sep. 2006.
[39] I. Viola and E. Gröller, "Smart visibility in visualization," in *Proc. 1st Eurographics Conf. Comput. Aesthetics Graph. Vis. Imaging*, 2005, pp. 209–216.
[40] V. Interrante, H. Fuchs, and S. M. Pizer, "Conveying the 3D shape of smoothly curving transparent surfaces via texture," *IEEE Trans. Vis. Comput. Graph.*, vol. 3, no. 2, pp. 98–117, Apr. 1997.
[41] N. Elmqvist and P. Tsigas, "A taxonomy of 3D occlusion management for visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 5, pp. 1095–1109, Sep./Oct. 2008.

[42] N. Elmqvist, A. V. Moere, H.-C. Jetter, D. Cernea, H. Reiterer, and T. Jankun-Kelly, "Fluid interaction for information visualization," *Inf. Vis.*, vol. 10, no. 4, pp. 327–340, 2011.
[43] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vis. to Think*. Burlington, MA, USA: Morgan Kaufmann, 1999.
[44] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification envelopes," in *Proc. 23rd Annu. Conf. Comput. Graph. Interactive Techn.*, 1996, pp. 119–128.
[45] S. Calderon and T. Boubekeur, "Bounding proxies for shape approximation," *ACM Trans. Graph.*, vol. 36, no. 5, 2017, Art. no. 57.
[46] G. Cipriano and M. Gleicher, "Text scaffolds for effective surface labeling," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 6, pp. 1675–1682, Nov./Dec. 2008.
[47] G. Sellers, R. S. Wright, and N. Haemel, *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Boston, MA, USA: Addison-Wesley, 2013.
[48] A. Mahdavi-Amiri, T. Alderson, and F. Samavati, "A survey of digital earth," *Comput. Graph.*, vol. 53, pp. 95–117, 2015.
[49] R. Gasteiger, M. Neugebauer, C. Kubisch, and B. Preim, "Adapted Surface Visualization of Cerebral Aneurysms with Embedded Blood Flow Information," in *Proc. 2nd Eurographics Conf. Visual Comput. Biol. Med.*, 2010, pp. 25–32.
[50] VisItUsers, "Aneurysm data," 2016. [Online]. Available: https://tinyurl.com/jcdmdlv, Accessed on: Dec. 11, 2016.
[51] S. R. de Galarreta, A. Cazón, R. Antón, and E. A. Finol, "The relationship between surface curvature and abdominal aortic aneurysm wall stress," *J. Biomechanical Eng.*, vol. 139, no. 8, 2017, Art. no. 081006.
[52] S. Glaßer, K. Lawonn, T. Hoffmann, M. Skalej, and B. Preim, "Combined visualization of wall thickness and wall shear stress for the evaluation of aneurysms," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2506–2515, Dec. 2014.
[53] T. Sherif, N. Kassis, M.-É. Rousseau, R. Adalat, and A. C. Evans, "Brainbrowser: Distributed, web-based neurological data visualization," *Frontiers Neuroinformatics*, vol. 8, 2015, Art. no. 89.
[54] B. Fischl, M. I. Sereno, and A. M. Dale, "Cortical surface-based analysis: II: Inflation, flattening, and a surface-based coordinate system," *Neuroimage*, vol. 9, no. 2, pp. 195–207, 1999.
[55] T. Tallinen, J. Y. Chung, F. Rousseau, N. Girard, J. Lefèvre, and L. Mahadevan, "On the growth and form of cortical convolutions," *Nature Phys.*, vol. 12, no. 6, pp. 588–593, 2016.
[56] S. Rusinkiewicz, "Estimating curvatures and their derivatives on triangle meshes," in *Proc. Int. Symp. 3D Data Process. Vis. Transmiss.*, 2004, pp. 486–493.
[57] M. Sarletu and G. Klein, "Hardware-accelerated ambient occlusion computation," in *Proc. Vis. Model. Vis.*, 2004, pp. 331–338.
[58] W. J. van der Laan, S. Green, and M. Sainz, "Screen space fluid rendering with curvature flow," in *Proc. Symp. Interactive 3D Graph. Games*, 2009, pp. 91–98.
[59] M. Corsini, P. Cignoni, and R. Scopigno, "Efficient and flexible sampling with blue noise properties of triangular meshes," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 6, pp. 914–924, Jun. 2012.
[60] J. A. Trangenstein and J. B. Bell, "Mathematical structure of the black-oil model for petroleum reservoir simulation," *SIAM J. Appl. Math.*, vol. 49, no. 3, pp. 749–783, 1989.

Allan Rocha received the MSc degree in computer science from PUC-Rio, working in illustrative visualization. During this period, he had also worked as a researcher and software engineer developing visualization solutions for the oil and gas industry. Currently, he is a PhD Candidate in computer science, pursuing his research at the Illustrares and VISAGG groups at the University of Calgary. His research combines aspects from scientific visualization, information visualization, non-photorealistic rendering, visual design, interaction design and real-time rendering, to tackle the problem of visualizing and interacting with multivariate spatial data.

Julio Daniel Silva received both MSc and PhD degrees in mathematics from the Instituto Nacional de Matemática Pura e Aplicada (IMPA), Brazil. He is a senior research associate with the University of Calgary, Canada. His research interests include pure and applied aspects of partial differential equations and numerical analysis in the context of fluid dynamics and geometric modelling, heterogeneous parallel computing, and the visualization of all of this.

Usman R. Alim received the PhD degree in computer science from Simon Fraser University, in 2012. Since 2012, he has been with the Department of Computer Science, University of Calgary where he is currently an associate professor. He is the director of the Visualization and Graphics Group (VISAGG) which focuses on addressing a diverse range of fundamental and applied problems in Data Visualization and Computer Graphics. His current interests include multivariate data visualization, large scale data visualization, visualization in immersive environments, and statistical and numerical methods for visualization.

Sheelagh Carpendale is a full professor with the University of Calgary where she holds a Tier 1 Canada research chair in information visualization and an NSERC/AITF/SMART industrial research chair in interactive technologies. She has many received awards including the E.W.R. NSERC STEACIE, a BAFTA; and is a member of the ACM CHI Academy. She directs the Innovations in Visualization (InnoVis) research group and initiated the interdisciplinary graduate program, Computational Media Design. Her research interests include information visualization, interaction design, and qualitative empirical research. By studying how people interact with information both in work and social settings, she works towards designing more natural, accessible and understandable interactive visual representations of data.

Mario Costa Sousa received the MSc degree from PUC-Rio, Brazil, in 1994, and the PhD degree from the University of Alberta, Canada, in 1999. He is an associate professor of computer science with the University of Calgary, Canada. He leads the Illustrares research group, a multidisciplinary team working on fundamental and applied research of interactive visual computing in science and engineering. His research interests include non-photorealistic rendering, sketch-based interfaces and modeling, illustrative visualization, visual analytics, and human-data and computer interaction. He was a recipient of an eight-year industrial research chair in Scalable Reservoir Visualization sponsored by the Canadian government and the industry sector.