Interactive Data Deformation Techniques to Improve Feature Visibility in Scientific Visualization

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Cheng Li, B.E., M.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2018

Dissertation Committee:

Han-Wei Shen, Advisor

Jian Chen

Roger Crawfis

Abstract

Visualization has been widely applied in scientific research in order to obtain various kinds of information from complex datasets. The direct perspective projection of a three-dimensional dataset frequently faces several limitations that hinder the visibility of data features, and thus hurts the visual cognition of the data. Data deformation is a widely used strategy to improve feature visibility by removing occlusions or enhancing feature layout. Facing the increasing data size and complexity in modern scientific research, advanced designs of interactive deformation systems are needed to tackle many new challenges, such as preserving data features during the deformation, handling special relation between the virtual camera and the data, building more intuitive connection between the data and the operator, etc. In this dissertation, I will present three new interactive deformation techniques that contribute to data exploration in different scenarios with the goal of improving the feature visibility.

The first proposed technique targets the need of feature preservation when conducting data deformation. For this aim, I propose a new physically-based data exploration system that preserves data features in the context by setting the mesh properties according to interesting data attributes selected by users. The designed interfaces use a metaphor of virtual retractor, which reflects the cutting and splitting of data that the system is simulating.

The second proposed technique is to address the unique occlusion situations that occur when exploring a three-dimensional dataset in an egocentric mode. I tackle this problem by

proposing a new automatic deformation method. The method detects the part of the data that blocks the camera, and deforms it to create a clean view. With the proposed technique, the system performs the deformation automatically without interrupting other user interactions that are currently being performed.

The third proposed technique aims to enable users to deform a dataset as if handling a real object in hand. The system utilizes a touchscreen, so users can directly perform deformation with their fingers. Users can drag interesting data features and move them along with the finger. By directly using fingers to perform deformation, our system not only works intuitively, but also benefits from coordinating multiple fingers to create more deformation effects that are hard to conduct by one mouse.

Dedicated to my family

Acknowledgments

I would like to thank my beloved family for their support of pursuing my doctoral degree. My parents, Qinge Sui and Zongnian Li, have been giving me consistent care ever since my born. My wife, Li Zhang, has been accompanying me during the hardest time of being a student. And my little kid Elbert, who has delighted my life and encouraged me to be a better person.

I would like to express my sincere gratitude to my advisor, Professor Han-Wei Shen. Dr. Shen is the most important person that successfully turns me into a researcher. He has trained me in various aspects of conducting academic research, which will be a precious treasure in my future professional life. He has always been instructive and patient at every phase of my progress, especially when encountering obstructions or failures. He has also been supportive of my research interest and career plan. His passion, vision, and honesty towards research have deeply impacted my working habit and will benefit me in a long run. I feel really lucky to have had such a great advisor. At the same time, I want to thank my committee members, Dr. Jian Chen and Dr. Roger Crawfis, for their help in guiding the formation of this dissertation, as well as Dr. Huamin Wang who has helped the proposal of this dissertation.

I want to thank all my collaborators in my research publications during the past years, including Dr. Xin Tong, Dr. Joachim Moortgat, Mr. Junpeng Wang, and Mr. Subhashis

Hazarika. Thank external users, including Dr. Wei Zhu and many others, for providing domain feedback to my works. Thank Ms. Wenxi Yang for helping creating the accompanying video of paper submission. Thank my labmates Chun-Ming Chen, Wenbin He, Souyma Dutta, Xiaotong Liu, Ayan Biswas, Kewei Lu, Tzu-Husan Wei, Ko-Chih Wang, and Jiayi Xu, for the inspiring discussions we had, as well as the fun life we spent together. Thank all my teachers in OSU, and all my friends in Columbus, and this period of study and life will be an unforgettable memory of my whole life.

My appreciation also goes to Dr. Punam Saha from The University of Iowa, who has opened up the gate of academic research to me and helped me achieved my master degree, as well as all lab mates from his group I had worked together with. Thank my advisor in undergraduate school, Dr. Zhijian Ou, from Tsinghua University, and all classmates who shared cherish memories with me. Thank my mentor, Austin Zhao, during my internship in Facebook, and my mentors, Dr. Christine Sweeney and Dr. Patrick McCormick, during my internship in LANL, who helped me learn a lot of knowledge and skills during the internships that further helped my academic works. Thank my intern mentors before joining OSU, including Dr. Shuo Li from General Electric and Mr. Chi Zhang from Nufront Software, who helped me improve various skills and get better knowledge of my field of study.

Vita

2006 - 2010	B.E., Electronic Engineering, Tsinghua University, China.
2010 - 2013	M.S., Electrical and Computer Engineering, The University of Iowa, USA.
2013 - present	Ph.D. student, Computer Science and Engineering, The Ohio State University, USA.
June - August, 2017	Research Intern, Los Alamos National Lab, USA.
May - August, 2018	Intern Software Engineer, Facebook Inc., USA.

Publications

Research Publications

Junpeng Wang, Subhashis Hazarika, Cheng Li, Han-Wei Shen "Visualization and Visual Analysis of Ensemble Data: A Survey". *IEEE Transactions on Visualization and Computer Graphics,* 2018.

Cheng Li, Joachim Moortgat, Han-Wei Shen "An Automatic Deformation Approach for Occlusion Free Egocentric Data Exploration". *IEEE Pacific Visualization Symposium (PacificVis),* 2018.

Cheng Li, Han-Wei Shen "Winding Angle Assisted Particle Tracing in Distribution-Based Vector Field". *ACM SIGGRAPH Asia Symposium on Visualization,* 2017.

Cheng Li, Xin Tong, Han-Wei Shen "Virtual Retractor: An Interactive Data Exploration System Using Physically Based Deformation". *IEEE Pacific Visualization Symposium (PacificVis),* 2017.

Xin Tong, Cheng Li, Han-Wei Shen "GlyphLens: View-dependent Occlusion Management in the Interactive Glyph Visualization". *IEEE Transactions on Visualization and Computer Graphics,* 2017.

Fields of Study

Major Field: Computer Science and Engineering

Table of Contents

Page

List of Tables

List of Figures

Figure **Page**

Chapter 1: Introduction

1.1 Motivation

Scientific visualization has been widely applied as an important tool for knowledge discovery from complex spatial datasets. Compared to simply presenting numbers, visualization can convey information from the data in an intuitive and comprehensive way, and stimulate users' visual perception more efficiently. However, the benefits can only be obtained when three-dimensional data elements are arranged effectively on the output devices. Otherwise, the visibility of important or interesting features can be damaged, and the intuitiveness and effectiveness of information capture are compromised.

A frequent type of degeneration of feature visibility is caused by occlusion. The projection of three-dimensional objects onto a two-dimensional plane leads to loss of depth values of data elements, which frequently places occlusion upon farther data elements. While occlusion sometimes can provide helpful depth cue, important data features can be blocked by visual obstacles or become less visible. The obstacles differ from each other in different aspects, thus introduce varying challenges in different situations. The obstacles themselves may also contain implicitly defined features that should not be simply ignored. Another type of cases that hurts the feature visibility is the unexpected layout of features. Two features can be too close such that they block each other, and they may be tangled

in complex ways, making it hard to separate them. Related features may also be located far away from each other, which makes the visual comparison of them impossible. In this situation, displacing the features to desired new locations can be expected by users.

Data deformation has been a widely used data manipulation strategy to improve feature visibility. Data deformation can be defined as the techniques that directly change the shapes or locations of data elements [13]. Interactive deformation systems have been used in various visualization applications to solve different kinds of problems, such as Focus+Context visualization [81], feature enhancement [21], occlusion removal [71], surgical simulation and practicing [46], etc. Compared to other choices of visual manipulation, such as using cutaway and transparency, or using multiple viewpoints, deformation avoids extra visual load caused by different transparency strategies across the domain, as well as keeping the perspective viewpoint [29].

Besides the long history of deformation studies, modern scientific research is facing new challenges which cannot be satisfied by existing deformation techniques. For example, the data fidelity has been consistently improved, and more complex and tiny features can occur in the visual context. At the same time, the improvement of various types of hardware devices has created new opportunities for innovative deformation techniques that are hard to achieve previously. For example, the improvement of GPUs can boost many computation-demanding methods to interactive rates; the improvement of immersive devices poses requirements for algorithms to work in a virtual environment; the dissemination of touch-enabled input devices brings new connections between data and operators. In this dissertation, I will study several new challenges of the deformation strategy, and propose three new interactive deformation techniques. In the next subsection, I will briefly introduce the three new techniques one by one.

1.2 Problem Statements And Proposed Techniques

1.2.1 Physically Based Deformation with Feature Preservation

The first problem tackled in this dissertation is to improve feature preservation during data deformation. Compared to other occlusion removal techniques such as using transparency or cutaway, using deformation preserves the context near the revealed target rather than attenuating or discarding it, which is an important advantage. However, the context is still changed and its original structure can be destroyed. As the data resolution increases, it becomes more frequent that interesting details of the data can exist in the context around the target, or in the obstacles that block the target. The demand of keeping features in the context is less attended in traditional non-physically based deformation methods. As a consequence, to inspect the hidden target, users may have to pay the cost that important features in the context are altered too much.

The goal of the first proposed technique in this dissertation is to better preserve features in the context during the deformation. The solution is to adopt a physically-based deformation, and encode data features in the context into the physical properties of the deformable mesh. First I will introduce some background of physically-based deformation. Correa *et al.* [23] defined data deformation by using a displacement function $\vec{D}()$ upon every data element λ. Let the original position of λ to be $P(\lambda)$, and its new position to be $P'(\lambda)$, the displacement function $\vec{D}()$ will make

$$P'(\lambda) = P(\lambda) + \vec{D}(\lambda), \tag{1.1}$$

where $P(), P'(), \vec{D}()$ are all in R^3. Based on the displacement function $\vec{D}()$, deformation methods can be classified into physically-based and non-physically-based [23]. Non-physically-based methods are also called empirical methods [22], since they use a few

empirical functions to define $\vec{D}()$. Physically-based methods embed the physical properties of the data into the deformation by mechanical laws. This is commonly realized by using a mesh, and then the function $\vec{D}()$ is essentially to produce a one-to-one mapping between the original mesh and the deformed mesh, with the mesh decomposed into basic mesh elements (e.g., cubes or tetrahedra) during actual computation. Physically based deformation grants users more power to control the deformation by manipulating each individual cube or tetrahedron of the mesh. It not only simulates the shape altering with more aesthetically plausible animation, but also provides users more channels to understand data properties through the deformation process, if the data attributes are properly encoded into the mesh properties [54, 72]. Although studied a lot in the computer graphics area, physically-based deformation used to hinder visualization applications when the high computation cost could not satisfy the requirement for the interactive rate in early days [21, 23]. However, along with the development of GPUs, much progress has been made to utilize the benefit of physically-based deformation [54, 70, 72, 81, 82].

The proposed technique uses an elastic mesh that covers the obstacles that block the interesting target, with a void in the middle. After adding forces to the mesh, the void will be enlarged to reveal the inner structure which was originally occluded. The mesh is constructed with local data properties taken into account, such as data density or gradient. This setting preserves features of the context, by deforming more on regions with less interest, and deforming less on regions with more user interest. With this technique, the visibility of features in the context can be improved during the deformation, since their shapes are less altered. The technique is integrated in a system that is realized by a metaphor of *virtual retractor*, due to the resemblance to a medical retractor in surgeries.

1.2.2 Automatic Deformation for Egocentric Data Exploration

The second challenge that modern deformation research faces is the unique occlusion problems that happen in the egocentric visualization mode, which places the camera inside the bounding box of the dataset. The egocentric visualization mode is becoming more and more prevalent as the data size increases and more tiny structures may occur in the data interior. It also attracts more attention as the development of virtual reality (VR) techniques. The reason for the unique occlusion problems in the egocentric mode is that the camera can be too close to data elements. While in exocentric mode the obstacle usually locates with a significant distance from the camera, in egocentric mode, the obstacle can directly collide with the camera and cover the view. This phenomenon can happen when users navigate the camera so the camera approaches opaque data elements; or when the camera stays still, but opaque elements move towards the camera. In such cases, the scene will only contain these few opaque elements and become very uninformative. Traditional occlusion management techniques may not work equally well in egocentric mode. Also because the occlusion is more likely to happen when users are performing some interactions to change the scene, spending effort to manually remove occlusion at the same time can be less efficient. Several strategies can sometimes handle this type of occlusion, such as restricting the locations where the camera can go. The restriction strategy can be less effective for free data exploration, and make certain data elements less accessible.

The second proposed technique in this dissertation is an automatic deformation system that serves data exploration tasks using egocentric mode. It will automatically check the camera and the data at each frame, to see if the camera is blocked by nearby obstacles or not. If any blocking happens, it first processes the data to circle out the obstacles. Then the obstacles will be moved far away from the front of the camera, to create a pathway for the

straight view. Therefore rays from the camera can reach farther, instead of terminating very early after a short distance. As a result, the viewer will be able to see rich data features far away behind the obstacles. The automation of the method is realized by a state transition model, to fuse it into the visualization pipeline and make it applicable for many different data types and different exploration tasks.

1.2.3 Object-In-Hand Deformation Using Touchscreen And Physically Based Deformation

The last problem this dissertation focuses on is to provide more human-like operations to conduct deformation. Traditional deformation interactions are mostly conducted using the mouse+keyboard combination, and users may need to learn complex mouse or keyboard operations to control the deformation. This type of interaction has made long time contribution in the history of deformation research and applications. However, it still bears limitations, including the mechanical barrier between the data and human operators, and the limitation of the deformation effects that can be achieved.

The goal of the proposed technique is to design more intuitive interactions to conduct the deformation, and thus strengthen the connection between the data and human operators. The technique follows an *object-in-hand* metaphor. This metaphor has served the design of several popular interaction systems, such as the scene-in-hand camera control [83]. Similar ideas of deforming an object as if using a person's two hands have driven non-physically-based designs before, such as cracking or peeling metaphors based on affine transformation or trigonometric functions. But users' feeling of *in-hand* may not be strong if such designs are implemented by virtual widgets controlled by the mouse or keyboard.

The proposed interactive deformation system enables users to reshape the data by real hands through a touchscreen input. Users can drag data features and move them along with

the finger. Users can also press their fingers to hold other parts of the data fixed during the deformation. Users can also perform cutting on the data using a finger. As the data elements are sticking to users' fingers through the touchscreen and acting together with the fingers, the *object-in-hand* metaphor can be intuitively realized. The deformation utilizes the power of a physically-based deformable mesh, which can spread the influence of the fingers to the data in a physically-based manner, and displace data elements in more sophisticated and flexible ways. The object-in-hand interaction scheme is not only natural to users, but also allows one user to use multiple fingers to perform different operations with different configurations (e.g., dragging direction) at the same time. As a result, users can displace features in more ways than previous methods that initiated the deformation from one virtual widget. By displacing the features to new locations with less visual clutter or to form a better layout, their visibility can be effectively improved.

1.3 Contributions

In summary, this dissertation makes contributions to the family of data deformation strategy, by utilizing modern devices to solve several front edge problems in scientific data visualization area. An itemized list of the specific contributions made by this dissertation is given below:

- Propose a deformation technique that uses data attributes to control the physically based deformation, which can better preserve features in the context.
- Propose a deformation technique that works automatically by checking the camera status, to solve the occlusion when exploring the dataset in egocentric mode.

- Propose a deformation technique that follows the object-in-hand metaphor, which enables users to alter the data directly using their fingers on a touchscreen.

For each proposed technique, a new system is designed and presented, which encapsulates the technique with friendly and powerful interactions, to provide users pleasing control experiences. All systems utilize the power of GPUs to interactively respond to user inputs, finish the heavy but parallelizable computation, and present intermediate steps of the deformation to form smooth animations. All systems have flexible applicabilities on multiple data types and different visual applications, which share common characteristics but also possess unique challenges to be attend correctly. The details of each proposed technique will be introduced in Section 3, 4, and 5, which also provide the corresponding designs of the interactive systems, and thorough case studies to further support the contributions.

Chapter 2: Background and Related Work

2.1 Data Deformation For Scientific Visualization

I summarize the literature of deformation research in 3D visualization according to the chosen displacement function $\vec{D}()$ in Equation 1.1. More specifically, I will review the works which use spatially continuous functions as $\vec{D}()$, which use discontinuous functions as $\vec{D}()$, and which do not rely on specifically defined $\vec{D}()$.

The first category of related works contains the deformation techniques whose function $\vec{D}()$ are spatially continuous. This is mostly seen in Focus+Context visualization, by enlarging a focus region, and shrinking its nearby context regions. As a result, the visualization can render important or interesting regions of the data with larger space, so users can receive better visual stimulation. LaMar *et al.* [51] magnified interesting regions using local affine transformation. The Magic Volume Lens [80] emphasized interesting regions in a fisheye manner. Zhao *et al.* [89] performed the Focus+Context deformation using conformal mapping theory to achieve angle distortion minimization. Li *et al.* [52] performed the deformation in four-dimensional space, and project the deformation back to 3D space when needed. Wang *et al.* constructed a cubical mesh by dividing the space domain uniformly, and deformed the mesh using physically-based rules to enlarge views for interesting area, for surface visualization [81] as well as volume rendering [82]. Tao

et al. [70] also used a cubical mesh to reposition the streamlines of a flow field, to expand regions that contain important flow features or complex structures. Tong *et al.* [72] used a tetrahedral mesh to displace glyphs, while trying to keep the relative locations of the glyphs less disrupted. This type of deformation can also be used for occlusion management, to enlarge an empty space in front of a partially occluded target. This strategy has been used for flow visualization [70], particle visualization [72], and volume visualization [74]. However, this strategy requires the existence of the empty space, since the continuous function $\vec{D}()$ cannot break continuous obstacles to create an empty space to enlarge.

The next category of deformation techniques uses spatially discontinuous $\vec{D}()$. Such techniques play important roles for occlusion management. Compared with using continuous function $\vec{D}()$, this strategy does not require an existing empty space, since the discontinuity can break the space to create a new empty space. The early work of McGuffin *et al.* [59] treated a volume by a stack of layers, and defined the deformation by transforming the layers. Correa *et al.* [23] proposed displacement mapping to describe the deformation, as the Equation 1.1. His later works include incremental designs of the function $\vec{D}()$, such as the one to better separate focus and context [20], and the one to better separate medical features [21]. Bruckner and Gröller [10] exploded different parts of the obstacles into sides of the image. Islam *et al.* [39] studied the applications of splitting of a volume, or parts of the volume. Birkeland and Viola [9] designed a view-dependent peeling method, to roll up the layer of the obstacle to review the target behind. Tong *et al.* used lens to reveal hidden features in flow field visualization [71] and particle visualization [72], by replacing the obstacles according to the shape of the lens. Li *et al.* [54] enabled users to control the behavior of the deformation by setting the property of the physically-based deformation

mesh. Li *et al.* [53] studied the occlusion problems in egocentric visualization and virtual environment, and proposed an automatic solution driven by the camera location.

The last category places less restrictions on $\vec{D}()$, but focuses on general questions of deformation technologies. Chen *et al.* [14] used spatial transfer functions to define deformation. Some papers worked on improving performance to ensure the interactive rate [30, 85], which was quite challenging in early days even for non-physically-based deformation. Walton and Jones [78] studied the mapping between the original volume and the deformed volume for empirical deformation, to directly use the mapping for visualization and avoid the reconstruction of the deformed volume. Recently for physically-based volume deformation, GPU based rerasterization methods [34, 73] have been consistently studied to reconstruct the deformed volume with less time cost.

2.2 Occlusion Management Strategies

One category of occlusion management methods that directly attenuate the objects between the camera and the target can be referred to as Virtual X-ray methods. Magic lense [8, 76, 80] rendered the obstructions with higher transparency. Cutaway [11, 26, 55] methods removed the occluding part, to achieve a fully transparent effect. Augmented Reality (AR) [5, 56] techniques overlapped the image of the occluded target over the real photos to provide a see-through effect. For direct volume rendering, besides eliminating voxels according to their positions or density values, Rezk-Salama and Kolb [62] also considered the voxel's occurrence on the casted ray; and Hurter *et al.* [37] selected voxels in feature spaces, such as color or intensity distribution. The Virtual X-ray methods can lose important context information provided by the attenuated part, which may not always be the preferred situation.

Another category of methods employs more than one view points or projections. Multiple viewpoints methods [4, 25, 61, 86] looked for alternative view points that could observe the target without occlusion. Elmqvist and Tsigas [28] let users switch between perspective and parallel projections. These methods all require the existence of at lease one alternative view or projection which is not occluded, but the existence is not guaranteed, such as when the target is fully enclosed inside another object.

Deformation is also widely applied for occlusion management. This part of existing works has been reviewed in the last section (Section 2.1). In the end, Elmqvist and Tsigas [29] wrote a survey paper about the occlusion removal techniques, and provided a taxonomy of them.

2.3 Interactions

Most interactive visualization systems utilized the common mouse or keyboard input. However, specific designs are needed to transfer the 2D commands received from a mouse into 3D space. Basic virtual widgets such as lines, circles or arrows were used to take command in early times [59]. Lens was a metaphor frequently adopted [8, 71, 72, 76, 80], usually used to denote a center and/or a transition region of the deformation. Probe [67] was another metaphor similar with lens but has more 3D sense. Controlling multiple virtual 3D widgets, such as lenses, at the same time was studied in [76], but was concluded as a difficult problem without general solutions. Touch screens were used to explore 3D scientific datasets and were proved to have more degrees of freedom than the mouse [44, 88]. 3D input devices have drawn much attention too because of the extra dimension offered. Looser *et al.* [56] gave users a handheld trackball to control the lens more easily. Besançon *et al.* [6] made a tablet into a tangible tool by tracking its location and angle to control the

clipping plan. Motion cameras were applied to take advantages of the inputs with direct 3D coordinates [40, 72].

Besides user input, the output of the result image can also be enhanced by special interaction tools besides regular screen monitors. Benefiting from the development of modern output devices, researchers are pursuing advantageous applications of 3D display. CAVE [24] tried to provide immersive experience for visualization by using multiple projection screens. Jackson *et al.* [40] adopted a desktop-scale stereoscopic display to help improve users' understanding of data. Head-mounted displays were used to provide Augmented Reality (AR) [56] or Virtual Reality (VR) [72, 75] images, with better depth cue rewarded by the immersive visualization.

2.3.1 Scientific Visualization on Touchscreen

Among all related works about human-computer interactions, more papers about performing visualization on touchscreens are presented here, since this topic is closely related to the third technique proposed in this dissertation. As one of the most typical tactile input devices, touchscreen has always been attracting researchers' attention. Many studies have been done to evaluate the effectiveness of touch input. Kin *et al.* [43] reported improved performance for certain tasks compared to mouse-based input. Besançon *et al.* [7] compared touch input with mouse and tangible input regarding to the precision and completion times. Yu *et al.* [88] designed experiments specifically focusing on scientific datasets and common sci-vis tasks, and found preference of the touch interaction over mouse on several tasks. Many designs have been proposed to serve the exploration of scientific datasets on touch-based workstations. Isenberg [38] surveyed several scientific visualization systems using large display surfaces, mainly by tables and walls. Lundström *et al.* [57] designed a medical

data visualization system on a large size touch table, in order to perform surgery planning and supported collaboration of multiple users. Klein *et al.* [44] designed a touch-based workstation to explore fluid dynamics data, with implementation of tools such as navigation, cutting plane operation, and seed placement. Coffey *et al.* [18] used touch input to navigate inside a virtual reality (VR) environment, and connected the touch control with the data through a miniature. Sollich *et al.* [66] focused the designs on time-varying scientific data, and proposed the concept of 4D bookmarks as the solution. Besançon *et al.* [6] combined touch input with tangible input together to perform exploration of three dimension datasets. In the end, there are also many studies that did not focus on touch input, but did provide the touchscreen as an alternative to traditional devices, such as [87].

2.4 Camera-Data Collision

Camera control is an important topic existing in many fields of computer graphics research and applications. Several surveys can be referred to [17, 41]. For this dissertation, I narrow down this topic, by discussing methods that can relieve degeneration effects caused by camera colliding with data elements in egocentric mode of rendering. Some of the occlusion management methods reviewed in Section 2.2 were also combined with camera control systems, but they mostly focused on exocentric viewpoints. A walkthrough scenario is common when rendering virtual buildings or puzzles. For this scenario, Chittaro and Scagnetto [16] made walls of corridors transparency by letting users click on the wall. Cohen-Or *et al.* [19] compared several culling method. Elmqvist *et al.* [27] applied dynamic transparency scheme working for pre-selected targets. Knödel *et al.* [45] created a sketch based walkthrough system to create illustrative cutaway renderings. Camera control is also a key factor for effective visualization of 3D scientific datasets. Volume clipping is a basic

operation for observing 3D datasets, and many immersive visualization systems used a clipping plan to avoid blocking the camera [6, 18, 35]. Guided tour can be done by viewpoint evaluation using information theory (such as entropy), and then forming a camera path by traversing the best viewpoints. Early works [31, 42, 69, 77] only focused on exocentric viewpoints, and the camera paths were restricted on an observation sphere surrounding the data domain. Ma *et al.* [58] examined the egocentric viewpoint quality for flow datasets, and created an optimal camera path inside the 3D data domain. In medical applications, such as colonography [15, 36], a preprocessing of the dataset could place restrictions on camera navigation, for example, trapping the camera inside certain body cavities.

Chapter 3: Virtual Retractor: An Interactive Data Exploration System Using Physically Based Deformation

3.1 Introduction

3D visualization has been widely applied to assist people's understanding of various types of complex datasets. The ability to thoroughly and intuitively explore data has been a major goal pursued by many researchers. To visually explore three-dimensional datasets, one significant challenge is occlusion management. While occlusion sometimes can provide helpful depth cue, it may also block the view and hide the information that users may be interested in. Many occlusion management methods have been proposed to allow users to see hidden structures of the data, such as filtering with transparency or using multiple viewpoints. Since many techniques remove the blocking objects, important context stored in them might be lost. Compared to the filtering techniques or multi-view approaches, deformation is a promising way for occlusion management and data manipulation. It usually discards less data elements compared to the filtering techniques, thus preserves more context near the focus. A well designed deformation scheme can make the resulting image look natural, for example deformation that follows a real physical procedure.

The way to deform data for visualization can be roughly classified as physically based and non-physically based [21, 23]. The former involves forces applied to an elastic mesh,

updates the vertices at each iteration, and repeats this procedure to propagate the changes. The latter adopts simpler space transformation which usually can be expressed by analytic functions. Compared to non-physically based deformation, physically based deformation provides users with more control of the process, by manipulating each node or edge of the mesh individually. Due to this advantage, it is widely used for simulations of physical behaviors. However, physically based deformation is much less exploited for scientific data visualization. When incorporating local data attributes into the nodes or edges of the elastic mesh, data features can directly affect the deformation. This characteristic provides users a unique way to understand the data, thus it has been applied in several Focus+Context visualization studies [70, 72, 81, 82]. An attempt to perform occlusion management was also made in [72], but the occlusion cannot be completely removed when the objects are distributed in certain undesired ways, mentioned as a limitation in the paper. Interactivity is another important factor for data exploration. Compared to passive exploration techniques which focus on visualizing specific predefined features, direct manipulation of data gives users more freedom to select and manipulate the parts of the data they want [29]. To fully embrace the freedom, good designs of interaction techniques are very much needed to provide a smooth and powerful control experience.

In this chapter, we introduce a new data exploration system that allows direct manipulation of data with unique cutting and splitting capabilities. The procedure is carried out by deforming a tetrahedral mesh that has a void in the middle to simulate the incision created by the cut. The splitting operation enlarges the void to allow users to observe the inner structure which was originally occluded. Our mesh is constructed with the local data properties taken into account, such as local data density or gradient. This setting has multiple advantages, such as deforming the data in a physically more natural way, helping

users' comprehension of certain data properties, as well as keeping interesting features of the data. Our interaction is realized by a metaphor of virtual retractor, because the result of the deformation has an effect similar to using a medical retractor in surgeries. The user control can be specified by not only a traditional mouse, but also 3D input devices such as motion cameras. Our output presents the intermediate steps with animations to provide users a better picture of the deformation. Besides a common monitor, the output image also works well on head-mounted displays such as a virtual reality headset.

We demonstrated multiple case studies to show that our system is a powerful tool for data exploration. Our model can be applied to different datasets and visualization techniques. In this work, we tested the model using particle datasets and volume datasets. Our case studies show that compared to the previous work, our system can fully remove occlusion, and during the deformation, our system preserves the shape of interesting features with a general framework serving multiple choices of data property. We also conducted one of the case studies with a domain expert, and received positive feedback to our system.

This work has been published in IEEE Pacific Visualization Symposium in 2017 [54]. An accompanying video has been uploaded to Youtube at https://youtu.be/74xm3UZqQjI, subject to possible changes of the hosting website.

3.2 Algorithm Overview

The goal of our design is to provide a powerful system to explore 3D datasets. The basic requirements include:

- RQ1: Completely remove occlusion.
- RQ2: Preserve interesting features during deformation.
- RQ3: Provide smooth interaction.

Before going into details, we first give a brief overview of our system, and show how it satisfies the three requirements. Our main manipulation on the dataset is to simulate a cutting and splitting procedure. So when there are obstacles between the camera and the target, we cut a seam on the data domain, then push the obstacles to the left and the right side. This action is very common in our life, such as unzipping and opening a suitcase, opening up a double gate, or doing retraction in a medical surgery, therefore the concept is familiar to general users.

The cutting action is simulated by letting users draw a line segment. Our virtual retractor is then created based on this segment, by treating it as an incision on the dataset cut by the user. The virtual retractor, as the magenta cube in Figure 3.1a, will automatically control the following deformation, such as its location and shape. In this way, the users' desire is responded. Users are also able to update this retractor, since trial and error is also very much needed for interactive data exploration. This will fulfill our Requirement RQ3, and more details of the design will be given in Section 3.4.

After the cutting action, the splitting action is simulated with a physically based elastic model on a tetrahedral mesh. This mesh covers the user-interested area given by the virtual retractor. Different from the fully-connected mesh used in the previous studies as in Figure 3.1d, our mesh is designed with certain nodes which are spatially close but not connected, as the nodes marked by red and blue in the cross section views Figure 3.1b and Figure 3.1c. These nodes correspond to the incision cut by the user. To observe the user's target as shown by the yellow dot in Figure 3.1b, the part of the mesh between the camera and the target will open up, as the mesh changes from Figure 3.1b to Figure 3.1c. Then the obstacles, as the cyan dots in the Figure 3.1b, will be displaced following the change of the mesh. This will make the target visible without occlusion, which fulfills the Requirement RQ1. By

setting the physical properties of mesh according to the data attributes, our model can make the interesting features of the data less distorted during the deformation. On the contrary, regions with less interest are deformed more to make space for the interesting features. In this way, the Requirement RQ2 is fulfilled. The detailed algorithm is introduced in Section 3.3.

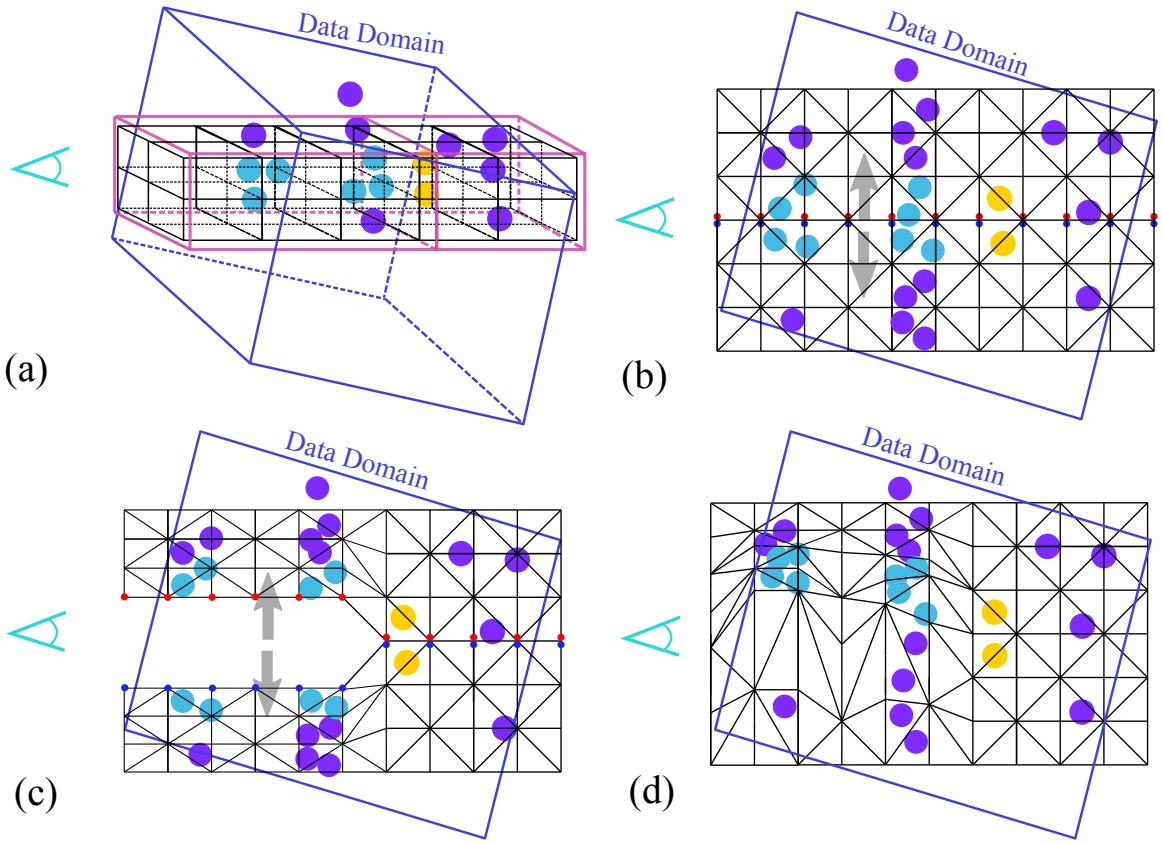
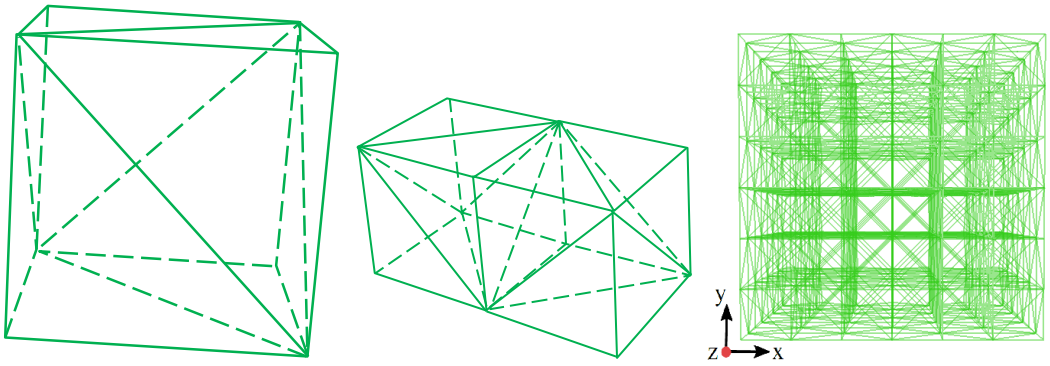


Figure 3.1: System overview. The dots represent an example particle dataset. The yellow dots are the user's target occluded by the cyan ones. The magenta cube represents our virtual retractor. The mesh is shown in black. (a): 3D view of the system. (b)(c): Cross section views of the system. (b): Before deformation. (c): After deformation. (d): The fully-connected mesh used in previous work [72], for comparison.

3.3 Deformation Algorithm

3.3.1 Mesh Construction

Our physically based deformation is performed on a 3D tetrahedral mesh. This type of mesh contains a stack of connected cubes. Each cube is further divided into five tetrahedrons, as shown in Figure 3.2a. The neighboring cubes must be divided in certain directions, as the way in Figure 3.2b, such that for each pair of adjacent cube faces, the triangles on these two faces match with each other. By stacking up a large amount of such tetrahedralized cubes, a mesh can be formed to cover a large 3D space, as in Figure 3.2c.



(a) 5 tetrahedrons. (b) Neighboring cubes. (c) A tetrahedral mesh.

Figure 3.2: (a) Tetrahedralization of one cube is to decompose it into 5 tetrahedrons. (b) Tetrahedralization of neighboring cubes in correct directions. (c) A tetrahedral mesh based on regular grid.

In order to simulate a splitting procedure, the mesh should have a certain structure like an incision, thus the tetrahedrons at the two sides of the incision can move away from the incision like being split. A fully-connected regular 3D mesh, as shown in Figure 3.2c, was widely used in previous researches. Such design of mesh can stretch or compress the space where it is embedded, therefore the occlusion can be relieved in certain situations [72],

showed as the left cluster of cyan dots in Figure 3.1d. But since no cutting action has been simulated, this type of mesh cannot break the continuous space to create any void. As a result, the occlusion cannot be removed for dense obstacles, such as the right cluster of cyan dots in Figure 3.1d.

To achieve our goal, we designed a special mesh, which uses some close but not connected nodes to create a cutting incision, as in Figure 3.3c. Starting from a regular tetrahedral mesh as Figure 3.2c, first let us denote each node p by their 3D indices (x_p, y_p, z_p), where x_p, y_p, z_p are integers within the ranges $[0,X], [0,Y], [0,Z]$. At each layer of the z direction, as the cross section view in Figure 3.3a, we find the nodes with the middle y index value, except the two nodes on the two sides in x direction, as the blue dots shown in the figure. Then we duplicate these nodes at their original positions, as the red dots shown in Figure 3.3b, and redefine the connected edges and faces accordingly. These colored nodes are used for simulating the incision, and we denote them by a set I, i.e.,

$$I = \{p \mid y_p = Y/2 \ \wedge \ x_p > 0 \ \wedge \ x_p < X\} = \{I_+ \cap I_-\} \tag{3.1}$$

where the sets I_+ and I_- represent the upper (red) dots and lower (blue) dots in Figure 3.3b respectively. Note that the nodes in I_+ are initialized at the same positions with the corresponding nodes in I_-, but here we draw them with a small distance for a clearer illustration.

The position and the size of the mesh is automatically decided by the virtual retractor, as shown in Figure 3.4a. The retractor has a cuboid shape. We define three faces on the cuboid: front face, center face, and back face. They are three quadrilaterals of the same shape. We describe the quadrilaterals by the major direction $\vec{r}$, the length of the major axis l, and the width w. The front face and the back face are placed to tightly cover the data domain. The center face is placed right in front of users' interested area, so it separates the

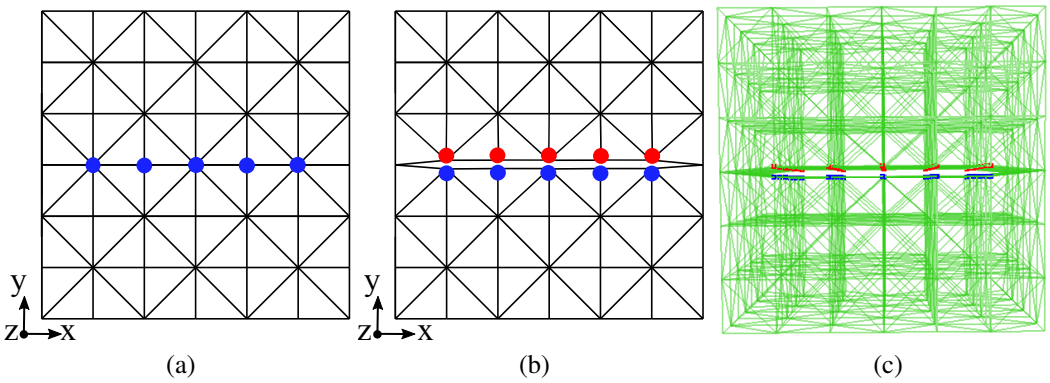


Figure 3.3: Our mesh design. (a) is a cross section from the previously used mesh in Figure 3.2c. (b) is a cross section from our mesh design. It has an incision created by duplicating the blue nodes in (a). (c) gives a 3D view of our mesh design.

deformation region with the interesting area. The center c of the center face is defined as the center of our retractor. When created, the view-dependent retractor always faces the camera, which means the three faces are perpendicular to the direction from the center c to the camera, denoted by $\vec{v}$. The mesh lays in a way that its boundaries exactly overlap with the boundaries of the retractor. Its Z direction points from the back face to the front face, and X direction is parallel to the major direction $\vec{r}$.

By placing the mesh in this way, the incision nodes in the set I are coplanar with the camera. With our mesh design, there is no edge in the space between the nodes in I_+ and I_-. If we make the nodes in I_+ and I_- move in opposite directions, the empty space between them can be enlarged. Since the opened empty space does not correspond to any region in the original data domain, no data objects will be presented there. Therefore, the occlusion can be completely removed, and a passage is created for the sight line. In this way, our Requirement RQ1 is achieved.

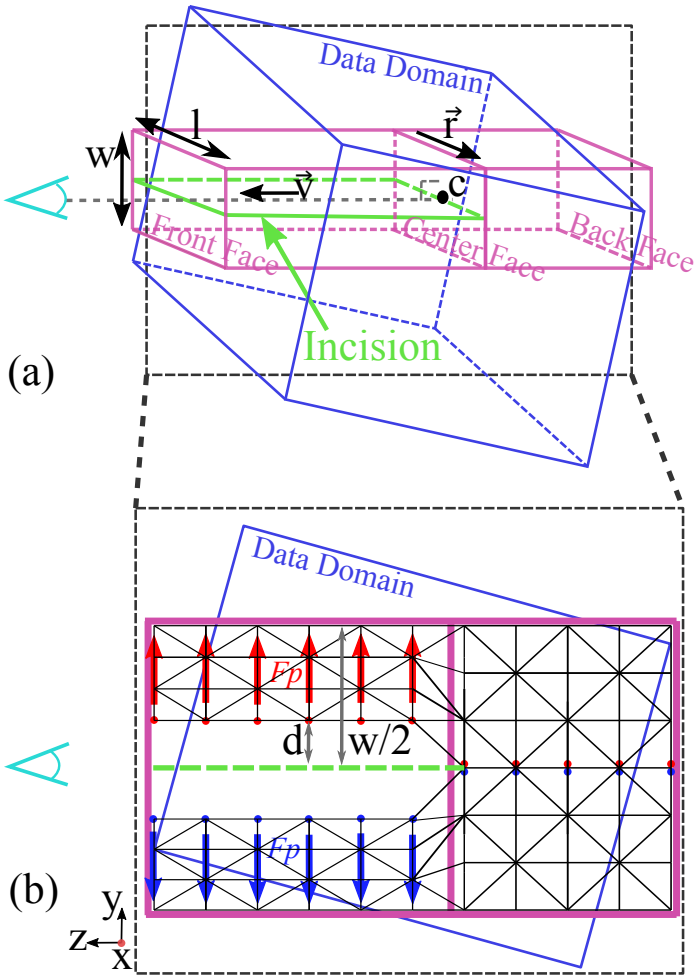


Figure 3.4: (a) Components and parameters of the virtual retractor. (b) A cross section of (a) at the position of the grey frame. The forces applied on the nodes are illustrated as red and blue arrows.

3.3.2 Mesh Properties Setting

In this section we introduce how to set the physical properties of the mesh. As long as the necessary mesh properties are well set, we can apply any proper physically based simulation model to deform the mesh. In our experiment, we applied a projective dynamic method proposed by Wang [79], which is a GPU-friendly parallel solution that can achieve

impressive image quality while keeping real-time interactivity. To use this model, we need to set the forces applied on mesh nodes for moving them, and the stiffness of each tetrahedron to serve as a resistance to the movement.

The forces should be designed to create movement of splitting from the incision. To split something in real life, a person can put the two hands or a real retractor into the incision, and push towards the two sides. Inspired by this real world example, we apply forces perpendicular to the incision, to the nodes forming the incision, i.e., the nodes in the set $I = \{I_+ \cap I_-\}$ defined in Equation (3.1). The magnitude of the forces are initialized uniformly, but decreases when the nodes move away, based on the distance to the incision. In stead of applying the force to all nodes in I, we only apply it to the nodes no deeper than the depth of the incision, which is defined by the depth of the center face of the virtual retractor. If the depth is denoted as c_z, we have the force F_p defined as

$$F_p = \begin{cases} f \times h(d) \times (0,1,0) & \text{if } p \in I_+ \ \wedge \ p_z > c_z \\ f \times h(d) \times (0,-1,0) & \text{if } p \in I_- \ \wedge \ p_z > c_z \\ (0,0,0) & \text{otherwise} \end{cases} \tag{3.2}$$

where p_z is the z coordinate of the node p. Note that both p_z and c_z are regarding to the local space of the mesh. f is an adjustable constant for users to control the overall force magnitude. $h(d)$ is a function that controls the reduction of the force, using the distance d from the node to the incision, as shown in Figure 3.4b. Our design of $h(d)$ aims to be continuous at both the zero and the first order of derivative. It should also involve the retractor width w, since it is used by users to specify the desired expansion of the deformation region. The following function satisfies our goal, which first normalizes d by half the width $w/2$ (as in Figure 3.4b):

$$\begin{aligned} &d_n = d/(w/2) \\ &\text{h(d)} = \begin{cases} 1 & d_n < 0.45 \\ (cos(10\pi \times (d_n - 0.45)) + 1)/2 & 0.45 \leq d_n \leq 0.55 \\ 0 & d_n > 0.55 \end{cases} \end{aligned} \tag{3.3}$$

Next we need to set the stiffness of each tetrahedron in the mesh. The most straightforward setting is to use a uniform stiffness for all the tetrahedrons in the mesh. But first, this is not natural, because for a the real object with uneven density distribution, it is not expected to manifest the same resistance to external forces everywhere. Secondly, using uniform stiffness cannot fully utilize the benefit of the physically based mesh. When all the tetrahedrons are behaving homogeneously, the deformation becomes more predictable, and may reduce to simple transformation that can be summarized with a few analytic functions. Therefore the result it obtained has little merit over non-physically based methods. At last, different regions usually contain different amount of features. Treating all the regions uniformly cannot give priority to important regions containing more features over other regions.

To accomplish our Requirement RQ2, we apply the principle of the data-driven mesh by setting the stiffness of each tetrahedron according to a local data attribute of the region it covers. This local attribute represents certain interesting features that users want to preserve during the deformation. Therefore we set the stiffness of the tetrahedron monotonically increasing with the value of this attribute. In this way, the interesting regions will have high stiffness, and become harder to be deformed than other regions. On the contrary, the regions with less interest will be deformed more easily. As a result, the shape distortion caused by the deformation is mainly distributed at the less important regions, while the shapes of the interesting features are preserved. In summary, for each tetrahedron, we have the stiffness g defined as:

$$g = g_0 + \alpha \times A^c \tag{3.4}$$

where A represents a local data attribute selected by users, and g_0, α and c are all constant parameters. g_0 is used as the minimal value to avoid the stiffness being too low. The choices

of these parameters are data dependent, and for this reason we used empirical values for each tested dataset. Previous Focus+Context visualization works also considered about data-driven meshes. As an improvement, our stiffness-setting framework is more general for various types of data exploration tasks. It can theoratically incorporate the previous data properties used for mesh setting, including cube classification [81], voxel color [82], cube entropy [70], and particle density [72]; it works for both particle datasets and volumetric datasets used in our study; it allows users to choose the data attribute by their own desire, or even extend it for new type of feature definitions. In this work, we have tested three strategies of stiffness setting, which are *density based*, *transferred density based*, and *gradient based*.

The *density based* setting uses the density of the region covered by each tetrahedron. The density of a tetrahadron is computed by dividing the mass m by the volume size V. For a mass-less particle dataset, we treat the number of particles inside each tetrahedron as the mass. For a volumetric dataset, the sum of voxel values inside the tetrahedron is used as the mass. Therefore the attribute A in Equation (3.4) is defined as:

$$A = \frac{m}{V} \tag{3.5}$$

This setting can aid users to have a better understanding of the density distribution over the space during the animation of deformation. It is the most natural one among our three choices, because it is like when people are splitting an uneven object using hands in real life, they can feel the differences of the resistance from different parts of the object.

The *transferred density based* stiffness setting is more flexible when exploring a volume dataset. For certain imaging techniques, the voxel value may not be a direct representation of mass or density. Also in some situations, users may be more interested in regions with certain voxel values that are not necessarily very high. For these cases, we define a transfer function $f()$ for every voxel p to compute a transferred value $f(p)$ from the original voxel

value. Note that although with a similar name, this transfer function is not used to compute a color for rendering, but is rather used to compute a *density* of users' interest. Then to compute the stiffness g for a tetrahedron T in Equation (3.4), A is defined as:

$$A = \frac{\sum_{p \in T} f(p)}{V} \tag{3.6}$$

. By designing a specific transfer function, users can better maintain the original structure of certain voxel values of their own choice, instead of being limited to only voxels with high values.

The *gradient based* stiffness method tries to consider the steepness of data values in the deformation. It uses the magnitude of the gradient of each voxel to compute the stiffness. For a tetrahedron T, the chosen attribute A in Equation (3.4) is set by:

$$A = \frac{\sum_{p \in T} \|\nabla p\|}{V} \tag{3.7}$$

. This setting gives more firmness to regions with more data value changes, serving the situations when such regions are considered to be more informative. Both the transferred density based and gradient based stiffness methods may not be as veridical as the density based method. But by preserving the interesting features decided by users, they provide extra assistance to explore and understand the dataset.

A few more details are given below to complete the mesh properties setting. To restrict the mesh deformation within the designated region, we fix the positions of the nodes at the boundary of the x and y axes. It means, the nodes $\{p = (x_p, y_p, z_p) | x_p = 0 \vee x_p = X \vee y_p = 0 \vee y_p = Y\}$ will not be moved during the deformation. Although mostly not needed, a stop command can be given to the system to end the deformation, in case the iterations of the computation encounter local oscillations.

Besides controlling the tetrahedron stiffness by data attributes, there may also be other ways to utilize the physically based mesh by setting other mesh variables appropriately. For example, setting different forces at each node may also achieve some useful effect. But this is a non-trivial question and needs more study to make a solid conclusion. Also applying different forces is further from our metaphor of virtual retractor, therefore it is treated as beyond the scope of our current study.

3.3.3 Data Deformation by Mesh

With the above settings, our system adopts the projective dynamic method to evolve the mesh a little at each iteration step. The iteration continues until the deformation converges as the force and the stiffness come to a balanced state. Meanwhile at each iteration, we compute and visualize a deformed dataset according to the mesh deformation. The visualizations of these deformed datasets form an animation, which helps users comprehend the whole deformation process. In this subsection, we introduce how the data is deformed according to the deformation of the mesh.

After the deformation starts, at any time step we have a new deformed mesh, as well as the original mesh. Each tetrahedron T occurs in both meshes, but with its vertices being at different coordinates. For a given point p, we define the function to compute its barycentric coordinate regarding to T as $b_0 = \beta_0(p)$ for the original T, and $b_1 = \beta_1(p)$ for the deformed T.

For a particle dataset, given an input particle p_0, before the deformation, we first find the tetrahedron T it belongs to and its barycentric coordinate b_0. After the deformation, we place this particle at the new position which will have the same barycentric coordinate

regarding to T, i.e., $b_1 = b_0$. Thus its new position p_1 can be computed as $p_1 = \beta_1^{-1}(b_1) = \beta_1^{-1}(b_0) = \beta_1^{-1}(\beta_0(p_0))$, where $\beta_1^{-1}()$ is the inverse function of $\beta_1()$.

For a volume dataset, transforming every voxel using the above method will cause an uneven distribution of voxels, which may reduce the quality of ray casting rendering. Instead, we apply a basic rasterization method introduced in [34]. This method creates a new volume with the same resolution of the original volume. For each tetrahedron T in the deformed mesh, we first compute its bounding box $[X_1, X_2], [Y_1, Y_2], [Z_1, Z_2]$. Then the bounding box is scanned, and every voxel p_1 of the new volume covered in the bounding box will be processed. After computing the barycentric coordinate $b_1 = \beta_1(p_1)$, we check whether the voxel is inside T. If so, we find a position p_0 in the original mesh, whose barycentric coordinate regarding to T is the same with p_1, by $p_0 = \beta_0^{-1}(b_0) = \beta_0^{-1}(b_1) = \beta_0^{-1}(\beta_1(p_1))$. Then we sample a value at p_0 in the original volume, and assign this value to the voxel p_1 in the new volume. The gradient of the new volume will also be computed after the new volume is constructed.

3.4 Interaction

An effective data exploration system needs intuitive user interaction designs, to guarantee that users can observe their targets easily and accurately. In this section, we describe our design that allows users to interactively define and modify the deformation through the metaphor of the virtual retractor. We show our interactions not only on regular 2D devices such as the mouse and screen, but also on 3D input and output devices. We discuss both the advantages and disadvantages of these devices in our system.

3.4.1 User Input Operations

The task of the input system is to receive users' instructions of creating and updating the virtual retractor. Since the retractor automatically defines the mesh, it conveys the users' detailed requests of the deformation to the system.

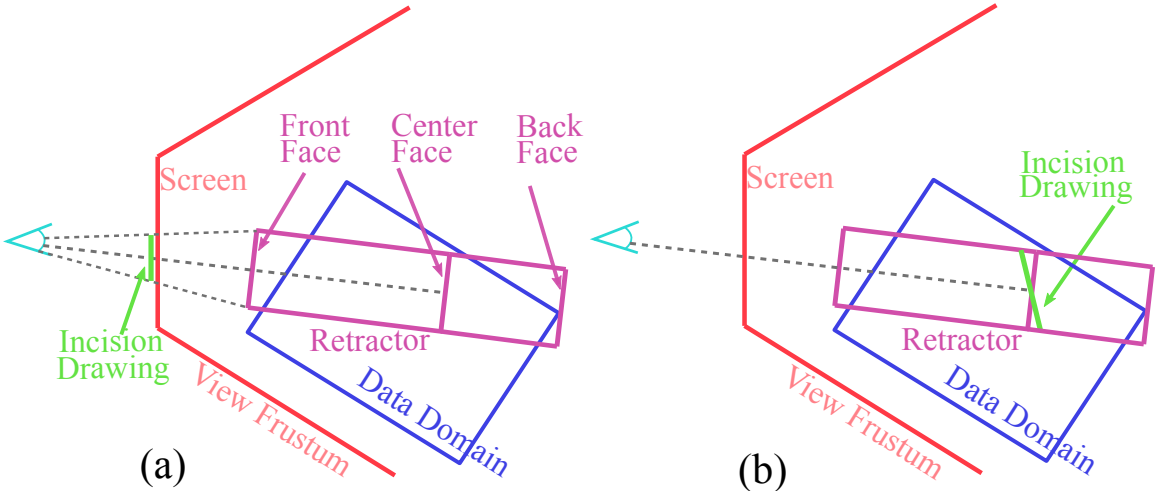


Figure 3.5: (a) Creating a retractor by drawing a line segment on screen. (b) Creating a retractor by drawing a line segment in 3D space.

First we introduce how to create and update a virtual retractor using a mouse. The mouse is a 2D input device which only provides a 2D screen coordinate directly. To simulate the action of cutting an incision, users can draw a line segment on screen, as the green line in Figure 3.5a. However, from the line segment on screen we cannot specify a line segment in 3D space because of lacking depth information. We interpret it as the projection of the center line of the front face, which is the closest face to the camera. In this way, when the retractor is created and rendered, the projection of the actual incision can have maximum overlapping with this 2D segment. With this interpretation and the information of the data

domain, we can compute the position of the front face and the back face, but we still do not know the position of the center face, since it can be anywhere between the front face and the back face. Therefore, we need to preset a default depth of the center face. Now with one more default setting of the width w, all the parameters in Figure 3.4a of the retractor can be computed. One disadvantage of creating the retractor in this way is, when users are drawing the 2D line segment, it may not be easy to map this segment on screen to the 3D position in the data domain. Therefore creating a satisfying retractor at the first try may not always be easy. Also the necessity to preset the depth of the center makes this more difficult. Besides the cut placement, most updating operations have to be specified in 2D screen space too. For example, translating the retractor in 3D space has to be done in two steps: first dragging the retractor in the plane parallel to the screen, and then changing the depth. A complete list of interaction operations is shown in Table 3.1.

Next we use a Leap Motion [84] as an example to show how our system makes use of motion camera devices to take input. The Leap Motion can capture the positions of users' fingers. Its greatest advantage compared to a mouse is that it specifies 3D positions of the gesture signals, thus it has the ability to interact with 3D objects more intuitively. To utilize this, users can move the hand to create a cut by drawing a 3D line segment in space, as the green line in Figure 3.5b. This segment will be used to directly decide the center face. It may need a little adjustment to become exactly perpendicular to the sight line. Nevertheless, drawing in the object space gives users direct sense of the depth of the cut they are creating. After the creation of the retractor, updating the parameters can also be conducted in 3D. For example, moving the center c can be done directly by moving a hand freely, and we do not need an extra operation to adjust the depth like when using a mouse. To distinguish the user commands from casual hands movements, we utilize the finger pinch action as a

Table 3.1: The interaction techniques to create and update the virtual retractor.

Operations	Mouse	Motion Camera
create a retractor	draw a 2D segment	draw a 3D segment
move horizontally	drag the center	drag the center with fingers
change depth	scroll wheel	
change size	drag the boundary with mouse	drag the boundary with fingers
change orientation		
change force	drag a slider on panel	three fingers pinched & move

replacement of mouse button pressing. Table 3.1 gives more details about operation designs, and a comparison between the 2D and 3D interaction devices. One disadvantage of the motion camera occurs when we draw a symbol of the hand, which indicates its current location. In some cases the hand symbol may be occluded by the data, which is not a problem when drawing the mouse cursor on screen. In our implementation, we increased the transparency of the data to make the hand symbol visible. But still it may not fit certain datasets if the extra occlusion management of the hand symbol is too distracting. Besides, for users without prior experience of motion cameras, additional training needs to be given.

Besides controlling the virtual retractor, general interactions to control the view frustum, such as zooming, panning, and rotating, are also implemented, but will not be discussed in details here. Together with the retractor management, our Requirement RQ3 is fulfilled.

3.4.2 Immersive Output by Head-Mounted Display

Besides the traditional monitor screen, in our study we also explored the use of head-mounted display (HMD) to display our image as virtual reality (VR). The experiment was conducted with an Open-Source Virtual Reality (OSVR) Hacker Dev Kit 1.3. The immersive 3D environment is created by rendering two separate images, one for each eye of the user.

With the help of the headset, the two images are fused to create a stereoscopic visualization. The two images are generated using slightly different modelview matrices, since the two eyes are located slightly differently. The creation of our retractor is computed using one matrix of them without noticeable artifacts. Using the VR visualization, the two images create the parallax, which provides a better perception of the relative depth of nearby objects. This not only benefits the output, but also make the 3D interaction using motion cameras more straightforward. Since users need to wear a headset and cannot see things outside the headset, mouse or keyboard control can be difficult. In this case, the motion camera is a better option. Besides, the head tracking information can provide extra degrees of interactivity. The position and orientation of the HMD can be used to control the view frustum of the rendering.

3.5 Case Study

In this sections, we provide multiple case studies to demonstrate the benefits of our system. We show how our model can cut and split the occluding data, and compare it with previous methods. We compare several mesh stiffness settings regarding to the preservation of features. We demonstrated the system to a domain expert and received positive feedback.

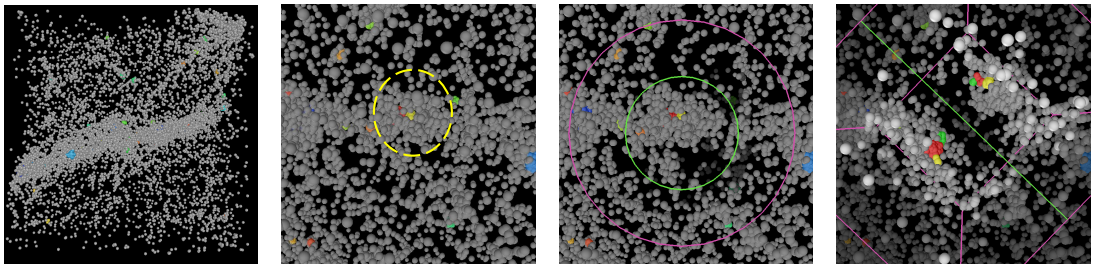
In the figures provided in this section, we draw the front face and the center face of our virtual retractor in magenta together with the data, to denote the deformation region. The back face is omitted. We draw the incision as in Figure 3.4a, which is projected as a green line on screen. These options can be easily changed by users in our system.

3.5.1 Complete Occlusion Removal

The first dataset we tested is a cosmological simulation dataset [2]. It simulates the evolution of the universe starting from its birth. At each time step, the dataset contains

a large amount of particles representing dark matters, as well as many halos containing different numbers of particles. The spatial distribution of particles and halos is one of the most interesting properties that users want to see. However, rendering the large number of particles and halos would result in severe occlusion. We select a region containing 16,829 particles and render them as spheres, as Figure 3.6a. If a particle belongs to a halo, it is rendered with a color uniquely assigned to this halo. If a particle does not belong to any halo, it is colored grey. In Figure 3.6b we show a region marked by a yellow circle in a closer view. This region contains heavily clustered particles, and the inner structure of this cluster is occluded. We can only see two halos in red and yellow from the surface. Figure 3.6c shows the application of the previous method in [72]. This method places a uniform tetrahedral mesh, as in Figure 3.2c, over the domain. By stretching this mesh, the cluster is enlarged and the particle distribution does become looser. But since the nearby tetrahedrons will always remain connected during the deformation, the obstruction cannot be broken. Thus we can only see limited parts of the two halos. Figure 3.6d shows the result of applying our method. We can see that by adding an incision, the cluster is separated into two halves, and the two parts are repositioned with enough distance. Therefore we can clearly see the inner structure of this cluster, where interestingly a new green halo is revealed, and we also get a better sense of the relative positions and sizes of these three halos.

The second dataset is a simulation dataset of the viscous fluid, using the Finite Pointset Method (FPM) [3]. It simulates the dissolving process when adding salt to the top of a cylinder container filled with water. We render 14,784 particles by spheres as in Figure 3.7a. Each particle is colored by its concentration value, and clustered particles with high concentration values can form a structure called viscous finger, which is of interest but may be occluded by other particles. We zoom in towards a region where the particles on the



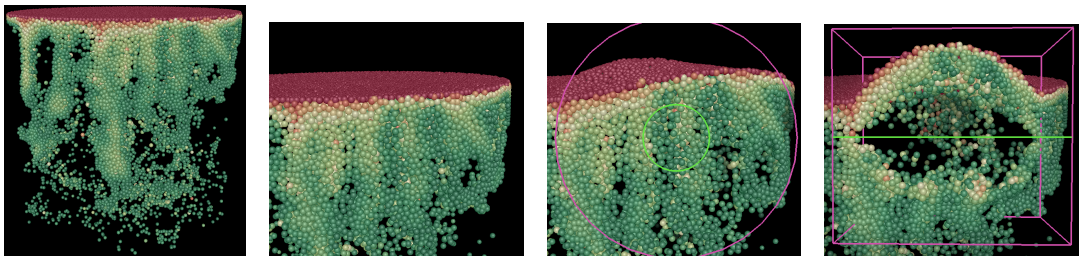
(a) Overview of cosmology dataset rendering. (b) Original visualization. (c) The method in [72]. (d) Using our method.

Figure 3.6: Exploring the cosmology dataset. Each sphere represents a simulated particle. A particle is colored grey if it does not belong to any halo. Otherwise, it is colored by the ID of the halo it belongs to. From (b), we can see two halos in the cluster circled by the yellow curve. In (d), one more halo in green is exposed by our method. As a comparison in (c), the deformation technique without cutting [37] cannot break the cluster or reveal the hidden green halo.

outside occlude the vision to observe the inner structure, as in Figure 3.7b. In Figure 3.7c, applying the previous method in [72] cannot solve this issue. Similar with the last case, when the obstacles are too dense, just stretching or compressing the tetrahedrons is hard to create an empty space to see through. Figure 3.7d shows that our method successfully displaces the obstacles, and reveals a hidden cluster behind the obstacles, as the cluster formed by dim particles in the figure. As a comparison, our mesh is designed with a void, thus it can deal with the case when the particles are very dense.

3.5.2 Feature Preservation

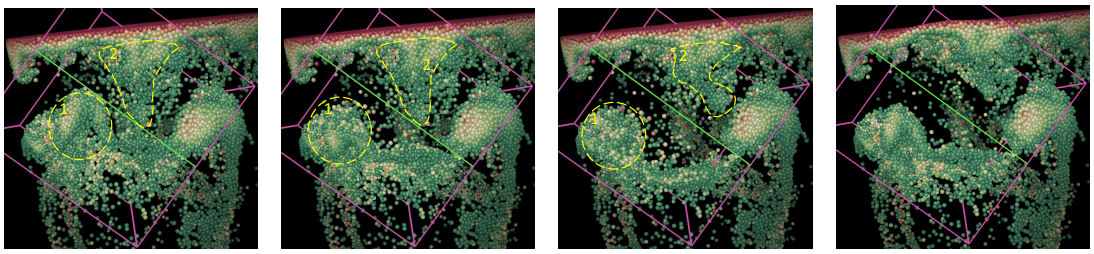
First we use a different ensemble run of the viscous fluid dataset, to show that our density based mesh setting can preserve the spatial structure of the high density region during deformation. This demonstration uses 19,483 particles. Figure 3.8a-3.8c show the deformed datasets with increasing forces applied. We circle two clusters by yellow and mark them



(a) Overview of the viscous fluid visualization. (b) Original visualization. (c) The method in [72]. (d) Using our method.

Figure 3.7: Exploring the viscous fluid dataset. Particles are colored by the concentration values using RdYiGn color map, where red is used for high value and green for low value. A hidden cluster is revealed by our method in (d), but not by the method in (c).

by 1 and 2, and their shapes are well kept during the deformation. When a small force is applied as in Figure 3.8b, cluster 1 is moved first, since there is larger empty space on its left and top. The displacement of cluster 1 makes use of the empty space, to minimize the distortion of the cluster itself. When the force is increased in Figure 3.8c, cluster 1 is flipped upwards, and cluster 2 is moved into the empty space on its right. Our stiffness setting makes the tetrahedrons that include denser particles harder to deform, and the tetrahedrons covering sparser particles more likely to deform. When the obstacles are pushed to the side, some space has to be sacrificed, and in this way the less dense space will be confiscated first. Therefore, the dense space, which contains more important features in this case, can be kept well. Figure 3.8d gives a comparison of using uniform density to set the mesh stiffness. In this kind of deformation, no local data properties have been incorporated into the mesh. Therefore the dense space and the empty space are deformed without difference. It causes the interesting clusters equally compressed together with less important regions, and the empty spaces are less utilized.



(a) Original view (no force). (b) Applying a small force. (c) Applying a larger force. (d) Using uniform stiffness.

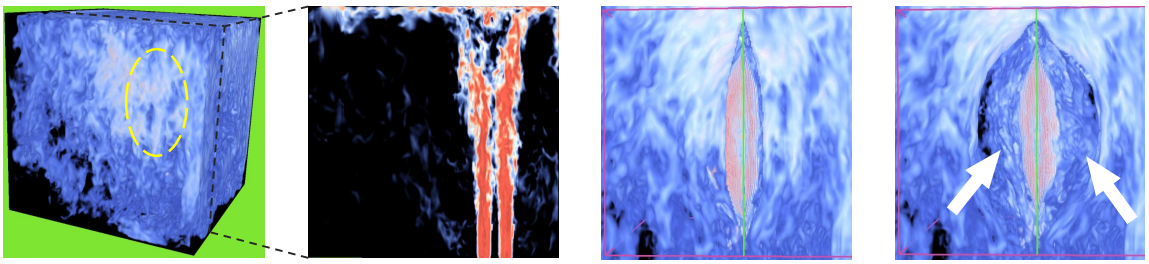
Figure 3.8: Examples showing the effects of the density based stiffness setting. Two clusters are circled by yellow dashed curves. (a)-(c) show the deformation using increasing forces. The shapes of the clusters are well preserved, and the empty space near them are compressed to make space for the clusters. (d) shows a comparison of using uniform stiffness setting, which compresses the clusters and the empty space in the same way, and destroys some shape features.

Next we test the applicability of our transferred density based stiffness setting using a volumetric NEK dataset. This dataset is a vector dataset with a $256\times256\times256$ dimension, generated by an eddy simulator using the Nek5000 code [32]. It shows the movement of two streams of coolant entering a machine. We rendered the vector magnitude using direct volume rendering, as shown in Figure 3.9a. The data values on the surface are in middle range and occlude the inner domain. From the cross section view in Figure 3.9b, we can see that inside the domain, there are two cylinder regions with high valued voxels, as well as large area of low value voxels. At the region circled by yellow in 3.9a, we apply our method with the density based stiffness setting to split the surface and look for the cylinder region, as in Figure 3.9c. Since under the surface there are regions with low voxel values, the tetrahedrons there have lower stiffness and are easier to be deformed than the tetrahedrons covering the surface. So the deformation pushes the incision surfaces inside and make them not visible. This behavior is authentic and helps users feel the emptiness under the surface

even without seeing it directly. But in other cases users may want to see the data on the incision surfaces. Then we can define a transfer function $f(p)$ to compute the transferred density, and set the stiffness by the Equation (3.6). We want to give lower stiffness to the tetrahedrons on the surface covering middle range values, and give higher stiffness to the tetrahedrons inside with low and high value voxels. So we define $f(p)$ as:

$$f(p) = \begin{cases} (0.2-p)/0.2 & \text{if } p < 0.2 \\ 0 & \text{if } 0.2 \leq p < 0.5 \\ (p-0.5)/0.5 & \text{if } 0.5 \leq p \end{cases} \tag{3.8}$$

where p represents the voxel value after being normalized to $[0,1]$. The result is shown in Figure 3.9d. It is generated using the same view, mesh and force with Figure 3.9c, except for the mesh stiffness. The two white arrows point to the two incision surfaces which are visible now.



(a) Overview of the NEK dataset.

(b) A cross section showing the inner structure of the dataset.

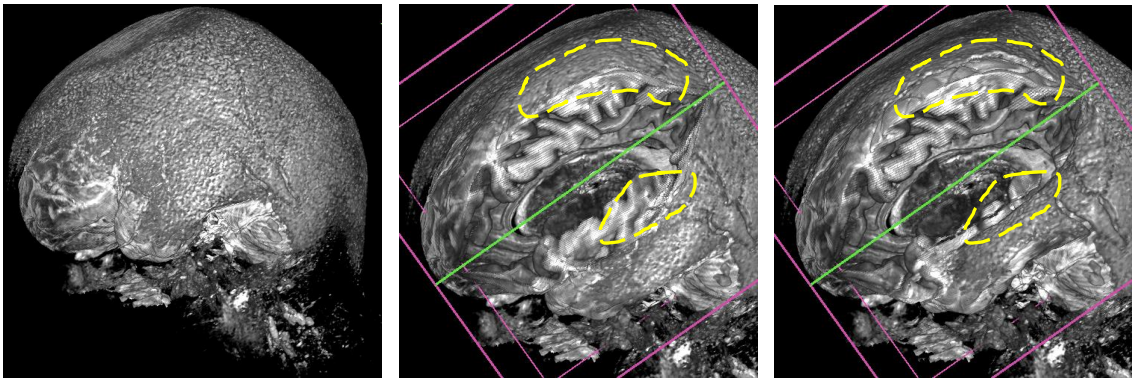
(c) Using density based stiffness setting.

(d) Using transferred density based stiffness setting.

Figure 3.9: The example showing the effect of the transferred density based stiffness setting. The dataset is the vector magnitude of NEK dataset. The warm-cold color map is used, where red represents high value. (b) is a cross section view of (a). (c) and (d) are enlarged views of the deformation at the region circled by yellow dashed curve in (a). (c) and (d) are generated using different stiffness. The white arrows in (d) mark the position of its difference with (c).

At last we use an Magnetic Resonance (MR) volume dataset to demonstrate our gradient based mesh stiffness setting. The dataset is the T2 image from a brain MR scan,

released by the Human Connectome Project (HCP) [1] funded by NIH. The volume size is 320×320×256. Figure 3.10a shows the original rendering. The face of the head was removed by the publisher to hide the identity. Figure 3.10b and 3.10c show how we use our method to cut an incision to explore the inner structure of the volume. In both images the same frustum, mesh and force are applied, except the mesh stiffness as the only difference. In Figure 3.10b, the stiffness is set by the density. Since there are many low value voxels between the skull and the cerebral cortex, these two parts are pushed close to each other. In Figure 3.10c, the stiffness is set by the gradient. The transition region from the skull to the cerebral cortex has large difference in the voxel values, so tetrahedrons covering this region have high stiffness. So these two parts are well separated for users to observe. The yellow curves in both images highlight the difference.



(a) The original image.

(b) Deform with density based stiffness.

(c) Deform with gradient based stiffness.

Figure 3.10: The example showing the effect of the gradient based stiffness setting. The dataset is an MR volume dataset. (b) and (c) are generated using the same configurations except for the stiffness. Differences of (b) and (c) can be found in regions marked by yellow dashed curves.

3.5.3 User Feedback

We conducted a case study of our system using the cosmological dataset with a domain scientist. The domain scientist is a PhD candidate in astronomy area with excellent research experiences and publications. He had given suggestions to our system throughout the development of our technique. He had not used a deformation related visualization technique before, so to the best of his knowledge, he assured the novelty of our system in his area. He agreed that our system could successfully remove the occlusion, which was very helpful for observing the halos and surrounding particles, as well as understanding their reciprocal activities. With the animation of deformation, he could easily recover the original shape of the deformed data. As for the tetrahedron stiffness setting, he mentioned that when the context was not important, he might prefer the simple uniform stiffness setting. On the other hand, he agreed that when the context was interesting, he preferred our density based stiffness setting. He raised a case when our system is very favorable: when the deformed region contained a halo which was highly related to the focus target, being able to preserve the original shape of this halo is very satisfying. He was very excited about our motion camera and immersive headset devices. The enhanced depth cue benefiting from the virtual reality visualization impressed him very much. However, he worried about the learning procedure of the motion camera could be too laborious for some users, but he was sure that after getting familiar with the operations, the data exploration could be effective and fun.

The domain expert also gave some suggestions about our future system design. Although our deformation was very valuable, he also hoped to see more techniques to be added to our system. For example, if adding operations such as clicking to show particle attributes, halo snapping, and automatic viewpoint selection, the system could be more useful in the domain research. He was also eager to see the application of our system to time-varying

data. However, we all agreed that combining the animation of deformation and the data throughout time could be challenging, but it could be an interesting future work.

3.6 Performance

In this section, we show that our system can run in real time, by testing the frame rate. The test was performed on a desktop computer, with an Intel(R) Core(TM) i7-2600 3.4GHz CPU, 16GB RAM, and an NVIDIA GeForce GTX 980 Ti graphics card. For the cases shown in Figure 3.6d, 3.7d, 3.8c, 3.9d, and 3.10c, we achieved frame rates (FPS) of 34.2, 29.6, 26.2, 28.5, and 20.4 respectively, at an image resolution of 800×800.

We further repeat the cases shown in Figure 3.8c and Figure 3.9d, with different number of tetrahedrons in the mesh, and different data sizes, to show how they affect the performance. Figure 3.11a shows that for both the particle datasets and the volume datasets, the mesh complexity has small influence on the performance. The first reason is that the computation on each tetrahedron can be highly parallel. The second reason is that the rendering took longer time than the deformation. Volume datasets appear to be a little more sensitive to the mesh size than the particle datasets, because the rasterization procedure (described in Section 3.3.3) involves more mesh-related computation. Figure 3.11b and 3.11c show how the performance reduces as the data size increases, because both the computation and the rendering need more time for larger data.

When the virtual retractor is updated, the mesh needs to be reconstructed. The mesh construction only needs to be done once before the deformation, so the computation of frame rate does not include the time cost of this step. Nevertheless, the mesh construction is also very fast after being parallelized on GPU. Among all the cases demonstrated in Section

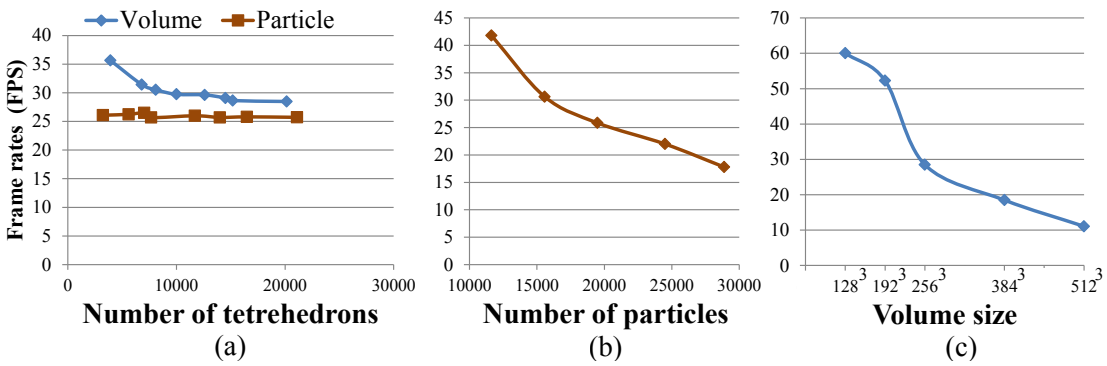


Figure 3.11: (a) Frame rate (FPS) when changing the number of tetrahedrons in the mesh. (b) Frame rate for different sizes of particle datasets. (c) Frame rate for different sizes of volume datasets.

3.5, the time costs to build the mesh are all less than 0.1 second. So updating the mesh does not hurt the performance either.

3.7 Conclusion

We presented a new data exploration system that adopts a physically based deformation method to manipulate scientific datasets. Users can simulate cutting on the dataset, then the dataset can be split to remove the occlusion. We showed the advantages of our physically based mesh, which sets the stiffness of the tetrahedrons according to the data features users are interested in. While the deformation is used to manage the occlusion problem, the data-driven mesh setting helps to keep the shape of features during the deformation. Our system also provides interactive designs to help users better control the exploration process. We have applied our system on exploration tasks for particle rendering and volume rendering.

One limitation of our system is that it is restricted by some general drawbacks of the physically based simulation, such as relatively high computing cost and difficulties in parameter tuning. This trade-off is unavoidable when using the physically based deformation. Nevertheless, it is still affordable in many cases. Another limitation is that currently our system can only be applied on particle datasets and volume datasets. For line or surface based rendering, first we need to perform extra topology related computations, when the vertices of the same line or surface are split by our algorithm. Otherwise, the line or the surface will lay across the incision. Then the estimation of mesh stiffness should also take the topology into consideration. We will study on this topic in our future work. Additionally, in the future we plan to apply suitable perception metrics that can guide our design of occlusion management strategies. By making the visual performance of occlusion management methods more quantifiable, we can compare our method with non-physically based methods in multiple aspects, to provide users better comprehension of the data exploration process.

Chapter 4: An Automatic Deformation Approach for Occlusion Free Egocentric Data Exploration

4.1 Introduction

3D data visualization facilitates knowledge discovery from scientific datasets, thus plays important roles in modern scientific research. Occlusion is a common problem challenging the effectiveness of 3D visualization. As a natural result of depth differences of data elements, important data information can be blocked from users' view. A lot of studies have been conducted to provide occlusion free visualization; however, most of them address situations in an exocentric mode, for which the camera is placed outside of the bounding box of the dataset. With improved data acquisition techniques and increased data resolution, a dataset can also be visualized in an egocentric mode, which places the camera inside the dataset, and navigates the camera in a first-person view. Egocentric visualization has a wide application in various scientific problems [48]. Its benefits include directly visualizing internal data features underneath the surface, and obtaining more details at a close distance [15, 36, 58]. It is also frequently needed to create an immersive virtual reality environment levaraging CAVEs [24] or consumer-grade VR headsets [68, 75], which have been validated for improvement of user performance for various visualization tasks when compared to traditional output devices [33, 49, 50].

However, egocentric data visualization suffers from occlusion problems in different ways compared to those in previous studies targeting the occlusion happened in exocentric views. While in the latter case the camera is still far from the obstacle, in egocentric mode, occlusion can happen when a few opaque data elements occur right in front of the camera. Figure 4.1(a) gives an example of navigating inside an MRI brain dataset. When the camera moves forward and enters the solid cluster, it only receives a meaningless scene as Figure 4.1(b). Besides camera navigation, when the camera is fixed at a location which is currently not blocked, interactions with the data can change the scene and produce new occlusion too. For example, changing the transfer function for volume rendering can highlight regions that block the camera. Another example is exploring time varying datasets, since even if the camera stays at a good location in the currently rendered time step, data elements in new time steps may come over and block the camera. While an occluded scene in exocentric mode may still provide certain information about the obstacle, when the camera and the obstacle are almost overlapping in egocentric mode, the view is fully occupied by only a few elements as Figure 4.1(b). From such a scene with little information, it is hard for users to even distinguish the obstacle, and the sense about the current location and direction can also be lost. Also some previous strategies require user interactions being performed. This does give users more power, and works well for static visualization in previous research. But as the examples we just named, occlusion in egocenteric data exploration frequently happens at the same time when users are performing some other interactions to change the scene. In such cases, spending extra interactions to resolve the occlusion can be less efficient.

Several studies have been conducted to enhance camera behavior when it collides with solid data elements. One popular method in the computer graphics community is to attenuate the polygons that block the camera, such as using higher transparency or cutaway [16].

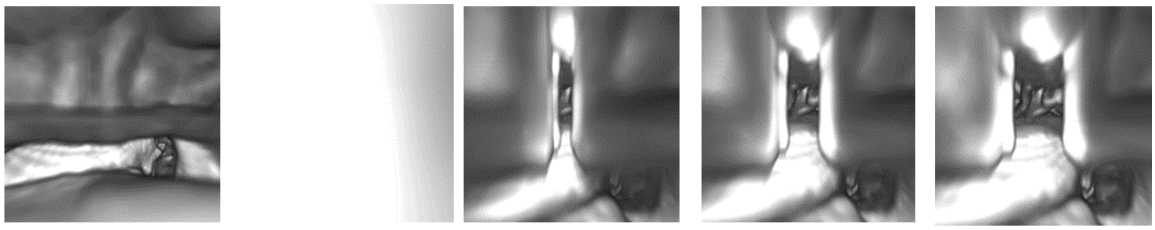


Figure 4.1: Screen shots from performing camera navigation in an egocentric mode, using an MRI brain dataset. (a) gives the original position of the camera. Without our method, moving the camera forward will see a scene as (b), where the camera is trapped into opaque voxels and the view is blocked. Our method can detect this degeneration, and deform the obstacle automatically using animations as shown by (c)(d)(e).

Although transparency adjustment or cutaway can be applied to data visualization too, they both lose information contained in the attenuated data elements. Another solution is to use a restricted navigation principle. For example, it is popular in colonography field to forbid the camera from leaving the colon cavity [36]. Another similar strategy is to compute an optimal path for the camera regarding to certain metrics [58]. These camera restriction strategies are more suitable for particular datasets or tasks, but are less effective for free data exploration in a general sense.

Data deformation is a promising way for occlusion management and data exploration, which usually discards less data information compared to the filtering techniques. Although deformation has been frequently adopted in exocentric mode of data visualization, its potential for egocentric exploration is not well exploited yet. In this chapter, we propose a new automatic deformation approach for free-form data exploration in egocentric mode. Using our method, the occlusion resulted from the camera and opaque data elements being too close will be solved, leading to more informative scenes, without extra manual interactions needed. Our method deforms the obstacles to create a pathway for the camera

and view straight. The deformation is performed using animation, such as the screen shots shown in Figure 4.1(c)-(e). The automation of our deformation technique is realized by monitoring the data, the camera, and the rendering configurations at every frame and reacting properly. As a result, when users are already occupied by necessary interactions required by other exploration tasks, their interactions will not be interrupted by extra operations to perform occlusion management. We applied our method on various types of scientific datasets, including volumes, polygon based isosurfaces, and particle datasets. We studied the utility of our method for different applications, including camera navigation, selecting different isovalues, changing the transfer function, and exploration of time varying data. We also collaborated with a domain expert who applied our technique in his own research, to receive feedback from actual users.

This work has been published to IEEE Pacific Visualization Symposium 2018 [53]. An accompanying video has been uploaded to Youtube at https://youtu.be/yA3eVcYWSe8, subject to possible changes of the hosting website.

4.2 Automatic Deformation Management

In this section, we introduce our deformation management strategy to provide occlusion free exploration automatically. In short, our system monitors possible updates of the scene at each frame. Then through a state transition model, our system decides the proper deformation-related reaction before visualizing the data. Basic reactions include triggering a deformation when needed, managing the deformation to provide smooth transition between scenes, and restoring the data when the deformation is not needed any more. The state transition model is shown in Figure 4.4; however, to clearly introduce this model, we will first present several components involved, such as detecting occlusions, regions of

deformation, and three types of animations. Then at the end of this section, we introduce how this model organizes the whole deformation system.

4.2.1 Occlusion Detection

The first concept adopted in our method is to detect whether occlusion will happen at a given three-dimensional spatial location. The detection is usually performed for the current location of the camera. However, it may also be needed for a location even if no camera has been placed there yet. Due to this reason, being occluded or not is treated as a property of the location instead of the camera. We call the kind of locations where the camera can be occluded as *occluded location*. On the contrary, a location for camera where occlusion will not happen is called a *clear location*.

The decision of being occluded or clear is data dependent; though, some general ideas can be summarized. Firstly being occluded or clear is a local property, so it can be detected fast using advanced location query methods, or parallel methods on GPUs. Secondly any location outside of the data domain is essentially clear. In the following, we list the basic methods we applied for common types of scientific datasets.

Volumetric Dataset: For a volumetric dataset, occlusion detection of a location depends on the values of local voxels, as well as the current transfer function for direct volume rendering. We select a density threshold and a distance threshold. Under the current transfer function, if the density of any neighboring voxel within the distance threshold is higher than the density threshold, the location is treated as an occluded location, otherwise as a clear location. As an example, the left image of Figure 4.2(a) gives a slice of a brain dataset, and if a linear transfer function is used, the right image shows the voxels whose density higher

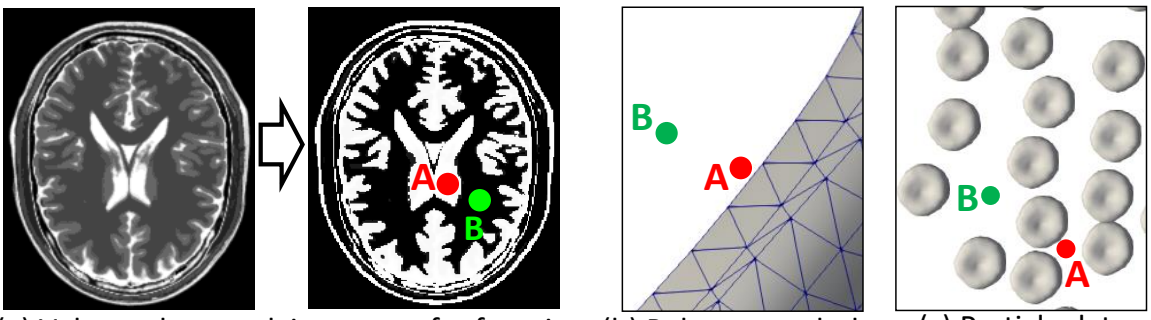


Figure 4.2: Examples of clear/occluded locations. Three data types are shown. Location A is occluded, while B is clear in all cases.

than the threshold depicted as white, and other voxels as black. Therefore, in this image, location *A* is an occluded location, and location *B* is a clear location.

Polygon Mesh Dataset: For a polygon mesh dataset, we use a parallel GPU method to find the distance from the location to each polygon of the mesh. A location is treated as occluded if there exists a polygon that is closer than a threshold, otherwise clear. Figure 4.2(b) gives an example using a triangular mesh. In this example, location *A* is an occluded location, and location *B* is a clear location.

Particle Dataset: Although particle datasets are frequently rendered using polygon mesh models, our system treats them differently. The reason is that our deformation method may need to break a mesh for polygon mesh datasets, but for particle datasets, since each particle is relatively small, it may not be effective to break the polygon mesh representing a single particle. In this project, we study particles that are relatively sparse, so the occlusion is mostly caused by one or a few particles. When deciding whether a location is occluded, we use the distances from the location to the center of each particle, and check if there is any

particle closer than a threshold. In the example of 4.2(c), location A is an occluded location, and location B is a clear location.

4.2.2 Deformation Region Creation

When a new deformation is needed, our system will first try to find the local obstacle that causes the occlusion. We want the deformation to make as little impact to the data as possible. Therefore, we define a deformation region to mark out the responsible local obstacle, and only perform deformation within this region.

Figure 4.3(a) gives an example of a deformation region using the pink rectangle. In this figure, white is used to denote opaque data elements. To generate the deformation region, we first define a central axis of the region. We use the camera direction $\vec{v}$, as in Figure 4.3(a), as the direction of the central axis. This choice aims to take care of the current camera direction with higher priority than other possible camera directions. The central axis will go through the current camera location. Its two ends are decided by the most proximal clear locations along $\vec{v}$ and the opposite direction of $\vec{v}$, as the location A and B in Figure 4.3(a). These two locations are found by an iterative sampling method. Starting from the camera location, each time we move the sample location along $\vec{v}$ (or its opposite) for a small step. If the new sample location is occluded, iteratively move the sample location forward, until when we find the first clear location, which will be the end of the axis. After finding the central axis, the full deformation region will be completed with a user selected shape model, whose details will be introduced later in Section 4.3.1. In the following we continue using a cuboid shape as an example for method description. Now if central data elements are displaced outwards, view sights can reach as far as the two ends of the central axis, which

are very possible to be in open areas that can provide rich information, as the example shown in Figure 4.3(f).

After being generated, the deformation region may need to be updated in later frames, after certain user interactions are performed. The reason is that, if in the new scene the occlusion detection result of the same location can be different, the two ends of the central axis are not guaranteed to be at the most proximal clear locations. For example, the transfer function used to get Figure 4.3(a) is changed and now gives us Figure 4.3(b). Then we update the deformation region from the original one in the left image of Figure 4.3(b) to the one in the right image. The update is conducted by adjusting the two ends of the axis in a similar iterative way. This updating strategy requires the old and the new deformation regions to have overlap. It can be satisfied if between nearby frames, opaque data elements, such as the white obstacle in Figure 4.3(b), are moved in a spatially continuous manner. Since such kind of continuous movement is more common in scientific simulations than data elements teleporting for a long distance, the assumption is commonly satisfied in real research.

4.2.3 Deformation with Animation

Our deformation is performed with animation to change the scene smoothly. The animation can create and enlarge an empty space inside a deformation region, which we call *tunnel* in this chapter, as marked in Figure 4.3(c). To cover all possible situations of deformation, three types of animation will be needed, which is opening animation, closing animation, and mixture of opening and closing. The following paragraphs give more details.

The opening animation is triggered when a new deformation is needed to resolve occlusion. A timer is created to be associated with the animation. The expiration time

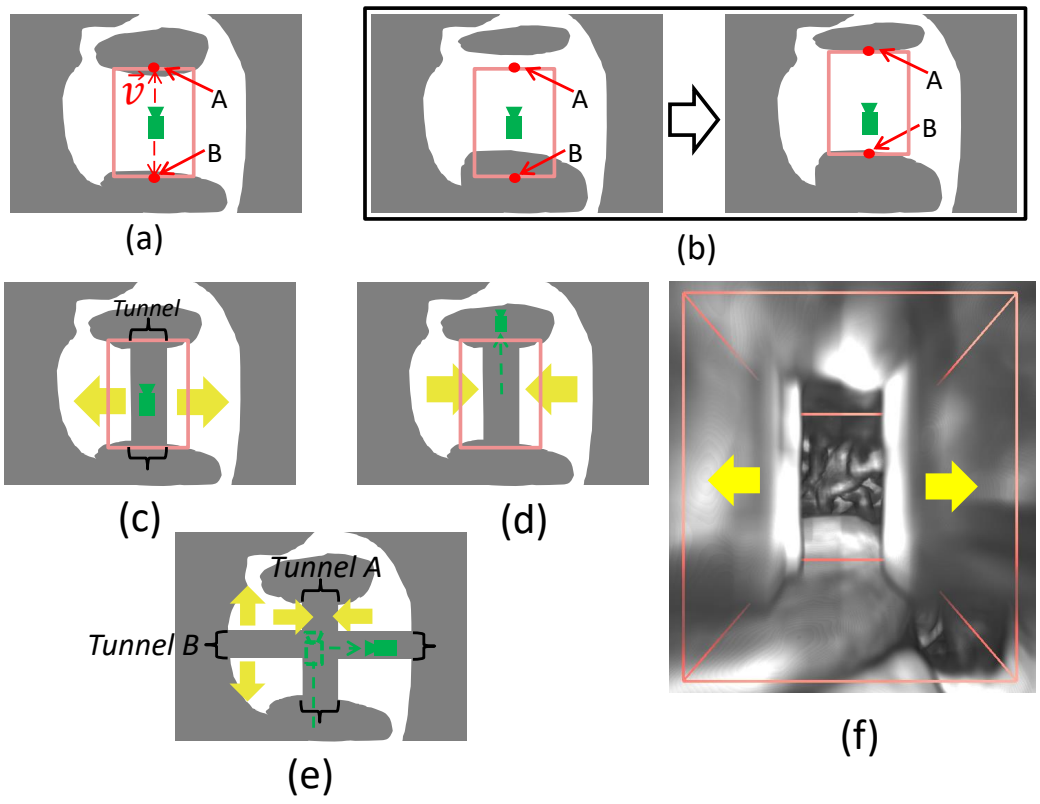


Figure 4.3: Illustration of deformation regions and tunnels. White is used for opaque data elements, and grey is for empty space. The pink frames represent deformation regions. (a) Compute the deformation region. Segment AB forms the central axis. (b) Update the deformation region when needed. (c) Tunnel opening. (d) Tunnel closing. (e) An example of mixed animation and two tunnels existing at the same time. (f) A 3D view of the deformation region.

for this timer is a preset length T, usually a few seconds, denoting how long it takes the animation to finish. During the following time frames t, $0 \leq t \leq T$, the size of the tunnel will increase from 0 to a preset scale, and we compute the displaced positions of data elements and update the rendered scene accordingly. Note that now the occlusion detection method for a location can be slightly different. During the process of opening, some solid data elements inside the deformation region can still be close to the camera, but eventually they

will move far away, so we do not want them to mislead the occlusion detection. Since we can know the final shape of the tunnel before the opening finishes, locations inside the final tunnel are always treated as clear.

The closing animation happens when an existing deformation is not needed any more, so data elements in the deformation region will be restored to original. For example, the camera leaves the deformation region and goes to a clear location in the original data, as in Figure 4.3(d). The animation is done as the reverse procedure of the opening animation. Therefore, in the order of Figure 4.1(e), (d), (c), they can also be treated as screen shots from a closing animation. A timer is also used to control the size of the tunnel to be continuously reduced from its current scale to 0. Once the timer expires, the associated deformation region is not needed any more and will disappear.

The mixed animation of opening and closing will be needed when the camera goes to an occluded location from a created tunnel. Figure 4.3(e) gives an example, where the camera first entered an opaque region and the deformation created Tunnel A; then from Tunnel A the camera goes to another occluded location. In this case, an opening animation is needed to create Tunnel B, while Tunnel A is closed at the same time with a closing animation. Note that the central axis of Tunnel B is computed using the original dataset, so it will go through the whole opaque obstacle, instead of being cut off at Tunnel A. Therefore, when the camera in Figure 4.3(e) looks left, the view sight can still reach far. At each frame of the animation, we first compute an intermediate shape of the data by only applying the closing deformation corresponding to Tunnel A, then on the intermediate shape apply the opening deformation corresponding to Tunnel B. If the camera goes into more occluded locations before Tunnel A fully closes, even more tunnel deformations can exist at the same time. We can used a queue to manage those tunnels and associated timers.

4.2.4 Organize The Deformation System

To automatically handle all possible updates made by user interactions, and manage the deformation animations in an uniform and efficient way, we have designed a state transition model to organize the deformation system. A state transition diagram is given in Figure 4.4. There are five states as the five rectangles shown in the figure, which are data in the original shape, in a static deformed shape, and in one of the three possible types of animations. Events that can trigger a transition are drawn in dashed ellipsoids in the figure. There are three main kinds of events. The first kind is time-out event of the timers associated with the animations, which takes the highest priority. The second kind is the occlusion detection for the current camera location using the original dataset. The third kind is also deciding whether the camera location is clear or not, but using the current dataset which might has been deformed. To be concise in the figure, we use *C* to denote clear, *O* to denote occluded, *Ori* to denote the original data, and *Cur* to denote the current data.

To control the deformation system at each frame, one edge of the transition diagram will be executed. The head of the executed edge is decided by the ending state of last frame. Then by analyzing which event is happening in the current frame, the system can pick a proper edge originating from the head, and the ending state of the current frame will be transitioned to the tail of this edge. In this way, all types of datasets and all exploration tasks studied in this chapter can be processed uniformly. As an example, suppose the system was as Figure 4.3(c) at the end of the previous frame. So in the current frame, we will find an edge originating from the *Opening Animation* state. If we detected that the camera location is occluded in both the original data and the current data, then perhaps Figure 4.3(e) happens, and the system goes to the *Mixed Animation* state, along the edge going right-downwards.

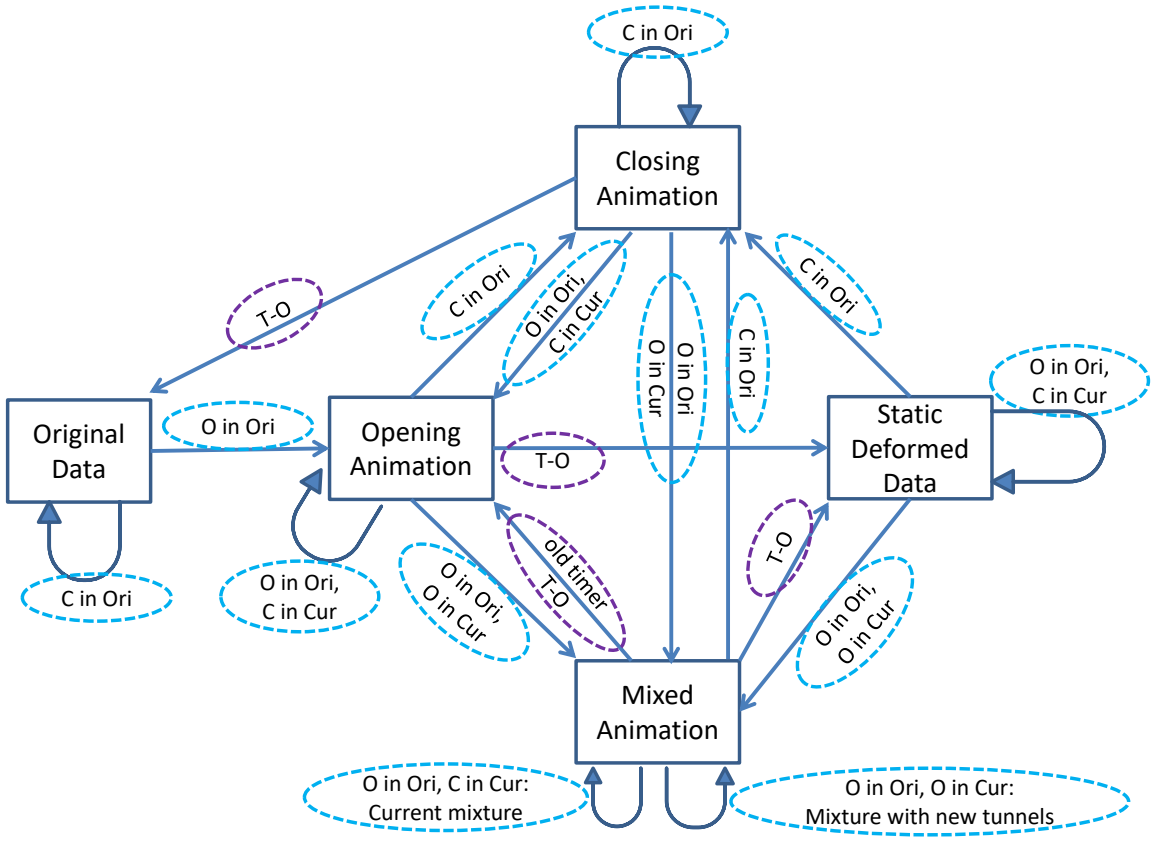


Figure 4.4: The state transition diagram of our deformation system. Rectangles represent states; arrows represent transitions between states; and dashed ellipsoids represent events. *T-O*: time out. *C*: clear. *O*: occluded. *Ori*: original data. *Cur*: current data.

4.3 Data Elements Deformation

In last section, to describe our deformation system, we utilized the concept of the deformation region, but without going into its details. In this section, we will complete the plot by introducing what is happening inside the deformation region, including possible selections of its shape, and how data elements inside are deformed.

4.3.1 Shape Models

While previously in Section 4.2.2 we introduced how to determine the central axis of the deformation region, here we show how to define the whole shape besides the axis. We have designed two shape models, the *Cuboid Model* and the *Circular Model*. Different models may best suit different situations, depending on the type of datasets or tasks.

The shape model gives a boundary to separate data elements to be displaced, and decides the specific formulae to displace them. At any time frame, it maps a location p in the original space to a new location p' in the deformed space. Then, for polygon mesh datasets, we move each vertex from its original location p to p'. For particle datasets, we move the particle center from its original location p to p', and for all polygon vertices used to compose this particle, keep their relative locations to the center unchanged. For volumetric datasets, we create a new volume with the same resolution to represent the deformed data, and for its voxel at location p', find the location p in the original space, retrieve the scalar value there to assign the voxel at p'. Besides creating a new volume, it is also possible to use ray deflecting [47] to achieve the deformation effect when rendered by ray casting, which when sampling at a location p' along a ray, returns the voxel value at location p. In our models, the mapping functions that involve p' and p are the same as creating a new volume. We will give more mathematical details in the following.

4.3.1.1 Cuboid Model

This model uses a cuboid to define the deformation region. The length l of the central axis is automatically determined by our system, and the width w and the height h, as in Figure 4.5(a) and (b), are user selected parameters. The camera direction $\vec{v}$ and its up direction $\vec{u}$ will determine a center plan of the cuboid, as the blue plan in Figure 4.5(b). The

cross product $\vec{w}$ of $\vec{v}$ and $\vec{u}$ will serve as the direction that data elements are moved along. Data will be deformed from the center plan towards two sides to create a cuboid tunnel.

The created tunnel has the same length l and height h, and the width of the tunnel w' will gradually change between 0 and $w/2$ using an opening or closing animation. Let uniform change be applied to w', so by the passing time t and the expiration time T of the associated timer, we can easily compute the current value of w'. As in Figure 4.5(a), let the distance from the boundary of the tunnel to p' be x', and the distance from the center plane to p be x. Let the data be uniformly deformed, so we will have $x'/(w-w') = x/w$. So a vertex at p in a polygon mesh dataset, or a particle centered at p, will be displaced to position p' by:

$$p' = p + \vec{w}\,(x - \frac{w'}{2} - \frac{(w-w')x}{w}). \tag{4.1}$$

For volume datasets, to create the deformed volume data, the value of the voxel at p' is assigned by the value of the original data at position p:

$$p = p' + \vec{w}\,(x' + \frac{w'}{2} - \frac{wx'}{w-w'}). \tag{4.2}$$

An extra operation is needed for polygon mesh datasets. Because just moving the vertices will not disassemble the original connected mesh to create a break, we may need to first modify the original mesh to create such a break. This is a one-time operation when a state transition requires a new deformation region, and can be finished in real time using GPUs. For the cuboid model, the break of the original mesh will happen at the central plane of the deformation region. To realize this, we use a triangular mesh shown in Figure 4.6 as an example. We first find all triangles that intersect with the central plane. Then we modify each of these triangles into three new smaller triangles with two new vertices on each intersecting edges. This intuitive operation is enough to fulfill our needs in our tested cases, although more sophisticated geometry modeling method might be available too.

4.3.1.2 Circular Model

The circular shape is also a popular choice in occlusion management studies and is often named by certain kind of *lens*, due to its resemblance to real camera lens. We use a cylinder shape with a selected radius r as a parameter, and the pink frames in Figure 4.5(c) and (d) give the boundary of the region. The tunnel created by this region will have the same central axis. Similarly with the cuboid region, we set the maximum radius of the tunnel as half of the radius of the deformation region r, and let the current tunnel radius r' change with a constant rate before the associated timer expires. The deformed position p' of a data element will lay on the ray pointing from the center of the cross section to its original position p. Denote the direction of this ray as $\vec{w}$. We also uniformly compress the data into the space between the tunnel and the boundary of the deformation region, so we have $x'/(r-r') = x/r$. Therefore, when knowing p and its distance to the central axis x, the data element will be displaced to:

$$p' = p + \vec{w}\,(r' + \frac{(r-r')x}{r} - x). \tag{4.3}$$

And when knowing p' and its distance to the tunnel x', its original position is computed by:

$$p = p' + \vec{w}\,(\frac{rx'}{r-r'} - r' - x'). \tag{4.4}$$

In our study we found that the circular model is suitable for particle datasets. However, although we use a volumetric dataset for demonstration in Figure 4.5(d), it is not a good choice for volumetric datasets or polygon mesh datasets. While the cuboid model breaks the space with a rectangle plane, this model breaks the space with the central axis which is only a segment. Therefore, data elements around the central axis will be distorted too much after deformation. On the contrary, when deforming particle datasets, no break is directly applied on data elements, and only displacement of individual particles will occur. In our case study

in Section 4.5.4, we will also show a scenario when the circular model works better than the cuboid model to prove its value.

4.3.2 Hint for Displaced Data Elements

Besides providing a pleasing rendering, our animation also serves as a tool to remind users of the change, which lowers the chance that users being confused or mistreating the deformed data as the original one. To further improve the assurance, similar with [72], we use color as an extra visual cue, to encode the distance of displacement for elements that are not at their original positions. Figure 4.7 shows several examples adopting this strategy. In each case, we select a hint color which is in large contrast with the original color of the deformed data elements. Then during the deformation, the color of a deformed data element will be its original color blended with the selected hint color. The blending weight of the hint color is positively correlated with the distance from the current location to the original location. We also set this as an option that can be turned on or off, so when users want to closely inspect the deformed obstacle, they can turn this feature off in order not to be disturbed.

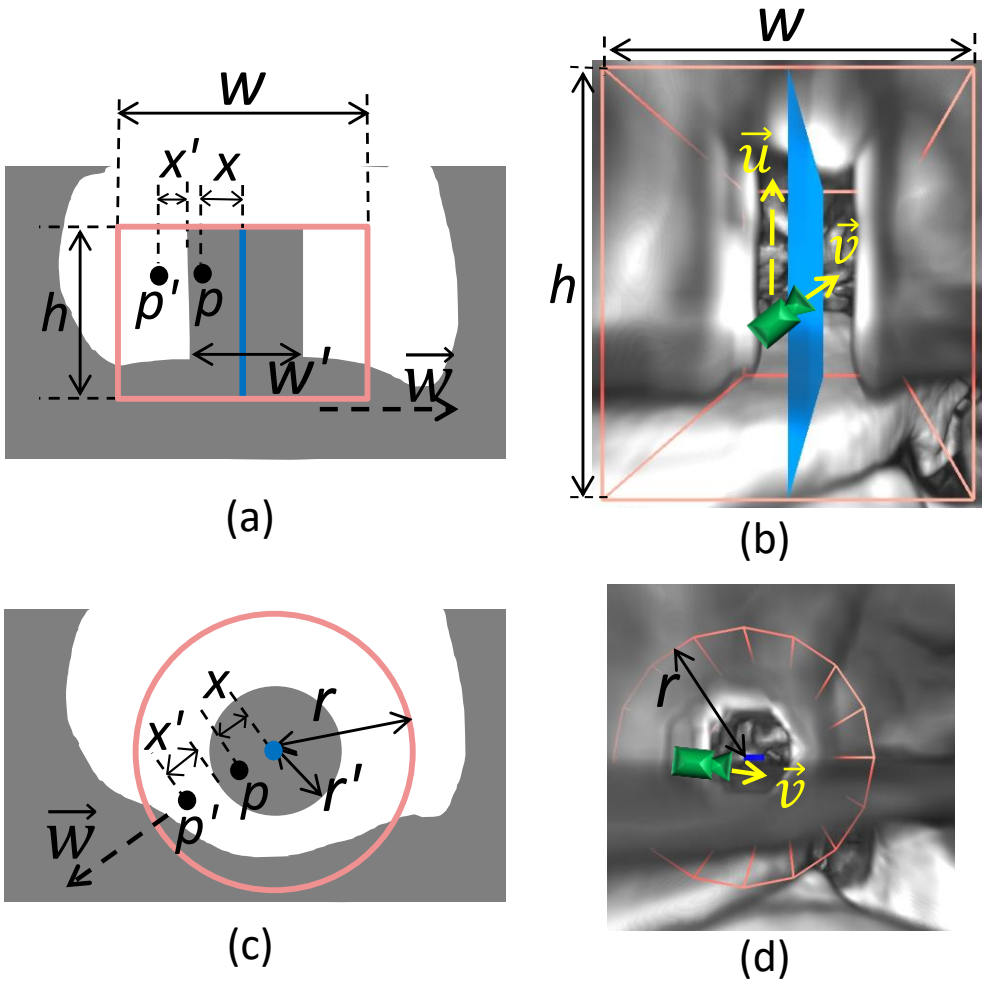


Figure 4.5: Parameters of two shape models, and mapping of locations between the original and deformed spaces. (a)(b): cuboid model. (c)(d): circular model. (a)(c): cross sections perpendicular to $\vec{v}$ in (b) and (d).

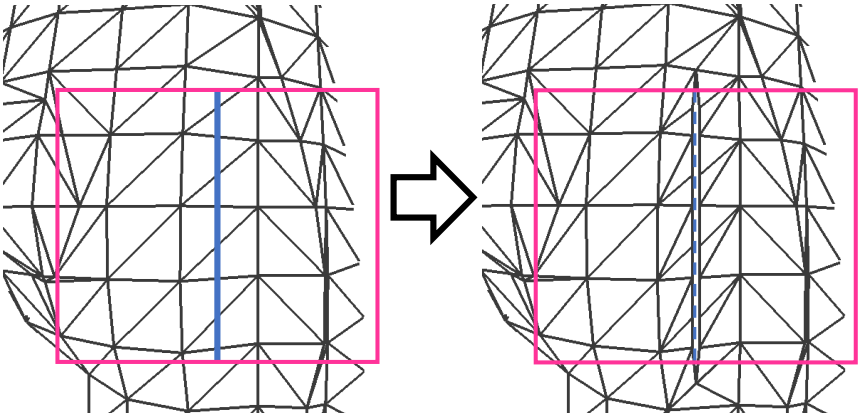


Figure 4.6: Cutting the triangular mesh to create a break for the cuboid shape model.

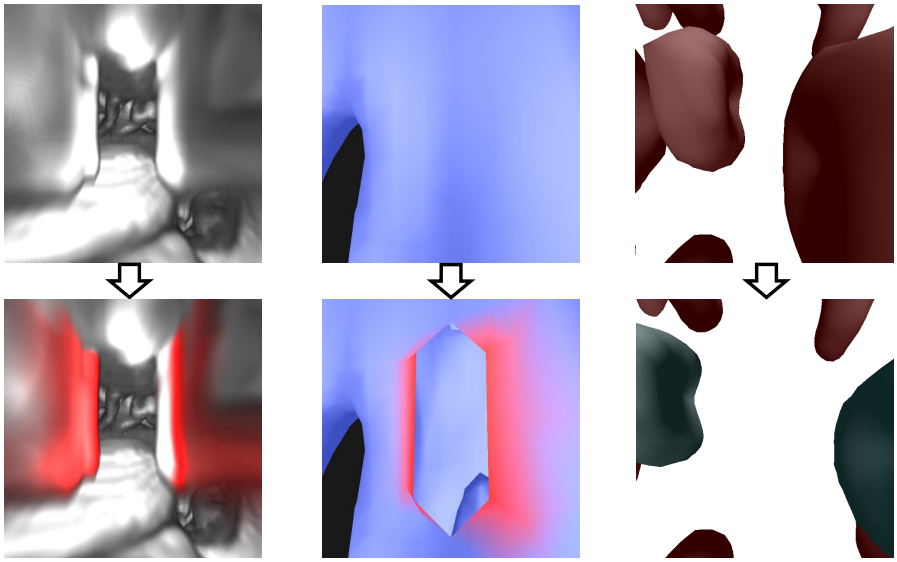


Figure 4.7: Examples of using color as a hint for displaced data elements. Red used in (a) and (b), and cyan used in (c).

4.4 Implementation

One application of our approach is for camera navigation. Our algorithm only takes the current location, front direction, and up direction from the camera. Therefore, in theory, most navigation schemes will work as long as these configurations are available. In our experiment, we follow the *flying* navigation strategy proposed by [63], which fits the egocentric environment in a natural way. We use mouse dragging to change the camera direction. We use keys from a keyboard and mouse wheel scrolling to pan the camera. We also testify the camera navigation applications using the Leap Motion detector and Open-Source Virtual Reality head-mounted display as in Section 3.4. Camera panning is done by hands moving, and camera rotating is done by head rotating. Figure 4.8 shows an example when a user is performing navigation using our system with the motion detector input and HMD output.

To study other exploration tasks besides camera navigation, we provide interactions to change isovalues for isosurface study, change the transfer function for direct volume rendering, and change the rendered time step of a time varying dataset. All these interactions are implemented through QT widgets. To make the scenes change smoothly, we use slider bars to change corresponding values.

We also provided regular interfaces, implemented by QT widgets, to enable users to adjust the parameters related to the deformation method. As a summary of the description in the previous sections, users are able to set the threshold values for occlusion detection, set the time cost to fully open a tunnel, and select the deformation model. The width and the height of the cuboid model, and the radius of the circular model are also adjustable. Once set at the beginning, changing parameters is not a requirement for data exploration, so it is not conflict with the automation principle.

Figure 4.8: A user performs navigation using our system with a motion detector and an HMD.

When users are not familiar with the global context of the data, egocentric visualization can be combined with exocentric views together, as an application of the Overview+Detail strategy. We also provided users an easy switch between the two views. Using advanced designs to better combine these two modes is an important research topic, relying on concrete tasks and available devices [18, 58, 68]. Creating a comprehensive system is beyond the scope of this chapter, since our work only focuses on the egocentric part.

4.5 Case Study

In this section, we demonstrate several case studies using various types of datasets for different exploration tasks. Since our deformation procedure uses animations, we provide accompanying video for each case study.

4.5.1 Camera Navigation

We first demonstrate a camera navigation case using an MRI scan dataset, which is the one in the teaser image. It is an MRI T2 scan of a brain phantom, provided by McConnell Brain Imaging Centre of Montreal Neurological Institute, McGill University, with a size of $181 \times 217 \times 181$. Figure 4.9(a) gives an external view of the data. We start the navigation from the location in Figure 4.1(a). There the camera is inside the left hemisphere, and facing the cortex between the two hemispheres. After deforming the cortex as in the teaser image, we move the camera forward into the tunnel, and turn left to observe the deformed cortex as in Figure 4.9(b). We can have a good sense of its thickness, as well as its spatial relations with both hemispheres, and the corpus callosum below. In Figure 4.9(c), we move the camera backwards to enter the solid cortex again, and trigger a mixed animation similar to the case in Figure 4.3(e). In Figure 4.9(d), we leave the tunnel and enter the right hemisphere. Next we move to a location below the corpus callosum as Figure 4.9(e), where rich brain tissues exist. We can directly penetrate these tissues as in Figure 4.9(f), and go to the mirrored location in the left hemisphere as in Figure 4.9(g). At last, we deformed the outer cortex as Figure 4.9(h), and move to the thin space between the skull and the brain cortex to observe these two close-by layers as in Figure 4.9(i). The navigation procedure of this case demonstrates that we can move to any cavity region of the data without abruptly changing the image or taking a detour, and we can also gain knowledge of the occluding cortex or tissues when passing the tunnels.

The next case of camera navigation uses a geological simulation dataset provided by a domain expert who collaborates with us on this work. It simulates injection of carbon dioxide (CO_2) into a brine-saturated deep saline aquifer. When CO_2 dissolves into the aqueous phase it increases the local density, which is gravitationally unstable and triggers

so-called *gravitational fingering* [60, 64]. The ensuing non-linear gravito-convective mixing forms highly complex flow patterns and distributions of dissolved CO_2 that are challenging to visualize and interpret. Isosurfaces are a common tool to visualize this type of 3D data in geosciences. The iso-surfaces in Figure 4.10(a) illustrate the many thin and long fingers descending from the top of aquifer. However, it is difficult to visualize how the fingers are connected and what happens inside a finger, because a given isosurface blocks the view of its interior.

As a comparison, our collaborator believes our method significantly improves the interpretation of such simulation results. We select three isosurfaces with isovalues of 0.0006, 0.0011, 0.0014 for the molar fraction of dissolved CO_2 in brine, colored by the isovalue, and explore the fingers using our method. Green is used as a hint color for displacement, as introduced in Section 4.3.2. First, we face a finger of the lowest isovalue 0.0006, as Figure 4.10(b). We move forward to open this finger as in Figure 4.10(c), and find another finger of isovalue 0.0011 inside. Since the finger of isovalue 0.0006 is still visible, we can compare their shapes and separation. Next we move towards this inner finger and break it as in Figure 4.10(d), but the next finger of isovalue 0.0014 is also opened up. Thus we know that their separation is very small. If we want to keep the shape of the red finger, we can use a lower distance threshold for determining whether a location is occluded or clear. But with a threshold too low, the surface can only open up when the camera is very close, which may not be expected. Alternatively, we can also consider the normal of the polygons for occlusion detection, by only using a lower threshold when a location is behind the polygon. Thus the central axis of the deformation region will only cover the finger of isovalue 0.0011, and we can separate the two surfaces to get Figure 4.10(e). In Figure 4.10(f), we raise the camera up to compare the top parts of the fingers. When we

move outwards from the current location, Figure 4.10(g) gives a screen shot of a mixed animation, when the tunnel on the finger of isovalue 0.0011 is closing, while the tunnel on the finger of isovalue 0.0006 is opening. Figure 4.10(h) gives a view from the bottom of a finger, since the tips of fingers are also interesting features for domain researchers. We break the tip of the outer finger, and compare the tips of two fingers at the same time.

4.5.2 Isovalue Adjustment

The next exploration task we demonstrate is to change the isovalues for isosurfaces. We continue using the same geological simulation dataset as before, to study the fingers of dissolved CO_2. In the navigation case we suppose users have specific isovalues to study. Now we experiment how our method works when no target isovalues are specified, but users are trying out different isovalues to find interesting ones. As stated before, we implemented slider bars for users to change isovalues consecutively. New surfaces are computed by the VTK marching cube filter. The camera is placed as in Figure 4.11(a), and initially the isosurfaces are computed by isovalue 0.001 and 0.0013. The surface of isovalue 0.001 is colored blue, but the surface of 0.0013 is not visible now. When we reduce the isovalue 0.001 to 0.0009 as Figure 4.11(b), or to 0.0008 as Figure 4.11(c), the surface becomes very close to the camera, occupies the whole view with little information available. Since users are currently using the slider bar to change the isovalue, it can be distracting to switch to camera operations, or manually remove the occlusion to achieve a good view. When the isovalue is reduced to 0.0006, the view is abruptly changed to Figure 4.11(d), when the blue surface moves from the front of the camera to the back of the camera. If our automatic deformation is enabled, the whole procedure will be as Figure 4.11(a), 4.11(e), 4.11(f), 4.11(g), 4.11(d). As a comparison, each single view is informative rather than the whole view being blocked

by simple plain blue, and no abrupt change of the scene will happen as the change from Figure 4.11(c) to 4.11(d). At the same time, the opening of isosurfaces is fully automatic, so users can focus on isovalue adjustment, instead of needing extra interactions to achieve informative views. When we change the isovalue back to 0.001, since the surface is far enough from the camera, the opened tunnel can also smoothly close as Figure 4.11(h) to restore the surface.

4.5.3 Transfer Function Adjustment

The third type of exploration task we demonstrate is to change the transfer function used for direct volume rendering. Because direct volume rendering with a transfer function of narrow peaks can achieve a similar effect with using isosurface, we redo the previous study of changing isovalues by using direct volume rendering and controlling the transfer function. The transfer function we adopt has two narrow peaks with Gaussian shapes to reveal the isosurfaces of two values, and similarly we color these two values with blue and red, as in Figure 4.12(a). The blue surface is generated by a peak value of 0.001 in the transfer function. We reduce the peak value using a slider bar too. Without the deformation method, we get images like Figure 4.12(b) and Figure 4.12(c), until a sudden change happens at peak value 0.0006 as Figure 4.12(d), when the surface has moves to the back of the camera. If our automatic deformation is enabled, using green as the hint color, the whole procedure of transfer function adjustment will be as Figure 4.12(a), 4.12(e), 4.12(f), 4.12(g), 4.12(d). Again we achieve an exploration experience which not only has smooth transition but also has informative scenes at every moment, without extra user interactions beyond the original transfer function adjustment operations. When we move the peak value back to 0.001 as in Figure 4.12(h), the surface can also be restored decently with a closing animation.

4.5.4 Time Varying Data

In this section, we show the last type of exploration task, using a time varying blood cell dataset [65]. It is a hydrodynamic simulation that studies the behavior of red blood cells when interacting with both fluid and obstructions, which in this case are several columns in the field. The simulation has 27 time steps, and each step has a number of cells varying from 1182 to 1184. Figure 4.13(a) gives an overview of the field, where cells are colored red, and the solid columns and walls on the boundary are colored by brown. Users can change the current time step consecutively, or play multiple time steps using animation. In the latter case, for frames between the given time steps, cells are placed at intermediate positions by linearly interpolating their positions in nearby two time steps.

While the blood cells are drifting by the fluid, their collision with the columns is important, so we place the camera closer to the columns. Figure 4.13(c) gives several screen shots of displaying the whole simulation sequence without our method. We can see that since cells may move to the front of the camera, there are several moments that we cannot see anything in the view. Even through the animation is played automatically and users now are free to perform interactions, quickly respond properly while the animation continues is a hard job. In fact, this challenge has been mentioned but not solved in some early studies using manual deformation methods [54, 72]. We apply our automatic deformation method using the circular shape model. Since each cell has an orientation $\vec{v}$ and is roughly in a flat column shape as in Figure 4.13(b), we used two threshold values to decide whether a location is occluded or clear. More specifically, if a location and the center of a cell has a distance less than 2.5 along the direction $\vec{v}$, and has a distance less than 4 perpendicular to $\vec{v}$, the location is determined as occluded. Displaced cells are blended by cyan. Figure 4.13(d)

shows screen shots at the same frames with Figure 4.13(c). We can see that now no cells will ever stay right in front of the camera to block our view.

Next we use another view to show the benefit of the circular model over the cuboid model for this case. As in Figure 4.14, we study a few time steps in which a cell is moving from the left of the camera to the right. When it is close enough to the camera, if using the cuboid model, the cell will be pushed to the left, as the left image of Figure 4.14(a). In later time steps, its original location is on the right side of the center plan of the deformation region, and its deformed location is even more right, as the right image of Figure 4.14(a). Figure 4.14(c) illustrates this behavior, where the blue dash line is the projection of the central plan. The dashed particles show the original positions, and the solid particles are at the deformed positions. The black particles are in the last time frame, while purple ones are in the new time frame. As a result, in the two nearby frames, the cell is being transported for a long distance. This behavior can happen when particles pass through the center plan of the deformation region. For volumetric or polygon mesh datasets, the transported elements are only a thin layer and not easily distinguishable, but sparse particle datasets can amplify the phenomenon. For this problem, using the circular model will be a good solution. From the same view, screen shots of using the circular model are shown by Figure 4.14(b). The long transportation can only happen when particles pass through the central axis of the deformation region, as in Figure 4.14(d), which has a much lower chance than passing through the center plan of the cuboid deformation region. Even if that happens, we also set a threshold for the amount of location change, measured by the angle that a cell has rotated in nearby frames regarding to the central axis, as angle θ in Figure 4.14(d). When this angle is larger than the threshold, we set the actual deformed location by rotating it backwards, and that is why in the screen shots in Figure 4.14(b), the deformed particle in cyan passes

by the camera following an arc-shaped route. In this way, we can control the change of a particle between consecutive frames in a natural manner to always get a smooth transition.

4.5.5 Performance

With the help of modern graphics cards and GPUs, the algorithms used in our system are highly parallelizable, which enables our system to reach an interactive rate. The computation of deformation can be parallelized by each element of the dataset, such as each voxel, each vertex, or each particle. We tested the performance of our system in a desktop computer, with an Intel(R) Core(TM) i7-2600 3.4GHz CPU, 16GB RAM, and an NVIDIA GeForce GTX 980 Ti graphics card. All case studies have been performed at an interactive rate, and the lowest FPS we achieved was 38.

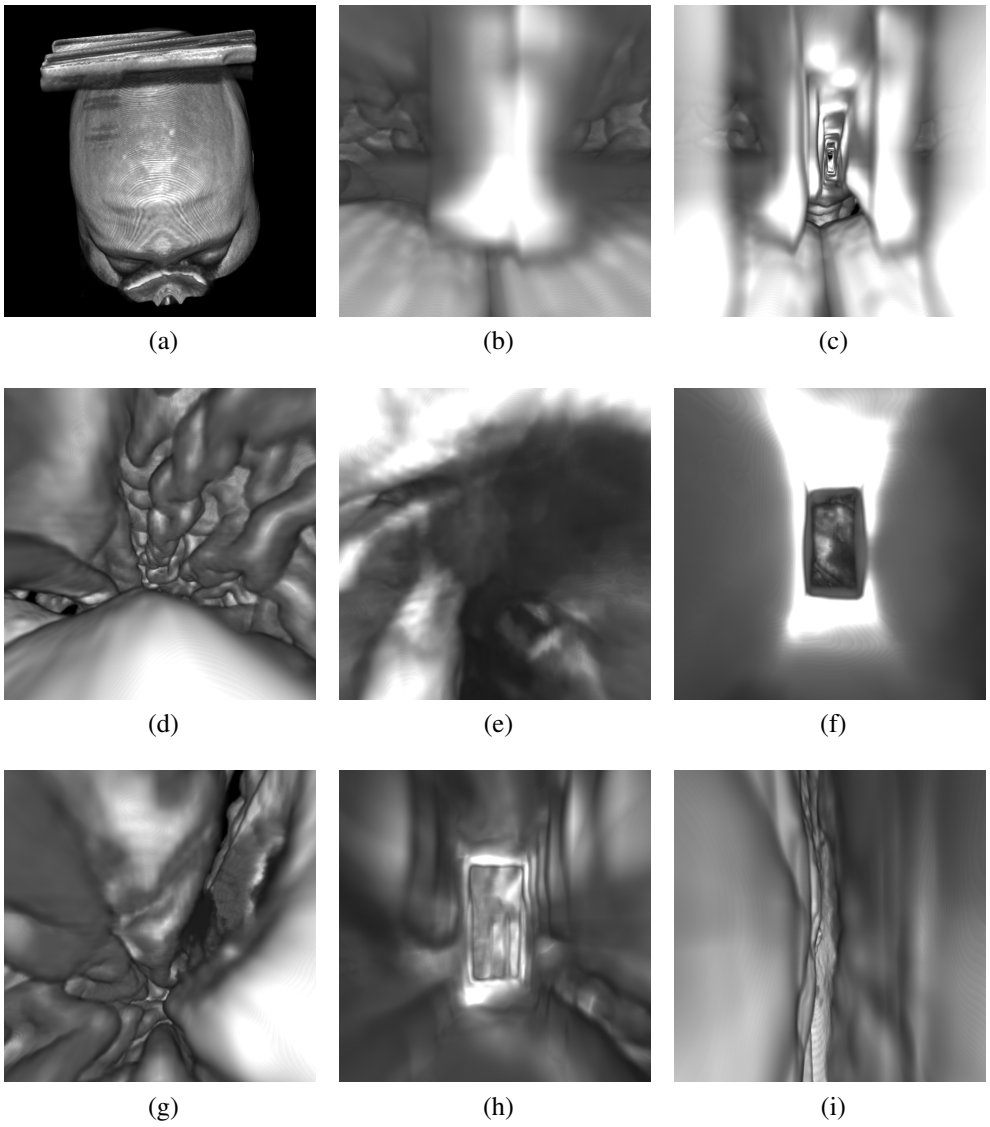


Figure 4.9: Navigation to explore the brain phantom MRI dataset. Linear grayscale transfer function is used. See text for details.

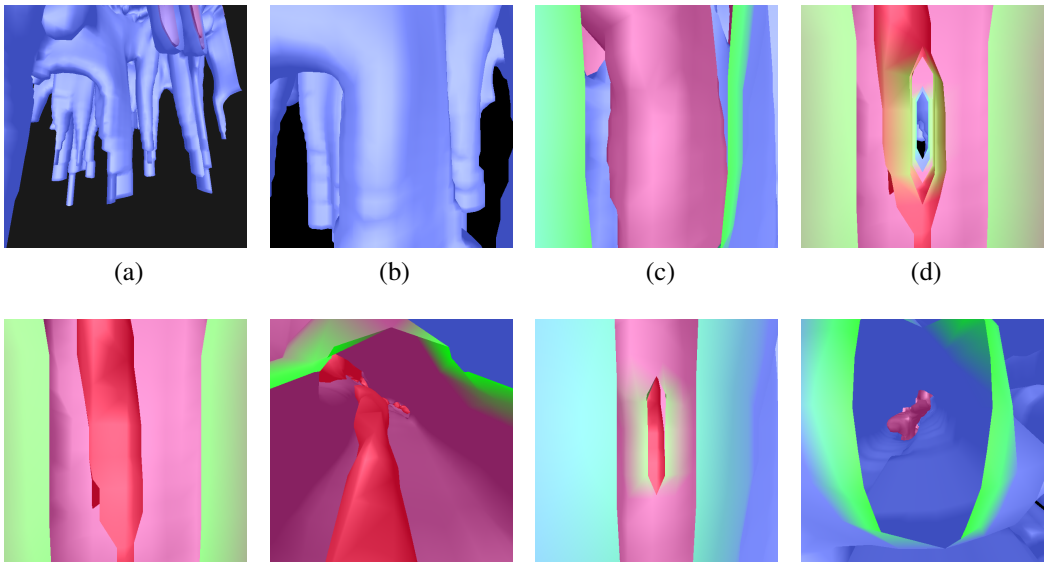
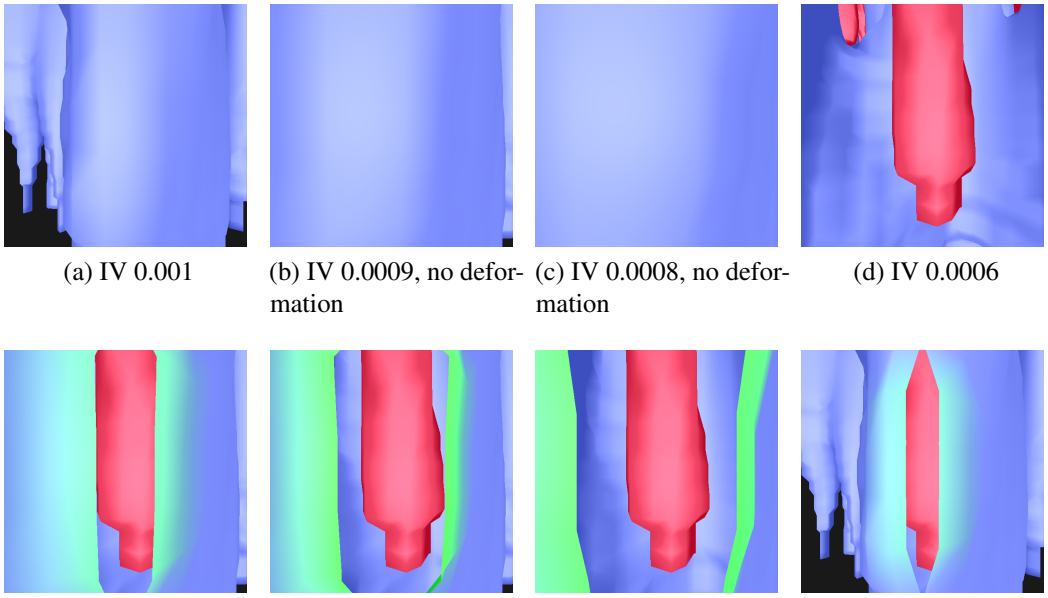


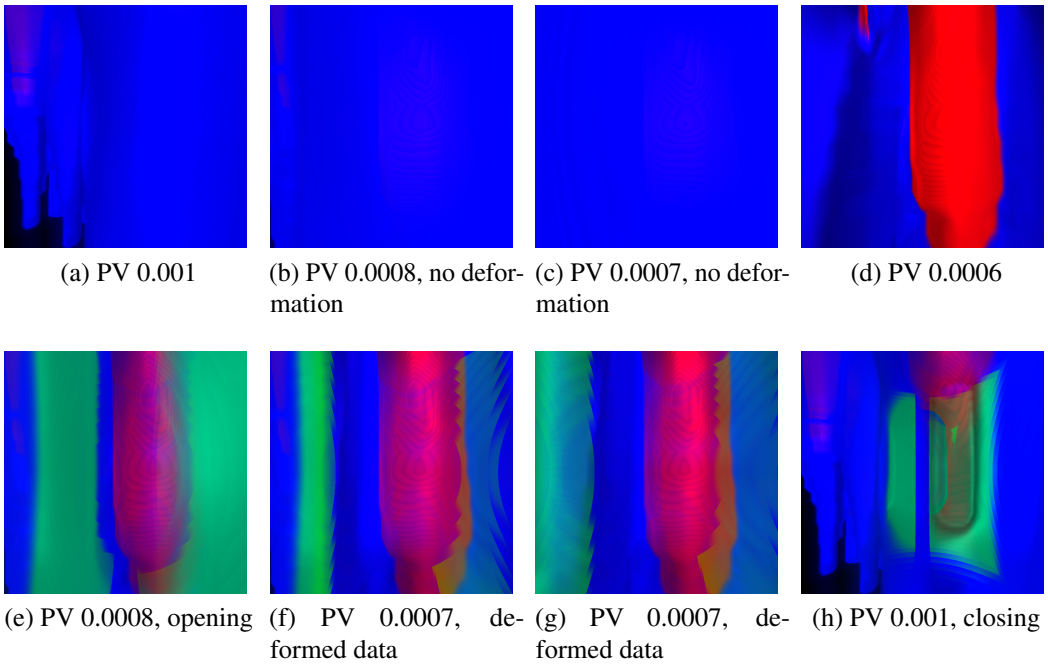
Figure 4.10: Navigation to explore the isosurfaces of the geological dataset. Blue, pink, red are used for the low, middle, and high isovalues respectively. Green is used as a hint color for displacement. See text for details.



(a) IV 0.001 (b) IV 0.0009, no deformation (c) IV 0.0008, no deformation (d) IV 0.0006

(e) IV 0.0009, opening (f) IV 0.0009, opened surface (g) IV 0.0008, opened surface (h) IV 0.001, closing

Figure 4.11: Exploring the geological dataset using isosurfaces. Change the isovalue of the blue surface, while the camera is fixed. *IV* short for isovalue.



(a) PV 0.001 (b) PV 0.0008, no deformation (c) PV 0.0007, no deformation (d) PV 0.0006

(e) PV 0.0008, opening (f) PV 0.0007, deformed data (g) PV 0.0007, deformed data (h) PV 0.001, closing

Figure 4.12: Exploring the value layers of the geological dataset using direct volume rendering. The transfer function has two narrow peaks. Change the value of one peak to change the position of the blue surface, while the camera is fixed. *PV* denotes the peak value.

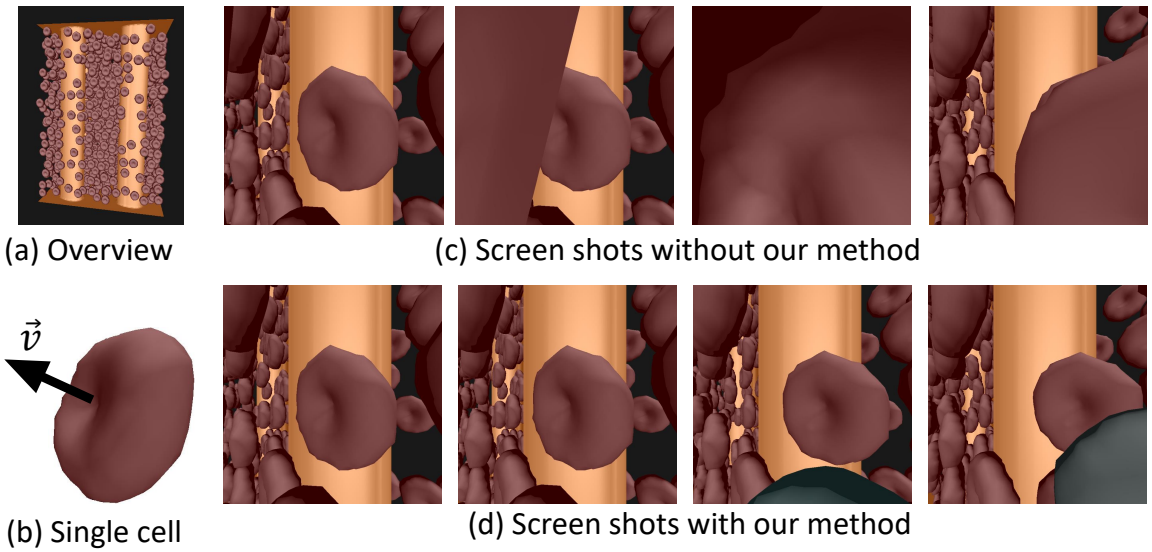


Figure 4.13: Exploration of the time varying blood cell simulation, compared with not using our method. Cyan color is blended on displaced cells.

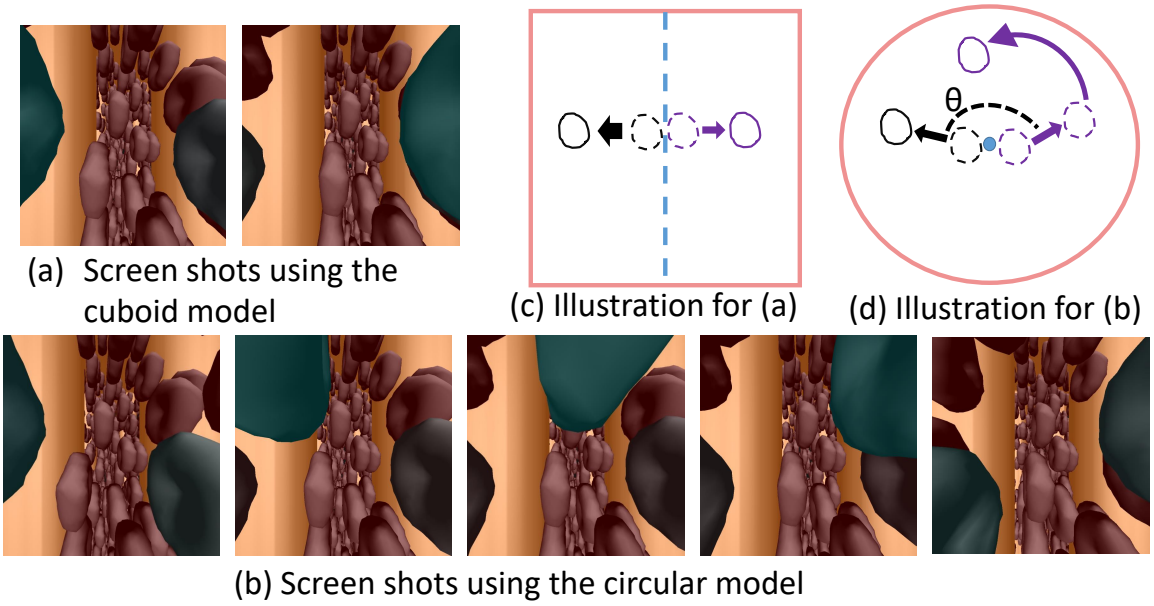


Figure 4.14: Exploration of the time varying blood cell simulation. Compare two shape models to show that the circular model can avoid abrupt change of cell locations. Cyan color is blended on displaced cells.

4.6 Domain Expert Feedback

We have conducted several studies together with the domain expert who provided the geological simulation datasets, produced for his own research. The expert is a professor with 10 years of experience working in Earth Sciences. He is experienced with widely used visualization tools including ParaView, VisIt, and Tecplot. However, to visualize the complex fingering structures, basic 3D techniques from these tools, such as volume rendering or isosurfaces, all result in various degrees of data occlusion when the visualization is projected on 2D media. As a result, it is hard to make sense of the large scale structure of the fingers, because they obscure each other. A combination of isosurfaces and slices is his common choice when using the existing tools. But firstly, slices do not show 3D connectivity; secondly, it needs cumbersome navigation to place the slice and the camera at desired locations to observe an interesting finger, while exocentric navigation is very unintuitive in this case.

To better visualize the fingers and internal features, the expert suggested us to apply deformation on his simulation data at an early stage of this study. After trying out our method, he believed the combination of an egocentric camera view point with a deformation of occluding features greatly improves the flexibility and user experience of exploring such 3D data-sets, compared to exploring isosurfaces using the existing tools. With the deformation, he could easily compare a finger with another finger inside, and had more awareness when the camera collided with any finger surface during the navigating. After a short learning period, he was able to easily use our system by himself to perform interactions. He was excited to use our method to prepare images for his future publications, and has promoted our work to other researchers from his lab.

During the development of the project, he provided many suggestions, such as selecting isovalues dynamically, and adjusting the threshold of the deformation. He has confirmed his favoring of several implementation choices, including using the cuboid shape model, using animated deformation, and applying color-highlighting of the data that were manipulated. The manipulation of data itself is not necessarily a concern. The first reason is that visualization techniques, including isosurfaces, already contain numerical estimations and differ from the actual data, and such effort of mentally reconstructing the actual data is common and manageable for him. Secondly, different visualization techniques are typically explored in a complementary fashion, so the weakness of one technique can be remedied by another. For example, although using isosurfaces is common, whole volume rendering and slice-by-slice checking are still necessary to retrieve data values in different manners. We all agree that a whole system comprising multiple techniques could be built to better serve such type of data, as an application-oriented future work. It may be useful to involve the global view too, by implementing a small inset that indicates where the zoomed-in local egocentric view is located within the full data-set. It can also be implemented with more freedom within the deformation region, such as other deformation models, or a dynamic region of transparency, as alternatives. Another exciting opportunity would combine this visualization method with a VR headset, to allow a fully immersive and intuitive use of the egocentric viewpoint and navigation.

4.7 Discussion

Our method was originally inspired by the camera navigation task, for which updating the location of an existing deformation region, as happened in Figure 4.3b, will never be needed. Other exploration tasks can also perfectly fit in our framework, with an assumption

that the location change of opaque data elements between frames is continuous, as stated in Section 4.2.2 in order to update the deformation region. Although the assumption can be easily satisfied, even if not satisfied, our system can still provide occlusion free exploration decently, because deformation regions used for opening animations can always be computed from scratch. The only defect is that closing animations can be compromised if information of closing regions cannot be received from early frames properly. For specific applications, it is possible to be exempted from this assumption by modeling the movement of the data elements directly, and move the deformation region together. But such methods are more data-dependent, instead of a general way as our method.

In our work, we adopted two intuitive and common shape models to define the deformation region. Because our whole system does not rely on the adopted shape model, it also has a potential to be extended with other shape models. For example, the current cuboid model opens up the obstacles to the left and right. When dealing with short and horizontally aligned objects, we can consider about a deformation model that slides to the top and bottom. Another example is when deforming thick obstacles, it is possible to slowly displace the front part of the obstacle first, then eventually displace the rear part. This can make the appearance of new elements behind the obstacle gradual but less surprising, which can further improve the temporal coherence of the animation. Some conventional deformation techniques reviewed in Chapter 2 may also be integrated. While the fitness cannot be asserted without careful experiments, we believe our pioneering work still creates many possibilities for extensions. The best shape model will always be decided by data and exploration tasks, so future work can also be specific data-driven or application-driven studies, and more shape models can be tested to reveal the most advantageous choice.

Besides new shape models, occlusion management strategies other than deformation are also possible to be employed within the deformation region. For example, using transparency is a possible way, which is conceptually easier. However, using transparency also bears some disadvantages, such as decreased depth cues [29], and the loss of information contained in the transparent data elements [10, 12, 59, 72]. We believe using transparency is also a promising idea, and we do not want to claim an absolute preference between it and our method. However, under the absence of a similar research using the transparency strategy, it may not be trivial to address all technique issues without a thorough research. For example, for a volume dataset, since each time the processed obstacle is different, it is not easy to automatically decide a transfer function to achieve the desired transparency effect. Another example is when the camera is close to, or even located precisely on, a polygon mesh, part of the mesh clipped out by the near plane of the view frustum can be distracting or even misleading. Therefore, future research is needed to fully comply the idea.

4.8 Conclusion

In this chapter, we proposed an automatic deformation method in order to achieve occlusion free camera view. Our method solved several different occlusion problems during egocentric data exploration, which were less attended by previous studies. The automation allowed users to focus on their exploration tasks, without having to put extra effort to manipulate the camera. Our system monitored the camera and the data in a state transition model, and reacted accordingly with smooth animations. Various data types and exploration tasks were studied to demonstrate the advantages of our method.

Chapter 5: Object-In-Hand Feature Displacement Using Physically-Based Deformation

5.1 Introduction

Scientific visualization plays an instrumental role in many applications by presenting spatial datasets in a visually accessible form. However, it is very frequent that important data features are located in positions that are hard to see in the spatial domain. When projected to a 2D screen, the intuitive photorealistic placement of data features may not always work the best regarding to information presentation. For example, a region of interest can be hidden behind some obstacles; multiple interesting features may cover each other; or two features may locate far away that makes the visual comparison difficult. With the goal of making more effective visualization, various kinds of manipulation can be conducted beyond the basic perspective projection of the data.

Data deformation is one such manipulation that directly changes the shapes or locations of data elements [13]. It has been used in various applications to solve different kinds of problems. Its major advantage compared to other choices of visual manipulation, such as using cutaway and transparency, or using multiple viewpoints, is that it avoids placing extra visual load by adopting a uniform transfer function in the whole domain, as well as keeping the perspective viewpoint [29]. Deformation methods can be classified into physically-based

and non-physically-based [23]. Physically-based methods embed the physical properties of the data into the deformation by mechanical laws, which is commonly realized by using a mesh. Physically based deformation grants users more power to control the deformation by manipulating each individual cube or tetrahedron of the mesh. More background about the physically-based deformation has been given in Chapter 1 and 3.

In this chapter, we propose an interactive system to efficiently displace features using physically-based deformation, to achieve an *object-in-hand* effect. Similar ideas of deforming an object as if using a person's two hands have driven non-physically-based designs before, such as cracking or peeling metaphors based on affine transformation or trigonometric functions. However, with the power of a physically-based deformable mesh, we are able to displace data elements in more sophisticate and flexible ways. Benefiting from this advantage, we design interaction tools that enable users to reshape the data by real hands through a touchscreen input. Users can move multiple fingers on the touchscreen to conduct dragging, and data elements under the fingers will move along with the fingers. At the same time, users can use other fingers to press and hold on the touchscreen, in order to make the shapes and locations of the underneath data elements unchanged during the deformation. Our system also supports cutting on the data by using a finger to draw the incision, which can either make the dragging only affect desired area, or separate an interesting feature off from the main body of the data. Since one user can use multiple fingers performing different operations with different configurations (e.g., dragging direction) at the same time, their combinations can create a large space of deformation effects, which can be hard to achieve using previous physically-based deformation systems that initiated deforming forces from only one circular or rectangular model [54, 70, 72, 81, 82].

Using the proposed object-in-hand alike operations, we can displace interesting data features from a visually cluttered region to nearby empty spaces to make them less occluded; we can also displace far away data features to a closer distance for easier visual comparison. By incorporating local data attributes into the deformation mesh, the displacing operations present smooth and realistic processes. We studied how this is done for various types of datasets, including particle data, volumetric data, and streamlines. We also study how these data types differ in the methods to deform the data along with the mesh. In the end, we present case studies using each of these data types, and successfully achieved improved visual cognition of data features by our deformation system.

This work has been submitted to IEEE Pacific Visualization Symposium in 2019.

5.2 Deformation Algorithm

The overall design idea of our deformation system is to manipulate the dataset as if it was an object in a person's hands. This idea has served the design of several popular interaction systems, such as the *scene-in-hand* camera control [83]. If applied for data deformation, a human's multiple fingers should be able to control the data shape in a fine level by flexible ways. Therefore, we firstly build our system on a touchscreen to empower the fingers of the operator; next, to naturally spread the influence of the fingers to the data, a physically-based deformable mesh will be employed. By properly making use of the touchscreen and handling the settings of the mesh, we are able to simulate several actions of real hands, listed as below:

- Drag: Users can move data elements along with the finger movement, as if dragging an object in hands using fingers.

- Hold: Users can press data elements from being deformed, as if holding an object to protect it from being pulled away.
- Cut: Users can cut on the dataset, as if cutting an object with a knife.

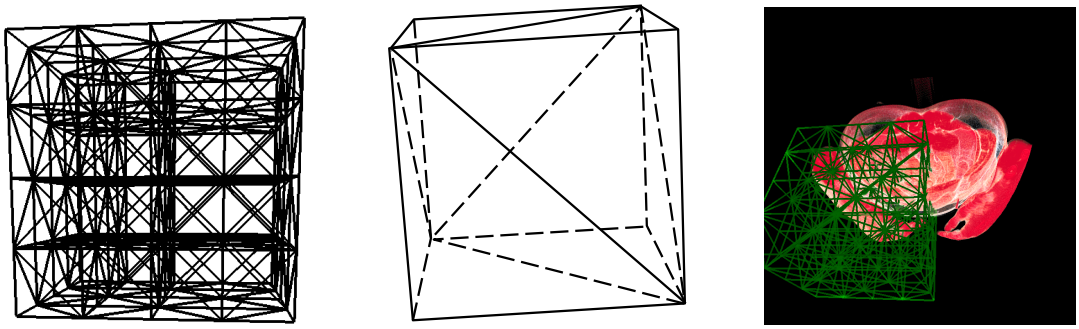
These are three basic operations of our deformation system, which are used in combination to achieve various deformation effects. The dragging operation is the fundamental one among the three since it initiates the deformation, while the other two are optional. The holding operation is to preserve the shape and location of certain part of the data from being affected by the dragging operation. The cutting can also control the extent of the dragging's influence, while it can further utilize the dragging result for feature separation too.

In the following, we will first introduce how to construct the mesh in Section 5.2.1. Then in the next three subsections, we present how to make use of the physically-based mesh to realize the three basic operations, i.e., Drag, Hold, and Cut.

5.2.1 Mesh Construction

The first step of our data manipulation process is to place a mesh that covers an interesting region. We adopt a structured tetrahedral mesh as in Figure 5.1a. This type of mesh is stacked by cubes of the same size, and each cube is further divided into 5 tetrahedra as in Figure 5.1b. Its shape is decided by the resolution (number of cubes) along each of the three dimensions, and the spacing indicated by the length of each cube. Without loss of generosity, we place the three axes of the mesh parallel to the three axes of the data domain.

The location, resolution, and spacing are all parameters that users can adjust, and the related interaction schemes are given later in Section 5.3.1. Due to the fact that the extent of the mesh depicts the region where the deformation will occur, the goal of adjusting the location and the shape of the mesh is to cover the feature of users' interest. Figure 5.1c



(a) Overall mesh. (b) Individual cube. (c) Mesh Placement.

Figure 5.1: Placing the mesh. (a) The overall cube stacking of the structured mesh. (b) The five tetrahedra inside one cube. (c) An example of placing the mesh covering the target feature.

gives an example, that when we plan to drag the arm of the data, we place the mesh that covers the arm. Leaving certain margin around the feature is also preferred, in order to accommodate the deformed shape of the feature. Note that the mesh resolution in Figure 5.1c is intentionally reduced for the purpose of easier view. Since we aim to achieve fine operation of local features, our mesh spacing needs to be small. This is also the reason why we do not use a fixed global mesh that covers the whole domain, since in that case the computational speed will be too low.

After the location and the shape of the mesh have been set, the next step is to set the physical properties of the mesh. There are three major properties we exploit to control the deformation effect and realize our design. The first property is the tetrahedra stiffness, which is operation-dependent but data-related, and the algorithm for this part will be introduced in Section 5.4.1. The other two properties are the external forces and the fixed vertices, which are decided by users' finger operations. Here we briefly discuss the concepts of them, before we move to more details of the operations in later subsections.

The external forces are forces added to the mesh vertices to drive the deformation. The forces make the vertices leave their original locations, and update their new locations at every frame. When the vertices move away, the internal inertia of the mesh will generate counteracting internal forces that tend to offset the movement of the vertices, until a balanced state is achieved. The internal mechanics of the mesh has been widely studied in computer graphics area, and in our work we adopted the projective dynamic method from [79].

The fixed vertices are those vertices whose locations are forced to remain the same during the deformation. The most necessary usage of this setting is to fix the vertices on the boundary of the mesh to secure its overall location. Otherwise, the deformation will eventually end up as the translation of the whole mesh along the direction of the added force. We record these vertices by a set $S_{boundary}$ as:

$$S_{boundary} = \{v \mid v \ on \ the \ boundary\}, \tag{5.1}$$

and set them fixed.

5.2.2 Simulate Dragging Actions

The dragging operation starts when a user presses a finger on the touchscreen, as the example shown in Figure 5.2a. We first find the contacted data element underneath the touch point, for which the details will be introduced later in Section 5.3.1. We model the contacted part of the data by a small spherical region, and name such sphere as *force sphere*. In order to indicate the sphere to users but not occlude too much data, we draw the sphere as a circle, as the blue circle under the finger in Figure 5.2a. Assume no overlap happens between different force spheres, since the corresponding fingers will never overlap.

The dragging operation will create external forces upon the mesh vertices. Specifically, the force on to each vertex is decided by two factors, the force spheres, and the finger

positions. Denote each force sphere by S_i^f and their set as $S^f = \{S_i^f\}$, where $i \in [0,N)$, with N being the number of force spheres; denote the position information of each moving finger by P_i and their set as $P = \{P_i\}$, where $i \in [0,N)$ too. For any vertex v, denote the contribution from the two factors, S^f and P, by $F_{sphere}(v,S^f)$ and $F_{finger}(v,P)$ respectively, then we can compute its force by

$$F(v) = \alpha \times F_{sphere}(v,S^f) \times F_{finger}(v,P), \tag{5.2}$$

where α is a constant scalar to control the magnitude in an overall manner. In the following we analyze how the two factors determine the value of $F_{sphere}(v,S^f)$ and $F_{finger}(v,P)$.

First we present how the force spheres decide $F_{sphere}(v,S)$ in Equation 5.2. If a vertex is outside of all the force spheres, we want the force on it to be zero; and if the vertex is inside of any force sphere, we want the force to have a positive magnitude. Therefore for a vertex v at location $p(v)$, we have

$$F_{sphere}(v,S^f) = \begin{cases} 0, & \text{if } \{i \mid p(v) \ in \ S_i^f\} = \emptyset \\ f(v,S_I^f), & \text{if } \{i \mid p(v) \ in \ S_i^f\} = \{I\} \end{cases} \tag{5.3}$$

where $f(v,S_I^f)$ is a positive scalar, and because of the non-overlapping between spheres, at most one sphere S_I^f can be found such that v is inside of it. We could have set $f(v,S_I^f)$ simply to be constant 1. However, data inside a sphere may not always be isotropic. For example in Figure 5.3, when the sphere center c_I^f is on a high density feature, the left bottom of the sphere is covering background region, and the right top part has reached another feature. Therefore we may not want to assign vertices B or C high force magnitude, while keeping the force magnitude on vertex A high. To achieve this goal, we initiate the value $f(v,S_I^f)$ as 1, and then sample a series of locations on the segment connecting the vertex v and the center c_I^f, as the green segments in Figure 5.3. Then at these sample locations we retrieve the tetrahedra stiffness $\{s_0,s_1,...,s_j,...\}$ there, which is data-dependent as defined

later in Equation 5.7, and can be an indicator of the data isotropy. If any sample stiffness s_j is less than the stiffness $s(c_I^f)$ at the sphere center, we decay the value $f(v, S_I^f)$ by a ratio of $s_j/s(c_I^f)$. Now taking the example in Figure 5.2a, when the user presses the arm of the data, the deformation will not degenerate if the force sphere unintentionally intersects the abdomen a little.

Next we present how the finger movement decides $F_{finger}(v,P)$ in Equation 5.2. The dragging action for a finger i starts by an initial 3D location denoted P_i^{init}, and then the finger moves along the touchscreen to a new location P_i^{cur}, as the example shown in Figure 5.2c and 5.2d. We interpret this dragging action as users' intention to move the data element at location P_i^{init} to location P_i^{cur}. Similarly, for a vertex at other locations $p(v)$ inside the force sphere, users intent to move them with the same trend computed by $P_i^{cur} - P_i^{init}$, like the arrow shown in Figure 5.2c. Therefore, its target location should be $p_{target}(v,i) = p(v) + (P_i^{cur} - P_i^{init})$. When the current deformed location of this vertex $p_d(v)$ has not reached this target, the force added on this vertex will point to the target from $p_d(v)$, as shown in Figure 5.2d. Thus we have

$$F_{finger}(v,P) = \begin{cases} 0, & \text{if } \{i \mid p(v) \ in \ S_i^f\} = \emptyset \\ p_{target}(v,I) - p_d(v), & \text{if } \{i \mid p(v) \ in \ S_i^f\} = \{I\} \end{cases} \qquad (5.4)$$
$$p_{target}(v,I) = p(v) + (P_I^{cur} - P_I^{init})$$

and similar with in Equation 5.3, at most one sphere S_I^f can be found that encloses v due to the non-overlapping of spheres. This force not only gives a direction following the finger trace, but also tunes the force magnitude by the current distance to the target. During the dragging, the initial location P_i^{init} will remain the same, but the current location P_i^{cur} is updated at every frame. When the vertex v is driven closer to its target location, the magnitude of its force will decrease. But please be advised that v will never reach its target location, since at that time the force $F_{finger}(v,P)$ will be zero, therefore cannot counteract

the internal resistance of the mesh. Figure 5.2b gives an example, where the force sphere is rendered by the green circle, and its final balanced location is between the finger position and its initial location. An extra benefit of this choice is to reduce the chance of the data being occluded by users' fingers, as the dragged data in Figure 5.2b being visible.

5.2.3 Simulate Holding Actions

When a person has an object in hands and uses one hand to drag parts of it, she can usually use the other hand to support the object and protect other parts of the object from being altered. To simulate this situation in our system, when a finger on the touchscreen has never moved since the beginning of touching, it will protect the shapes and locations of contacted data elements from being changed by other moving fingers. Similar to force spheres, we model the contact between a finger and the data using a sphere named by *anchor sphere* S_i^a. Implementing the holding effect is by setting fixed vertices of the physically-based mesh. Mark the vertices inside the anchor spheres using the set S_{anchor} as:

$$S_{anchor} = \{v|\ \exists i\ s.t.\ p(v)\ in\ S_i^a\}. \tag{5.5}$$

Combining with the already fixed boundary vertices as Equation 5.1, we have the set of fixed vertices as

$$S = S_{anchor} \cup S_{boundary}. \tag{5.6}$$

It is worth noting that if we treat an anchor sphere as the same with a force sphere, under our force computation method, the steady finger will also generate forces to resist the movement of underneath mesh vertices. But this can only reduce the change but not fully eliminate it. Our choice using the fixed vertices guarantees to prevent changing the corresponding parts of the mesh.

5.2.4 Simulate Cutting Actions

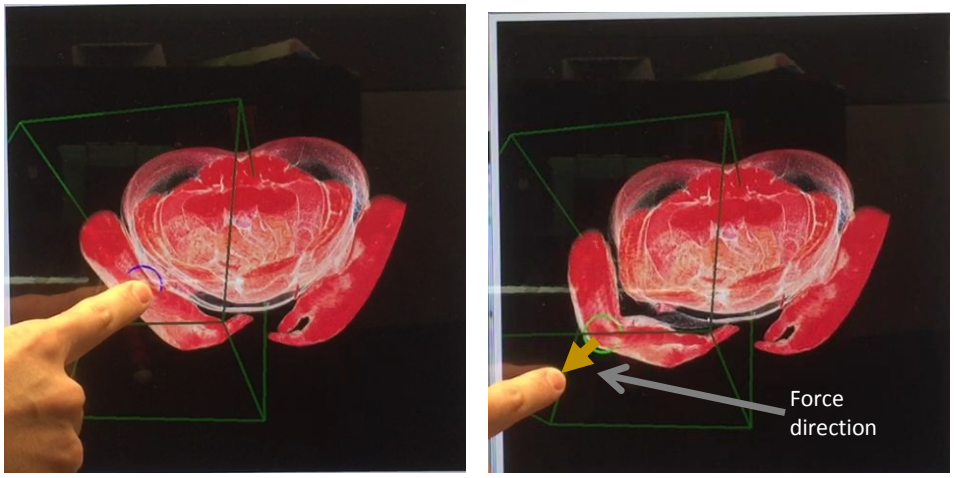
When a person tries to tear an object in hands but finds it difficult only by dragging, she may seek the help of a knife to achieve the goal. Our cutting operation simulates this situation. The cutting is done by drawing a curve on the screen, as the yellow curve in Figure 5.4a. The trace of curve will first be transformed into a Bezier curve. Then the 2D Bezier points are projected along the view rays to form a surface in the 3D domain. The mesh will be cut by this surface, which means the topology of the mesh will be changed. Cutting the mesh precisely along the surface can be very time consuming due to the need of adding new tetrahedra onto the current mesh structure. To keep our system interactive, we simplify our mesh restructure by only separating existing cubes of the mesh, without adding new tetrahedra. This way keeps the time cost low, and it will not reduce the visual quality in our applications, because our cutting action is mostly performed between multiple features, where it is usually less dense and frequently visualized as the background color.

The cutting actions designed in our system has two different modes. The first mode is a partial cut, for which when projecting the 2D curve on screen to a 3D surface, the surface does not go through the whole mesh, but stops at an adjustable depth. The surface with the yellow boundary in Figure 5.4b gives an example. As a result, the top part of the mesh is broken, and forces added on one side of the mesh are hard to impact the other side. Therefore the partial cut can protect the data on the other side from being deformed, similar to the holding operation. Another way to use the partial cut is to explicitly separate two connected features and enlarge the space between them, so we can achieve less occluded view.

The other mode is the full cut, for which the cutting surface goes through the mesh. In this case, the original mesh is cut into two parts, as the mesh shown in Figure 5.4d.

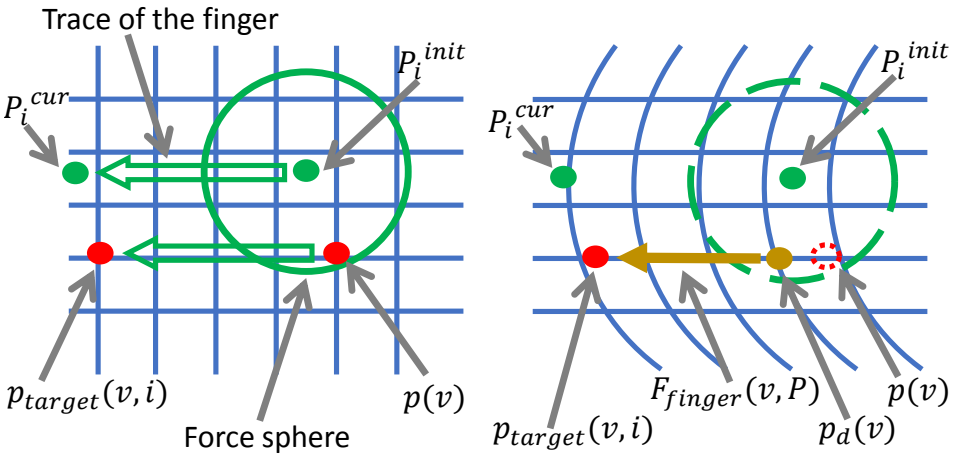
The difference from a partial cut is, the full cut is used to fully segment an interesting feature away from the whole dataset. After the segmentation, we can conduct translation and rotation only on the segmented feature to reorganize the whole view. Figure 5.4e gives an example, where after separating one arm of the data, we move it nearby the other arm to do visual comparison of them. The segmented feature was in a deformed shape at the time of cutting, but the visual comparison may prefer the original shape. To deform the feature back after the cut, we keep the external force added upon this segment, but remove all its vertices from the set of fixed vertices defined in Equation 5.6. Without restrictions from the boundary or the anchor spheres, a vertex v can eventually move to its target $p_{target}(v,I)$ as in Equation 5.4, and leave the force magnitude as zero. The final deformation effect of the segmented feature is same with a translation from the original position by $P_i^{cur} - P_i^{init}$ as the example in Figure 5.4c, but with a smooth deforming animation.

When combined with dragging, our cutting module can be more effective than regular clip planes. The original space between two interesting features can be curvy, making it hard to segment them directly using a clip plane on the original data. Such as the original view in Figure 5.2a, the tight space between the arm and the abdomen is located above the leg part in the data, so directly placing a clipping plane there will also cut the leg and cannot do the segmentation cleanly. Our deformation can enlarge the space by dragging, so the space is not fully clogged by the leg behind as in Figure 5.4a, and the separation by using a planer cut becomes possible. More examples will be shown later in the case studies.



(a) Original data.

(b) Deformed data.



(c) Original mesh.

(d) Deformed mesh.

Figure 5.2: The trace of the finger movement creates the force that pointing from the current locations of the affected data elements, towards their target locations decided by the current finger position. The frame of the mesh covered region is rendered as the green frame in (a), (b), and other demonstration figures in the chapter. Dash shapes in (d) represent the locations before deformation.

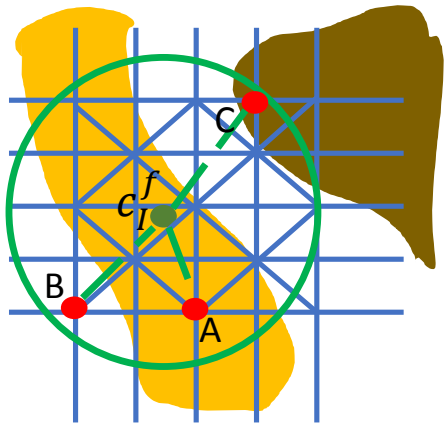
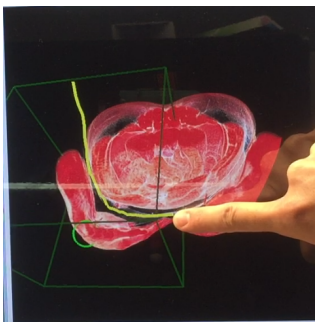
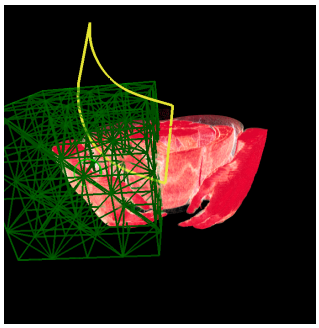


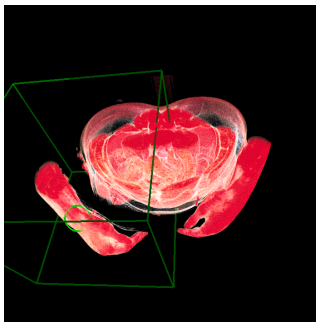
Figure 5.3: Illustrating the computation of $F_{sphere}(v, S^f)$ as in Eqaution 5.3. Vertices inside the sphere but are weakly connected to the center (e.g., vertices B and C) are given decayed force magnitude.



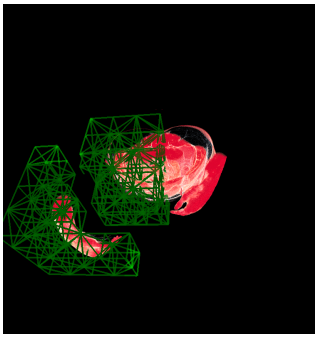
(a) Perform cut by one finger.



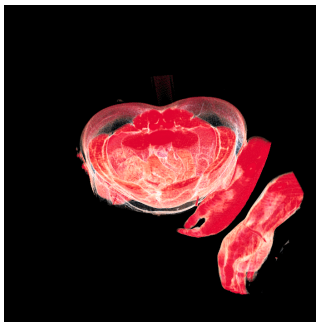
(b) A particle cut.



(c) Result of a full cut.



(d) Mesh splited by a full cut.



(e) Realign the segmented feature.

Figure 5.4: Illustrating the cutting operation. The mesh resolutions in (b) and (d) are intentionally reduced for easier view.

5.3 Interactions

In last section we mainly introduced the underlying algorithm of how the operations affect the deformation. In this section, we focus on the specific interaction schemes of conducting these operations on our system.

5.3.1 The Workflow on Touchscreen

The overall workflow to interactively manipulate features using our system is plotted as Figure 5.5. The cycle of operations begins by placing the mesh at the desired location. Next users will place fingers on the touchscreen, then started to drag to deform the data. After releasing fingers, users can adjust the deformation by further dragging, or conduct cutting on the data. If a full cut is performed, users can make transformation upon the segmented feature. In the end, users can accept the current deformed shape of the data, and continue the data manipulation by creating a new mesh to repeat the cycle again. We also store important states of the data to support rolling back. In the next, several major steps in the workflow will be introduced with more details.

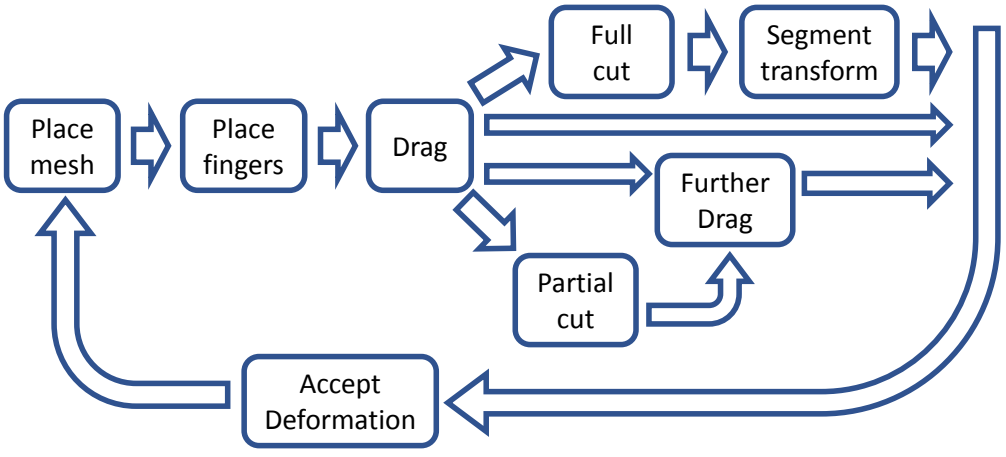


Figure 5.5: The workflow of using our system on a touchscreen.

5.3.1.1 Placing The Mesh

Our system asks users to place a mesh by clicking the touchscreen. The mesh is centered at the position of the finger click. Here we first introduce how to transform the 2D screen touch point into a 3D location in the domain, which will also be used later to identify where a dragging finger is indicating.

Identifying 3D Location from Screen Position. When a finger touches the screen, we can get a 2D coordinate in the screen space. We can compute its coordinate in the camera space, and find the ray from the camera towards this pixel. Then we find the first data element along the ray, as shown in Figure 5.6a. For direct volume rendering using ray casting, we march along this ray until the first location that the blended opacity has reached a user defined threshold. For sparse data representations such as particles or streamlines, we search for data elements within a small range of the touch point, as the green line in Figure 5.6a, and adopt the least deep one. This method of location selection is straightforward and works well in our applications. Our work is not focused on sophisticate feature selection, although advanced methods may benefit our work too. If no data element has been found along this ray, our system will perform other interactions, such as view matrix rotation.

By default, only the boundary frame of the mesh will be rendered, as the green frame in Figure 5.2a and other demonstration figures. To adjust the location of the mesh, users can press the frame and drag. Changing the size of the mesh can be done by either changing the resolution or the spacing, so these two operations are done by separate slider bars in an auxiliary panel to avoid ambiguity.

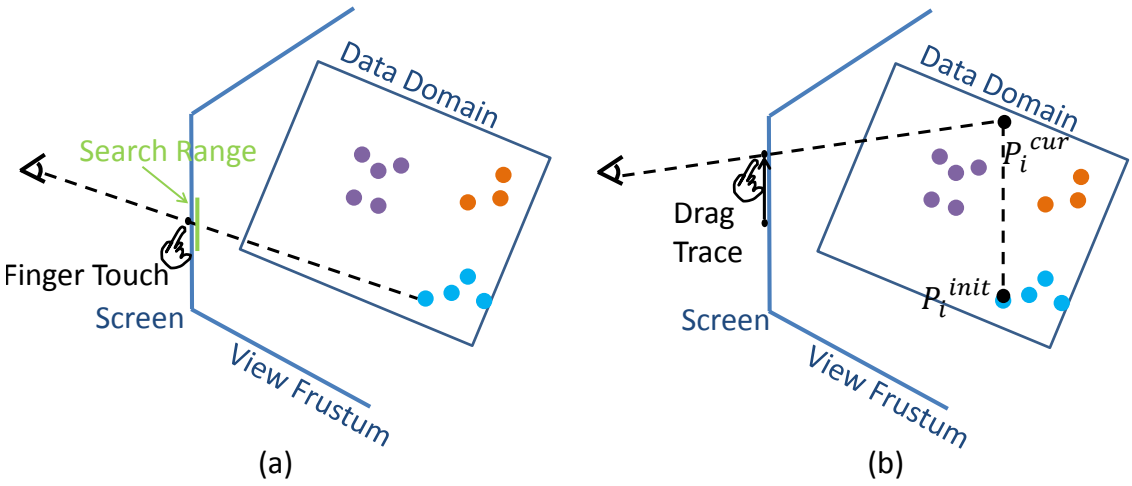


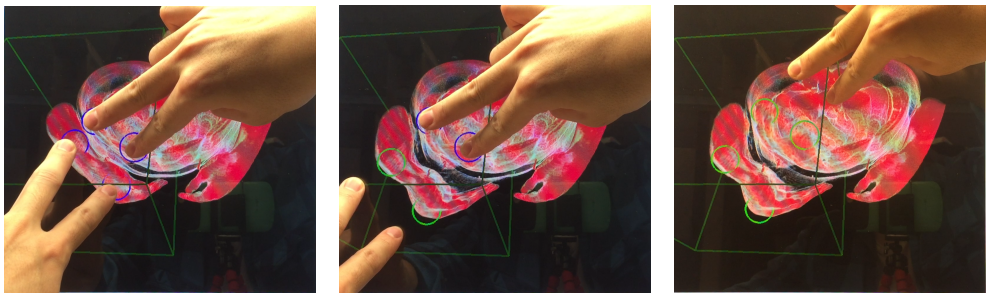
Figure 5.6: Two major situations to find the 3D location from a 2D touch point. (a) Find the 3D location of the first hit data element on the ray. (b) Find the 3D location on the ray that has the same depth as the initial dragging location.

5.3.1.2 Dragging And Holding Interactions

The dragging interaction starts with identifying the 3D location P_i^{init} of the initial finger touch position from its 2D screen space coordinate, which is done using the same way as placing the mesh. After moving, the new finger position P_i^{cur} is set to have the same clip depth with P_i^{init}, as the example in Figure 5.6b.

When users put several fingers on the screen at the beginning and before any finger is moving, we do not know whether a finger is used to do dragging or holding. Therefore, at first all spheres are treated as anchor spheres. As an example in Figure 5.7a, where four fingers are touching the screen, four anchor spheres are rendered underneath using blue circles. Then, any finger that moves on the screen will transfer its corresponding sphere into a force sphere, and mesh vertices inside the sphere will be added force and move together with the finger. The example in Figure 5.7b shows the left two fingers moving to drag, and

there corresponding spheres become force spheres rendered by green circles. At the same time, two other steady fingers preserve their anchor spheres. This sphere transformation is single-directed, which means a force sphere cannot be transferred back to an anchor sphere, because its interior mesh vertices have already deformed and cannot comply with the concept of the anchor sphere. As a result, the maximum times of sphere type transformation is a constant (no more than ten times), leading to an acceptable computational cost.



(a) (b) (c)

Figure 5.7: The process of dragging. (a) Before any finger moves, all touched fingers create anchor spheres. (b) Moving fingers turn their anchor spheres into force spheres. (c) During further dragging, adjustment can be made by dragging current force spheres, or dragging anchor spheres to turn them into force shperes.

After a finger doing dragging is released, its last finger position is recorded to compute $P_i^{cur} - P_i^{init}$, which continues to act as the force to preserve the deformed shape in later frames, without users' endeavor to retain the fingers forever.

Users can fine tune the shape by conducting further dragging multiple times, too. When users touch one or more force spheres and drag them, their stored P_i^{cur} are also updated by the new finger movement, so underneath data elements are also moved by the new dragging. We do not allow adding a new force sphere at this stage, since the underlying mesh has

already been deformed, so the affected region does not comply with the spherical concept. But our system accepts dragging an anchor sphere to turn it into a force sphere at this stage, since their underlying mesh has never been changed. In the example in Figure 5.7c, we dragged the two anchor spheres to turn them into force spheres.

5.3.1.3 Cutting Interactions

The cutting is started by clicking a button in the auxiliary panel. The mode of full cut or partial cut, and the particle cut depth, can also be tuned from the panel. The cut is performed by drawing a curve, and we automatically lengthen or shorten its two ends to make the ends just reach the boundary of the mesh area. After a partial cut, users can still conduct further dragging in the same way. After the full cut, the data will deform back to the original shape as analyzed in Section 5.2.4, and after that, we don't do physically-based deformation before entering the next workflow cycle. On the contrary, we support translating and rotating the segmented feature, so users can displace it in a large range to do view comparison with other parts of the data. The translation and rotation interactions can be the same with translating or rotating the whole domain, and which one to interact is decided by whether the finger touch location is on the segment.

5.3.2 Compare with Other Interaction Devices

We have also implemented our system using traditional mouse plus keyboard devices. Most interactions using a mouse are similar to the interactions using one finger. However, besides following more closely to the object-in-hand metaphor, another advantage of using touchscreen becomes obvious when multi-finger operations cannot be translated to mouse interactions. Firstly, users need to place each anchor sphere and force sphere one by one after placing the mesh, before starting dragging. This costs several more operations. Secondly,

each force sphere needs to be dragged one by one, which also costs many extra operations, and loses the smooth transition of an intact deformation. An alternative is to use the same dragging direction for all force spheres, which can make the transition smooth, but reduce the amount of possible effects that can be achieved.

There are still certain advantages of using mouse, too. Besides the easier access, using mouse to set the locations of the mesh, anchor spheres, and force spheres can be more precise. Our method illustrated in Figure 5.6a can be less precise in situations where the depth values of local data elements vary fiercely. In this case, a mouse can reduce the chance of wrong location selections.

There is still a limitation of our current system that exists on both the touchscreen and the mouse devices. These devices are still 2D input devices so they lack the ability to easily handle the depth dimension. One issue is the force is mostly added at surface area of the data. Although extra adjustment of the force spheres can be done to lay force at deeper parts of the data, this hurts the smoothness of the whole workflow. Another shortage it causes is the inconvenience to add vertical force on force spheres. Although our horizontal dragging is useful in many scenarios, when a vertical dragging is desired, users need to rotate the domain first to make the planned force direction horizontal, and then conduct the dragging. A possible solution to this limitation is to explore 3D input devices, which will be a future work of us.

5.4 Data-Mesh Engaging

5.4.1 Data-Driven Tetrahedra Stiffness

The stiffness of the mesh tetrahedra is an important property that directly affects the deformation behavior of the mesh. A tetrahedron with high stiffness is harder to be deformed,

so its shape can be better preserved. On the contrary, a tetrahedron with low stiffness is easier to be deformed, so it is generally stretched or compressed more than the tetrahedra with high stiffness values. The principle to set the data-driven tetrahedra stiffness is, to make the data elements creating heavier visual clues in the visualization covered by more stiff tetrahedra. Thus when being deformed, the shapes of such data elements are affected less. This is firstly more authentic, since when manipulating a real object in hands, the denser part of the object is often harder to be changed than the softer part. It also better keeps the shape of data elements with more visual clues, which are usually more important.

Inspired by the attribute-driven stiffness setting framework proposed in [54], we set the stiffness $s(t)$ of a tetrahedron t by:

$$s(t) = s_0 + \beta \times d(t)^{\gamma}, \tag{5.7}$$

where s_0 is a constant to avoid the stiffness being too low, and β and γ are polynomial coefficients empirically set. $d(t)$ is an estimate of the density of visual clues of the region covered by a tetrahedron. It is computed according to the specific type of the data. For each data type used in this study, we list our choice as below:

- Particle data: we use the count of particles covered by the tetrahedron as $d(t)$.
- Volumetric data: we compute $d(t)$ by the opacity of the covered voxel values in the adopted transfer function.
- Streamlines: we use the sum of lengths of streamlines intersecting this tetrahedron, weighted by the local curvature of the streamlines, since more curvy traces may contain more information.

One situation of our work different from previous studies is that, we are using a local mesh to manipulate features, so the spacing of the mesh can be very small. Therefore there

can be huge variance of stiffness values for neighboring tetrahedra, especially for sparse representations such as the particle data and streamlines. In such cases, we use gaussian smoothing to process the stiffness values of neighboring tetrahedra, to make the mesh behavior more stable.

5.4.2 Deforming The Data According To The Mesh

By now we have discussed the process of deforming the mesh, and in this subsection we present how to deform the data according to the deformation of the mesh. As introduced before, for mesh based deformation, the displacement function $\vec{D}()$ in Equation 1.1 is defined by the mapping of the mesh base element, which is the tetrahedron here. For data element λ at location $P(\lambda)$, suppose it lays inside the tetrahedron T before the deformation with barycentric coordinate $b_o(P(\lambda))$. The key idea is, when the tetrahedron T is deformed and its barycentric coordinate function changes to $b_d(P)$, we will make the new location of λ, $P'(\lambda)$, have the same barycentric coordinate regarding to T, i.e., $b_d(P'(\lambda)) = b_o(P(\lambda))$.

Based on the above reasoning, for each data type used in this study, we give the method to compute the deformed locations of data elements as below:

- Particle data: The deformed location of a particle λ can be directly derived as $P'(\lambda) = b_d^{-1}(b_o(P(\lambda)))$, where $b_d^{-1}()$ is the inverse function of $b_d()$.
- Volumetric data: To make the deformed volume smooth, we first build a new volume at the same resolution. Then we rasterize each deformed tetrahedron T, and for any voxel λ it encloses at $P'(\lambda)$, find the corresponding location before deformation by $P(\lambda) = b_o^{-1}(b_d(P'(\lambda)))$, then assign λ with the value at $P(\lambda)$ in the original volume. When a part of the volume is deformed out of the data domain, we create an extra

small volume, to process and render the protruding part together with the whole volume.

- Streamlines: Normally the nodes of a streamline are processed as the particle data, and the polyline topology is kept unchanged. If a streamline intersects the cutting plan after a cutting operation, it will break down into shorter new lines, which also requires duplicating the nodes at the break points. If the intersection happens several times on a streamline, we keep the topology between the first and last intersection points, therefore at most three new lines will be generated.

5.5 Case Study

In this section we show case studies conducted on three different types of datasets. We present the figures using screen shots to avoid possible block by the operator's hands.

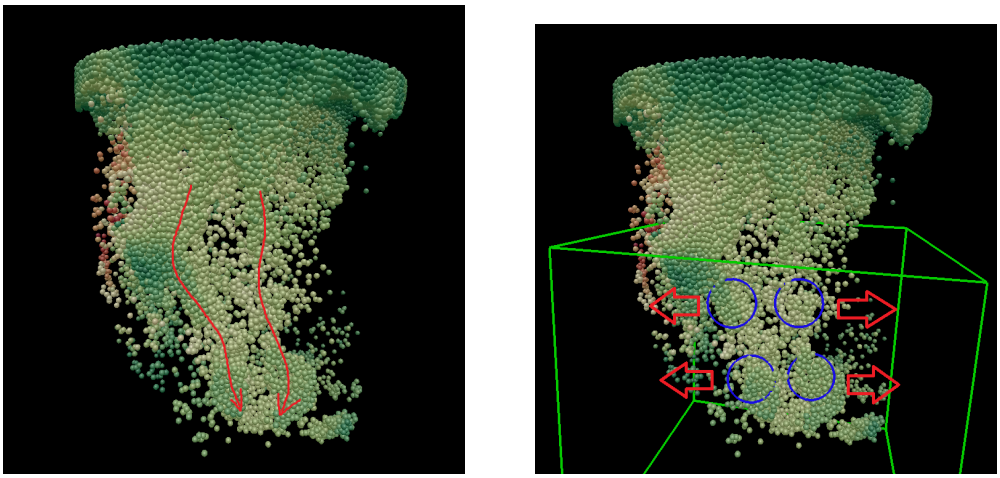
5.5.1 Particle Data

In this section, we apply our system upon a viscous fluid simulation dataset [3]. The data is generated by the finite pointset method, to describe the dissolving process of salt in water in a cylinder container, when the salt was added from the top of the container. Close by particles can form a long-shaped structure called viscous finger, which is of big interest to researchers. However, viscous fingers have highly varying shapes, and the inter-finger activities may further complicate their final appearance, making the visual comparison hard.

We first render a time snap of the simulation as in Figure 5.8a. We color the particles by its speed, where green particles are the slowest, white ones are faster, and red ones are the fastest. We notice two viscous fingers are very close to each other, marked by the two red curves in the figure. If we want to separate them using a cutting plane, we need to be

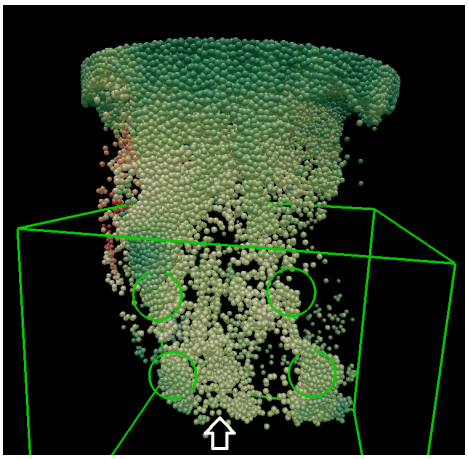
cautious if the cutting plane can correctly handle the particles behind the surfaces of the two fingers. A few of these underneath particles can be seen from the tiny split of the two fingers. From this view it is hard to tell if these particles belong to the left finger or the right one, so directly placing a cutting plane, or a fish-eye lens, to enlarge the split may not handle these particles properly. Using our system, a user can drag these two viscous fingers in opposite directions to separate them, using four fingers of two hands, as the red arrows shown in Figure 5.8b. At the moment when just touching the screen, all four fingers of the user create anchor spheres as the blue circles in the figure. Then the fingers start to move, turning the anchor spheres into force spheres, and successfully separate the two viscous fingers as in Figure 5.8c. Now we can discover that the left finger has a larger thickness, and most of the underneath particles, as pointed by the white arrow in the figure, belong to the left finger instead of the right finger. This is discovered naturally because our method utilizes the hidden connection relations between particles by encoding them into tetrahedra stiffness, and correctly keeps the affiliation during the deformation.

We then render another time snap as in Figure 5.9a. We notice a checkmark-shaped viscous finger as circled in the figure. There is another finger at the opposite side of the domain whose shape is very similar to this one, but due to their locations, it is impossible to make these two fingers visible at the same time. We then hope to cut this finger off and move it closer to the opposite one for visual comparison. This finger is closely attached to two upstream fingers both on its left top and its right top, so directly cutting it off is hard. We then hold the two upstream fingers, and drag this finger to the left bottom as in Figure 5.9b. As a result, the originally very small space between the checkmark-shaped finger and its upstream fingers is enlarged, with only two narrow links left, so we can easily do a full cut operation as the yellow curve in the figure. After the cut, the separated viscous finger



(a) Two close features.

(b) Drag in opposite directions.

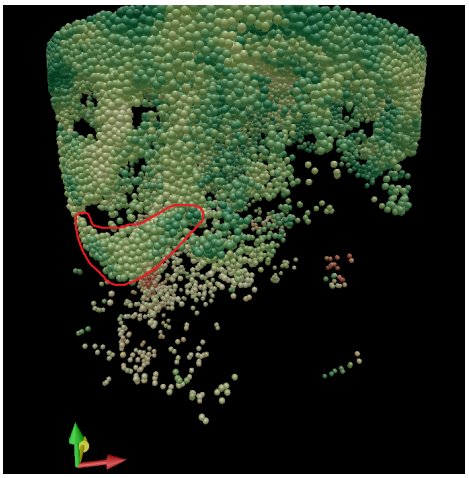


(c) Two features separated.

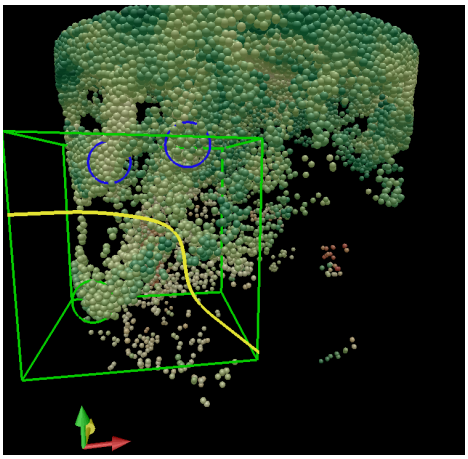
Figure 5.8: An example of using our method on the viscous fluid dataset.

will slowly recover its original shape as in Figure 5.9c. Then we displace it to the other side of the domain using affine transformations conducted by users' dragging and rotating. We place it as pointed by the arrow in Figure 5.9d. The cartesian coordinate system represented by three arrows from the origin point in the figures helps indicate that we are now seeing the other side of the domain. The target viscous finger to compare with is circled by red. Now since these two fingers are placed side by side, we can easily observe that they have similar

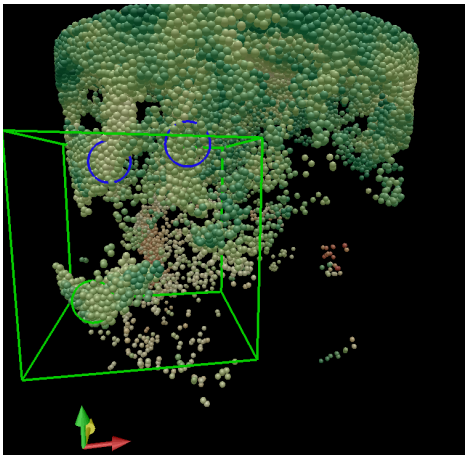
sizes and shapes, but the target finger has increasing particle speed from left to right, but for the first finger, particles at the right end have lower speed values.



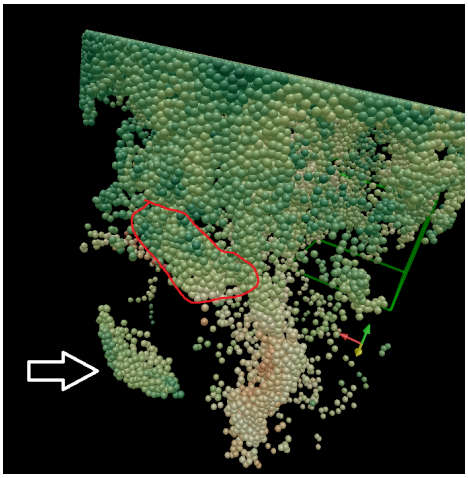
(a) A checkmark-shaped feature.



(b) Drag the feature away from upstreams, then cut.



(c) After the cut, the feature deforms back to its original shape.



(d) Move the feature close to another checkmark-shaped feature for comparison.

Figure 5.9: The second example of using our method on the viscous fluid dataset.

5.5.2 Volume Data

In previous sections when describing the algorithms and interaction schemes, we have used a dataset from the Visible Human Project, led by U.S. National Library of Medicine. It is the pelvis part of a male cadaver, with a size of $512\times512\times140$.

Next we demonstrate another case study using an MRI scan of a tomato, created by Information and Computing Sciences Division at Lawrence Berkeley Laboratory. The size of the data is $256\times256\times164$. We cut it at the middle position of the x axis, and get a half tomato as Figure 5.10a. From our culinary experience we know that a tomato contains several empty cells inside, which are named *locules*. Figure 5.10b shows the result after using our deformation method, from which we can find three locules in this dataset. However, in the original view as Figure 5.10a, only the right locule is visible, but the left and middle one are surrounded by *septa* (the plural form of *septum*) or *placenta*, which block our view to observe the locules. Since the locules are distributed irregularly, changing the cutting plane may not help. Also the septa not only exist in front of a locule but also on its back, so making the septa transparent cannot display the whole locule correctly.

In order to view the left locule of the data without occlusion, we first dragged the bottom part of the left septum through the force spheres in Figure 5.10c to flip it up as in the figure. At the same time, since the inner pericarp does not occlude our view but forms the wall of the locule, we hold a finger on it as an anchor, shown by the blue circle in Figure 5.10c. Now we accept the current deformation, and enter the next workflow cycle to flip up the top part of the septum, as Figure 5.10d. But as in the figure, the top part of septum is attached to the inner pericarp much more closely than the bottom part. Therefore, using anchor spheres may not be enough to hold the inner pericarp unchanged. So we conduct a partial cut as the yellow curve in Figure 5.10e to break the connection. Now with the break in the mesh,

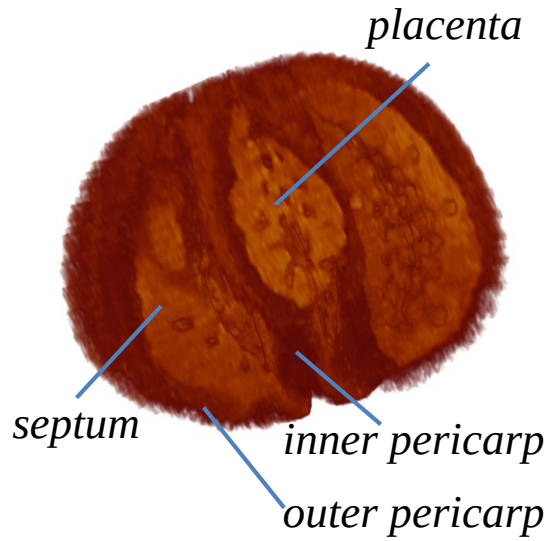
the force spheres on the left of the cut will not affect the inner pericarp on the right, so the inner pericarp will slowly revert back to its original shape in Figure 5.10f; meanwhile, we can further drag the top part of the septum to continue enlarging the distance, since it has also been freed from the constraint of the inner pericarp. In the end, the blocking septum ends with a shape similar to the results of natural hand squeezing, which conforms the object-in-hand design; at the same time, the shape of the neighboring inner pericarp is little affected due to the holding and cutting operations, which better helps us understand the shape of this locule. Following the similar steps as in Figure 5.10g and 5.10h, we flip up the placenta by dragging and cutting to reveal the middle locule. In the end, we gain visual access to all three locules.

5.5.3 Streamline Data

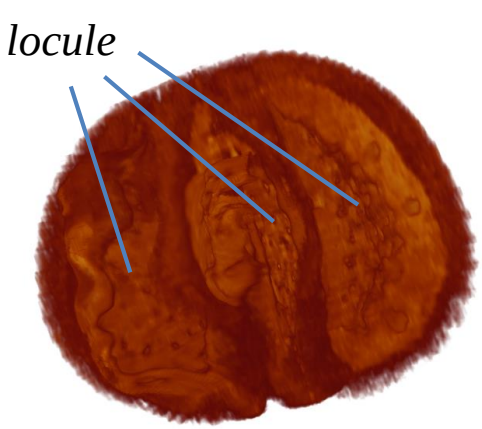
In this section we demonstrate applying our method on the streamlines traced from the Plume dataset. The dataset is a flow simulation describing the thermal down flow occurring on the surface of the sun, created by the National Center for Atmospheric Research. We generated 558 streamlines from uniformly distributed seeds in the top part of the domain. The tracing is computed by the ParaView software, with the Runge-Kutta 4-5 method in both the forward and backward directions. We color the traces by line curvature as in Figure 5.11. There are many small size vortices attached to the central bundle of long streamlines, and Figure 5.11a shows one of them marked by the brown circle. However, these vortices are blocked by many other streamlines, and are far away from each other for comparison.

In Figure 5.11b, we press the middle finger of the right hand to hold the vertex to remain its location, as the magenta circle shown in the figure. At the same time, use the index and ring finger of the right hand, and two fingers of the left hand, to drag the long streamlines in

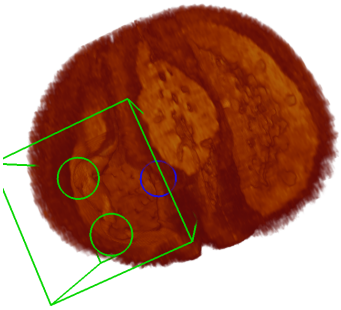
front of the vertex as if plucking a guitar's strings, which reduces the occlusion they caused. Then we switch to another view as Figure 5.11c, which also benefits from the displacement of the occluding long streamlines; but more importantly, we can press the anchor sphere covering the vortex, and drag it to the new location as in Figure 5.11d. At this location, we can easily do a cut to split it off the original bundle, which would be very hard to cut in the original position without interfering nearby streamlines. After the cut, both the vortex and the main body of the data will recover to the original shapes, as in Figure 5.11e. We can then move this vortex freely in the domain. In Figure 5.11f, we move it to the side of another vortex marked by the brown circle. Now we can easily sense that the new vortex is larger in size than the original one, but the original one has higher curvature in the center, indicated by deeper red color.



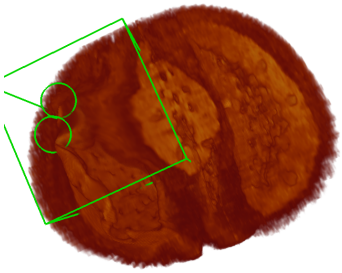
(a) The original data.



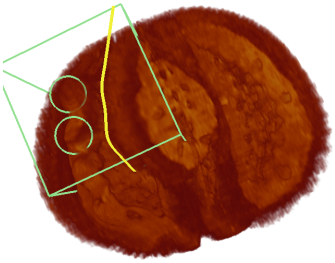
(b) Final deformed shape.



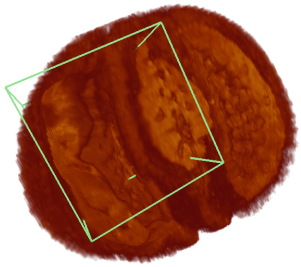
(c) Flip up the bottom septum while hold the inner pericarp.



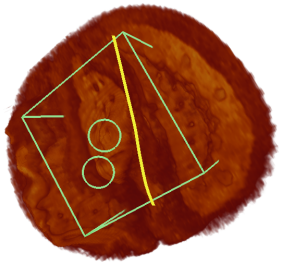
(d) Flip up the top septum while pulling the inner pericarp together.



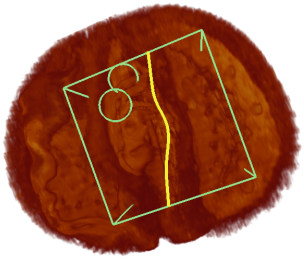
(e) Cut to separate the septum and the inner pericarp.



(f) Reveal the left locule.

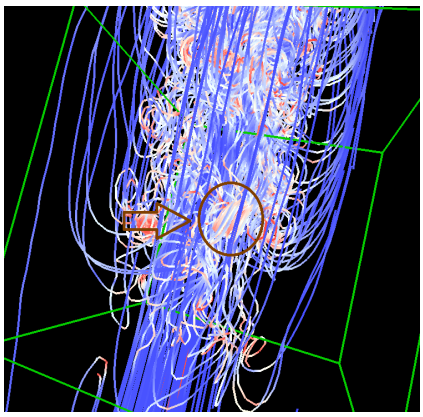


(g) Flip up the bottom placenta.

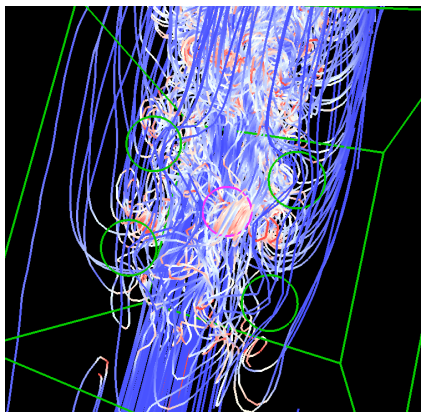


(h) Flip up the top placenta.

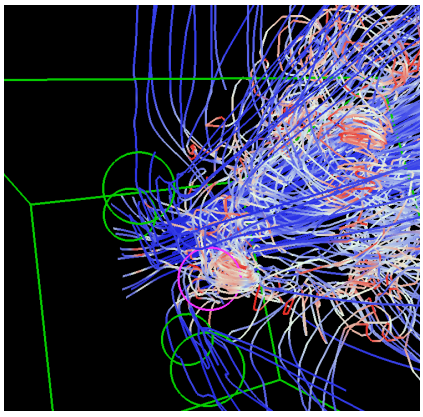
Figure 5.10: The example of using our method on a scan data of tomato. (c)-(h) are inner steps to achieve the final deformed shape as (b).



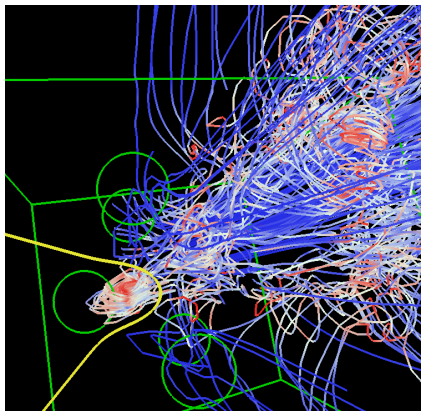
(a) A smaller vortex covered by other streamlines.



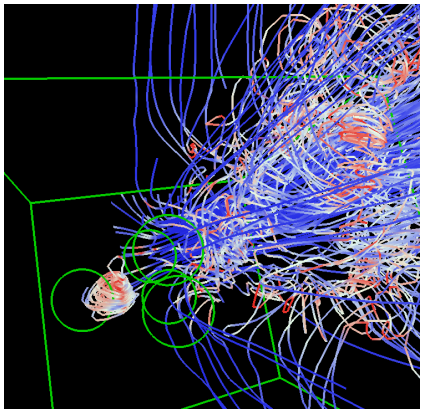
(b) Hold the vortex while displacing occluding lines.



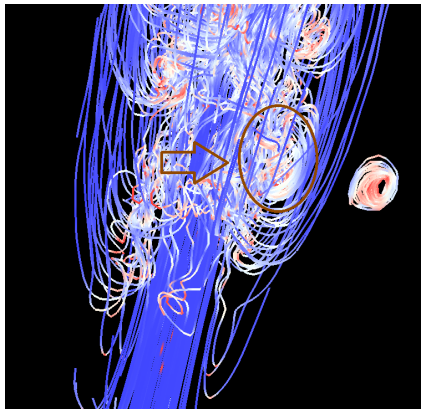
(c) Another view to drag the vortex.



(d) Drag and cut the vortex.



(e) After the cut, all parts deform back but keep separated.



(f) Displace the vortex for comparison with another vortex.

Figure 5.11: The example of using our method on streamlines of the plume data.

5.6 Discussion

In this chapter, we proposed our method to displace features by the power of a physically-based mesh. A feature in a dataset can be neighbored with complex structures that reduce the visualization quality, but there is usually certain amount of space between the feature and the neighbors. One fundamental capability of our physically-based dragging operation is that it can utilize the tiny and irregular empty spaces, by naturally enlarging them for less occlusion or feature separation. Previous works [72, 74] placed a lens in such empty spaces, but our system grants more freedom by using multiple fingers and different kinds of finger operations. Moreover, although our cutting operation mathematically breaks a connecting mesh so it can work on non-empty regions, but in our case studies, we still mostly cut on empty or sparse regions between features to separate them, in order to serve our major goal of feature displacement. We do not claim that the cutting operation is designed for cutting one single feature into two parts, or other applications based on this, which follows a different research track.

One limitation of our work has been discussed in Section 5.3.2, which is about the weakness on the depth dimension due to the 2D essence of the touchscreen. Our future work will be extended to 3D input devices, which is expected to better simulate the object-in-hand situations. Besides handling the depth dimension easier, using 3D input devices may also be able to simulate more types of movements of human hands beyond the current finger based operations. For example, we can model the force from the whole palm to simulate a push action.

Another future work direction is to lessen the burden of placing a local mesh. We will study heuristic algorithms to automatically decide a local mesh that can be satisfying in most of the time. In that case, users' feeling of object-in-hand can be more authentic.

5.7 Conclusion

Based on the continuing research on empirical deformation, in this chapter, we have proposed our advance on utilizing physically-based deformation to manipulate a dataset as an ductile object. The adoption of the touchscreen enabled users to displace data features with real hands, which enhanced the feeling of object-in-hand. We described our method using the word *dragging* due to the uniform directed force applied on one force sphere. But as we have shown in our case studies, the deformation can also achieve effects similar to squeezing, twisting, plucking, gripping, and various other actions, by properly combining multiple and manyfold dragging operations. We also strengthened our system with the holding operation to help preserve the shape of desired regions, as well as the cutting operation to help further separate features. The results showed that by using our deformation system, we could achieve less occlusion or better layout, for the visualization of different types of datasets.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

Compared to simply visualizing basic data elements on the screen, data manipulation is frequently pursued to improve the visibility of important or interesting features. Data deformation has been one of such manipulation strategies ever since the very early stage of visualization research history. However, its heavy computation cost hindered interactive applications in early days. Its popularity only started about a dozen of years ago, together with the improvement of parallel-capable hardware. In my opinion, the research of deformation in visualization is still a young field, and possesses many challenges as well as opportunities.

In this dissertation, three new deformation techniques have been proposed to tackle several new challenges of feature visibility issues in modern research, and make contributions to this field in various aspects. Firstly, I proposed a deformation technique that encodes data attributes into the mesh to conduct the physically based deformation. In this way, data features can lead to stiffer mesh elements, so they can be better preserved during the deformation. Secondly, I proposed an automatic deformation technique to solve the occlusion when exploring the dataset in an egocentric mode. By checking the relative status of the virtual camera and the data, occlusions can be automatically removed, which will not interrupt user's data exploration operations that happen at the same time. Thirdly, I

proposed a deformation technique that is conducted using fingers on a touchscreen. This allows users to deform the data as if handling a real object in hands, which not only makes the interactions more natural and intuitive, but also utilizes the creativity of humans' finger movement. All proposed techniques are integrated into newly designed systems running at interactive rates. Various case studies have been demonstrated, using different types of scientific datasets and different data exploration applications, to consolidate the effectiveness of the proposed techniques.

6.2 Future Work

The future research directions for each proposed technique have been presented in the individual chapter corresponding to each technique. In this section, I discuss some future research topics that are related to multiple techniques or general deformation research in visualization area.

Deformation on Time Varying Datasets. Applying deformation upon time varying datasets is still a challenging problem. The fundamental reason is that interactive deformation techniques need to show the deformation process with animations, while visualizing time varying datasets can also highly rely on animations. The second proposed technique in this dissertation provided an application on a time varying blood cell simulation dataset and combined the two animations, which to the best of my knowledge, is the first trial along this direction in visualization area. However, there is still no uniform way of combining the two sources of animation. Therefore, many deformation techniques, including the first and the third one proposed in this dissertation, cannot be directly applied to time varying datasets. In the future, some case-by-case solutions may need to be developed for specific applications. Moreover, it will be more beneficial to design a widely applicable framework

that can combine the two sources of animation, so general deformation techniques can be easily incorporated onto time varying datasets.

Deformation in Virtual Environment. When a person has an object in hand, it is usually much easier for the person to change its shape, than change the color or transparency, or view the object in a non-perspective way. This fact tells that compared to other visual manipulation methods, deformation has a larger potential to be conducted in a more real way. The road to achieving this goal follows two branches: improving the computation, and improving the interaction. The first branch can be advanced by the physically-based deformation, which has a short history and still has much to be exploited. The second branch is promoted by utilizing modern human-computer interaction devices. The third proposed technique in this dissertation combines these two branches using a touchscreen. However, devices with more DOF (degree of freedom) than the touchscreen, such as 3D input devices, are still less adopted for deformation. It can be expected to achieve much more authentic deformation in future research, by either improving one of the two branches, or combining and enhancing them together.

More Affordable Deformation. Accompanying the big power of deformation, it has a relatively high cost when being applied in actual scientific research, compared to alternatives such as using transparency. The causes are multifold. Firstly, most interactive deformation methods depend on parallel devices. For example, all deformation techniques proposed in this dissertation are experimented using CUDA, which requires NVIDIA GPUs. The techniques can be transferred to other parallel devices, but the extra programming cost to fit an unknown target platform can be high. Secondly, as a young and less exploited area, deformation techniques lack a systematic grammar, making it hard to design a uniform programming library like VTK, or a general purpose tool like PARAVIEW, to broadly

promote the application of deformation. All these problems can be studied in the future, in order to reduce the cost of scientists to apply deformation easily in their daily research, with minimal effort of programming required.

Bibliography

[1] Human connectome project, https://www.humanconnectome.org/.

[2] VIS Contest 2015, http://darksky.slac.stanford.edu/scivis2015/.

[3] VIS Contest 2016, http://www.uni-kl.de/sciviscontest/.

[4] Maneesh Agrawala, Denis Zorin, and Tamara Munzner. Artistic multiprojection rendering. In *Rendering Techniques 2000*, pages 125–136. 2000.

[5] Ryan Bane and Tobias Hollerer. Interactive tools for virtual x-ray vision in mobile augmented reality. In *Mixed and Augmented Reality, 2004. Third IEEE and ACM International Symposium on*, pages 231–239, 2004.

[6] Lonni Besançon, Paul Issartel, Mehdi Ammi, and Tobias Isenberg. Hybrid tactile/-tangible interaction for 3d data exploration. *IEEE transactions on visualization and computer graphics*, 23(1):881–890, 2017.

[7] Lonni Besançon, Paul Issartel, Mehdi Ammi, and Tobias Isenberg. Mouse, tactile, and tangible input for 3d manipulation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 4727–4740. ACM, 2017.

[8] Eric A Bier, Maureen C Stone, Ken Pier, William Buxton, and Tony D DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 73–80, 1993.

[9] Åsmund Birkeland and Ivan Viola. View-dependent peel-away visualization for volumetric data. In *Proceedings of the 25th Spring Conference on Computer Graphics*, pages 121–128, 2009.

[10] Stefan Bruckner and M Eduard Gröller. Exploded views for volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1077–1084, 2006.

[11] Michael Burns and Adam Finkelstein. Adaptive cutaways for comprehensible rendering of polygonal scenes. In *ACM Transactions on Graphics*, volume 27, page 154, 2008.

[12] M Sheelagh T Carpendale, David J Cowperthwaite, and F David Fracchia. Extending distortion viewing from 2d to 3d. *IEEE Computer Graphics and Applications*, 17(4):42–51, 1997.

[13] Min Chen, C Correa, Shoukat Islam, Mark W Jones, P-Y Shen, Deborah Silver, Simon J Walton, and Philip J Willis. Manipulating, deforming and animating sampled object representations. In *Computer Graphics Forum*, volume 26, pages 824–852. Wiley Online Library, 2007.

[14] Min Chen, Deborah Silver, Andrew S Winter, Vikas Singh, and N Cornea. Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, pages 35–44. ACM, 2003.

[15] Rui CH Chiou, Arie E Kaufman, Zhengrong Liang, Lichan Hong, and Miranda Achniotou. Interactive path planning for virtual endoscopy [colon ct]. In *Nuclear Science Symposium, 1998. Conference Record. 1998 IEEE*, volume 3, pages 2069–2072. IEEE, 1998.

[16] Luca Chittaro and Ivan Scagnetto. Is semitransparency useful for navigating virtual environments? In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 159–166. ACM, 2001.

[17] Marc Christie, Patrick Olivier, and Jean-Marie Normand. Camera control in computer graphics. In *Computer Graphics Forum*, volume 27, pages 2197–2218. Wiley Online Library, 2008.

[18] Dane Coffey, Nicholas Malbraaten, Trung Bao Le, Iman Borazjani, Fotis Sotiropoulos, Arthur G Erdman, and Daniel F Keefe. Interactive slice wim: Navigating and interrogating volume data sets using a multisurface, multitouch vr interface. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1614–1626, 2012.

[19] Daniel Cohen-Or, Yiorgos L Chrysanthou, Claudio T. Silva, and Frédo Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, 2003.

[20] Carlos Correa, Debora Silver, and Mi Chen. Illustrative deformation for data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1320–1327, 2007.

[21] Carlos Correa, Deborah Silver, and Min Chen. Feature aligned volume manipulation for illustration and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1069–1076, 2006.

[22] Carlos Correa, Deborah Silver, and Min Chen. Constrained illustrative volume deformation. *Computers & Graphics*, 34(4):370–377, 2010.

[23] Carlos D Correa, Deborah Silver, and Min Chen. Discontinuous displacement mapping for volume graphics. In *Eurographics/IEEE VGTC Workshop on Volume Graphics*, pages 9–16, 2006.

[24] Carolina Cruz-Neira, Daniel J Sandin, and Thomas A DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the cave. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 135–142, 1993.

[25] Jian Cui, Paul Rosen, Voicu Popescu, and Christoph M Hoffmann. A curved ray camera for handling occlusions through continuous multiperspective visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1235–1242, 2010.

[26] Joachim Diepstraten and Thomas Ertl. Interactive cutaway illustrations. *Computer Graphics Forum*, 22(3), 2003.

[27] Niklas Elmqvist, Ulf Assarsson, and Philippas Tsigas. Employing dynamic transparency for 3d occlusion management: Design issues and evaluation. *Human-Computer Interaction–INTERACT 2007*, pages 532–545, 2007.

[28] Niklas Elmqvist and Philippas Tsigas. View-projection animation for 3d occlusion management. *Computer Graphics*, 31(6):864–876, 2007.

[29] Niklas Elmqvist and Philippas Tsigas. A taxonomy of 3d occlusion management for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1095–1109, 2008.

[30] Shiaofen Fang, Su Huang, Rajagopalan Srinivasan, and Raghu Raghavan. Deformable volume rendering by 3d texture mapping and octree encoding. In *Visualization'96. Proceedings.*, pages 73–80. IEEE, 1996.

[31] Miquel Feixas, Mateu Sbert, and Francisco González. A unified information-theoretic framework for viewpoint selection and mesh saliency. *ACM Transactions on Applied Perception (TAP)*, 6(1):1, 2009.

[32] Paul Fischer, James Lottes, David Pointer, and Andrew Siegel. Petascale algorithms for reactor hydrodynamics. In *Journal of Physics: Conference Series*, volume 125, page 012076, 2008.

[33] Andrew Forsberg, Michael Katzourin, Kristi Wharton, Mel Slater, et al. A comparative study of desktop, fishtank, and cave systems for the exploration of volume rendered confocal data sets. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):551–563, 2008.

[34] Jorge Gascon, Jose M Espadero, Alvaro G Perez, Rosell Torres, and Miguel A Otaduy. Fast deformation of volume data using tetrahedral mesh rasterization. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 181–185, 2013.

[35] Claudia Hänel, Benjamin Weyers, Bernd Hentschel, and Torsten W Kuhlen. Visual quality adjustment for volume rendering in a head-tracked virtual environment. *IEEE Transactions on Visualization and Computer Graphics*, 22(4):1472–1481, 2016.

[36] Lichan Hong, Shigeru Muraki, Arie Kaufman, Dirk Bartz, and Taosong He. Virtual voyage: Interactive navigation in the human colon. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 27–34, 1997.

[37] Christophe Hurter, Russ Taylor, Sheelagh Carpendale, and Alexandru Telea. Color tunneling: Interactive exploration and selection in volumetric datasets. In *2014 IEEE Pacific Visualization Symposium*, pages 225–232, 2014.

[38] Tobias Isenberg. Interactive exploration of three-dimensional scientific visualizations on large display surfaces. In *Collaboration Meets Interactive Spaces*, pages 97–123. Springer, 2016.

[39] Shoukat Islam, Deborah Silver, and Min Chen. Volume splitting and its applications. *IEEE Transactions on Visualization and Computer Graphics*, 13(2), 2007.

[40] Bret Jackson, Tung Yuen Lau, David Schroeder, Kimani C Toussaint, and Daniel F Keefe. A lightweight tangible 3d interface for interactive visualization of thin fiber structures. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2802–2809, 2013.

[41] Jacek Jankowski and Martin Hachet. Advances in interaction with 3d environments. In *Computer Graphics Forum*, volume 34, pages 152–190. Wiley Online Library, 2015.

[42] Guangfeng Ji and Han-Wei Shen. Dynamic view selection for time-varying volumes. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1109–1116, 2006.

[43] Kenrick Kin, Maneesh Agrawala, and Tony DeRose. Determining the benefits of direct-touch, bimanual, and multifinger input on a multitouch workstation. In *Proceedings of Graphics interface 2009*, pages 119–124. Canadian Information Processing Society, 2009.

[44] Tijmen Klein, Florimond Guéniat, Luc Pastur, Frédéric Vernier, and Tobias Isenberg. A design study of direct-touch interaction for exploratory 3d scientific visualization. In *Computer Graphics Forum*, volume 31, pages 1225–1234, 2012.

[45] Sebastian Knödel, Martin Hachet, and Pascal Guitton. Interactive generation and modification of cutaway illustrations for polygonal models. In *Smart Graphics*, pages 140–151. Springer, 2009.

[46] Toru Koyama, Hiroshi Okudera, and Shigeaki Kobayashi. Computer-generated surgical simulation of morphological changes in microstructures: Concepts of "virtual retractor" technical note. *Journal of Neurosurgery*, 90(4):780–785, 1999.

[47] Yair Kurzion and Roni Yagel. Interactive space deformation with hardware-assisted rendering. *IEEE Computer Graphics and Applications*, 17(5):66–77, 1997.

[48] Bireswar Laha, Doug A Bowman, David H Laidlaw, and John J Socha. A classification of user tasks in visual analysis of volume data. In *Scientific Visualization Conference (SciVis), 2015 IEEE*, pages 1–8, 2015.

[49] Bireswar Laha, Doug A Bowman, and John J Socha. Effects of vr system fidelity on analyzing isosurface visualization of volume datasets. *IEEE Transactions on Visualization and Computer Graphics*, 20(4):513–522, 2014.

[50] Bireswar Laha, Kriti Sensharma, James D Schiffbauer, and Doug A Bowman. Effects of immersion on visual analysis of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 18(4):597–606, 2012.

[51] Eric LaMar, Bernd Hamann, and Kenneth I Joy. A magnification lens for interactive volume visualization. In *pg*, page 0223. IEEE, 2001.

[52] Bo Li, Xin Zhao, and Hong Qin. Four-dimensional geometry lens: A novel volumetric magnification approach. In *Computer Graphics Forum*, volume 32, pages 122–133. Wiley Online Library, 2013.

[53] Cheng Li, Joachim Moortgat, and Han-Wei Shen. An automatic deformation approach for occlusion free egocentric data exploration. In *Pacific Visualization Symposium (PacificVis), IEEE*, pages 215–224. IEEE, 2018.

[54] Cheng Li, Xin Tong, and Han-Wei Shen. Virtual retractor: An interactive data exploration system using physically based deformation. In *Pacific Visualization Symposium (PacificVis), 2017 IEEE*, pages 51–60. IEEE, 2017.

[55] Wilmot Li, Lincoln Ritter, Maneesh Agrawala, Brian Curless, and David Salesin. Interactive cutaway illustrations of complex 3d models. *ACM Transactions on Graphics*, 26(3):31, 2007.

[56] Julian Looser, Mark Billinghurst, and Andy Cockburn. Through the looking glass: the use of lenses as an interface tool for augmented reality interfaces. In *Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, pages 204–211, 2004.

[57] Claes Lundstrom, Thomas Rydell, Camilla Forsell, Anders Persson, and Anders Ynnerman. Multi-touch table system for medical visualization: Application to orthopedic surgery planning. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1775–1784, 2011.

[58] Jun Ma, James Walker, Chaoli Wang, Scott Kuhl, and Ching Kuang Shene. Flowtour: An automatic guide for exploring internal flow features. In *Visualization Symposium (PacificVis), IEEE Pacific*, pages 25–32, 2014.

[59] Michael J McGuffin, Liviu Tancau, and Ravin Balakrishnan. Using deformations for browsing volumetric data. In *Visualization, 2003. IEEE*, pages 401–408.

[60] Joachim Moortgat. Viscous and gravitational fingering in multiphase compositional and compressible flow. *Advances in Water Resources*, 89:53–66, 2016.

[61] Voicu Popescu, Paul Rosen, and Nicoletta Adamo-Villani. The graph camera. In *ACM Transactions on Graphics*, volume 28, page 158, 2009.

[62] Christof Rezk-Salama and Andreas Kolb. Opacity peeling for direct volume rendering. In *Computer Graphics Forum*, volume 25, pages 597–606, 2006.

[63] Warren Robinett and Richard Holloway. Implementation of flying, scaling and grabbing in virtual worlds. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 189–192. ACM, 1992.

[64] Ebrahim Shahraeeni, Joachim Moortgat, and Abbas Firoozabadi. High-resolution finite element methods for 3d simulation of compositionally triggered instabilities in porous media. *Computational Geosciences*, 19(4):899–920, 2015.

[65] Xing Shi, Guang Lin, Jianfeng Zou, and Dmitry A Fedosov. A lattice boltzmann fictitious domain method for modeling red blood cell deformation and multiple-cell hydrodynamic interactions in flow. *International Journal for Numerical Methods in Fluids*, 72(8):895–911, 2013.

[66] Hendrik Sollich, Ulrich von Zadow, Tobias Pietzsch, Pavel Tomancak, and Raimund Dachselt. Exploring time-dependent scientific data using spatially aware mobiles and large displays. In *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces*, pages 349–354. ACM, 2016.

[67] Henry Sonnet, Sheelagh Carpendale, and Thomas Strothotte. Integrating expanding annotations with a 3d explosion probe. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 63–70, 2004.

[68] Erik Sundén, Ingemar Lundgren, and Anders Ynnerman. Hybrid virtual reality touch table: An immersive collaborative platform for public explanatory use of cultural

objects and sites. In *Eurographics Workshop on Graphics and Cultural Heritage*, 2017.

[69] Jun Tao, Jun Ma, Chaoli Wang, and Ching-Kuang Shene. A unified approach to streamline selection and viewpoint selection for 3d flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):393–406, 2013.

[70] Jun Tao, Chaoli Wang, Ching-Kuang Shene, and Seung Hyun Kim. A deformation framework for focus+ context flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 20(1):42–55, 2014.

[71] Xin Tong, John Edwards, Chichin Chen, H W Shen, Chris R Johnson, and Pak C Wong. View-dependent streamline deformation and exploration. *IEEE Transactions on Visualization and Computer Graphics*, 22(7):1788, 2016.

[72] Xin Tong, Cheng Li, and Han-Wei Shen. Glyphlens: View-dependent occlusion management in the interactive glyph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):891–900, 2017.

[73] Rosell Torres, Alejandro Rodríguez, José M Espadero, and Miguel A Otaduy. High-resolution interaction with corotational coarsening models. *ACM Transactions on Graphics (TOG)*, 35(6):211, 2016.

[74] Michael Traoré, Christophe Hurter, and Alexandru Telea. Interactive obstruction-free lensing for volumetric data visualization (in press). In *VIS 2018. IEEE Visualization, 2018.*, 2018.

[75] Will Usher, Pavol Klacansky, Frederick Federer, Peer-Timo Bremer, Aaron Knoll, Alessandra Angelucci, and Valerio Pascucci. A virtual reality visualization tool for neuron tracing. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):994–1003, 2017.

[76] John Viega, Matthew J Conway, George Williams, and Randy Pausch. 3d magic lenses. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, pages 51–58, 1996.

[77] Ivan Viola, Miquel Feixas, Mateu Sbert, and Meister Eduard Groller. Importance-driven focus of attention. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 2006.

[78] Simon J Walton and Mark W Jones. Volume wires: a framework for empirical non-linear deformation of volumetric datasets. 2006.

[79] Huamin Wang. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics*, 34(6):246, 2015.

[80] Lujin Wang, Ye Zhao, Klaus Mueller, and Arie Kaufman. The magic volume lens: An interactive focus+ context technique for volume rendering. In *VIS 05. IEEE Visualization, 2005.*, pages 367–374, 2005.

[81] Yu-Shuen Wang, Tong-Yee Lee, and Chiew-Lan Tai. Focus+ context visualization with distortion minimization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1731–1738, 2008.

[82] Yu-Shuen Wang, Chaoli Wang, Tong-Yee Lee, and Kwan-Liu Ma. Feature-preserving volume data reduction and focus+ context visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):171–181, 2011.

[83] Colin Ware and Steven Osborne. Exploration and virtual camera control in virtual three dimensional environments. *ACM SIGGRAPH computer graphics*, 24(2):175–183, 1990.

[84] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380, 2013.

[85] Rüdiger Westermann and Christof Rezk-Salama. Real-time volume deformations. In *Computer Graphics Forum*, volume 20, pages 443–451. Wiley Online Library, 2001.

[86] Menglin Wu and Voicu Popescu. Multiperspective focus+context visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(5):1555–1567, 2016.

[87] Lingyun Yu, Konstantinos Efstathiou, Petra Isenberg, and Tobias Isenberg. Cast: Effective and efficient user interaction for context-aware selection in 3d particle clouds. *IEEE transactions on visualization and computer graphics*, 22(1):886–895, 2016.

[88] Lingyun Yu, Pjotr Svetachov, Petra Isenberg, Maarten H Everts, and Tobias Isenberg. Fi3d: Direct-touch interaction for the exploration of 3d scientific visualization spaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1613–1622, 2010.

[89] Xin Zhao, Wei Zeng, Xianfeng David Gu, Arie E Kaufman, Wei Xu, and Klaus Mueller. Conformal magnifier: A focus+ context technique with local shape preservation. *IEEE Transactions on Visualization and Computer Graphics*, 18(11):1928–1941, 2012.