2019 IEEE Pacific Visualization Symposium (PacificVis)

Object-In-Hand Feature Displacement with Physically-Based Deformation

Cheng Li* Han-Wei Shen†

The Ohio State University

ABSTRACT

Data deformation has been widely used in visualization to obtain an improved view that better helps the comprehension of the data. It has been a consistent pursuit to conduct interactive deformation by operations that are natural to users. In this paper, we propose a deformation system following the object-in-hand metaphor. We utilize a touchscreen to directly manipulate the shape of the data by using fingers. Users can drag data features and move them along with the fingers. Users can also press their fingers to hold other parts of the data fixed during the deformation, or perform cutting on the data using a finger. The deformation is executed using a physically-based mesh, which is constructed to incorporate data properties to make the deformation authentic as well as informative. By manipulating data features as if handling an object in hand, we can successfully achieve less occluded view of the data, or improved feature layout for better view comparison. We present case studies on various types of scientific datasets, including particle data, volumetric data, and streamlines.

Index Terms: Deformation, Physically-Based, Touchscreen

1 INTRODUCTION

Scientific visualization plays an instrumental role in many applications by presenting spatial datasets in a visually accessible form. However, when projected to a 2D screen, the original placement of data features may not be the most informative presentation. For example, a region of interest can be hidden behind some obstacles; multiple interesting features may cover each other; or two features may be located far away that makes the visual comparison difficult. In order to create more effective visualization, various kinds of manipulation can be conducted beyond the basic perspective projection of the data. Data deformation is one such manipulation method that directly changes the shapes or locations of data elements [8]. It has been used in various applications to solve different kinds of problems, such as Focus+Context visualization [43], feature enhancement [12], occlusion removal [35], surgical simulation and practicing [22], etc. Compared to other choices of visual manipulation, such as using cutaway, transparency, or multiple viewpoints, using deformation avoids placing extra visual load by adopting a uniform transfer function in the whole domain, as well as keeping the perspective viewpoint [15].

Correa *et al.* [11] 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 $P'(\lambda)$. Then the displacement function $\vec{D}()$ will make

$$P'(\lambda) = P(\lambda) + \vec{D}(\lambda), \tag{1}$$

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 [11]. Non-physically-based methods are also called empirical methods [14], 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 an 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 the 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 helps data comprehension through the deformation process, since the local data attributes are encoded into the mesh to impact the deformation [27, 36]. 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 [11, 12]. However, along with the development of GPUs, much progress has been made to utilize the benefit of physically-based deformation [27, 34, 36, 43, 44].

Besides using physically-based deformation, another possible way to make the deformation interactions more physically alike is by utilizing users' hand gestures with advanced devices. For example, previous studies simulated cracking or peeling operations using non-physically-based deformation models with 3D controllers [23] or touchscreens [33]. In this paper, we combine the benefits of the physically-based deformation and users' hand gestures, to propose an interactive system that efficiently displaces features on a touchscreen to achieve an *object-in-hand* effect. 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, 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. By adopting the physically-based deformation, we are able to displace data elements in physically alike manners driven by data attributes, compared to previous hand-gesture systems using empirical deformation models [23, 33]. By allowing one user to use multiple fingers performing different operations with different configurations (e.g., dragging direction) at the same time, we 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 [27, 34, 36, 43, 44].

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 demonstrate case studies showing improved visual cognition of the data being achieved by our deformation system, and present our evaluation study conducted with domain users and show their feedback.

*e-mail: li.4076@osu.edu

†e-mail: shen.94@osu.edu

DOI 10.1109/PacificVis.2019.00011

2 Related Work

2.1 Data Deformation

We summarize the literature of deformation research in 3D visualization according to the properties of their adopted displacement function $\vec{D}()$ in Equation 1.

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 an important or interesting region, and shrinking its nearby context regions. LaMar *et al.* [24] magnified interesting regions using local affine transformation. Advanced stretegies of designing the transformation methods include the fisheye mannor [41], using conformal mapping [49], or transforming to four-dimensional space [25]. 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 [43] as well as volume rendering [44]. Similar ideas have also been studied for streamline applications [34] and glyph visualization [36]. 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 [34], particle visualization [36], and volume visualization [38]. 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 $\vec{D}()$. Compared to using continuous $\vec{D}()$ for occlusion management, this strategy does not require an existing empty space, since it can break the space to create a new empty space. McGuffin *et al.* [29] defined volume deformation by transforming a stack of layers. Correa *et al.* [11] proposed displacement mapping to describe the deformation, as the Equation 1. His later works include incremental designs of the function $\vec{D}()$, such as the one to better separate focus and context [13], and the one to better separate medical features [12]. Other interesting deformation metaphors include splitting a dataset outwards from the center [6, 19], rolling up the layer of the obstacle [5], and using a lens to inspect the internal of the data [27, 35, 36]. Li *et al.* [26] studied the occlusion problems in egocentric visualization and virtual environment.

The last category places less restrictions on $\vec{D}()$, but focuses on general questions of deformation technologies. Chen *et al.* [9] used spatial transfer functions to define deformation. Some papers worked on improving performance by using GPU textures [16, 46], or using the mapping between the original volume and the deformed volume [39]. GPU based rerasterization methods [17, 37] have also been consistently studied for physically-based volume deformation.

2.2 Scientific Visualization on Touchscreen

Studies have been done to evaluate the effectiveness of touch input compared to mouse or tangible input [4, 20], including those specifically focused on scientific datasets [48]. Many designs have been proposed to serve the exploration of scientific datasets on touch-based workstations. Reisman *et al.* [31] studied the Rotate-Scale-Translate interactions of 3D data. Isenberg [18] surveyed several scientific visualization systems using large display surfaces, such as tables and walls. Lundström *et al.* [28] designed a medical data visualization system on a large size touch table to perform surgery planning. Sultanum *et al.* [33] designed tabletop workstation for reservoir datasets. Klein *et al.* [21] designed a touch-based workstation to explore fluid dynamics data. Coffey *et al.* [10] used touch input to navigate inside a virtual reality (VR) environment. Sollich *et al.* [32] put their focus on time-varying scientific data. Exploration of three dimension datasets has also been conducted by combining touch input with tangible input [3] or pressure information [42]. Bruckner *et al.* [7] modeled the *directness* of interaction on touchscreens by the mapping between spaces. In the end, many studies provided the touchscreen as an alternative to traditional devices [47].

3 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 designs of several popular interaction systems, such as the *scene-in-hand* camera control [45]. If applied to data deformation, a user's multiple fingers should be able to control the data shape at a fine level in 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:

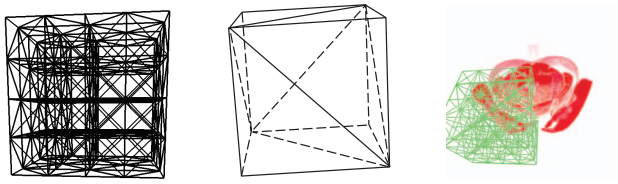
- Drag: Users can move data elements along with the finger movement, as if dragging an object using fingers.
- Hold: Users can press data elements to prevent them from being deformed, as if holding an object from being pulled away.
- Cut: Users can cut on the dataset, as if cutting an object with a knife.

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 3.1. Then in the next three subsections, we present how to make use of the mesh to realize the three basic operations.

3.1 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 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 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 generality, we place the three axes of the mesh parallel to the three axes of the data domain.



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

Figure 1: Illustration of the mesh used for deformation.

The location, resolution, and spacing are all parameters that users can adjust. Due to the fact that the mesh area 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 1c gives an example, where the data is the CT scan of the pelvis part of a male cadaver, from the Visible Human Project led by U.S. National Library of Medicine. 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 1c is intentionally reduced for the purpose of easier view. By default we only draw the frame of the mesh region to indicate the area, such as the green frame in Figure 2a. 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-independent but data-related, and the details will be introduced in Section 5.1. The other two properties are the external forces and the fixed vertices, which are decided by users' finger operations. Here we first briefly introduce the concepts of them, as the background for presenting our algorithm.

The external forces are forces added to the mesh vertices to drive the deformation. These 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 adopt the projective dynamic method [40].

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

$$S_{boundary} = \{v \mid v \text{ on the boundary}\}. \tag{2}$$

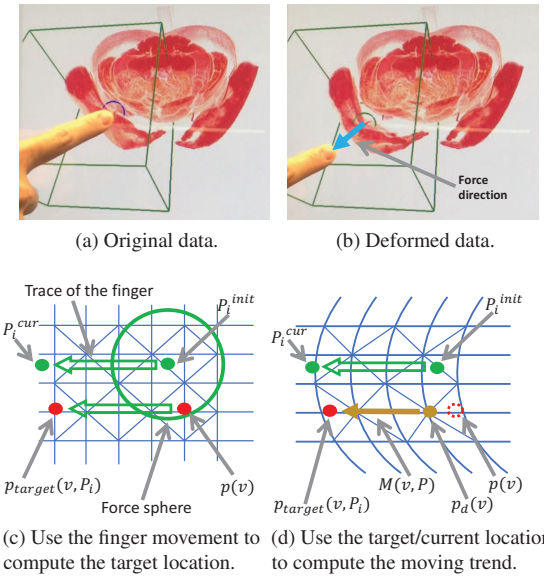
3.2 Dragging Actions

The dragging operation starts when a user presses the finger on the touchscreen, as the example shown in Figure 2a. We first find the contacted data elements underneath the touch point, for which the details will be introduced later in Section 4.1. We model the contacted part of the data by a small spherical region, and name such sphere as *force sphere*. To indicate the sphere without occluding too much data, we draw the sphere as a circle, as the blue circle under the finger in Figure 2a. Assume no overlapping of two force spheres, since the corresponding fingers will never overlap.

The requirement of our design of dragging is, for a force sphere created by a finger, data elements inside this sphere should follow the same trend of the movement of this finger; data elements outside this sphere are not directly affected by this finger. We achieve this requirement by properly design the external forces added upon the mesh vertices. The force on to a vertex v is computed by two factors, the degree of its affiliation with the force spheres, and its moving trend. The degree of affiliation with the force spheres is a scalar denoted by $d(v,S^f)$, where S^f is the set of all force spheres. The moving trend of a vertex is caused by the movement of fingers, and it is a vector denoted by $M(v,P)$, where P denotes the position information of all moving fingers. Then we can compute its force by

$$F(v) = \alpha \times d(v,S^f) \times M(v,P), \tag{3}$$

where α is a constant scalar to control the magnitude in an overall manner. When there are N moving fingers, we can represent $P = \{P_i | i \in [0,N)\}$ where P_i is the position information of each individual moving finger, and $S^f = \{S_i^f | i \in [0,N)\}$ where S_i^f represents each force sphere. Please note that although S_i^f is created at the initial finger touch position, it still models a region of data instead of any



(a) Original data. (b) Deformed data.

(c) Use the finger movement to compute the target location. (d) Use the target/current locations to compute the moving trend.

Figure 2: The moving trend of the data caused by the finger movement. (c) and (d) are cross sections of the mesh.

finger property, so it is not directly updated together with P_i. We analyze the two factors in more details in the following.

First we present how compute to the degree of affiliation $d(v,S^f)$ of a vertex v with the force spheres S^f. To satisfy our design requirement, we want a vertex to be only positively affiliated with a sphere that encloses it. Therefore we have

$$d(v,S^f) = \begin{cases} 0, & \text{v not in any force sphere} \\ f(v,S_I^f), & \text{v in force sphere } S_I^f \end{cases} \tag{4}$$

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 3, when the sphere center c_I^f is on a dense feature, the left bottom part 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 high force magnitude on vertex A . 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 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 Section 5.1, 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 an example as Figure 2a, when a user drags 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 fingers affect the moving trend $M(v,P)$ of vertex v in Equation 3. Our design requirement wants v to follow the moving trend of its corresponding finger, so we first need to model the moving trend of a finger i. Denote its initial 3D location by P_i^{init}, and at a new moment the finger moves along the touchscreen to a new location P_i^{cur}, as the example shown in Figure 2c. Thus we model the moving trend of this finger by $P_i^{cur} - P_i^{init}$, like the green arrow in Figure 2c. If following the same trend, a vertex originally

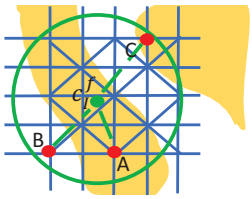


Figure 3: Illustrating the computation of $d(v,S^f)$ as in Eqaution 4.

at $p(v)$ inside the force sphere should be moved to a target location denoted by $p_{target}(v,P_i) = p(v) + (P_i^{cur} - P_i^{init})$. When its currently deformed location, denoted by $p_d(v)$, has not reached this target, the force added on this vertex will point to the target from $p_d(v)$, as shown in Figure 2d. Thus we have

$$M(v,P) = \begin{cases} 0, & \text{v not in any force sphere} \\ p_{target}(v,P_I) - p_d(v), & \text{v in force sphere } S_I^f \end{cases}$$
$$p_{target}(v,P_I) = p(v) + (P_I^{cur} - P_I^{init}) \tag{5}$$

and similar with Equation 4, 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, when the current finger location P_i^{cur} is updated, the force on the vertex is also updated to follow the new trend. 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 $M(v,P)$ will be zero, therefore cannot counteract the internal resistance of the mesh. Figure 2b gives an example, where the force sphere is rendered by the green circle, and its final balanced location is between the finger's initial and current locations. 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 2b being visible.

3.3 Holding Actions

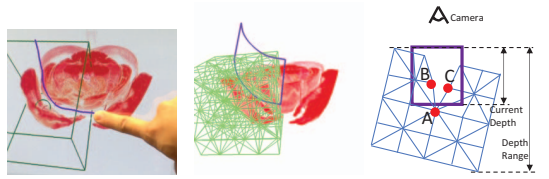
Using our force model, the dragging can affect the whole mesh region. Sometimes it is desired to keep certain data elements unchanged, regardless of their neighbors being deformed. For example, Tong *et al.* [36] implemented locking utility to keep selected glyphs always still. To simulate this situation 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 contacted data with a steady finger using a sphere named *anchor sphere*. Implementing the holding effect is by setting fixed vertices of the mesh. Combining with the fixed boundary vertices as Equation 2, we have the set of fixed vertices as

$$S = S_{boundary} \cup \{v|\ \text{v in an anchor sphere}\}. \tag{6}$$

It is worth noting that if we treat an anchor sphere as the same with a force sphere, under our force computation scheme, the steady finger can also generate forces to resist the movement of underneath mesh vertices. Also some previous research [27, 36] tuned mesh density to reduce the deformation of desired part of the mesh. These methods also work in certain circumstances, but they can only reduce the change instead of fully eliminate it. Our choice using the fixed vertices guarantees to prevent changing the corresponding parts of the mesh, and also gives users direct finger control that is closer to the object-in-hand situation we want to simulate.

3.4 Cutting Actions

While dragging will not change the topology of the mesh, previous research [27] showed that breaking a connected mesh to create discontinuity in the space can generate useful effects that cannot be achieved using connected mesh. We simulate the cutting action to break the mesh. The cutting is done by drawing a curve on the screen, as the purple curve in Figure 4a. The trace of curve will be first transformed into a Bézier curve, then projected along the view rays to form a cutting surface in the 3D domain. Changing the mesh topology 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 restructuring 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 features, where it is usually less dense and frequently visualized as the background color.



(a) Perform cut by one finger. (b) A partial cut. (c) Define the cut depth.

(d) Result of a full cut. (e) Mesh splited by a full cut. (f) Realign the segmented feature.

Figure 4: Illustrating the cutting operation. The mesh resolutions in (b) and (e) are intentionally reduced for easier view.

The cutting action 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 purple boundary in Figure 4b gives an example. Figure 4c illustrates how the depth is defined. In the camera space, the range of mesh area along the z dimension defines a range of possible depth that the bottom of the cutting plane can reach. Therefore, by normalizing this range, the depth of the cut can be denoted by a value within $[0,1]$. 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 on two sides of the cut, so we can drag them in oppositely directions more easily to achieve less occluded view. The depth can also be adjusted anytime during the exploration, after recording the camera information at the moment when the cut was created. As the example in Figure 4c, we divide Vertex A into two new vertices if the depth increases, or merge Vertex B and C if the depth decreases.

The other mode is a full cut, for which the cutting surface goes through the mesh area. In this case, the original mesh is cut into two parts, as the mesh shown in Figure 4e. For this mode, a value of cut depth is not needed. It is used to fully segment an interesting feature away from the whole dataset. Figure 4f gives an example, where after separating one arm of the data, we can conduct translation and rotation only on the arm to reorganize the whole view, and move it nearby the other arm to do visual comparison. 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 forces added upon this segment, but remove all its vertices from the set of fixed vertices defined in Equation 6. Without restrictions from the boundary or the anchor spheres, a vertex v can eventually move to its target $p_{target}(v, P_i)$ defined in Equation 5, and leave the force magnitude to zero. The final deformation effect of the segmented feature is the same with a translation from the original position by $P_i^{cur} - P_i^{init}$ as the example in Figure 4d, 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 from the original view in Figure 2a, the tight space between the arm and the abdomen is located above some tissues, so directly placing a clipping plane there will also cut those tissues and cannot do the segmentation cleanly. Our deformation can enlarge the space by dragging and make the space not clogged by the tissues behind as in Figure 4a, and the separation by using a planer cut becomes possible.

4 Interactions

In this section, we present the interaction design of our system, to conduct the operations introduced in last section.

4.1 The Workflow on Touchscreen

The main advantage of our interaction design is the flexible multi-finger collaboration through dragging and holding. For previous touch-based designs using empirical deformation, only a fixed number of fingers restricted by the deformation model can make an impact to the deformation, e.g., defining a cross section at the middle of two touch points [33]. Benefiting from the physically-based mesh, our system enables each finger to make its own impact to the deformation. Thus, the key motivation of our design is to support an arbitrary number of fingers to do dragging or holding at the same time. Other operations, such as cutting, are not necessarily to be performed concurrently with dragging.

The overall interaction strategy is summarized by a workflow plotted in Figure 5. A cycle of the workflow begins by placing the mesh at the desired location. Next users can place any number of fingers on the touchscreen, then start to drag or hold the data. After releasing fingers, users can adjust the deformation by further dragging, or conduct cutting on the data. In the end, users can accept the current deformed shape of the data, and continue the deformation by repeating the cycle again. We also store important states of the data to support rolling back, by clicking a button in an auxiliary panel. This is very important for data exploration, because before the exploration, a user may not know the exact deformation procedure to achieve the most desired visualization, and therefore, trial-and-error is frequent in actual applications. Examples of rolling back include restoring a realigned cut segment, canceling the existing cut, and canceling the existing deformation. By saving snapshots of deformed data on disk, it also supports reverting the deformation that has been adopted, so users can always restore the original shape once finishing inspecting the current region of interest. Next, several major steps in the workflow will be introduced with more details.

4.1.1 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.

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 6a. For direct volume

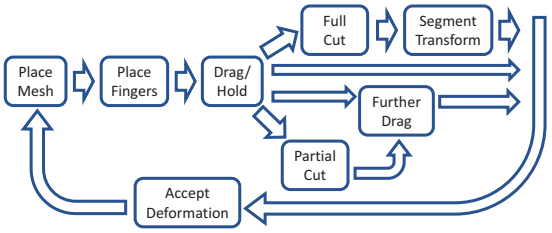
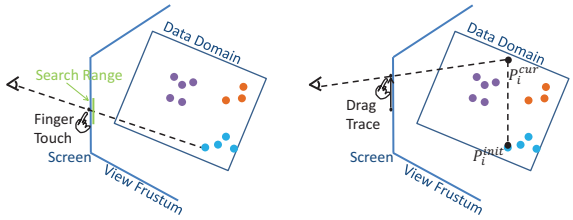


Figure 5: The workflow of using our system on a touchscreen.

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 6a, and adopt the nearest one. This method of location selection is straightforward and works well in our applications. Our work is not focused on sophisticated feature selection, although advanced methods may benefit our work too.



(a) Find the 3D location of the first hit data element on the ray.

(b) Find the 3D location on the ray with the same depth after dragging.

Figure 6: Find the 3D location from a 2D touch point.

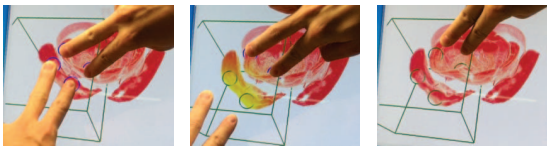
To adjust the location of the mesh, users can press the frame and drag. Changing the size of the mesh can be done by either changing the resolution or the spacing, so these two operations are done by separate slider bars in the auxiliary panel to avoid ambiguity.

4.1.2 Dragging And Holding Interactions

The dragging interaction starts with identifying the 3D location of the initial finger touch position P_i^{init}, which is done in 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 illustrated in Figure 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 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 forces to be moved. The example in Figure 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 been 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), making the computational cost acceptable. Now since the data shape has been deformed, we can optionally blend a different color to the visualization as a hint. The ratio of the blended color is positively related with the distance the data element

is deviated from its original position. Figure 7b shows an example of using yellow as a hint color.



(a) Initial touch. (b) First dragging. (c) Further Dragging.

Figure 7: An example of the process of dragging.

After a finger doing dragging is released, its last finger position is recorded to compute $P_i^{cur} - P_i^{init}$, which continues to create forces to preserve the deformed shape, 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 new finger movements, so underneath data elements are also moved by the new dragging. We do not allow adding a new force or anchor 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 during the further dragging, since their underlying mesh has never been changed. In the example in Figure 7c, we drag the two anchor spheres to turn them into force spheres.

Since sometimes keeping a finger static to perform holding may be challenging, we use extra strategies to help ease the operation. Each time after any finger starts to move, we use a time threshold (in our experiments, 2 seconds) to speculate users' intention of holding. That means, after this time threshold, an anchor sphere will not be turned into a force sphere in the current touch event, regardless of the finger movement. Even if this speculation is wrong, users still can turn this sphere in the further dragging phase as stated above.

4.1.3 Cutting Interactions

The cutting is started by clicking a button in the auxiliary panel. The mode of full cut or partial cut, and the partial 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. A shorter cut whose ends do not reach the domain boundary is technically possible, but not covered in the current stage. After a partial cut, users can still conduct further dragging in the same way, or adjust the depth of the cut through a slider on the panel. After the full cut, the data will be deformed back to the original shape as analyzed in Section 3.4, 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 initial touch location is on the segment.

4.2 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, the advantage of using touchscreen becomes obvious when multi-finger operations cannot be easily translated to mouse interactions. Firstly, users need to place each anchor or force sphere one by one. Secondly, each force sphere needs to be dragged one by one, too. These cost many extra operations, and lose the smooth transition of an intact deformation. The main advantage of using mouse, besides the easier access, is that using mouse to set the locations of the mesh, anchor spheres, and force spheres can be more precise than using fingers.

5 Data-Mesh Engaging

5.1 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. Our principle to set the tetrahedra stiffness is, to make the data elements creating heavier visual clues in the visualization covered by more stiff tetrahedra. 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 from past work [27], we set the stiffness $s(t)$ of a tetrahedron t by:

$$s(t) = s_0 + \beta \times d(t)^{\gamma}, \tag{7}$$

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 the tetrahedron. For each data type used in this study, we set $d(t)$ 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.

Compared to previous works, our stiffness setting for the particle data is the same [36], but are different compared to the studies that used voxel values for volumes [27] or flow entropy for streamlines [34]. We want to mention that the best choice is application specific as mentioned in previous work [27]. Besides, since we use a local mesh with small spacing, we also use gaussian smoothing to process the final tetrahedra stiffness to make the mesh behavior more stable.

5.2 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. For data element λ at location $P(\lambda)$, suppose it lays inside the tetrahedron T before the deformation with barycentric coordinate $b_o(P(\lambda))$. A common idea in data deformation research [27, 34, 36] 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 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 methods 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 with 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, with polyline topology unchanged. If a streamline

intersects the cutting plan after a cutting action, 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.

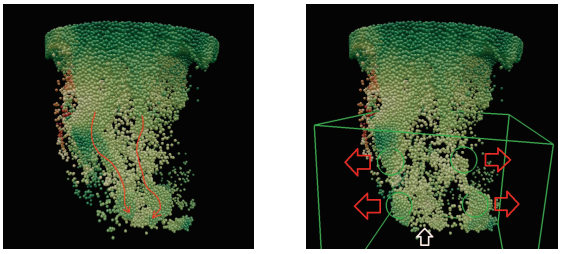
6 Case Study

In this section we show case studies conducted on three different types of data. We present the figures using screen shots to avoid possible blocking by users' hands, but we show the complete touch-screen operating procedures in our accompanying video.

6.1 Particle Data

In this section, we apply our system upon a viscous fluid simulation dataset [1]. 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 is added from the top of the container. Nearby 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.

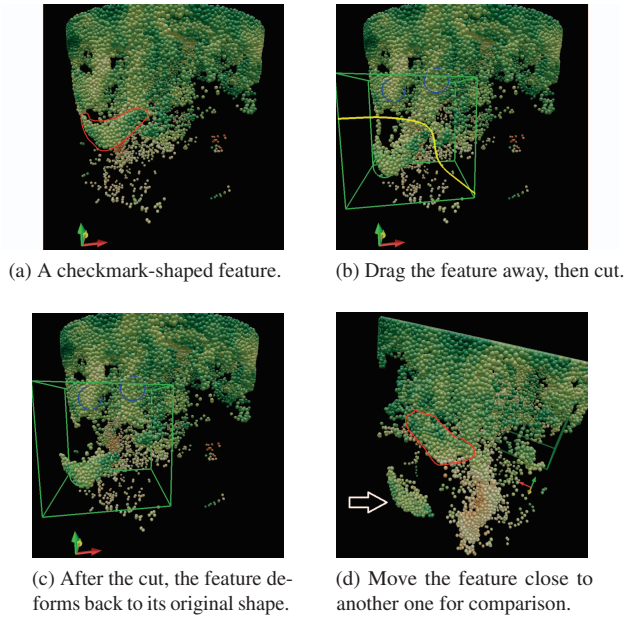
We first render a snapshot of the simulation as in Figure 8a. We color the particles by their speed, where green particles are the slowest, white ones are faster, and red ones are the fastest. We notice two viscous fingers are very close to each other, marked by the two red curves in the figure. If we 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. With our system, a user can drag these two viscous fingers in opposite directions, using four fingers of two hands, as the red arrows shown in Figure 8b. After dragging, we successfully separate the two viscous fingers as in Figure 8b. 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 preserves the relation during the deformation.



(a) Two close features. (b) Drag in opposite directions.

Figure 8: An example of exploring the viscous fluid dataset.

We then render another snapshot as in Figure 9a. We notice a checkmark-shaped viscous finger as circled by the red curve 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 towards the left bottom direction as in Figure 9b. As a result, the original 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 will slowly recover its original shape as in Figure 9c. Then we displace it to the other side of the domain using affine transformations conducted by dragging and rotating. We place it as pointed by the arrow in Figure 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, and 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, then cut.

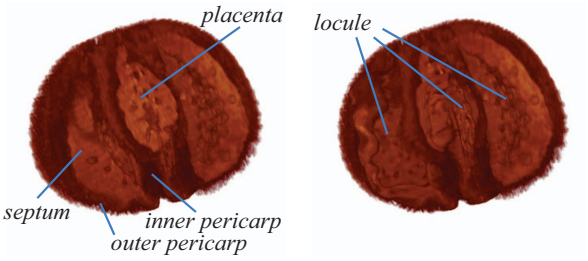
(c) After the cut, the feature deforms back to its original shape. (d) Move the feature close to another one for comparison.

Figure 9: The second example of exploring the viscous fluid dataset.

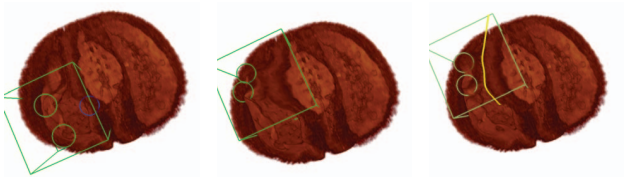
6.2 Volume Data

We demonstrate a 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 10a. From our culinary experience we know that a tomato contains several compartments inside, which are named *locules*. Figure 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 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 drag the bottom part of the left septum through the force spheres in Figure 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



(a) The original data. (b) Final deformed shape.



(c) Flip up the bottom septum. (d) Flip up the top septum. (e) Cut to separate the septum and the pericarp.

(f) Reveal the left locule. (g) Flip up the bottom placenta. (h) Flip up the top placenta.

Figure 10: The example of exploring a scan data of tomato. (c)-(h) are inner steps to achieve the final deformed shape as (b).

locule, we hold a finger on it as an anchor, shown by the blue circle in Figure 10c. Now we accept the current deformation, and enter the next workflow cycle to flip up the top part of the septum, as Figure 10d. But as in the figure, the top part of septum is attached to the inner pericarp much more stronger 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 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 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 10g and 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.

6.3 Streamline Data

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 11. There are many small size vortices attached to the central bundle of long streamlines, and Figure 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.

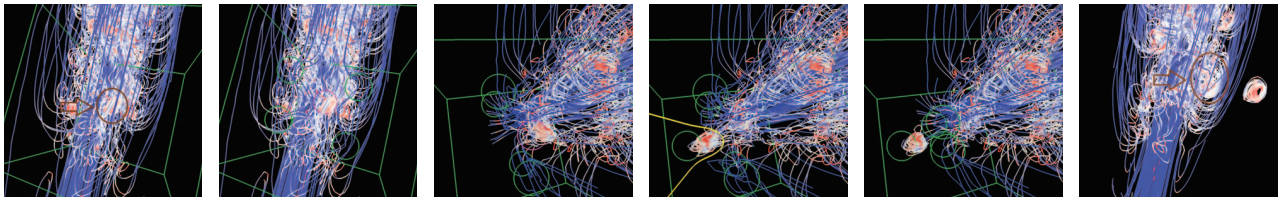
In Figure 11b, we press the middle finger of the right hand to hold the vertex to remain in its location, as the magenta circle shown in the figure. At the same time, we use the index and ring finger of the right hand, and two fingers of the left hand, to drag the long streamlines in front of the vertex as if plucking a guitar's strings, which reduces the occlusion they caused. Then we switch to another view as Figure 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 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 11e. We can then move this vortex freely in the domain. In Figure 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.

7 Domain User Evaluation

We demonstrated our system to two domain experts to evaluate the effectiveness of our system from potential users' perspective. In the following we refer to the two individual experts as User A and User B. User A is a professor from geoscience field who is experienced in compositional flow simulations, therefore we conducted the evaluation with him mainly using the particle datasets involved in Section 6.1. User B is a professor from mechanical engineering field who frequently works with aeromechanical flow field simulations, so we conducted the evaluation with him mainly using the streamline data involved in Section 6.3. The evaluation studies were performed by bringing our workstation to the domain users, introducing our systems, and asking the users to try out by themselves.

Overall, both users gave positive feedback to our system. They both agreed that our method is potential to benefit real research works. User A encounters occlusion a lot caused by finger-shaped clusters, and he was fond of the ability to drag a front cluster aside to reveal those behind, and to move interesting clusters aside for individual inspection. User B commented that our system made the viewpoint selecting more flexible, since the scenes at the originally imperfect camera locations can be manually improved. Both users believed our interaction scheme is intuitive, and dragging data elements together with fingers is a sensible choice. They also confirmed the usefulness of the holding and cutting actions. User A mentioned that the holding and cutting actions took longer time to master than the dragging action, especially the partial cut which was related with a half-broken mesh. However, he still thought the learning effort was manageable, and was able to use all actions after several minutes of guided practice. User B paid much attention to the task of keeping selected features unchanged while only deforming the surrounding context. Besides using holding, he also suggested that more ways could be adopted to achieve the anchoring effect, such as setting anchors based on data value. Both users also treated our design of visual metaphors, including the spheres, the cutting plane, and the mesh region boundary, as informative, as well as the strategy of blending a hint color for deformed data elements. In the end, they both discussed with us the potential applications of our method on other of their research data.

The users also proposed their expectations upon the future improvement of our system. Both users considered our workflow as effective, including the rolling back utility to help recover the original shape after inspecting the deformed data. However, they both



(a) A smaller vortex covered by other lines. (b) Hold the vortex while displacing obstacles. (c) Another view to drag the vortex. (d) Drag and cut the vortex. (e) After the cut, separated parts deform back. (f) Displace the vortex for comparison.

Figure 11: The example of exploring the streamlines of the Plume data.

expected more flexibility that can reduce the number of user operations. For example, the need to place a local mesh every time when starting a new deformation seemed tedious.

8 Discussion

A feature in a 3D dataset can be neighbored with complex structures that reduce the visualization quality, but it is also common that there is certain amount of space between the feature and the neighbors. One fundamental capability of our system is that it can utilize the tiny and irregular empty spaces. The physically-based dragging operation can naturally enlarge them for less occlusion or better separation. Previous works [36, 38] 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 in theory our cutting operation can be operated on the body of a feature, 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.

There is a limitation of our current system caused by the weakness to handle the depth dimension due to the 2D essence of the touchscreen. It has caused a few shortages. The first shortage is that the force and anchor spheres are mostly added at surface area of the data. This shortage does not hurt most cases, since displacing invisible data elements underneath the surface is uncommon in deformation research; nevertheless, it might be a restriction in certain situations. The second shortage is the inconvenience to add vertical forces 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. The last shortage is the depth of a partial cut cannot be determined directly at the time of drawing the cut. To resolve these shortages, we can design some gestures or widgets to adjust the depth of touch points. However, we may not be able to use the gestures/widgets at the same time with dragging or cutting. That means extra steps in the workflow will be needed. Facing the trade-off between workflow smoothness and depth handling, our current design prioritizes the former for dragging/holding since this can already handle many cases, but chooses to use widgets to control the cut depth which is indispensable. In the future, we will explore advanced input devices to attend both the smoothness and the depth control at the same time. One possible direction is to utilize a touchscreen with pressure sensor, whose effectiveness has been testified in several recent studies [2, 30, 42]. Another direction for future work is to exploit 3D input devices.

A few components of the current system may also be extended in future research. One aspect is to lessen the burden of placing a local mesh. While the performance of using a high resolution global mesh can be too below, this problem may be solved by using an adaptive mesh that uses low resolutions in empty regions. We may also study heuristic algorithms to automatically decide a local mesh that can be satisfying in most of the time. Another aspect is to provide more choices to set anchors in the deformation. We can add value-based anchors as a domain expert suggested. We can also enable users to draw an anchor region. Although unlike the current finger holding action, these new utilities are hard to be done together with dragging and thus need extra steps in the workflow, they still strengthen the power of the whole system. The last aspect is to accept different stiffness settings of the mesh. For example when working with CT-scan data, we may be able to estimate actual tissue stiffness characteristics based on the Houndsfield units. Better results can be expected if the choice of the stiffness is made by users according to the specific application.

9 Conclusion

Based on the continuing research on data deformation, in this paper, we have proposed our advancement on utilizing physically-based deformation to manipulate a dataset as an ductile object. The adoption of the touchscreen enabled users to displace data elements with hand gestures, which enhanced the feeling of object-in-hand. We described our method using the word *dragging* due to the uniformly directed forces 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.

Acknowledgments

The authors would like to thank Dr. Joachim Moortgat and Dr. Jen-Ping Chen from The Ohio State University to provide domain user feedback. This work was supported in part by UT-Battelle LLC Contract 4000159557, Los Alamos National Laboratory Contract 471415, and NSF Grant SBE-1738502, program manager Lucy Nowell.

References

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

[2] L. Besançon, M. Ammi, and T. Isenberg. Pressure-based gain factor control for mobile 3D interaction using locally-coupled devices. In *Proc. CHI*, pp. 1831–1842. ACM, Denver, 2017.

[3] L. Besançon, P. Issartel, M. Ammi, and T. Isenberg. Hybrid tactile/tangible interaction for 3D data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):881–890, 2017.

[4] L. Besançon, P. Issartel, M. Ammi, and T. Isenberg. Mouse, tactile, and tangible input for 3D manipulation. In *Proc. CHI*, pp. 4727–4740. ACM, Denver, 2017.

[5] Å. Birkeland and I. Viola. View-dependent peel-away visualization for volumetric data. In *Proc. SCCG*, pp. 121–128. ACM, New York, 2009.

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

[7] S. Bruckner, T. Isenberg, T. Ropinski, and A. Wiebel. A model of spatial directness in interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, To appear, 2018.

[8] M. Chen, C. Correa, S. Islam, M. W. Jones, P.-Y. Shen, D. Silver, S. J. Walton, and P. J. Willis. Manipulating, deforming and animating sampled object representations. *Computer Graphics Forum*, 26(4):824–852, 2007.

[9] M. Chen, D. Silver, A. S. Winter, V. Singh, and N. Cornea. Spatial Transfer Functions: a unified approach to specifying deformation in volume modeling and animation. In *Proc. VG*, pp. 35–44. ACM, Tokyo, 2003.

[10] D. Coffey, N. Malbraaten, T. B. Le, I. Borazjani, F. Sotiropoulos, A. G. Erdman, and D. 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.

[11] C. Correa, D. Silver, and M. Chen. Discontinuous displacement mapping for volume graphics. In *Proc. VG*, pp. 9–16. EG, Boston, 2006.

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

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

[14] C. Correa, D. Silver, and M. Chen. Constrained illustrative volume deformation. *Computers & Graphics*, 34(4):370–377, 2010.

[15] N. Elmqvist and P. Tsigas. A taxonomy of 3D occlusion management for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1095–1109, 2008.

[16] S. Fang, S. Huang, R. Srinivasan, and R. Raghavan. Deformable volume rendering by 3D texture mapping and octree encoding. In *Proc. Visualization*, pp. 73–80. IEEE, San Francisco, 1996.

[17] J. Gascon, J. M. Espadero, A. G. Perez, R. Torres, and M. A. Otaduy. Fast deformation of volume data using tetrahedral mesh rasterization. In *Proc. SCA*, pp. 181–185. ACM, Anaheim, 2013.

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

[19] S. Islam, D. Silver, and M. Chen. Volume splitting and its applications. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):193–203, 2007.

[20] K. Kin, M. Agrawala, and T. DeRose. Determining the benefits of direct-touch, bimanual, and multifinger input on a multitouch workstation. In *Proc. Graphics Interface*, pp. 119–124. Canadian Information Processing Society, Kelowna, 2009.

[21] T. Klein, F. Guéniat, L. Pastur, F. Vernier, and T. Isenberg. A design study of direct-touch interaction for exploratory 3D scientific visualization. *Computer Graphics Forum*, 31(3pt3):1225–1234, 2012.

[22] T. Koyama, H. Okudera, and S. Kobayashi. Computer-generated surgical simulation of morphological changes in microstructures: Concepts of "Virtual Retractor". *Journal of Neurosurgery*, 90(4):780–785, 1999.

[23] B. Laha and D. A. Bowman. Volume Cracker: A bimanual 3D interaction technique for analysis of raw volumetric data. In *Proc. SUI*, pp. 61–68. ACM, Los Angeles, 2013.

[24] E. LaMar, B. Hamann, and K. I. Joy. A magnification lens for interactive volume visualization. In *Proc. Pacific Graphics*, pp. 223–232. IEEE, Tokyo, 2001.

[25] B. Li, X. Zhao, and H. Qin. Four-dimensional geometry lens: A novel volumetric magnification approach. *Computer Graphics Forum*, 32(8):122–133, 2013.

[26] C. Li, J. Moortgat, and H.-W. Shen. An automatic deformation approach for occlusion free egocentric data exploration. In *Proc. Pacific Visualization*, pp. 215–224. IEEE, Kobe, 2018.

[27] C. Li, X. Tong, and H.-W. Shen. Virtual Retractor: An interactive data exploration system using physically based deformation. In *Proc. Pacific Visualization*, pp. 51–60. IEEE, Seoul, 2017.

[28] C. Lundstrom, T. Rydell, C. Forsell, A. Persson, and A. 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.

[29] M. J. McGuffin, L. Tancau, and R. Balakrishnan. Using deformations for browsing volumetric data. In *Proc. Visualization*, pp. 401–408. Seattle, 2003.

[30] S. Pelurson and L. Nigay. Bimanual input for multiscale navigation with pressure and touch gestures. In *Proc. ICMI*, pp. 145–152. ACM, Tokyo, 2016.

[31] J. L. Reisman, P. L. Davidson, and J. Y. Han. A screen-space formulation for 2D and 3D direct manipulation. In *Proc. UIST*, pp. 69–78. ACM, Victoria, 2009.

[32] H. Sollich, U. von Zadow, T. Pietzsch, P. Tomancak, and R. Dachselt. Exploring time-dependent scientific data using spatially aware mobiles and large displays. In *Proc. ISS*, pp. 349–354. ACM, Niagara Falls, 2016.

[33] N. Sultanum, S. Somanath, E. Sharlin, and M. C. Sousa. "Point it, Split it, Peel it, View it": Techniques for interactive reservoir visualization on tabletops. In *Proc. ITS*, pp. 192–201. ACM, Kobe, 2011.

[34] J. Tao, C. Wang, C.-K. Shene, and S. H. Kim. A deformation framework for focus+context flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 20(1):42–55, 2014.

[35] X. Tong, J. Edwards, C. Chen, H. W. Shen, C. Johnson, and P. Wong. View-dependent streamline deformation and exploration. *IEEE Transactions on Visualization and Computer Graphics*, 22(7):1788, 2016.

[36] X. Tong, C. Li, and H.-W. Shen. Glyphlens: View-dependent occlusion management in the interactive glyph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):891–900, 2017.

[37] R. Torres, A. Rodríguez, J. M. Espadero, and M. A. Otaduy. High-resolution interaction with corotational coarsening models. *ACM Transactions on Graphics*, 35(6):211:1–211:11, 2016.

[38] M. Traoré, C. Hurter, and A. Telea. Interactive obstruction-free lensing for volumetric data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1029–1039, 2018.

[39] S. J. Walton and M. W. Jones. Volume Wires: A framework for empirical non-linear deformation of volumetric datasets. *Journal of WSCG*, 14(1–3):81–88, 2006.

[40] H. Wang. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics*, 34(6):246:1–246:9, 2015.

[41] L. Wang, Y. Zhao, K. Mueller, and A. Kaufman. The Magic Volume Lens: An interactive focus+context technique for volume rendering. In *Proc. Visualization*, pp. 367–374. IEEE, Minneapolis, 2005.

[42] X. Wang, L. Besançon, M. Ammi, and T. Isenberg. Augmenting tactile 3D data exploration with pressure sensing. In *Visualization, Poster*. IEEE, 2017.

[43] Y.-S. Wang, T.-Y. Lee, and C.-L. Tai. Focus+context visualization with distortion minimization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1731–1738, 2008.

[44] Y.-S. Wang, C. Wang, T.-Y. Lee, and K.-L. Ma. Feature-preserving volume data reduction and focus+context visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):171–181, 2011.

[45] C. Ware and S. Osborne. Exploration and virtual camera control in virtual three dimensional environments. In *Proc. I3D*, pp. 175–183. ACM, Snowbird, 1990.

[46] R. Westermann and C. Rezk-Salama. Real-time volume deformations. *Computer Graphics Forum*, 20(3):443–451, 2001.

[47] L. Yu, K. Efstathiou, P. Isenberg, and T. 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.

[48] L. Yu, P. Svetachov, P. Isenberg, M. H. Everts, and T. 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.

[49] X. Zhao, W. Zeng, X. D. Gu, A. E. Kaufman, W. Xu, and K. Mueller. Conformal Magnifier: A focus+context technique with local shape preservation. *IEEE Transactions on Visualization and Computer Graphics*, 18(11):1928–1941, 2012.