SSI-DT79135031

**Doctoral Thesis**

# Development of a Real-time Brain Retraction Simulator Using Patient-specific Model
# （患者固有モデルを用いた 実時間脳組織圧排シミュレータの開発）

Kazuya Sase

March, 2017

Division of Systems Science and Informatics
Graduate School of Information Science and Technology
Hokkaido University

Doctoral Thesis
submitted to Graduate School of Information Science and Technology,
Hokkaido University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy.

Kazuya Sase

Thesis Committee: Prof. Atsushi Konno
Prof. Hajime Igarashi
Prof. Satoshi Kanai

2017

# Development of a Real-time Brain Retraction Simulator Using Patient-specific Model*

Kazuya Sase

**Abstract**

The progress of computer technology has enabled the real-time computing of physical phenomena. An application of the computing technique is surgery simulator, in which the deformation of soft organs is calculated in real-time. In addition, using haptic device, users can touch the virtual organs with artificial sense of touch. One difficulty in the clinical application of surgery simulator is the generation of the patient-specific model. When a surgeon wish to conduct a preoperative surgery planning, he/she need to prepare the models for mechanical analysis, e.g. finite element (FE) mesh, from medical images. Especially, the model generation for a brain is challenging because the structure of the brain is highly complex. From the point of the real-time application, the model is required to be simple for reasonable calculation cost. At the same time, the model is required to preserve the geometrical and topological features of the target organ. Although models that fulfill these requirements can be generated using current software, the generation task takes several hours or days with a specialist. Ideally, the model generation should be conducted by medical stuffs in a hospital and should not require special knowledge on the numerical analysis for them. In this thesis, a generation method of patient-specific FE mesh from medical images is proposed for enabling brain retraction procedure in virtual environment. The method relies on an idea of the use of nonconforming mesh. The input geometry is embedded in a regular hexahedral mesh. The deformation of the input geometry is interpolated by

*Doctoral Thesis, Division of Systems Science and Informatics, Graduate School of Information Science and Technology, Hokkaido University, SSI-DT79135031, March 1, 2017.

the hexahedral mesh in a master-slave manner. An advantage of this approach is the simplicity of the mesh generation compared with the conforming mesh generation. Thanks to the simplicity, the mesh generation can be performed quickly and robustly. Thus, a user can generate a patient-specific model without special knowledge on a specific software. A drawback of this method is the possibility of the loss of fine structure. For example, if multiple domains are included in a hexahedral element, they are considered as connected domains even when they are separated in the input geometry. To resolve this issue, an approach using element duplication is proposed. An efficient algorithm of the mesh generation for segmented medical image is also described. The effectiveness of the method in the topology preservation of brain fissure is presented. Additionally, a simulation method for a patient-specific nonconforming FE mesh is also proposed. The deformation of the brain tissues is calculated using corotational finite element method. The contact problem between brain tissues and surgical instruments is formulated using penalty method. For stabilizing the simulation, implicit time integration is utilized for dynamic problem. This framework enables to simulate the interaction between a brain retractor and a brain model with stable force feedback.

**Keywords:** surgery simulator, haptic rendering, real-time physics simulation, medical image processing, finite element method

# Contents

# List of Figures

# List of Tables

# Chapter 1. Introduction

## 1.1 Brain Retraction

In neurosurgery, brain spatulas are used to push the brain tissue aside and expose the lesion. To keep the lesion exposed, the surgeon fixes the spatulas to a frame installed on an operating table and then treats the lesion. The operation that exposes a lesion is called "retraction" and the equipments specialized for brain retraction are called brain retractor systems [12]. Because of the compression of the brain tissues, brain retraction may inhibit blood flow around the pressured region. The inhibition of blood flow reduces oxygen supply to the surrounding tissues. Damage to nervous tissue by blood flow inhibition causes dysfunction, such as hemiparesis and aphasia. Such damage caused by brain retraction is called brain retraction injury [13, 14].

Use of brain retraction is inevitable in most craniotomy approaches [15] (Fig. 1.1). Craniotomy is an operation in which part of the skull (bone flap) is removed to obtain a sufficient surgical field. After removal of the bone flap, the surgeon cuts the arachnoid to release connective tissue, after which the brain retraction is performed. In this way, the surgeons open a brain fissure to access a deep brain regions.

In addition to damage caused by blocked blood flow, mechanical injury can be induced by retraction. Inside the brain fissure, there is a fine web-like fiber structure (Figure 1.2). The fibers connects cerebral vessels and thin membranes that cover the brain parenchyma. If the fiber constraints are not removed sufficiently, the fibers can tear the walls of the small vessel [16].

To avoid injury during retractions, young surgeons should learn the anatomical structure, how to remove constraints between brain tissues, and where to position brain retractors. Young surgeons can learn such points from video material.

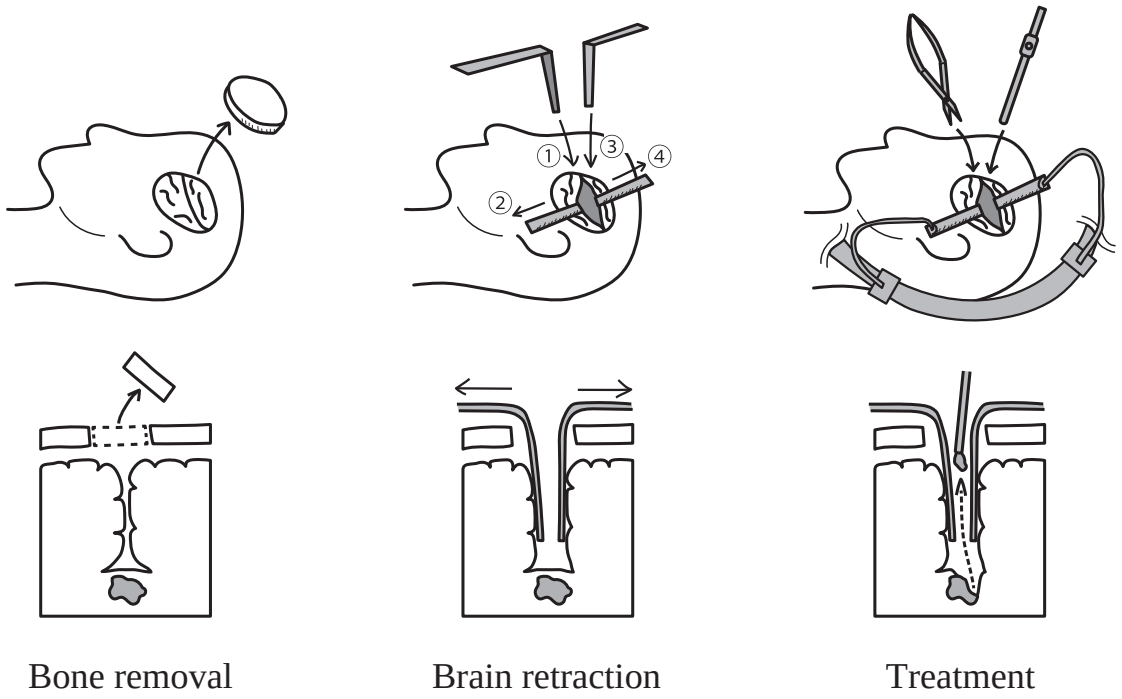



Figure 1.1: Brain retraction using brain spatulas during craniotomy.

However, they cannot learn the tactile cues from such visual media. They could also train using animal bodies. However, the structure of the organs are highly different from that of humans. Some surgeons might have the opportunity to train using donor bodies. However, such opportunities are very limited. Even when it is possible, the material properties of the body changes after death. One possible effective means of training is an artificial organ model constructed using 3D printers [17]. However, the resolution of the modeled geometry is limited by manufacturing processes and the reconstruction of material properties remains difficult. Eventually, the education of young surgeons is dependent on actual surgery underthe supervision of a professional surgeon. This obviously carries the risk of inducing retraction injuries during the trainings of young surgeons.

## 1.2 Surgery Simulator

For the training of young surgeon, virtual reality (VR) technology is expected to enhance the safety and efficiency of surgical training [18, 19, 20]. In the last

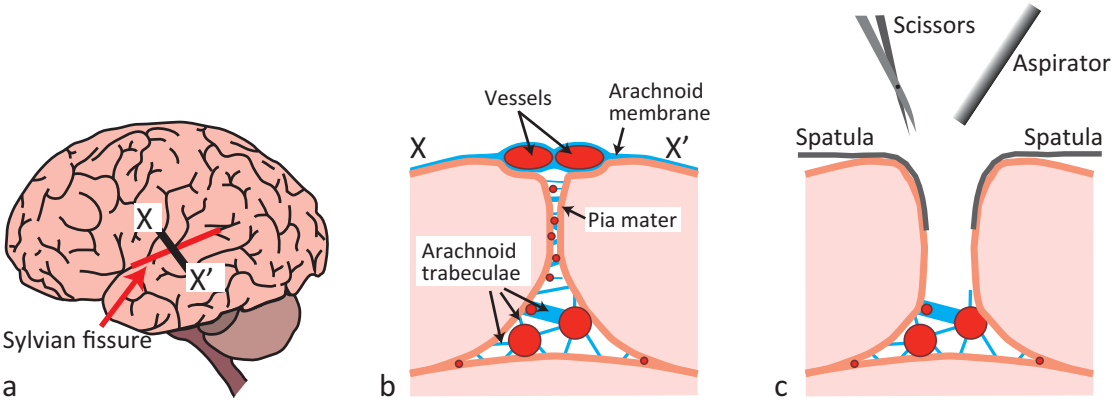



Figure 1.2: Overview of the opening of a cerebral fissure. (a) Sylvian fissure, (b) cross-sectional view of X-X', and (c) retraction using spatulas and dissection using scissors and an aspirator. Reproduced from [1].

two decades, VR training systems for laparoscopic surgery have increasingly been developed [21, 22]. Typically, a surgery simulator consists of a computer, a graphics display, and a user I/O device that allows input of the movement of surgical instruments and output of the reaction force to be applied to the user's hand. The development of a laparoscopic surgery simulator has been strongly funded because there is a great demand for training in laparoscopic surgery. Laparoscopic surgery requires surgeons to learn the usage of special instruments under the limited view from an endoscope. Current laparoscopic surgery simulators are well developed and are now commercially available [23]. Recently, following the success of laparoscopic surgery simulators, VR surgery simulators have been developed for other fields.

Recently, neurosurgery simulators have also been developed for education or preoperative planning purposes [24, 25, 26]. Delorme et al. developed a sophisticated neurosurgery simulator called NeuroTouch [27, 28]. NeuroTouch enables surgeons to train in a tumor resection procedure using an aspirator and bipolar forceps. NeuroTouch is now available as a commercial product named NeuroVR [29]. Alaraj et al. developed a simulator for training in cerebral aneurysm clipping [30]. In addition, some brain retraction simulators have been developed. Koyama et al. introduced a brain retraction simulator called Virtual Retractor [31]. In this simulator, the deformations of intracranial vessels are modeled using geometrical theory, but physical consistency was not considered. Hansen et al. developed a

brain retraction simulator using FEM [32]. They calculated the deformation of brain tissues and the reaction force for haptic feed back. However, the calculation speed is not sufficient for haptic rendering. Hasegawa et al. conducted a cerebellar retraction simulation using a high-resolution model [33]. In their study, they considered the nonlinear viscoelastic behavior of soft tissues. However, they did not achieve real-time simulation because of high-computational costs. In the physics-based simulator, the real-time computation of brain deformation is still a challenging problem.

## 1.3 Brain Model Construction

To simulate a brain retraction, a brain model is required. In particular, when we require reliable soft-tissues deformation, finite element method (FEM) is the most common approach [34]. FEM requires a discretized volumetric representation, such as tetrahedral or hexahedral mesh, termed the finite element (FE) mesh. The resolution (number of nodes or elements) of the FE mesh is related to the computational cost of FEM. To achieve a force feedback in brain retraction simulation, the soft-tissues' dynamic behavior must be calculated in real time. Therefore, the resolution of a FE mesh is limited. This limitation make the brain model construction difficult, because the brain has a highly complex structure [35]. In a retraction simulation, we cannot ignore the structure of a brain fissure that is to be opened using spatulas. In the previous real-time neurosurgery simulators [27, 30, 32], brain fissures were not modeled and only a small area surrounding the affected part was discretized to a volumetric mesh.

In general, the inner structure of human body can be obtained by medical imaging techniques, such as computed tomography (CT) or magnetic resonance imaging (MRI). The structure of the brain can be reconstructed from medical images. However, the meshing process for reproducing a brain fissure structure has not been automated well. Therefore, to develop a brain retraction simulator, we should consider how to generate a brain model containing brain fissures from medical images under limited resolution. Brain model construction is important in preoperative planning with a patient-specific model. Ideally, the model generation

should be conducted by medical staff in a hospital and should not require special knowledge about their numerical analysis [35].

## 1.4 Approach of This Study

This study aimed to develop a brain retraction simulation using a patient-specific model with force-feedback. To develop such a simulator, the following functions were developed.

- A generation method for a patient-specific FE mesh that preserves the structure of brain fissures with reasonable resolution.
- Fast and stable physics simulation method for haptic rendering of the contact between a retractor and soft brain tissues.

A generation method for patient-specific FE mesh from medical images is proposed to allow a brain retraction procedure in a virtual environment. The method relies on the concept of using nonconforming mesh. The input geometry is embedded in a regular hexahedral mesh (cartesian grid). The deformation of the input geometry is interpolated by the hexahedral mesh in a master-slave manner. An advantage of this approach is the simplicity of the mesh generation as compared with the conforming mesh generation. Thanks to the simplicity, the mesh generation can be performed quickly and robustly. Thus, a user can generate a patient-specific model without special knowledge on a specific software. A drawback of this method is the possibility of the loss of fine structure. For example, if multiple domains are included in a hexahedral element, they are considered as connected domains, even when they are separated in the input geometry. To resolve this issue, an approach using element duplication is proposed. An efficient algorithm for mesh generation from segmented medical images is also described. The effectiveness of the method in topology preservation of brain fissure is also presented.

Additionally, a simulation method for a patient-specific nonconforming FE mesh is also proposed. The deformation of brain tissues is calculated using a corotational FEM. The contact problem between brain tissues and surgical instruments is

formulated using a penalty method. For stabilizing the simulation, implicit time integration is utilized for dynamic problems. This framework allows simulation of the interaction between a brain retractor and a brain model with stable force feedback.

## 1.5 Outline of the Thesis

The thesis is structured as follows.

**Chapter 1 Introduction**

The background and purpose of this study is are described.

**Chapter 2 Patient-specific Model Generation**

The method of patient-specific model generation is described in this chapter. The method allows adjusting of the resolution of an output mesh while making preservation of features possible. To evaluate the method, meshes of various resolutions were generated and the abilities of feature preservation are compared.

**Chapter 3 Real-time Simulation of Soft-tissue Deformation**

The mechanical modeling of soft tissues and its calculation method using FEM is described in this chapter. Additionally, the acceleration method using GPGPU for real-time computation is described and its performance is shown in this chapter.

**Chapter 4 Haptic Rendering Based on Virtual Coupling**

The haptic rendering method based on virtual coupling is described in this chapter. Virtual coupling is a commonly used approach for stabilizing 6-DoF haptic rendering. The contact problem between a retractor and a brain model is formulated based on a penalty method. To stabilize the contact simulation, a dynamic system is formulated using implicit time integration.

**Chapter 5 Haptic Rendering for Embedded Volume**

A haptic rendering method for an embedded volume is described. A patient-specific FE mesh, generated by the method described in Chapter 2, is a

nonconforming orthogonal mesh. Thus, the boundary does not equal that of the surface of an input volume. This chapter describes how to address this issue and the results shows the effectiveness of the proposed collision handling method for simulating brain retraction.

**Chapter 6 Conclusion**

The background and purpose of this study are described.

# Chapter 2. Patient-specific Model Generation

## 2.1 Introduction

This chapter describes the method of patient-specific model generation. A patient-specific model is generated from medical images. Firstly, a medical image is segmented into anatomical regions. From the segmented volume FE mesh is generated. The FE mesh is a nonconforming orthogonal mesh which covers whole volume. Fig. 2.1 shows the overview of the typical approach of the patient-specific model generation and ours.

## 2.2 General Approach

Patient-specific model is generated from medical images obtained preoperative clinical examinations. In general, medical images are segmented into regions, e.g., anatomical regions and affected area, for analysing the geometry. This procedure is called segmentation. From the segmented region, surface of the volume is extracted. The surface is represented by a polygonal mesh. Using the surface mesh, the domain is discretized into volumetric meshes, e.g. hexahedral mesh, tetrahedral mesh, and their combinations.

In the above procedure, the segmentation and mesh generation is labor-intensive works. The segmentation techniques are significantly studied and the developed algorithm can be used in some open-source software, e.g. 3D Slicer [6]. Fig. 2.4 shows an example of an atlas-based segmentation. In the segmentation, a T1 weighted MR image was segmented automatically based on manually segmented MR image (atlas). The mesh generation methods are also well-studied. Some algorithms are implemented in open-source software, e.g. CGAL [36]. Our research

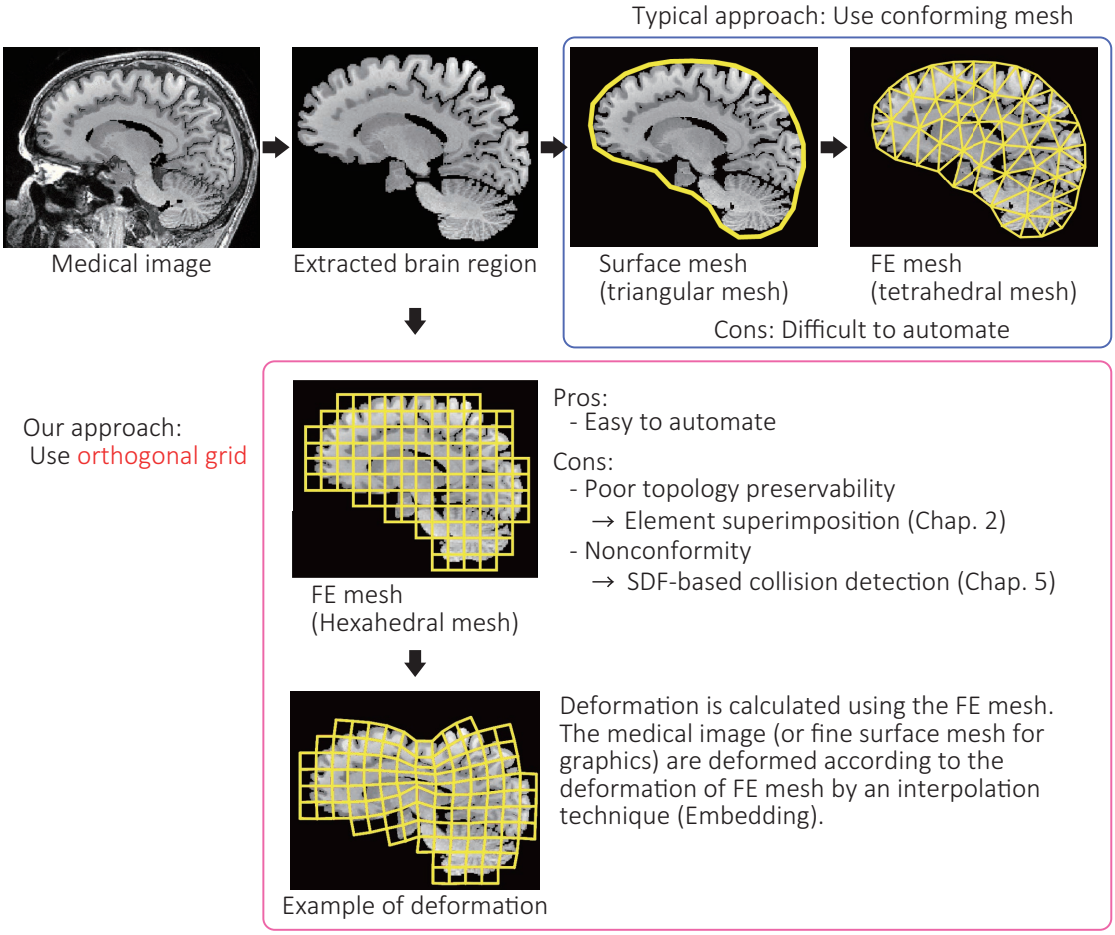



Figure 2.1: Typical approach of the patient-specific model generation and ours. The MR image printed in this figure is from the SPL brain atlas [2].

group have used a commercial software, GiD pre- and post- processor [37] provided by International Center for Numerical Methods in Engineering, Spain. Most of the volumetric mesh generation algorithms requires water-tight surface mesh as an input of the meshing process. Therefore, the surface mesh must be modified appropriately. In most case, such modification involves manual tasks. Meshing often takes several hours or days for meshing a complex domain, such as brain. Such labor-intensive tasks are hope to be automated.

To reduce the manual tasks of mesh generation, Pons et al. developed a volumetric mesh generation method that directly generates tetrahedral grid from a multi-segmented volume data [4]. This method enabled to avoid surface mesh generation and reduce the labor involved in patient-specific mesh generation. Boltcheva et

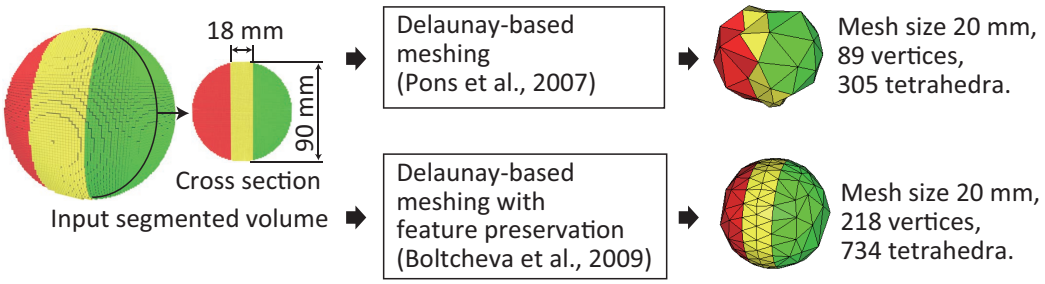



Figure 2.2: Examples of the results of tetrahedrization using Delaunay-based meshing [3, 4]. Adopted from [5].

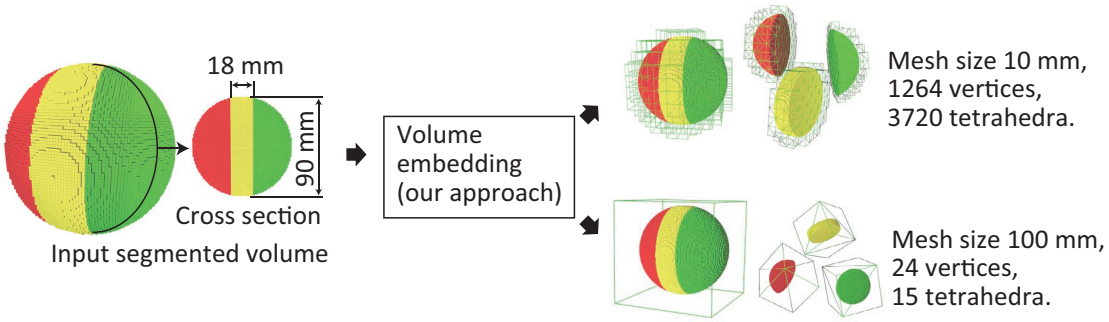



Figure 2.3: Examples of the results of tetrahedrization using our volume embedding. Adopted from [5].

al. extended the method to preserve the point and edge features according to constraints input by users [3]. Their methods are implemented in CGAL [36]. Some results of the mesh generation using [4] and [3] are shown in Fig. 2.2. The feature preservation worked well and good-quality mesh was obtained. However, the resolution of the mesh becomes fine when the feature preservation is applied. High resolution mesh increase the computational cost in FEM. Apparently, the ability of the detail preservation and the computational cost of FEM are in a trade-off relationship. Therefore, it is difficult to achieve the both of the preservation of the fine geometry and the reasonable computational cost that can be used in real-time surgery simulation.

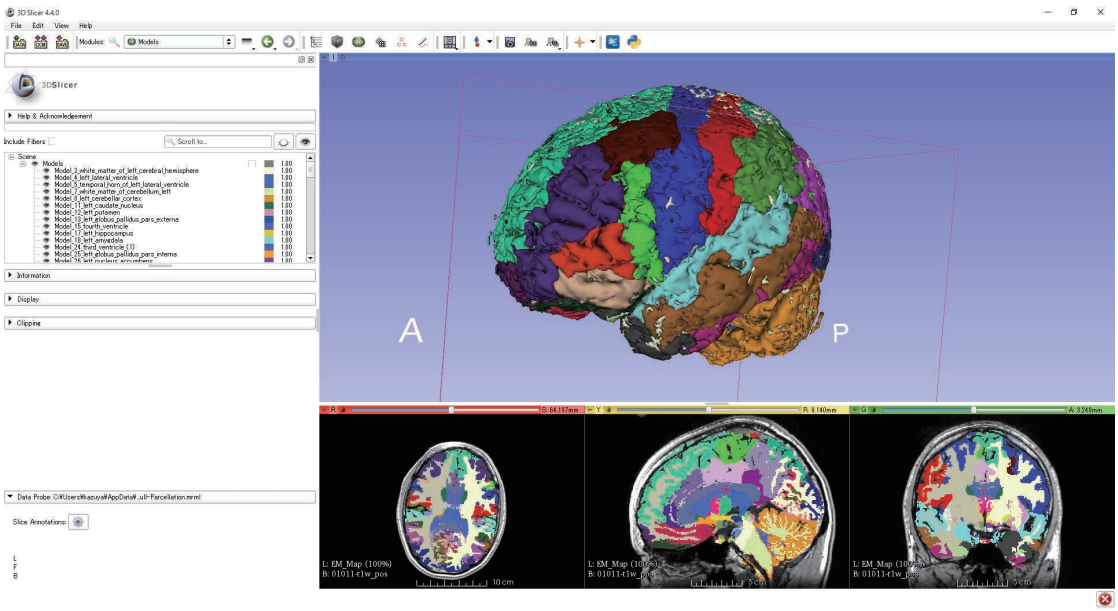



Figure 2.4: Example of an atlas-based segmentation implemented in 3D Slicer [6]. The input image was T1 weighted image from NAMIC: Brain Mutlimodality (case-01011) [7].

## 2.3 Embedding Approach

In the computer graphics community, deformations of fine geometries are often controlled by geometrical interpolation techniques using the movement or deformations of skeletons or master meshes. In the character animation, polygonal meshes are often deformed by interpolating the movement of skeletons associated with the skin meshes. Such animation techniques is called rigging and implemented in general 3D animation software such as Maya (Autodesk Inc.), 3ds Max (Autodesk Inc.), and Blender (an open-source freeware). The rigging can be achieved using a master mesh. In the approach, a polygonal mesh is covered by a coarser control mesh, which usually called *cage* [38]. The polygonal mesh is deformed by the control mesh in a master-slave manner. By combining this mesh-based rigging and physics simulation, physically-based deformation can be obtained with good visual plausibility. Such idea was introduced by Müller et al. [39]. They generated a regular grid that covers a fine polygonal geometry. The grid is deformed by FEM and the fine geometry is deformed by a geometrical interpolation technique. This approach is useful for trading off the computational cost of physics simulation and

the quality of visual representation. In this thesis, we call this approach *embedding* because this naming is conventionally used in the computer graphics community, e.g. in [40].

In the embedding approach, regular grid is often used for FE analysis. Thus, the boundary of the embedded fine geometry and that of the FE mesh is not conformed. Although the use of regular grid limits the accuracy of domain representation, it is easy to control the resolution of the FE mesh. Therefore, it can easily adjust the computational cost of FEM. Even if the resolution is decreased, the visual plausibility is preserved because the rendered model is different from the FE mesh. Instead of being Fig. 2.3 illustrates examples of the meshes used in volume embedding approach. Because the method uses a regular grid for FE mesh, they are able to reduce the number of vertices down as far as 24 vertices in this example. Such coarse resolution can not be achieved when we use a Delaunay-based meshing method as shown in Fig. 2.2.

One of the issues of the use of regular grid is the disability to separate proximate surfaces that included in a same cell. Off course we can avoid such issue by increase the resolution of the grid. However, it leads too high computational cost to compute the deformation in real time. To resolve this issue, Nesme et al. proposed a method using the superimposition of elements [40]. In their method, when multiple separated regions exist in a same cell, overlapped cells are generated for each region. The connectivity of the cells are modified according to embedded fine geometry. As a result, a topology-preserved FE mesh is obtained. However, they intended for embedding a polygonal 3D model in a FE mesh and not for medical images.

In this study, we developed a mesh generation algorithm based on regular grid with element superimposition. Our method that directly processes a segmented volume data. In addition, we consider the separation of attached boundary which specified by a user.

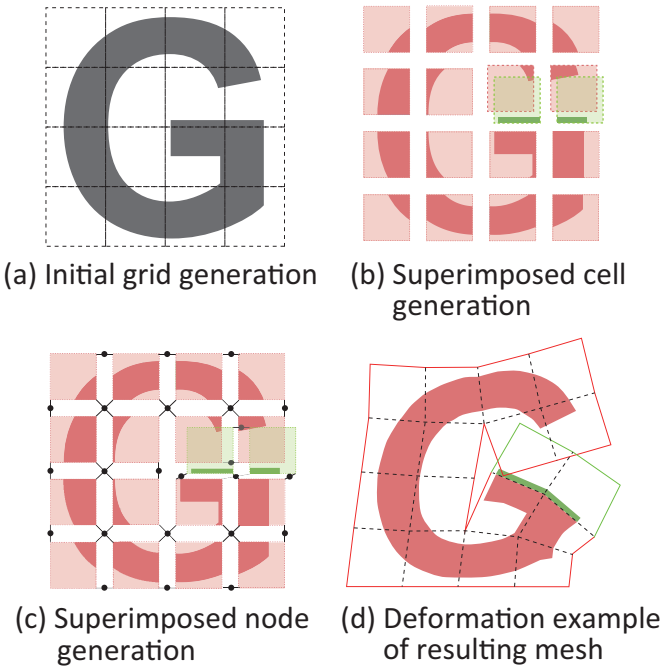



Figure 2.5: Algorithm overview. Reproduced from [5].

## 2.4 Topology Preservation

To preserve the topology of input structure, we uses superimposed nodes and cells. Fig. 2.5 shows an overview of the developed algorithm. By superimposing nodes and cells, the degrees of freedoms of the mesh deformation are added. As a result, the connections between volumetric regions are preserved as precise as possible. We also paid attention to the separation of connected multiple segments, e.g., the "Temporal lobe" segment and the "Parietal lobe" might be separated for simulating the brain retraction for the Sylvian fissure. To describe such user requests, we define separation label pairs (SLPs). The algorithm performs the hexahedral mesh generation according to the input volumetric data (label map) and user-specified SLPs.

The details of the input and output of our algorithm are described as follows.

**Segmented medical image (label map).** A segmented medical image, which also called label map, is volume data that store labels (integer value) at aligned points. A label represents a segment, e.g., 0 is "empty (air)", 1 is "gray matter", 2 is "white matter". Let $\boldsymbol{i} \in \mathbb{Z}^3$ be a voxel coordinate. We

denote a label at $\boldsymbol{i}$ as $L(\boldsymbol{i}) \in \mathbb{Z}$. Voxel coordinate $\boldsymbol{i}$ can be transformed into spatial coordinate $\boldsymbol{p} \in \mathbb{R}^3$ using a vector of the origin of the volume $\boldsymbol{p}_0 \in \mathbb{R}^3$ and a vector of spacing values (size of a voxel) $\boldsymbol{s} = [s_x, s_y, s_z] \in \mathbb{R}^3$ as $\boldsymbol{p} = \boldsymbol{p}_0 + \boldsymbol{s} \odot \boldsymbol{i}$, where $\odot$ denotes element-wise vector multiplication. $\boldsymbol{p}_0$ and $\boldsymbol{h}$ are available as properties of medical image data.

**Mesh size.** We define the mesh size $H \in \mathbb{R}$ as the approximate edge length of a resulting hexahedral cell. The algorithm determines the actual edge lengths in the $x$, $y$, and $z$ directions $\boldsymbol{h} = [h_x, h_y, h_z]$ by multiples of the spacing value of a volume, e.g., $h_x = \mathrm{ceil}(H/s_x)s_x$ for the direction $x$, where $\mathrm{ceil}(H/s_x)$ is the number of voxels along the $x$ axis. The edge lengths of a hexahedral cell become $n_{\mathrm{voxels}}h_x$, $n_{\mathrm{voxels}}h_y$, and $n_{\mathrm{voxels}}h_z$.

**SLPs.** An SLP is a pair of labels $\{L_a, L_b\}$, where $L_a$ and $L_b$ are the labels of segment $a$ and $b$, respectively. The algorithm processes the mesh generation to separate the segments that are specified by SLPs. The SLPs are a part of our algorithm and they should be prepared users. The SLPs need to be generated by users.

**Hexahedral mesh.** A mesh consists of nodes and cells. A node has a corresponding position and a cell has references to nodes (eight nodes for a hexahedral mesh). Note that a hexahedral mesh can be considered a rectangular mesh, where each cell has four nodes in 2D space. In this paper, although the schematic illustrations are sometimes described in 2D space with a rectangular mesh, all the algorithms are directly extended to 3D space with a hexahedral mesh.

**Superimposed nodes and cells.** A superimposed node is a node that coexists at the same position as another node. The number of superimposition is not limited. The same is true for superimposed cell.

## 2.5 Mesh Generation Algorithm

The first step is the generation of a regular grid (Fig. 2.5(a)). The bounding box of the input volume is calculated, and an regular grid is constructed inside

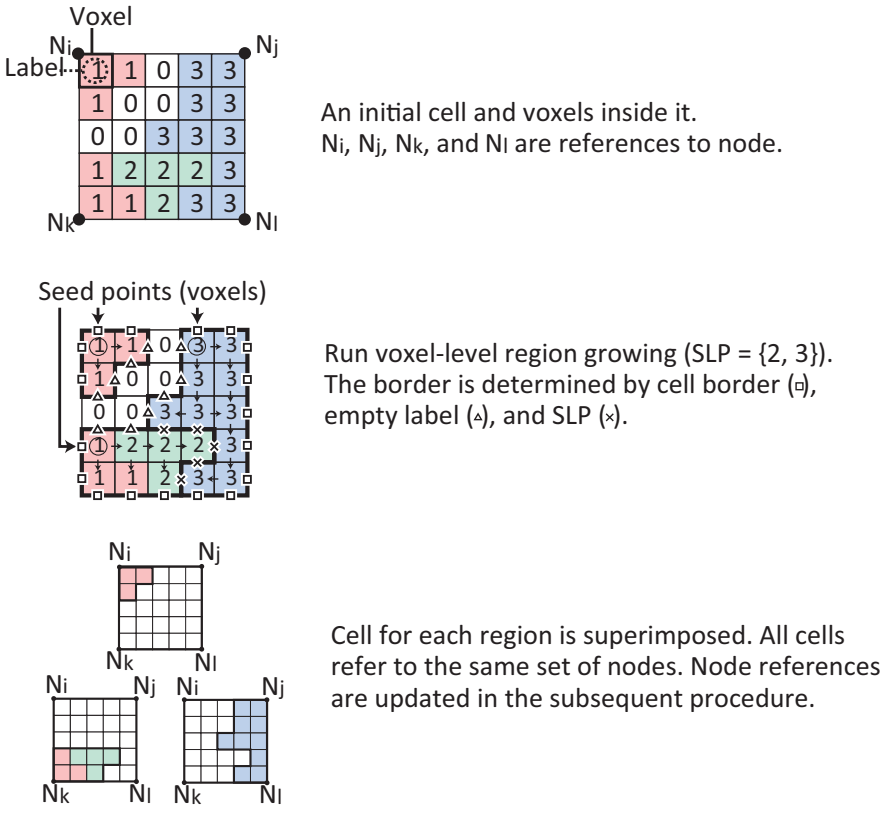



Figure 2.6: Generation of superimposed cells using voxel-level region growing. Adopted from [5].

the bounding box. The lattice bases are $[h_x, 0, 0]^T$, $[0, h_y, 0]^T$, $[0, 0, h_z]^T$.

The second step is superimposition of the cells (Fig. 2.5(b)). In order to detect multiple regions in a cell, local segmentation based on region growing is executed in each cell, as illustrated in Fig. 2.6. An initial seed point is arbitrarily selected from voxels inside the cell, and then, the label values of the neighbor voxels are compared. If the label value of a neighbor voxel is not "empty (air)" and is not registered in the SLPs, the neighbor voxel is added to the region. These procedures are repeated until no connected voxel is detected. After a region is obtained, the region growing algorithm is repeated until all voxels are checked. If multiple regions are detected, superimposed cell is generated for each region. If no region is detected, the cell is deleted. At the end of this step, all superimposed cells have the reference to the same set of nodes.

The third step is superimposition of nodes (see Fig. 2.5(c) and Fig. 2.7 for

details). This process modified the node references stored in cells. This is a node-independent procedure. At each node, cells that share the node are extracted and the connection between the cells are checked. This connection check is performed in a manner of cell-level region growing. The region growing is initiated from a seed point, in this case an arbitrary cell in the extracted cells. From the seed cell, connectivities with surrounding cells are checked. For the cell connectivity determination, the neighbor labels on the boundary of a pair of cells are checked. If a label pair does not have a label of "empty" and is not registered in the list of SLPs, the pair is considered to be connected. If at least one connected label pair is found, the two cells are determined to be connected. If multiple regions (subsets of connected cells) are detected, the node is superimposed for each region and the corresponding node references in the cells are rewritten to the generated superimposed node. If no region is detected, the node is deleted because it means no cells does not refer the node.

Optionally, small islands deletion can be useful. In most cases, a label map includes isolated small segments because of imaging noise or segmentation errors. Such segments can generate unmeaningful small pieces (island) in dynamic simulation. In order to delete these island, we measure the volume of the island by counting the number of cells for each cell island (a set of connected cells). If the volume of the island is smaller than a threshold, the cells included in the island are deleted.

## 2.6 Visualization

The surface polygons of segmented volumes are generated using the marching cubes method for visualization. As in [41, 40], the polygonal surfaces are deformed by interpolating the deformation of FE mesh obtained calculated by the FEM. To perform the interpolation, each surface vertex is associated with a tetrahedron in a preprocessing and the vertex is transformed using the barycentric coordinate defined in the associated tetrahedron. For real-time simulation, the voxels are rendered as a simple primitive such as a point or box because of the low rendering cost. For offline rendering of a captured deformation, a deformed volume can

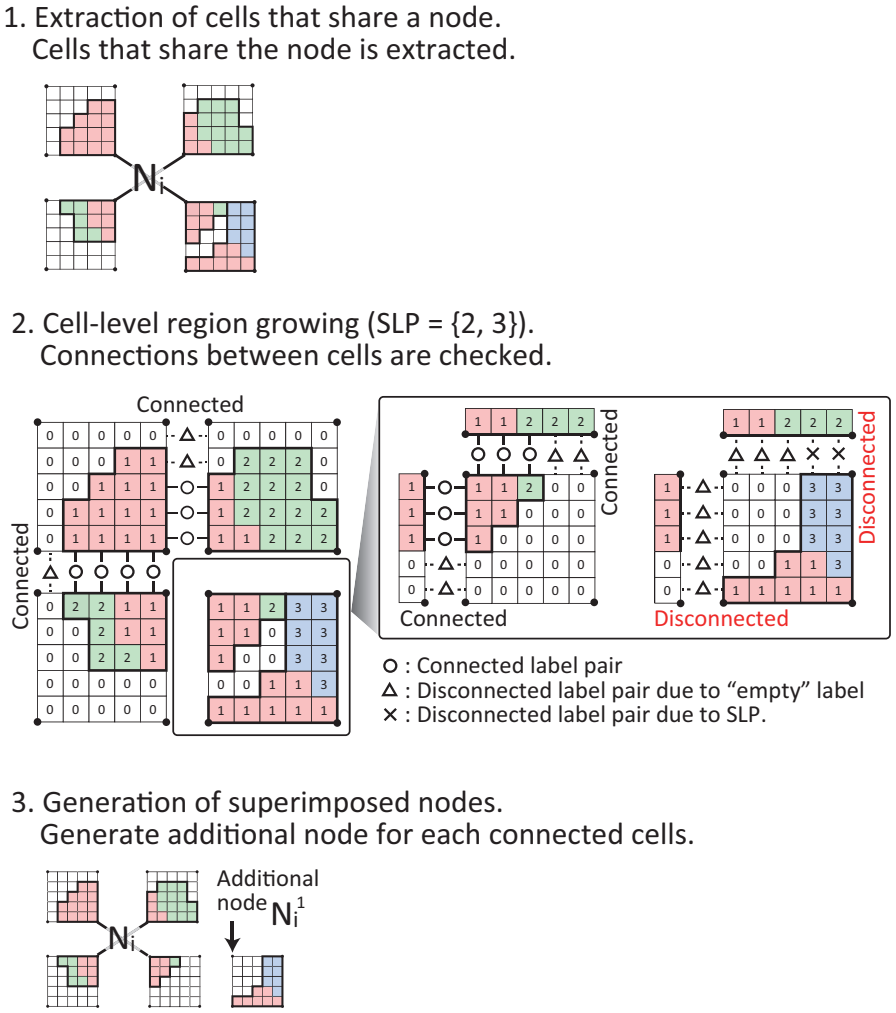
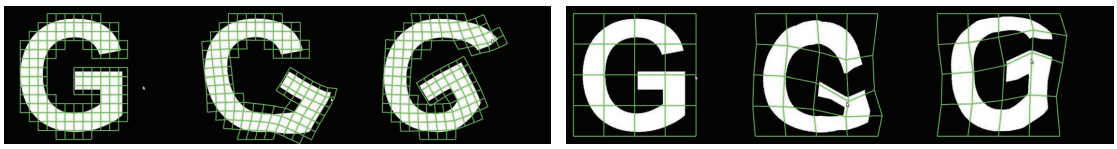



Figure 2.7: Generation of superimposed nodes using cell-level region growing. Reproduced from [5].

be exported in a common format of volume data, such as NRRD or DICOM. Although the general volume data format stores values on equally spaced points, the voxels no longer locate on the initial positions in a deformed volume. Therefore, the values by interpolating the values on volume points being exported from a deformed volume. this can be realized by interpolation method, such as the nearest neighbor search, trilinear interpolation, in the deformed volume for each voxel of the exported volume.



(a) (b)

Figure 2.8: Deformations of a "G"-shaped 2D image without topology preservation. Although fine mesh (a) preserved the topology of the small gap, coarse mesh (b) failed to preserve the topology.

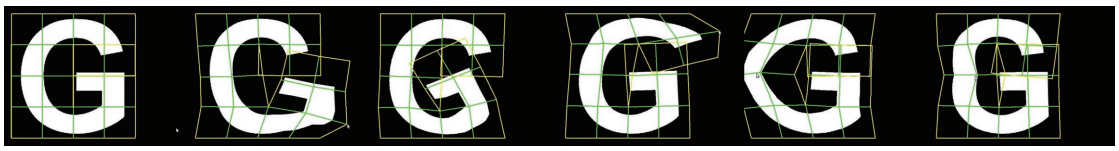


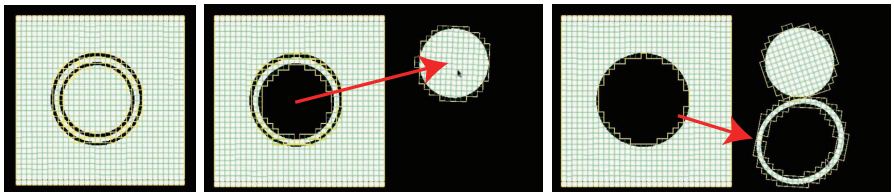
Figure 2.9: Deformations of a "G"-shaped 2D image with topology preservation. Even very coarse mesh preserved the topology of the small gap.

## 2.7 Results and Discussion

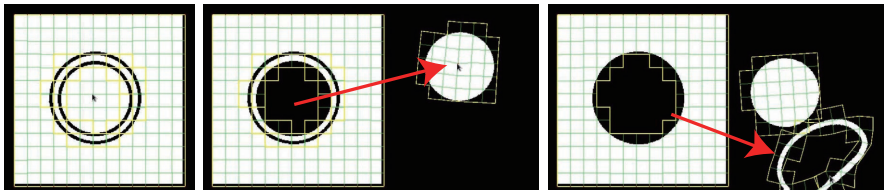
### 2.7.1 Conditions

Our method was implemented on a workstation with an six-core CPU (Intel Core i7-3960X overclocked to 4.5 GHz), 64 GB of RAM, and two GPUs, an NVIDIA K20c (2,496 CUDA cores) and an NVIDIA Quadro K5000 (1536 CUDA cores). GPU K20c was used for numerical calculation of FEM and GPU K5000 was used for graphics rendering. The proposed meshing algorithms which described in this chapter were implemented using OpenMP for the parallelization on multi-core CPU.

Meshes obtained by our algorithm were validated using an FEM (explained in Chapter 3). Each hexahedrron in obtained meshes is divided into five tetrahedra and a first-order tetrahedral element was used for FE model. The meterial properties of all tetrahedral element were set to same values; Young's modulus is 1000 Pa, Poisson's ratio is 0.4, density is 1.0 g/cm$^3$.



(a)



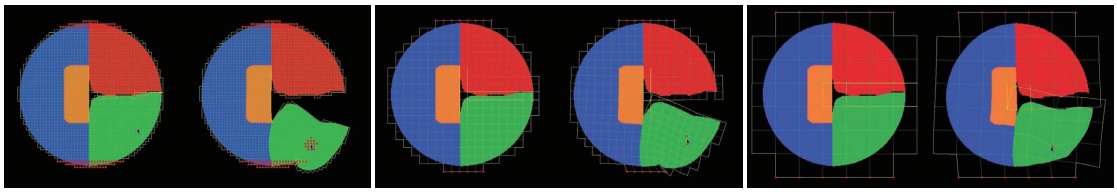
(b)

Figure 2.10: Deformations of a ring-shaped 2D image with topology preservation.

### 2.7.2 Evaluations in 2D

To validate the effect of our meshing method, some evaluations were conducted in 2D. Fig. 2.8 and 2.9 are the deformation examples of a "G"-shaped image without/with the topology preservation. In Fig. 2.8 (a), the shape is embedded in a fine orthogonal mesh and the deformation of the image is interpolated by the orthogonal mesh. Using such a fine mesh, visually plausible dynamic deformation was obtained. However, in Fig. 2.8 (b), the topology of the shape was not preserved: the right-top part and the right-bottom part of the "G" was connected because the separated multiple parts are integrated in a single cell. This is a drawback of the use of a regular mesh that we wish to avoid. Fig. 2.9 shows the deformation example of the topology preserved mesh generated by the proposed method. In the figure, the surface boundary edges are colored with yellow lines. Thanks to the superimposed cells, the right-top part and right-bottom part of "G" were separated in the generated FE mesh. As a result, visually plausible simulation with very coarse mesh was achieved.

We also conducted a simulation using a ring-shaped image with very small gap (Fig. 2.10). By our meshing method, the floating parts in the image (a circle and a ring) were separated in the obtained FE mesh. In the figure, we first dragged the circle region and then dragged the ring. In the both of fine mesh and coarse mesh, the circle and ring regions were separated as expected.



(a) (b) (c)

Figure 2.11: Deformations of a multi-labeled 2D image with topology preservation. The red and green parts are separated according to an SLP.

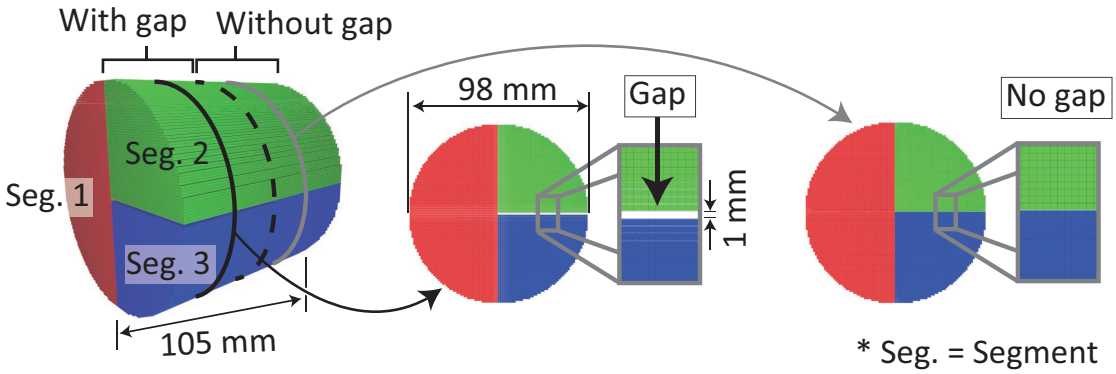
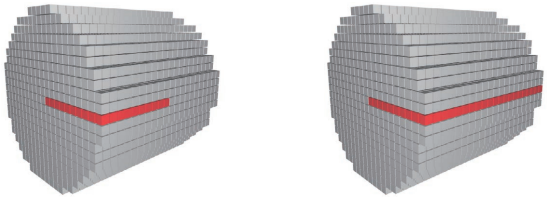



Figure 2.12: Cylinder model used for the evaluation. This model is a volume data and the voxel size is 1 mm.

We also conducted a simulation using a multi-labeled image. Fig. 2.11 shows the deformation of the image that is segmented into four regions, red, green, blue, and orange. The red, green, and orange regions are partly attached. To separate these regions, we input the SLPs {{red, green}, {red, orange}, {green, orange}}. As shown in the figure, these regions were separated in the obtained FE mesh. Even in a coarse mesh (Fig. 2.11 (c)), the SLP was processed well and the boundaries were separated.

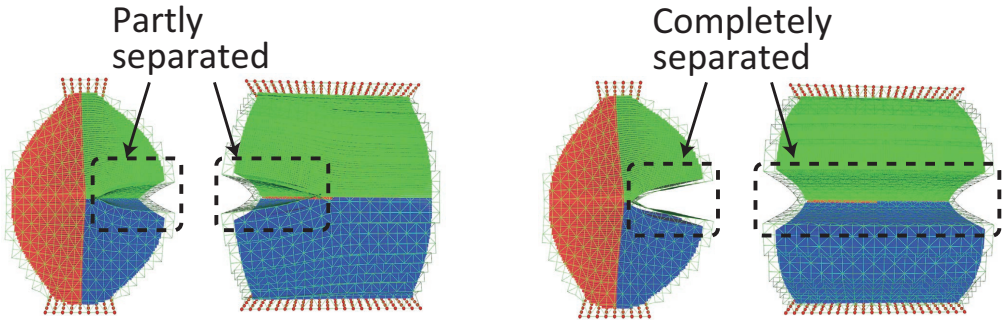
### 2.7.3 Evaluation Using a Cylinder Model

For preliminary evaluation in 3D problem, we used a cylinder volume model that divided into 3 segments (Fig. 5.5). In the figure, red, green, and blue parts represents segmented region labeled 1, 2, and 3, respectively. In the front-half part, there is a small gap. The width of the gap is 1 mm (equal to the length



(a) Without SLP (b) With SLP

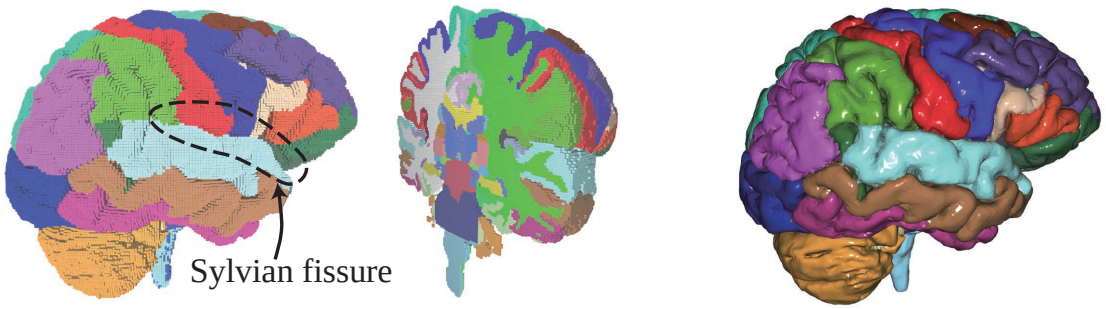
Figure 2.13: Distributions of superimposed cells in the evaluation using the cylinder model. Red cells represents superimposed cells.




(a) Without SLP (b) With SLP

Figure 2.14: Deformations results of the obtained meshes in the evaluation using the cylinder model. Green wireframes are FE meshes and red spheres are position-constrained nodes.

of a voxel). In the rear-half part, there is no gap between segments 2 and 3 (completely attached). Fig. 2.13 and 2.14 show the results of the proposed volume embedding with and without SLP $\{2, 3\}$. The results were obtained with a mesh size of $H = 5$ mm. Fig. 2.13 shows the distributions of the superimposed cells in the obtained FE mesh. In the figure, gray cells indicate normal cells, and red cells indicate superimposed cells. As seen in this figure, the use of SLP generated the superimposed cells on attached boundary between segment 2 and 3. Fig. 2.14 shows the examples of the mesh deformations from two different views. In this figure, the groove was separated even in the case without the SLP. However, the attached boundary between segments 2 and 3 was not separated. In contrast, using SLP, the boundary was separated as expected. This results show that our algorithm based on the SLPs worked well and is effective for the separation of completely attached segments.




(a) Label map and its cross-sectional view. Voxels are rendered as cubes.

(b) Surface rendering.

Figure 2.15: Label map of brain used in the evaluation. This data is called SPL brain atlas [2] and rendered in our simulator.

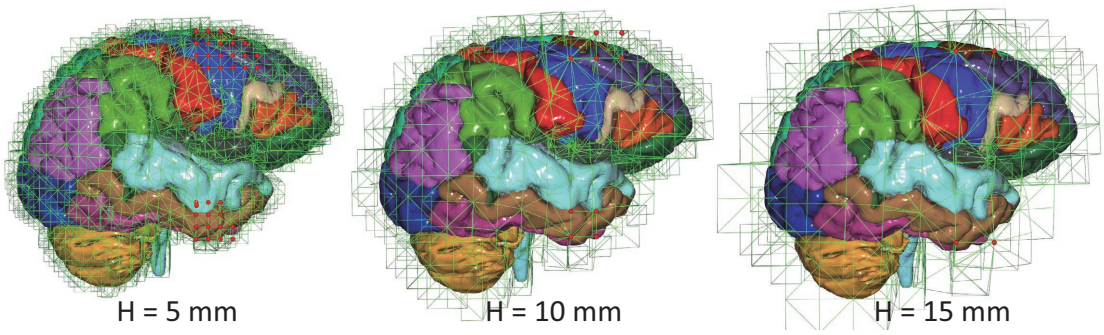



Figure 2.16: Deformation results in the evaluation using the brain model. The Sylvian fissures are opened in all cases by forced displacements. Green wireframes are FE meshes and Red spheres are position constrained nodes.

### 2.7.4 Evaluation Using a Brain Model

Fig. 2.15 shows the label map ($256 \times 256 \times 256$) used in the evaluation for meshing of a brain model, our target organ in this study. The label map is a part of SPL brain atlas dataset provided by Halle et al. [2]. In this evaluation, SLPs for the separation of the Sylvian fissure (described in Fig. 2.15) were generated manually by the authors. Full list of the used SLPs are shown in Appendix A. The several different mesh sizes were used for the comparisons with respect to the effect of the resolution. Fig. 2.16 shows the deformation results of the meshes obtained by our method. In this results, the forced displacements was imposed to open the Sylvian fissure. As seen in this figure, the Sylvian fissure was opened in

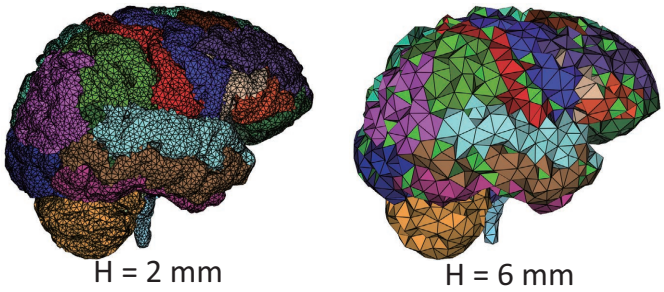



Figure 2.17: Meshes obtained by a method using Delaunay-based triangulation [4]

Table 2.1: Results of the evaluation using the brain model [2]. $N_{\text{vert}}$ and $N_{\text{tet}}$ are the number of vertices and tetrahedra, $T_{\text{mesh}}$ and $T_{\text{fem}}$ are the computational time taken for the mesh generation and a loop of FEM simulation, respectively. Reproduced from [5]

| Method | Mesh size (mm) | $N_{\text{vert}}$ | $N_{\text{tet}}$ | $T_{\text{mesh}}$ (s) | $T_{\text{fem}}$ (ms) |
|---|---|---|---|---|---|
| Pons, et al. [4] | 2.0 | 149,506 | 799,165 | 22.23 | - |
| Pons, et al. [4] | 4.0 | 44,421 | 95,740 | 7.43 | - |
| Pons, et al. [4] | 6.0 | 6,106 | 29,144 | 1.28 | - |
| Ours | 2.0 | 227,225 | 925,360 | 5.78 | - |
| Ours | 5.0 | 22,999 | 79,650 | 0.57 | 133.3 |
| Ours | 10.0 | 4,829 | 14,130 | 0.19 | 30.8 |
| Ours | 15.0 | 2,072 | 5,530 | 0.16 | 11.9 |

all cases as expected even in very coarse mesh with the mesh size of 15 mm.

To compare the method with a well-established method, we also conducted Delaunay-based meshing proposed in [4] (Fig. 2.17). We used an implementation in CGAL library [36] for Delaunay-based meshing. When we used Delaunay-based meshing with a too low resolution, as noted in [4], non-manifold meshes, that involves vertex singularity, was produced. Such singularity can cause the instability in FEM simulation. In addition, thin or small segments vanished in the obtained mesh. Furthur, the boundary of Sylvian fissures of the meshes look jaggy in coarse resolution, which lead to poor visual plausibility in the real-time simulations.

The quantitative results (number of nodes and tetrahedra and timing results)

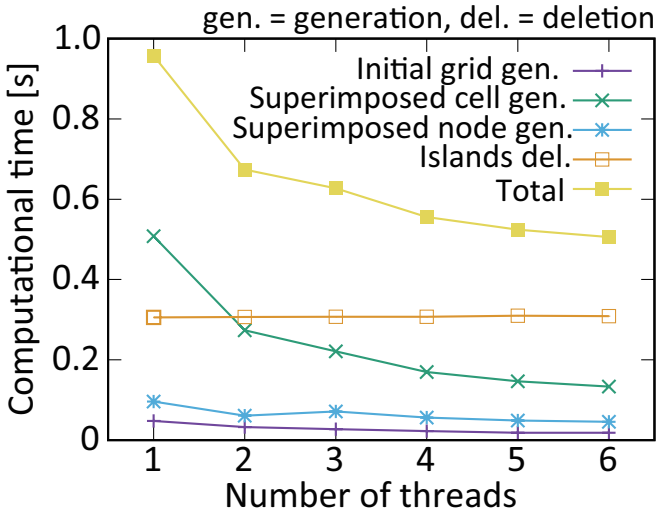



Figure 2.18: Parallel scalability. Adapted from [5]

are described in Table 2.1. The timing measurements of the FEM simulation (computational times spend on matrix assembler and linear solver with 20 iterations in conjugate gradient method) are performed for meshes obtained by our method. Note that the simulation was not executed for the mesh with $H = 2.0$ because the mesh was too large to store the related data in the GPU memory. This table shows that our meshing method is very fast and easily modify the resolution interactively. It would help users construct patient-specific model that have appropriate mesh resolution for real-time simulation. In this evaluation, we was able to find that the mesh size with 10 mm was a reasonable mesh for real-time simulation because the FEM calculation takes approximately 30 ms ($\approx$ 30 FPS).

We also investigated the applicability of the parallelization techniques. In our algorithm the superimposed cell generation and the superimposed node generation are cell-independent and node-independent procedures, respectively, and thus it can be parallelized straightforwardly on multi-core CPU. Fig. 2.18 shows the computational times of each algorithm stage of the mesh generation with different number of threads. In this result, good scalability was observed. This performance is suitable for current computer architecture that involves many core in a CPU package.

We show an example of postprocess application. Our volume embedding approach enables to deform the volume data and export the volume data by resam-

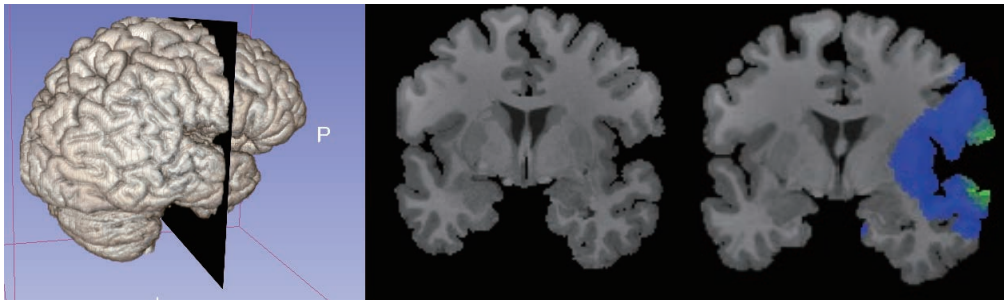



Figure 2.19: Visualization of deformed volume in 3D Slicer [6]. The left figure is the slicing position. The middle figure is the cross section image of the deformed volume. The right figure is the cross section image with stress field overlay.

pling the MRI intensites using interpolation techniques. The exported data can be visualized using viewers that supports medical image format. Fig. 2.19 shows a visualization example using 3D Slicer [6]. This was obtained by exporting a deformed volume with mesh size 5 mm. Note that the original MR image was obtained preoperatively and Sylvian fissure should be closed. In the visualization example (middle image), the Sylvian fissure is opened. This is because the image was deformed according to the displacements imposed in FE analysis as described above. Furthermore, the stress field inside the volume was overlaid (right figure). This is valuable , e.g., when assessing the risk of retraction injury with evidence of stress concentrations.

Although we have focused on the separation of the Sylvian fissure, our method can separate the boundary between differently segmented regions. Therefore, the other fissures can also be separated. To confirm this advantage, we tried to separate the cerebellum and longitudial fissure.

### 2.7.5 Evaluation Using a Abdominal Model

Although we have developed a meshing algorithm for brain retraction, our method can be used for simulations of the other organs. To provide an example, we conducted a simulation using a abdominal model. We used a label map that is a part of SPL abdominal atlas dataset provided by Talos et al. [42]. We resampled the label map to 1 mm voxel size and set the mesh size to 5 mm. To separate the

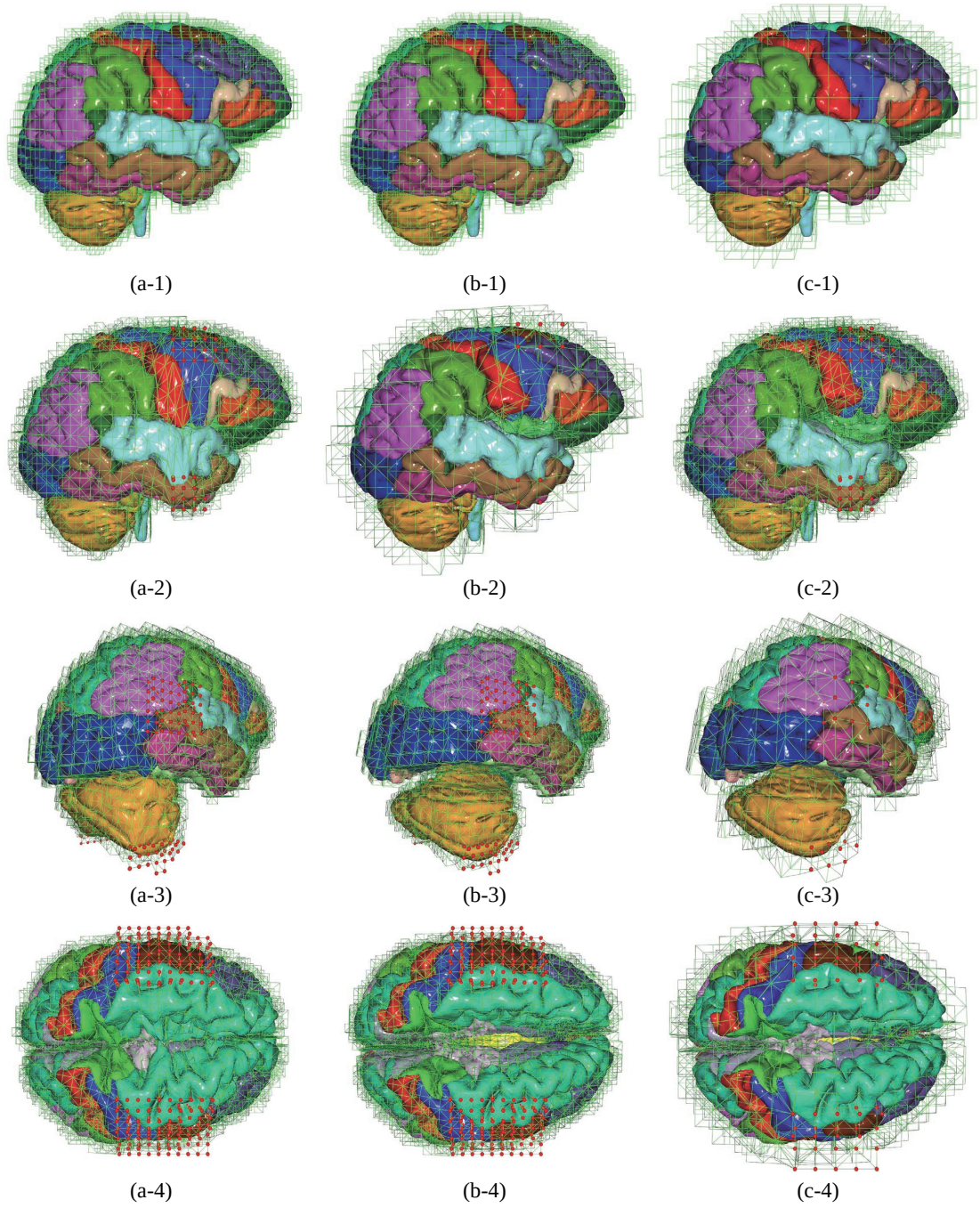



Figure 2.20: Deformation of meshes. (a-*): $n_{\text{voxels}} = 5$, without SLPs. (b-*): $n_{\text{voxels}} = 5$, with SLPs. (c-*): $n_{\text{voxels}} = 10$, with SLPs. (*-1): rest shape. (*-2): shape when the Sylvian fissure is opened. (*-3): shape when the cerebellum is pushed. (*-4): shape when the longitudial fissure opened.

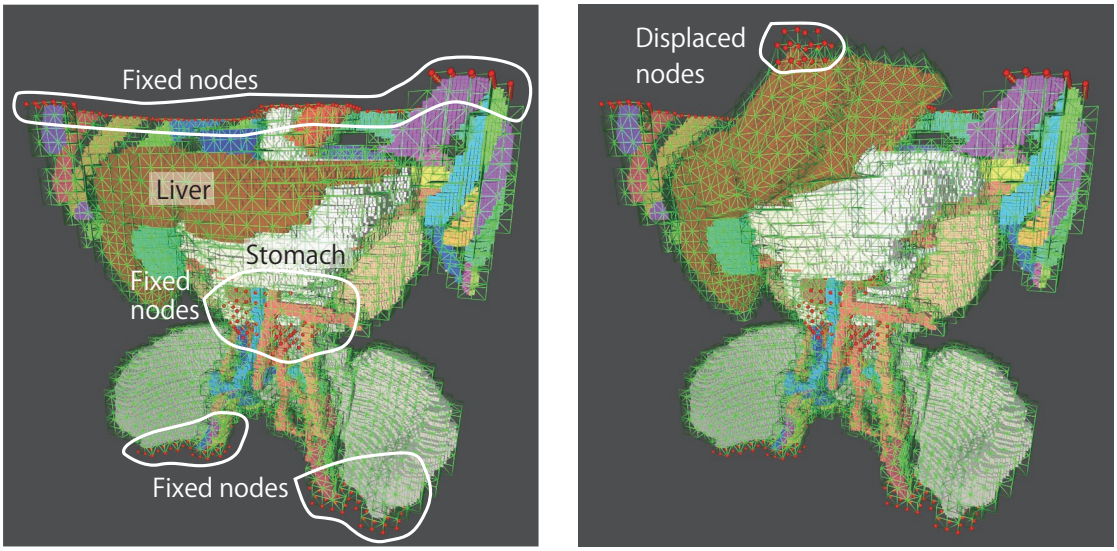



Figure 2.21: Result of the evaluation using a abdominal model. Adopted from [8]

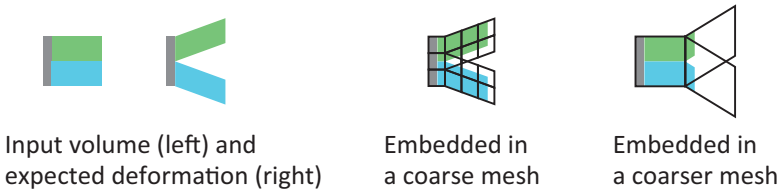



Figure 2.22: Limitation of the topological preservability.

liver and stomach, the SLP {3 (stomach), 13 (liver)} was input to the meshing algorithm. Fig. 2.21 shows the result of the simulation. In the simulation, the liver was displaced upperward and separated from the stomach. This results imply that our grid-based meshing algorithm can applied to various surgical situations that involves complex geometry.

## 2.8 Limitations

Finally, we discuss on the limitation of the topological preservability. As mesh size $H$ increases, small gaps with lengths that are smaller than $H$ were integrated in a cell as illustrated in Fig. 2.22. Thus, we need to determine the mesh size with a consideration of the boundary length that user wish to preserve.

Because we used regular grid for meshing the volumetric domain, the boundary conformity of the FE mesh was ignored. This approximation decreases in the

accuracy of the physics simulations. This problem can be mitigated considering the modification of the element stiffness matrix on the basis of the spatial distribution of labels inside a finite element [40]. To address this problem, the knowledge on the computational mechanics would be helpful. The method called Finite Cover Method and homogenization are related to this issue.

Furthermore, we have not discussed on the contact handling for embedded volumes. This topic is discussed in Chapter 5.

Moreover, the automated segmentation method should be customized for specific applications. In our evaluations, we used well-segmented models (atlas). However, in an actual situation, automated segmentation should be performed for efficient model generation. For the brain retraction simulation, we tried the implementation on 3D Slicer [6], which is called EMSegmenter with "MRI Human Brain Full Parcellation" predefined task. Although this implementation works well, some small segmentation errors can be generated in most cases. Fig. 2.23 shows a 2D deformation of a label map with segmentation failures. This is an example of deformations using a mesh generated by our method. The label map was generated using the 3D Slicer EMSegmenter module. The input MR image was NAMIC: Brain Mutlimodality "case01011" provided by National Alliance for Medical Image Computing [7]. In the label map, a part of parietal lobe is segmented temporal lobe (Fig. 2.23 (a)). Because our algorithm preserves the topology according to a label map, resulting mesh can deform unexpectedly if the map includes segmentation errors (Fig. 2.23 (b)). Although we can correct such error interactively in 2D (Fig. 2.23 (c)), this correction is difficult for 3D complex segmentation. In order to avoid the problem, precise segmentation method or intuitive correction interfaces are required.

## 2.9 Summary

In this chapter, a mesh generation method for volume embedding approach was introduced. This method preserves the connectivity of an input shape defined as segmented volume data using superimposed cells. This method also considers the separation of the boundary of differently labeled regions. Such boundaries

are described using SLP by a user and the proposed algorithm try to separate the boundaries even when they are attached without gaps. The proposed method succeeded in preserving the topology of the brain fissures with good efficiency and robustness.

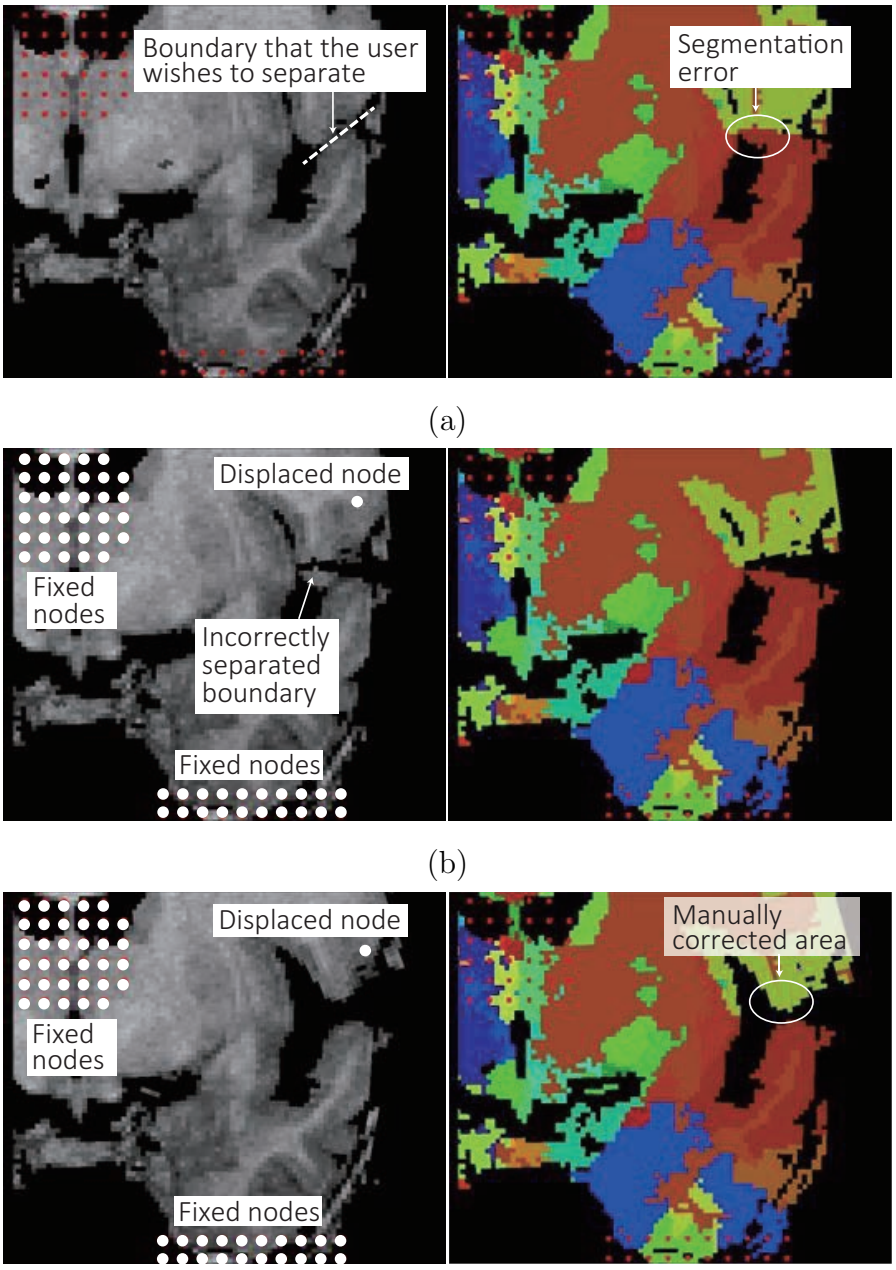



Figure 2.23: Deformation of a label map with segmentation failures and its corrected label map in 2D. Adopted from [9].

# Chapter 3. Real-time Simulation of Soft-tissue Deformation

## 3.1 Introduction

In this chapter, a real-time simulation method for soft-tissue deformation and fracture is described. The brain tissues are modeled as linear elastic material, and the deformation is calculated using FEM. The geometrical nonlinearity is considered by adopting a corotational formulation. Additionally, a fracture representation of brain tissues is described. The proposed methods are evaluated on the calculation speed and the stability in fracture simulation.

Please note that this chapter does not describe the consideration of nonconforming mesh proposed in chapter 2. This chapter focuses on the general procedure of FEM and its acceleration using a GPU. The contact problem considering nonconforming boundaries is described in chapter 5.

The contributions of the method proposed in chapter are described as the followings:

- An acceleration method using GPU for a corotational linear FEM with frequent change in the region of geometrical boundary condition.
- A stabilization method for the fracture simulation on the basis of the element removals is integrated in the proposed FEM framework.

In general, the most time-consuming procedures of FEM are the stiffness matrix assembly and solving linear simultaneous equation. In addition, the collision response procedure requires additional costs. In the FEM solver described in this chapter, the collision response is calculated by imposing forced displacements (ge-

ometrical boundary conditions). Because general FEM solvers do not consider the frequent changes in the regions applied geometrical boundary conditions, efficient matrix rearrangement algorithm have not been discussed in the field of real-time physics simulation. In this chapter, the implementation of the three procedures mentioned above, i.e., the stiffness matrix assembly, the linear solver, and the stiffness matrix rearrangement, are described by considering sparse-matrix storage formats (Section 3.4). All algorithms are designed for the paralellization on a GPU. However, these algorithms presented in this chapter assume that the mesh topology is not changed during a simulation. Therefore, topology changes such as node additions are forbidden. In order to enable to simulate the soft-tissue dissection under these limitations, we utilized a fracture representation based on the element removal approach. In this approach, fracture behaviors are computed by disabling the contributions to global stiffness of removed elements. However, it has been reported that the dynamic simulation become unstable, which can lead to vibration when the tetrahedral mesh becomes nonmanifold due to the element removals [43]. To address the instability issue, we also developed an element removal algorithm with topological singularities avoidance (Section 3.5). The concept of the avoidance algorithm is aggressive element removal. Its implementation is very simple and can be used as alternative to existing methods, e.g. [44]. Finally, the method was evaluated by conducting a blunt dissection and brain retraction simulations.

## 3.2 Related works

### 3.2.1 Collision Response

Recent studies on the haptic rendering for interaction with deformable environments have focused on efficient contact handling considering collisions between multiple soft objects as well as self-collisions [45, 46]. They have modeled contact response on the basis of Signorini's law and Coulomb's law, and the linear or nonlinear complementarity problem needs to be solved. Although they have developed efficient GPU implementations, the number of nodes is limited to several thousands for real-time simulation because of their high computational burdens.

The fastest method for collision response may be the penalty method [47]. In penalty method, contact forces are applied to contact nodes which formulated using the depth to the penetration in a colliding object. The magnitude of the contact force is often formulated by the multiplication of the penetration depth and penalty parameters. However, the determination of the penalty parameters for realizing stable simulation is difficult. To obtain a stable behavior, the time integration should be carefully formulated. This approach is described in Chapter 4.

The other approach is the position-constraint method. In this method, nodal displacements are directly imposed according to geometrical relationships. In an early study on a real-time deformable model for surgery simulator, Cotin et al. proposed a position-constraint method using the Lagrange multiplier method [48]. A few years later, Hirota et al. adopted a boundary-condition-based constraint [49]. These method are basically equivalent and both methods resulting in a large simultaneous linear equations. The method by Hirota et al. might be faster because the size of the linear equations is smaller than that of [48](see Section 3.3.3). The limitation of the position-constraint-based method is that the inability to compute accurate contact responses subject to Signorini's law or Coulomb's law. However, the computational cost is lower than that of the accurate methods (method based on Signorini's law or Coulomb's law), the position-constraint method can be utilized for a high-resolution models.

In general, the stiffness matrix of FEM is a large sparse matrix, in which most of the components are zero. Therefore, the stiffness matrix should be stored as a sparse matrix format. However, past boundary-condition-based studies have stored the stiffness matrix in dense matrix format. This might be faster for a very coarse mesh model (e.g. less than 1,000 nodes). However, when we consider to use a large and high-resolution mesh (e.g. mesh with approximately 10,000 nodes), memory consumptions and calculation cost become larger than the implementation with dense matrix format. Therefore, GPU-based implementations with sparse matrix format is required for fast calculation of high-resolution deformable models.

### 3.2.2 Fracture and Cutting

In the community of computer graphics, a lot of fracture simulation methods have been proposed (e.g. [50, 51, 52]). This section reviews several important algorithms developed for real-time fracture simulations.

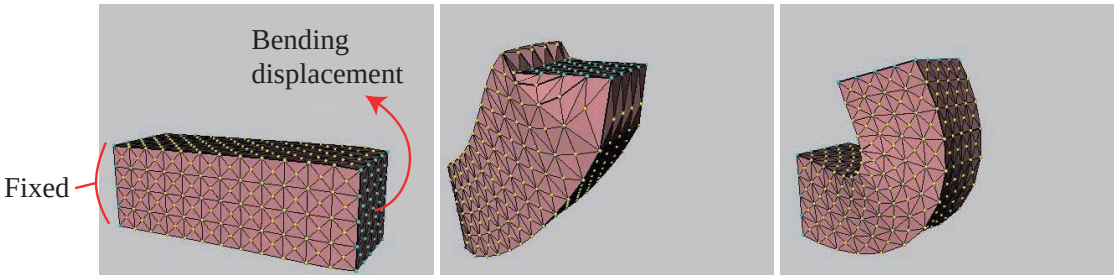
Mor and Kanade developed the model of knife cutting based on explicit mesh modification [53]. They investigated split patterns in which a tetrahedron is divided into smaller tetrahedra according to the path of the knife movement. This approach explicitly modifies a tetrahedral mesh by adding new nodes to it, and thus, the number of nodes are increased as the cutting procedure are conducted.

Jerabkova et al. have developed a method based on the extended FEM [54], in which a crack is represented by modification of interpolation functions of finite elements. This method does not requires the mesh topology modification and might free us from the complex implementation of explicit online mesh modification. However, this approach increase in the degree of freedom (DOF) as the cracks propagated and the computational cost can be increased.

In the early years of the studies of surgery simulation, Delingette et al. developed an method based on the element removal [55]. The main drawback of this approach is a loss of volume. However, it offers a low calculation cost and simple implementation. In this approach, no nodes are added as the fracture propagated. Therefore, the simulation can be run at the constant computational cost. Because we wish to make high-speed calculation top priority, we adopt this element removal approach.

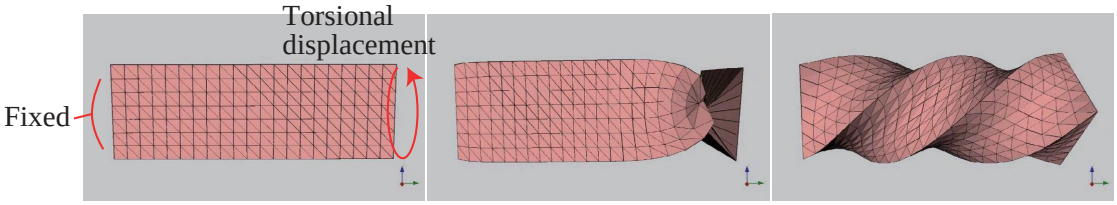
However, as pointed by Forest et al., element removals may lead a tetrahedral mesh to become nonmanifold [43], which means that the tetrahedral mesh has vertices or edges on which the thickness of the volumetric mesh cannot be defined. Such vertices and edges are called singular vertices and singular edges, respectively, and the singularity is called a topological singularity. Fig. 3.10 shows examples of topological singularity. Because the dynamic simulation can be unstable when the FE mesh is a nonmanifold geometry, topological singularities should be avoided.

Some techniques of topological-singularity avoidance have been discussed in the literature. Forest et al. introduced an algorithm based on node separation [43]. In their method, the singular vertices and singular edges are separated by adding




(a) Initial shape (b) Linear FEM (c) Corotational FEM

Figure 3.1: Bending deformations with linear FEM and corotatioanl FEM.




(a) Initial shape (b) Linear FEM (c) Corotational FEM

Figure 3.2: Torsional deformations with linear FEM and corotatioanl FEM.

copies of the singular vertices and edges. However, this approach increases the computational burden because the DOFs of the mesh increase with the node additions. Nakayama et al. developed an algorithm that suspends the element removals that cause topological singularities [44]. However, such delay algorithm does not correctly simulate realistic fracture phenomena because the stress will be concentrated at a singular vertex and singular edge and such vertex and edges should be disconnected immediately.

## 3.3 Finite Element Method

### 3.3.1 Corotational FEM

In this study, we adopted corotational FEM for calculating the deformation of soft tissues. In the corotational FEM, the geometrical nonlineality is considered by modifying the element stiffness matrix defined in infinitesimal strain theory. The corotational FEM can be implemented with small modification of linear FEM. This method enhances the visual plausibility of the simulation and is a reason-

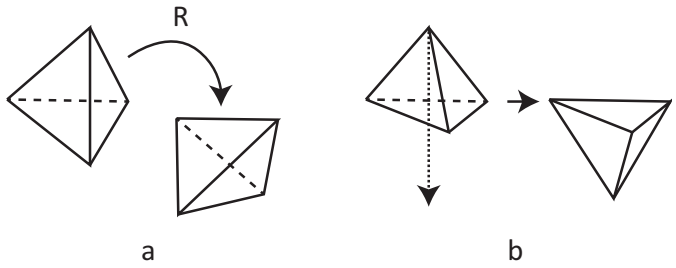



Figure 3.3: Deformation of a tetrahedron. (a) Geometrical nonlinearity. (b) Inversion of an element. Reproduced from [1].

able choice for trading off accuracy and computational cost [56]. In particular, corotational FEM can calculate the deformations involves element rotations such as bending (Fig. 3.1) or torsional deformation (Fig. 3.2) better than linear FEM. In this formulation, element strains is evaluated in the rotated coordinates, which are called corotational coordinates (see Fig. 3.3(a)). A corotational coordinate is defined based on the rotation component of a element deformation. The element stiffness equation of a linear FEM is described as

$$\boldsymbol{f}^e_{\text{ext}} = \mathbf{K}^e \boldsymbol{u}^e, \tag{3.1}$$

where $\boldsymbol{f}^e_{\text{ext}}$ and $\boldsymbol{u}^e$ are the element force vector and element displacement vector, respectively. The element displacement vector is defined as $\boldsymbol{u}^e = \boldsymbol{x}^e - \boldsymbol{x}^e_0$, where $\boldsymbol{x}^e$ and $\boldsymbol{x}^e_0$ are the current and initial nodal positions vectors, respectively. Let $\mathbf{R} \in \mathbb{R}^{3\times3}$ be the rotation matrix that represents th erotation of the element coordinates. In the corotationa formulation, the current position and force vectors are transformed using $\mathbf{R}$ and the element stiffness equation can be written as

$$\mathbf{R}^{e\mathrm{T}} \boldsymbol{f}^e_{\text{ext}} = \mathbf{K}^e_0 \left( \mathbf{R}^{e\mathrm{T}} \boldsymbol{x}^e - \boldsymbol{x}^e_0 \right), \tag{3.2}$$

where $\mathbf{K}^e_0$ is the element stiffness matrix of a linear FEM and

$$\mathbf{R}^e \triangleq \text{blockdiag}\left[\mathbf{R}, \mathbf{R}, \mathbf{R}, \mathbf{R}\right].$$

Eq. (3.2) is rewritten as

$$\boldsymbol{f}^e_{\text{ext}} = \mathbf{K}^e \boldsymbol{x}^e - \boldsymbol{f}^e_0, \tag{3.3}$$

where

$$\mathbf{K}^e = \mathbf{R}^e \mathbf{K}^e_0 \mathbf{R}^{e\mathrm{T}}, \tag{3.4}$$

$$\boldsymbol{f}^e_0 = \mathbf{R}^e \mathbf{K}^e_0 \boldsymbol{x}^e_0; \tag{3.5}$$

We call $\boldsymbol{f}_0^e$ the force offset vector.

The rotation matrix $\mathbf{R}$ is obtained by SVD of the deformation gradient tensor $\mathbf{F}$ [57]. This formulation is stable even if the elements are inverted (see Figure 3.3(b)). In the case of the first-order tetrahedral element, $\mathbf{F}$ transforms an edge vector of the initial shape $\boldsymbol{d}_{\mathrm{m}j}$ into an edge vector of the deformed shape $\boldsymbol{d}_{\mathrm{s}j}$ as $\boldsymbol{d}_{\mathrm{s}j} = \mathbf{F}\boldsymbol{d}_{\mathrm{m}j}$ $(j = 1, 2, 3)$. From this equation, $\mathbf{F}$ is calculated as $\mathbf{F} = \mathbf{D}_{\mathrm{s}}\mathbf{D}_{\mathrm{m}}^{-1}$, where $\mathbf{D}_{\mathrm{s}} = [\boldsymbol{d}_{\mathrm{s}1}\ \boldsymbol{d}_{\mathrm{s}2}\ \boldsymbol{d}_{\mathrm{s}3}]$ and $\mathbf{D}_{\mathrm{m}} = [\boldsymbol{d}_{\mathrm{m}1}\ \boldsymbol{d}_{\mathrm{m}2}\ \boldsymbol{d}_{\mathrm{m}3}]$. $\mathbf{F}$ can be represented as

$$\mathbf{F} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}, \tag{3.6}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, and $\boldsymbol{\Sigma}$ is a diagonal matrix. The rotation matrix is calculated as

$$\mathbf{R} = \mathbf{U}\mathbf{C}\mathbf{V}^{\mathrm{T}}, \tag{3.7}$$

where $\mathbf{C} \triangleq \mathrm{diag}\left[1, 1, \det(\mathbf{U}\mathbf{V}^{\mathrm{T}})\right]$ [58].

To obtain the rotation matrices of the tetrahedral elements, SVD of a large number of $3 \times 3$ matrices described in Eq. (3.6) must be performed. However, most existing parallel implementations of SVD are specialized for large matrices [59]. For SVD of a large number of small matrices, Bedkowski et al. introduced an algorithm for three-dimensional reconstruction using mobile robots [60]. In the present work, the algorithm introduced by Bedkowski et al. is modified. The modified algorithm is summarized as follows:

1. Diagonalize $\mathbf{F}^{\mathrm{T}}\mathbf{F}$ by the Jacobi eigenvalue algorithm as $\mathbf{F}^{\mathrm{T}}\mathbf{F} = \mathbf{V}^{\mathrm{T}}\mathbf{S}\mathbf{V}$, where $\mathbf{V}$ is an orthogonal matrix, and $\mathbf{S}$ is a diagonal matrix whose elements are the eigenvalues of $\mathbf{F}^{\mathrm{T}}\mathbf{F}$.

2. Construct a matrix $\boldsymbol{\Sigma}$ whose diagonal elements are the singular values of $\mathbf{F}^{\mathrm{T}}\mathbf{F}$. The singular values are obtained by calculating the square root of each diagonal element of $\mathbf{S}$.

3. Calculate $\mathbf{U} = \mathbf{F}\mathbf{V}\boldsymbol{\Sigma}^{-1}$.

4. $\mathbf{U}$ and $\mathbf{V}$ are used in Eq. (3.7).

In this algorithm, the eigenvalue approach is different from that of Bedkowski et al. They calculated the eigenvalues by obtaining the roots of a cubic polynomial.

On the other hand, we adopted the Jacobi eigenvalue algorithm to simplify the implementation.

### 3.3.2 Matrix Assembly

After $\mathbf{K}^e$ and $\boldsymbol{f}_0^e$ are obtained, they are assembled and integrated in the global stiffness matrix $\mathbf{K} \in \mathbb{R}^{3N_{\text{node}} \times 3N_{\text{node}}}$ and global force offset vector $\boldsymbol{f}_0 \in \mathbb{R}^{3N_{\text{node}}}$ , where $N_{\text{node}}$ is the number of nodes. The assembly is processed by gathering all element contributions using mesh topology. This procedure is referred to as matrix assembly. As the mathematical expression, the assembly can be written as the following equations.

$$\mathbf{K} = \sum_e \mathbf{L}^{e\mathrm{T}} \mathbf{K}^e \mathbf{L}^e, \tag{3.8}$$

$$\boldsymbol{f}_0 = \sum_e \mathbf{L}^{e\mathrm{T}} \boldsymbol{f}_0^e, \tag{3.9}$$

where $\mathbf{L}^e \in \mathbb{R}^{12 \times 3N_{\text{node}}}$ is the gather matrix, which extract element nodal values from global vectors. $\mathbf{L}^e$ consists of zeros and ones (Boolean matrix). Eq. (3.8) and Eq. (3.9) are just mathematical formulations, and the implementation of the assembly process is realized in an efficient manner (the matrix multiplication is not performed). We describe the efficient implementations in Section 3.4.3.

Let $\boldsymbol{f}_{\text{ext}}$ and $\boldsymbol{x}$ be the global external force vector and global position vector, respectively. The global stiffness equation can be written as

$$\boldsymbol{f}_{\text{ext}} = \mathbf{K}\boldsymbol{x} - \boldsymbol{f}_0. \tag{3.10}$$

This equation can be written in analogy with the global stiffness equation formulated in a linear FEM as

$$\boldsymbol{f} = \mathbf{K}\boldsymbol{x}, \tag{3.11}$$

where $\boldsymbol{f} = \boldsymbol{f}_{\text{ext}} - \boldsymbol{f}_0$.

### 3.3.3 Boundary Conditions

In this chapter, we consider that the contact response is calculated by applying the geometrical boundary conditions (Note that we also discuss on the contact

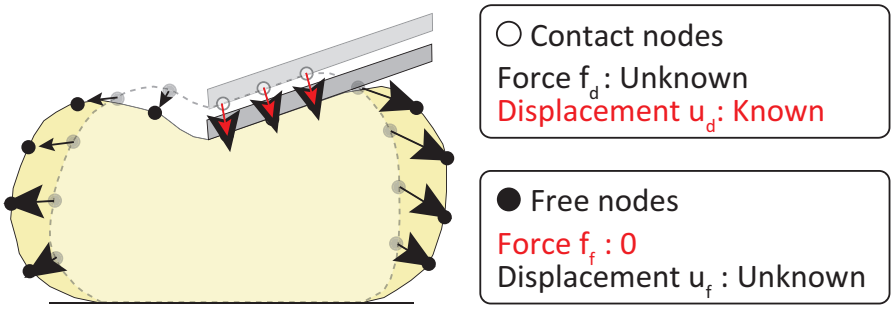



Figure 3.4: Contact nodes and free nodes. Reproduced from [1].

response based on penalty method in Chapter 4). We assume that the forced displacements for a deformable object due to contact of surgical instruments are obtained by collision detection algorithms. When a contact between the brain model and surgical instruments are detected, the forced displacements are imposed to the contact nodes (Fig. 3.4). Therefore, the displacements of the contact nodes are known, but, their external forces are unknown. In contrast, the displacements of free nodes are unknown, but their external forces are known (the external forces are zeros). Following the method proposed in Hirota et al. [49], the components in the matrix and vectors of Eq. (3.11) are rearranged into displacement-known and force-known variables, and the equation can be modified as:

$$\begin{bmatrix} \boldsymbol{f}_{\mathrm{f}} \\ \boldsymbol{f}_{\mathrm{d}} \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{\mathrm{ff}} & \mathbf{K}_{\mathrm{fd}} \\ \mathbf{K}_{\mathrm{df}} & \mathbf{K}_{\mathrm{dd}} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{\mathrm{f}} \\ \boldsymbol{x}_{\mathrm{d}} \end{bmatrix}, \tag{3.12}$$

where the suffixes d and f denote the components of the displacement-known and force-known nodes, respectively. In Eq. (3.12), $\boldsymbol{f}_{\mathrm{d}}$ and $\boldsymbol{x}_{\mathrm{f}}$ are unknown, and $\boldsymbol{x}_{\mathrm{f}}$ is calculated by solving the following linear simultaneous equation:

$$\mathbf{K}_{\mathrm{ff}}\boldsymbol{x}_{\mathrm{f}} = \boldsymbol{f}_{\mathrm{f}} - \mathbf{K}_{\mathrm{fd}}\boldsymbol{x}_{\mathrm{d}}. \tag{3.13}$$

After $\boldsymbol{x}_{\mathrm{f}}$ is obtained, $\boldsymbol{f}_{\mathrm{d}}$ is calculated as

$$\boldsymbol{f}_{\mathrm{d}} = \mathbf{K}_{\mathrm{df}}\boldsymbol{x}_{\mathrm{f}} + \mathbf{K}_{\mathrm{dd}}\boldsymbol{x}_{\mathrm{d}}. \tag{3.14}$$

We described only the formulation of the static FEM above. The dynamic formulation with implicit time integration becomes a mathematically similar equation to that of static FEM. The equation of motion for a deformable object is written

as

$$\mathbf{M}\ddot{\boldsymbol{x}} + \mathbf{C}\dot{\boldsymbol{x}} + (\mathbf{K}\boldsymbol{x} + \boldsymbol{f}_0) = \boldsymbol{f}_{\text{ext}}, \tag{3.15}$$

where $\mathbf{M}$ and $\mathbf{C}$ are a mass matrix and a damping matrix, respectively. $\mathbf{M}$ is a diagonal matrix determined by gathering the equivalent masses of all nodes from the node-share tetrahedrons: $m_i = \sum_{T_i} m_{T_i}/4$, where $m_i$ is the equivalent mass of node $i$, $T_i$ is a tetrahedron that shares node $i$, and $m_{T_i}$ is the mass of $T_i$. In general, $\mathbf{C}$ is determined on the basis of the material constitutive law. However, for simplicity, Rayleigh damping is adopted in this study:

$$\mathbf{C} = \alpha\mathbf{M} + \beta\mathbf{K}, \tag{3.16}$$

where $\alpha$ and $\beta$ are scalar values representing the damping effect, which are selected heuristically for stabilizing the simulation. Eq. (3.15) can be written in the same form as the linear FEM form as

$$\mathbf{M}\ddot{\boldsymbol{x}} + \mathbf{C}\dot{\boldsymbol{x}} + \mathbf{K}\boldsymbol{x} = \boldsymbol{f} \tag{3.17}$$

by defining a vector $\boldsymbol{f} = \boldsymbol{f}_{\text{ext}} - \boldsymbol{f}_0$. When we substitute $\boldsymbol{v}$ for $\dot{\boldsymbol{x}}$, the time derivatives of the variables are defined as

$$\dot{\boldsymbol{x}} = \boldsymbol{v}, \tag{3.18}$$

$$\mathbf{M}\dot{\boldsymbol{v}} = -\mathbf{C}\boldsymbol{v} - \mathbf{K}\boldsymbol{x} + \boldsymbol{f}. \tag{3.19}$$

In order to avoid numerical instability in the dynamic simulation, we adopt implicit time integration because it has unconditionally stable characteristics. Implicit time integration is formulated as

$$\boldsymbol{x}^{i+1} = \boldsymbol{x}^i + \Delta t\, \boldsymbol{v}^{i+1}, \tag{3.20}$$

$$\mathbf{M}\boldsymbol{v}^{i+1} = \mathbf{M}\boldsymbol{v}^i + \Delta t \left(-\mathbf{C}\boldsymbol{v}^{i+1} - \mathbf{K}\boldsymbol{x}^{i+1} + \boldsymbol{f}^{i+1}\right). \tag{3.21}$$

By substituting Eqs. (3.16) and (3.20) into Eq. (3.21), $\boldsymbol{v}^{i+1}$ can be obtained by solving the following equation:

$$\left((1 + \alpha\Delta t)\,\mathbf{M} + \left(\beta\Delta t + \Delta t^2\right)\mathbf{K}\right)\boldsymbol{v}^{i+1} = \mathbf{M}\boldsymbol{v}^i + \Delta t \left(-\mathbf{K}\boldsymbol{x}^i + \boldsymbol{f}^{i+1}\right). \tag{3.22}$$

As discussed in Section 3.3.3, the contact nodes move together with the rigid body; hence, $\boldsymbol{x}_{\text{d}}$ and $\boldsymbol{v}_{\text{d}}$ are known, whereas $\boldsymbol{f}_{\text{d}}$ is unknown. The forces applied to the

unconstrained nodes are zero, i.e., $\boldsymbol{f}_{\mathrm{f}} = \mathbf{0}$. Therefore, Eq. (3.22) can be rewritten as

$$\left((1+\alpha\Delta t)\,\bar{\mathbf{M}} + \left(\beta\Delta t + \Delta t^2\right)\bar{\mathbf{K}}\right)\bar{\boldsymbol{v}}^{i+1} = \bar{\mathbf{M}}\bar{\boldsymbol{v}}^{i} + \Delta t\left(-\bar{\mathbf{K}}\bar{\boldsymbol{x}}^{i} + \bar{\boldsymbol{f}}^{i+1}\right), \tag{3.23}$$

where

$$\bar{\mathbf{M}} = \begin{bmatrix} \mathbf{M}_{\mathrm{f}} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{\mathrm{d}} \end{bmatrix},\ \bar{\mathbf{K}} = \begin{bmatrix} \mathbf{K}_{\mathrm{ff}} & \mathbf{K}_{\mathrm{fd}} \\ \mathbf{K}_{\mathrm{df}} & \mathbf{K}_{\mathrm{dd}} \end{bmatrix},\ \bar{\boldsymbol{v}} = \begin{bmatrix} \boldsymbol{v}_{\mathrm{f}} \\ \boldsymbol{v}_{\mathrm{d}} \end{bmatrix},\ \bar{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x}_{\mathrm{f}} \\ \boldsymbol{x}_{\mathrm{d}} \end{bmatrix},\ \bar{\boldsymbol{f}} = \begin{bmatrix} \boldsymbol{f}_{\mathrm{f}} \\ \boldsymbol{f}_{\mathrm{d}} \end{bmatrix}.$$

Eq. (3.23) is rewritten as

$$\begin{bmatrix} \mathbf{A}_{\mathrm{ff}} & \mathbf{A}_{\mathrm{fd}} \\ \mathbf{A}_{\mathrm{df}} & \mathbf{A}_{\mathrm{dd}} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_{\mathrm{f}}^{i+1} \\ \boldsymbol{v}_{\mathrm{d}}^{i+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_{\mathrm{f}}^{i+1} \\ \boldsymbol{b}_{\mathrm{d}}^{i+1} \end{bmatrix}. \tag{3.24}$$

## 3.4 GPU Parallelization

This section describes a GPU implementation of the FEM described in Section 3.3. In this section, some performance measurements are shown. For the performance measurements, a CIARA KRONOS S810R workstation was used. The workstation mounts an Intel Core i7-3960X (six cores, overclocked to 4.5 [GHz]) CPU, 64 [GB] of RAM, and two GPUs: an NVIDIA K20c (2,496 CUDA cores) for numerical calculation and an NVIDIA Quadro K5000 (1,536 CUDA cores) for graphics rendering. The algorithms are parallelized using OpenMP for multithreading on a multicore CPU and NVIDIA CUDA for general-purpose computing on a GPU.

### 3.4.1 Simulation Procedures

Fig. 3.5 shows the flowchart of the boundary-condition-based contact simulation scheme. Before the real-time loop is started, the element stiffness matrices $\mathbf{K}_0^e$ and the reduction arrays explained in Section 3.4.3 are calculated. The procedures in the real-time loops are as follows.

**Element data calculation: $\mathbf{R}^{\mathbf{e}}$, $\mathbf{K}^{\mathbf{e}}$,** and $\boldsymbol{f}_0^e$ are calculated. The parallel implementations of this stage are described in Section 3.4.2.

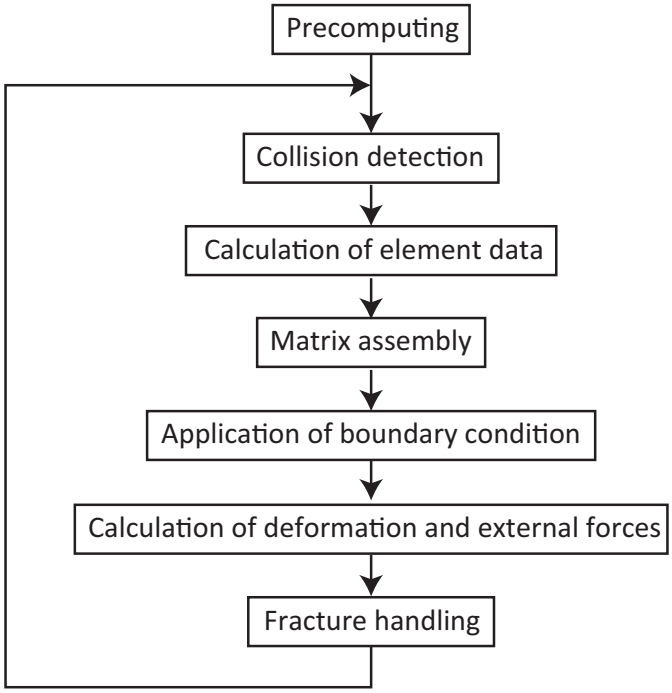



Figure 3.5: Flowchart of the simulation scheme. Reproduced from [1].

**Matrix assembly:** $\mathbf{K}$ and $\boldsymbol{f}_0$ are assembled. The parallel implementation of the assembly are described in Section 3.4.3.

**Collision detection:** Intersections between a deformable object (brain) and rigid objects (brain spatulas) are tested. The contact nodes of the deformable object and the corresponding forced displacements are determined. The discrete collision detection approach reported in [61] is adopted. This method can deal with collisions between a nonconvex deformable object and a rigid object.

**Boundary condition application:** On the basis of collision detection, a boundary condition is set. As mentioned in Section 3.3.3, a large sparse matrix is rearranged according to the boundary condition. The implementation details are described in Section 3.4.4.

**Calculation of deformation and forces:** The deformation is calculated by solving a system of linear equations. The linear equations are solved by the conjugate gradient method. Sparse-matrix dense-vector multiplications are

implemented by the sparse-matrix library CUSPARSE provided by NVIDIA Corp.

### 3.4.2 Element Data Calculation

An element stiffness matrix $\mathbf{K}^e$ is formulated as Eq. (3.4). We assume the material isotropy; thus, $\mathbf{K}^e$ is a symmetric matrix. Therefore, only the elements of the upper triangular matrix of $\mathbf{K}^e$ should be stored. Further, $\boldsymbol{f}_0^e$ is obtained in the same way as $\mathbf{K}^e$ (Eq. (3.5)). Finally, all $\mathbf{K}^e$ ($\mathbf{K}^1$, $\mathbf{K}^2, \ldots, \mathbf{K}^{N_{\text{elem}}}$) and $\boldsymbol{f}_0^e$ ($\boldsymbol{f}_0^1$, $\boldsymbol{f}_0^2$, $\ldots$, $\boldsymbol{f}_0^{N_{\text{elem}}}$), where $N_{\text{elem}}$ is the number of tetrahedral elements, are serialized and stored in the arrays *valuesKe* and *valuesF0e*, respectively. These element data calculation can be executed in parallel using one thread per element.

### 3.4.3 Matrix Assembly in a Sparse Storage Format

In the matrix assembly procedure, the element stiffness matrices (small dense matrices) are added into the global stiffness matrix (a large sparse matrix), as described in Eq. (3.8). Similarly, the element force offset vectors (small dense vectors) are added into the global force offset vector (a large dense vector), as described in Eq. (3.9). In this section, the implementation of the assembly of global stiffness matrix and global force offset vector is described.

In FEM, the global stiffness matrix is a large sparse matrix, in which most of the components are zero. Therefore, the matrix is usually stored in a sparse-matrix storage format to decrease memory consumption. In this work, we store the global stiffness matrix using the coordinate list (COO) sparse storage format at the stage of matrix assembly. This is because of the requirement of the algorithm of the matrix rearrangement (described in Section 3.4.4). The COO matrix format consists of three arrays: *values*, *rowIdx*, and *colIdx*. In this format, only the nonzero components of a matrix are stored in the array *values*. The row and column indices of the nonzero components are stored in the arrays *rowIdx* and *colIdx*, respectively. These arrays are stored in row-major order (sorting is required). For example, the

matrix

$$\begin{bmatrix} a & 0 & 0 \\ b & c & 0 \\ 0 & 0 & d \end{bmatrix}$$

is stored as the following three arrays.

$$\begin{aligned} \mathit{values} &= [a, b, c, d] \\ \mathit{rowIdx} &= [0, 1, 1, 2] \\ \mathit{colIdx} &= [0, 0, 1, 2] \end{aligned}$$

In the remainder of this section, *values*, *rowIdx*, and *colIdx* represent the arrays of $\mathbf{K}$ in the COO format.

In order to achieve a fast calculation of matrix assembly, we adopted the reduction list approach introduced by Cecka et al. [62]. When the mesh connectivity does not change during a simulation, *rowIdx* and *colIdx* are constant. Therefore, only *values* should be recalculated at every time step. The matrix assembly procedure involves lots of independent summations expressed as

$$\mathit{values}(i) = \sum_{j=1}^{N_{\mathrm{src},i}} \mathit{valuesKe}(\mathit{srcIdx}_i(j)), \tag{3.25}$$

where $\mathit{srcIdx}_i$ is an array that stores the source indices that point to the components of *valuesKe* for the summation of the $i$-th component of *values*, and $N_{\mathrm{src},i}$ is the number of components of $\mathit{srcIdx}_i$. The components of *srcIdx* are determined by the mesh topology. In a reduction list, the source indices and a negative-signed destination index are stored as

$$\mathit{reductionList}_i = [\mathit{srcIdx}(1), \mathit{srcIdx}(2), \ldots, \mathit{srcIdx}(N_{\mathrm{src},i}), -i]. \tag{3.26}$$

In general, a reduction list has one destination index as shown in Eq. 3.26. On the other hand, because our stiffness matrix $\mathbf{K}$ is a symmetric matrix, our reduction list stores two destination indices as

$$\mathit{reductionList}_i = [\mathit{srcIdx}(1), \mathit{srcIdx}(2), \ldots, \mathit{srcIdx}(N_{\mathrm{src},i}), -i, -\mathit{lower}(i)], \tag{3.27}$$

where $\mathit{lower}(i)$ is an index that points to the lower component of $\mathit{values}(i)$. An example of reduction-list-based summation is illustrated in Fig. 3.6. There is a

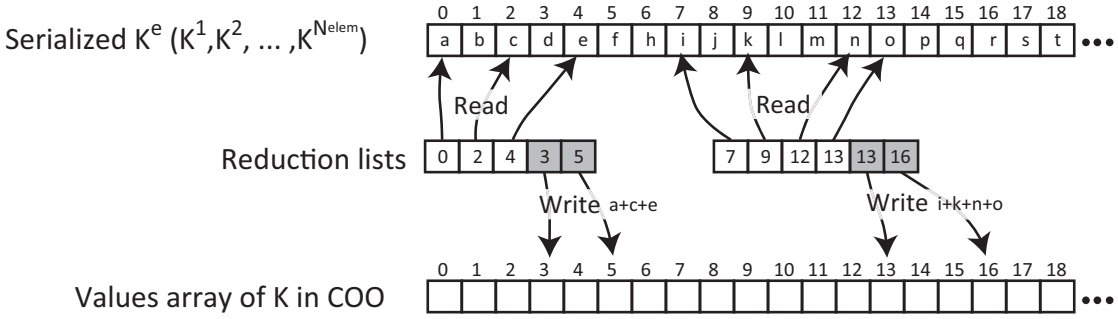



Figure 3.6: Reduction array. Reproduced from [1].

storage format specialized for a symmetric matrix that stores only upper or lower triangular entries as in the case of $\mathbf{K}^e$. However, we did not adopt such a storage format for $\mathbf{K}$ because it requires special treatment in the subsequent procedures, which can degrade the maintainability because of its complexity. Therefore, all the nonzero components of $\mathbf{K}$ are stored as described above. As noted in [62], this approach avoids atomic operations (conflict avoidance for parallel writing operations) because the destination addresses are independent of each other.

The assembly of $\boldsymbol{f}_0$ is performed in a similar way. After we obtain $\mathbf{R}^e$, all values of $\boldsymbol{f}_0^e$ are stored as an array. We construct the reduction list for the assembly of $\boldsymbol{f}_0$ in advance. The reduction is executed in a thread per component of $\boldsymbol{f}_0$, which enables the calculation of $\boldsymbol{f}_0$ without atomic operation. Because this reduction is not depends on the assembly of $\mathbf{K}$, assemblies of $\mathbf{K}$ and $\boldsymbol{f}_0$ are executed concurrently, e.g., on two GPUs.

### 3.4.4 Matrix Rearrangement

In this study, contact responses are calculated by applying geometrical boundary conditions (forced displacements) and the global stiffness matrix is rearranged according to the boundary conditions as described in Section 3.3.3. This rearrangement procedure includes permutation and separation of matrices.

Permutation is executed by using a permutation list. Permutation list includes the destination indices of the permutation. We describe the list as $list(i)$ for the permutation of the $i$-th index. The list is generated according to the boundary conditions. Fig. 3.7 is an example of permutation for in the case of one tetrahedron.

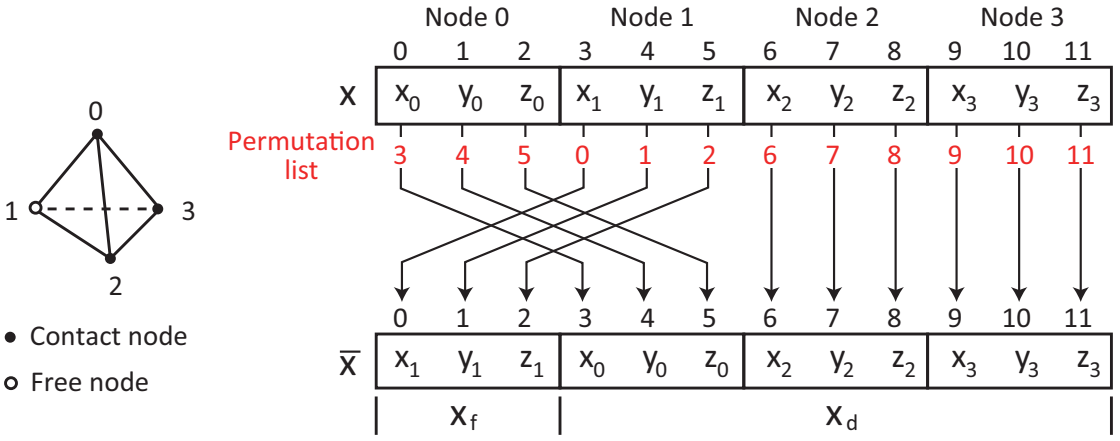



Figure 3.7: Permutation list. Reproduced from [1].

In this example, the nodes 0, 2, and 3 are constrained. The permutation procedure prepare an array that store the resulting permutated array, and copies the values of the contact nodes to the head of the array and those of the free nodes at the bottom. The Permutation is represented as $m'(list(i), list(j)) = m(i, j)$, where $m$ and $m'$ are the source matrix and permutated matrix, respectively. $m(i, j)$ denotes the $i, j$ component of matrix $m$. When a source matrix is stored as a dense matrix format, permutation is executed by simply copying the source components to the destinations. However, when the source matrix is stored in a sparse matrix storage format, the implementation of permutation differs.

In this study, the COO format is adopted for storing global stiffness matrix, as mentioned in Section 3.4.3. For the matrix stored in the COO format, permutation is performed as

$$\begin{aligned} rowIdx(i) &= list(rowIdx(i)), \\ colIdx(i) &= list(colIdx(i)). \end{aligned} \tag{3.28}$$

These operations do not conflict with each other, and the permutations are executed in parallel. Note that compressed sparse row (CSR) are also widely used for sparse matrix storage format. The CSR format is constructed by compressing *rowIdx* of the COO format. This approach reduces the memory usage compared to the COO format, and it can be parallelized for matrix–vector multiplication with good performance. However, the permutation of the components is more complex than that of the COO format. Although permutations can be achieved using the permutation matrix $\mathbf{P}$ as $\mathbf{M}' = \mathbf{P}^{\mathrm{T}}\mathbf{M}\mathbf{P}$, this implementation is not

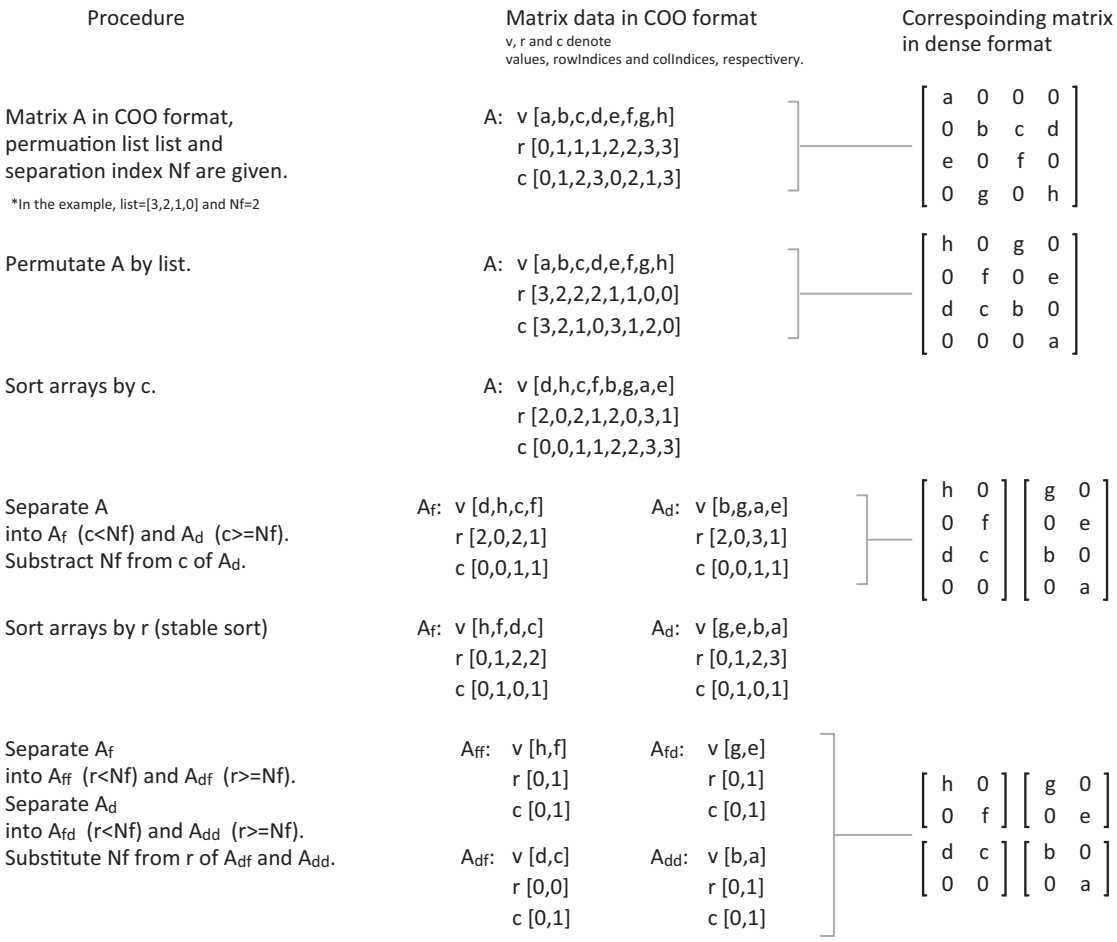



Figure 3.8: Algorithm of the matrix rearrangement with an example. Reproduced from [1].

efficient because additional arithmetic operations are required compared to the above-mentioned permutation in COO format. Therefore, the COO format was adopted as the sparse matrix storage format for our application.

Because COO format requires that the index arrays are sorted in the row-major manner, sorting must be performed when the index arrays are modified. We perform the sorting process in combination with a matrix separation process. Fig. 3.8 shows an overview of the procedure including permutation by taking a matrix $\mathbf{A}$ as an example. After the modification of the index arrays (Eq. (3.28)) is performed, three arrays are sorted by *colIdx*. Next, $\mathbf{A}$ is separated along its columns into $\mathbf{A}_\mathrm{f}$ and $\mathbf{A}_\mathrm{d}$. Let $N_\mathrm{f}$ be the separation index as used in Fig. 3.8. The entries whose column index is less than $N_\mathrm{f}$ are copied to $\mathbf{A}_\mathrm{f}$. The rest of the

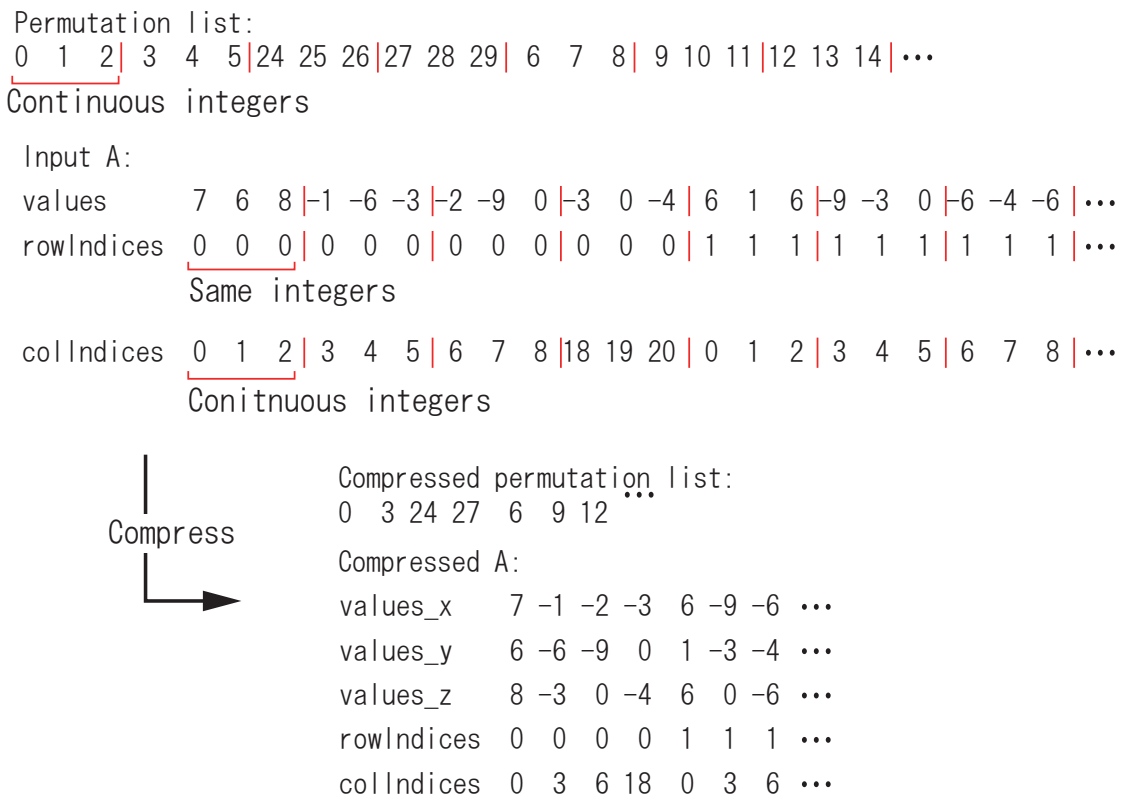



Figure 3.9: Example of input arrays and its compression. Reproduced from [1].

entries are copied to $\mathbf{A}_{\mathrm{d}}$. To fix *colIdx* to zero-base indices, $N_{\mathrm{f}}$ is subtracted from all the components of *colIdx* in $\mathbf{A}_{\mathrm{d}}$. Subsequently, three arrays (*values*, *rowIdx*, and *colIdx*) of $\mathbf{A}_{\mathrm{f}}$ and $\mathbf{A}_{\mathrm{d}}$ are sorted by their *rowIdx*. This sort must be stable sort, in which the original order is maintained when the compared components are same values. This is because, if the sorting is not stable, the ascending order of *colIdx* might be disturbed. After the sort, $\mathbf{A}_{\mathrm{f}}$ and $\mathbf{A}_{\mathrm{d}}$ are separated along their rows into $\mathbf{A}_{\mathrm{ff}}$, $\mathbf{A}_{\mathrm{df}}$, $\mathbf{A}_{\mathrm{fd}}$, and $\mathbf{A}_{\mathrm{dd}}$. Subtraction of *rowIdx* of $\mathbf{A}_{\mathrm{df}}$ and $\mathbf{A}_{\mathrm{dd}}$ is required for the same reason with that for $\mathbf{A}_{\mathrm{d}}$. Finally, the rearrange and separated matrices are generated in the COO format.

As described above, these procedures require sorting of large arrays, which is computationally demanding. To accelerate the sorts, we use the optimized implementation in the NVIDIA CUDA thrust library.

For further acceleration, the permutation list and the arrays (*values*, *rowIdx*, *colIdx*) of $\mathbf{A}$ can be compressed. Fig. 3.9 shows an example of the compression. Because three variables with respect to each node are relocated together, the

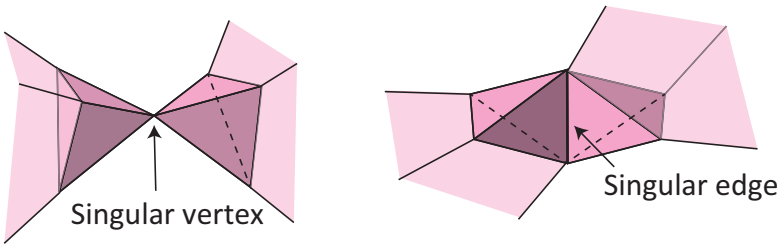



Figure 3.10: Topological singularity. Reproduced from [1].

permutation list is a combination of three consecutive integers. In addition, *rowIdx* and *colIdx* consist of a combination of the same three integers, e.g., (0, 0, 0), and a combination of three consecutive integers, e.g., (0, 1, 2), respectively. Therefore, sorting can be performed according to these three-integer blocks. In order to realize the sort per block, *values* is separated into three arrays, *values_x*, *values_y*, and *values_z*. The permutation list, *rowIdx*, and *colIdx* are compressed by storing only the first element of the consecutive-integer blocks. By this compression the size of the arrays to a third of their original size is reduced and the computational burden of sorting decreases. After $\mathbf{A}_{\mathrm{ff}}$, $\mathbf{A}_{\mathrm{fd}}$, $\mathbf{A}_{\mathrm{df}}$, and $\mathbf{A}_{\mathrm{dd}}$ are calculated, the compressed arrays are extracted in the uncompressed COO format.

## 3.5 Modeling of Dissection

### 3.5.1 Topological-singularity Avoidance

Element removing can lead a FE mesh to be a nonmanifold mesh [43], which means that the mesh has vertices or edges where the thickness of the volumetric mesh cannot be defined. Such vertices and edges are called singular vertices and singular edges, respectively. The singularity because of the singular vertex and singular edges is called a topological singularity. Examples of the topological singularity is illustrated in Fig. 3.10. Because the physics simulation can be unstable when the mesh becomes nonmanifold, we consider to avoid the topological singularities in our surgery simulator.

This section describes a simple and fast algorithm for topological-singularity avoidance that can be used for element-removal fracture representations. The

concept of the algorithm is aggressive element removing. One drawback of this approach is that the volume decreases as elements removals. However, if such drawback is allowed, the approach is fast and suitable for real-time applications. The summarized algorithm flow is described as follows.

1. **Fracture detection**: Detect the tetrahedra according to a fracture criterion and list them in a set $T_{\mathrm{rm}}$.

2. **Singularity verification**: Test whether the vertices and edges involved in $T_{\mathrm{rm}}$ are singular after the removal of the tetrahedra listed in $T_{\mathrm{rm}}$.

3. **Determination of additional tetrahedra to be removed**: If vertices or edges are predicted to be singular, the tetrahedra that include the singular vertices or edges are added to $T_{\mathrm{rm}}$.

4. Iterate **Singularity verification** and **Detection of additional tetrahedrons to be removed** until $T_{\mathrm{rm}}$ becomes empty.

In our simulator, the maximum principal stress is used as the criterion to detect the tetrahedra to be removed (in the fracture detection described above). That is, if the absolute value of the maximum principal stress exceeds a threshold, the element is added in $T_{\mathrm{rm}}$ (the set of tetrahedra to be removed). The removal criterion is written as

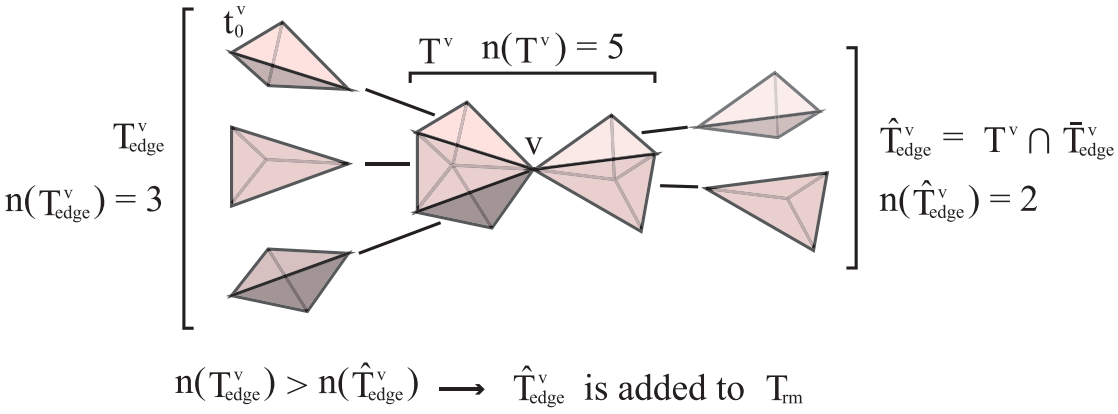
$$\max(|\sigma_1|, |\sigma_2|, |\sigma_3|) > \sigma_{\max}, \tag{3.29}$$

where $\sigma_i$ $(i = 1, 2, 3)$ is the maximum principal stress, which is obtained as the eigenvalue of the element stress tensor $\mathbf{S}^e$, and $\sigma_{\max}$ is the stress threshold. Because we use a constant-strain tetrahedral element, $\mathbf{S}^e$ is constant on an element. In the corotational formulation, $\mathbf{S}^e$ is obtained by applying the element rotation to $\mathbf{S}^e$ as

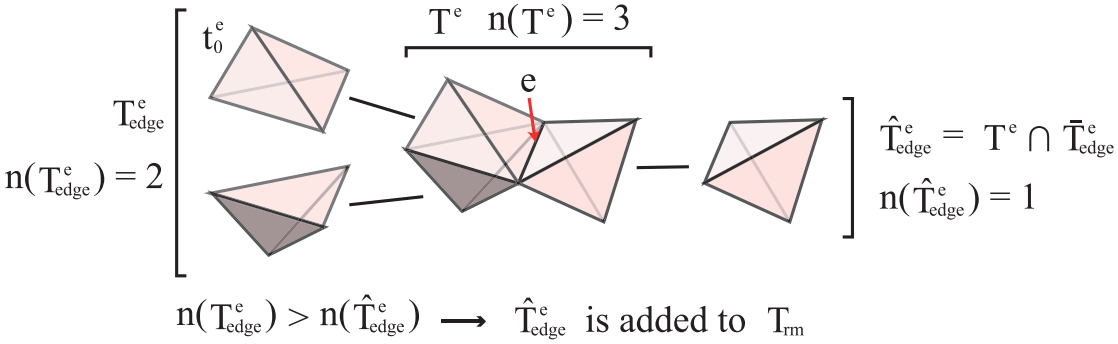
$$\mathbf{S}^e = \mathbf{D}^e\mathbf{B}^e\left(\mathbf{R}^e\boldsymbol{x}^e - \boldsymbol{x}_0^e\right), \tag{3.30}$$

where $\mathbf{D}^e$ is the strain–stress matrix and $\mathbf{B}^e$ is displacement–strain matrix.

In the phase of **singularity detection**, the sets of vertices $V_{\mathrm{rm}}$ and edges $E_{\mathrm{rm}}$ are extracted from the tetrahedra listed in $T_{\mathrm{rm}}$. Each vertex $v \in V_{\mathrm{rm}}$ and edge $e \in E_{\mathrm{rm}}$ is tested whether it is a singular vertex or edge. The singularity detection algorithm of vertex $v$ is described as the following. See also Fig. 3.11(a).




a




b

Figure 3.11: Topological singularity detection. (a) Singular vertex detection. (b) Singular edge detection. Reproduced from [1].

1. Extract $T^v$, a set of tetrahedra that share $v$.
2. Select an arbitrary tetrahedron $t_0^v \in T^v$.
3. Construct $T^v_{\text{edge}}$, a set of tetrahedra, that connects with $t_0^v$ by edges.
4. Select a tetrahedra $t_x^v \in T^v_{\text{edge}}$ and search for an edge-sharing tetrahedron as described in steps 2 and 3. Add the new edge-sharing tetrahedron to $T^v_{\text{edge}}$ and iterate until no entry is detected.
5. If $n(T^v) \neq n(T^v_{\text{edge}})$, $v$ is a singular vertex, where $n(\cdot)$ is the number of tetrahedra.

The algorithm of singular-edge detection goes in a similar manner with that for singular-vertex detection. The algorithm flow is described as the following. See also Fig. 3.11(b).

1. Extract $T^e$, a set of tetrahedra, that share $e$.
2. Select an arbitrary tetrahedron $t_0^e \in T^e$.
3. Construct $T^e_{\text{edge}}$, a set of tetrahedra, that connects with $t_0^e$ with edges **except** edge $e$.
4. Select a tetrahedron $t_x^e \in T^e_{\text{edge}}$ and search for an edge-sharing tetrahedron as described in steps 2 and 3. Add the new edge-sharing tetrahedron to $T^e_{\text{edge}}$ and iterate until no entry is detected.
5. If $n(T^e) \neq n(T^e_{\text{edge}})$, $e$ is a singular edge.

The phase of **detection of additional tetrahedrons to be removed** detects a set of additional tetrahedra to be removed, $T_{\text{add}}$, to avoid a topological singularity. When a singular vertex $v$ is detected, $T^v_{\text{edge}}$ and $\hat{T}^v_{\text{edge}} \left(= T^v \cap \bar{T}^v_{\text{edge}}\right)$ are defined. In order to avoid the volume loss as much as possible, the number of tetrahedra to be removed should be minimized. Hence, the smaller set between $T^v_{\text{edge}}$ and $\hat{T}^v_{\text{edge}}$ is selected as $T_{\text{add}}$ by comparing $n(T^v_{\text{edge}})$ and $n(\hat{T}^v_{\text{edge}})$. Similarly, when a singular edge $e$ is detected, the smaller set between $T^e_{\text{edge}}$ and $\hat{T}^e_{\text{edge}} = T^e \cap \bar{T}^e_{\text{edge}}$ is selected as $T_{\text{add}}$ for the same reason. After $T_{\text{add}}$ is determined, it is added to $T_{\text{rm}}$.

We shows the effect of the singularity avoidance. Fig. 3.12 shows some results of fracture simulations. Without topological-singularity avoidance (Fig. 3.12(b)), tetrahedra connected with a singular vertex or edge exhibit unstable dynamic behavior. On the other hand, with topological-singularity avoidance (Fig. 3.12(a)), stable behavior was obtained and the simulation can be continued in any fracture situation.

### 3.5.2 Implementation

In our fracture simulation the maximum principal stress of each tetrahedral element is required. It is computed in parallel using the GPU. The eigenvalues

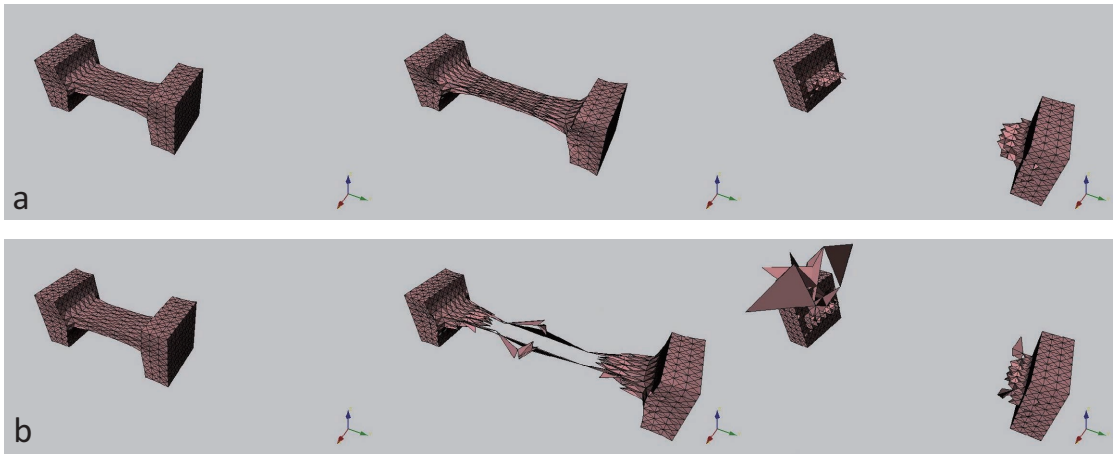



Figure 3.12: Examples of fracture simulations. These sequences show a fracture simulations of a soft tissue model executed (a) with and (b) without topological-singularity avoidance. Reproduced from [10].

of the stress tensor are calculated using the Jacobi eigenvalue algorithm. The topological-singularity avoidance is implemented on a CPU. It is because numerous conditional branchings and complex data structures for representing the mesh connectivity are required. However, it is not a cost-intensive procedure and is rapidly computed on a CPU.

## 3.6 Results and Discussion

### 3.6.1 Performance Evaluation of GPU Implementations

To evaluate our GPU implementation, we compare three implementations: (1) a CPU (no parallelization), (2) a six-core CPU with multithreading parallelization, and (3) a GPU parallelization. Cube-shaped models discretized by various numbers of tetrahedrons were used for the comparison. Simulation environment was constructed using a cube model. The cube model was procedurally generated with different resolutions. In the cube model, the surface vertices of the two opposite sides are constrained. The computational times of the matrix assembly and matrix rearrangement are shown in Fig. 3.13 and Fig. 3.14, respectively.

Fig. 3.13 and Fig. 3.14 show that the GPU parallelization was fastest among

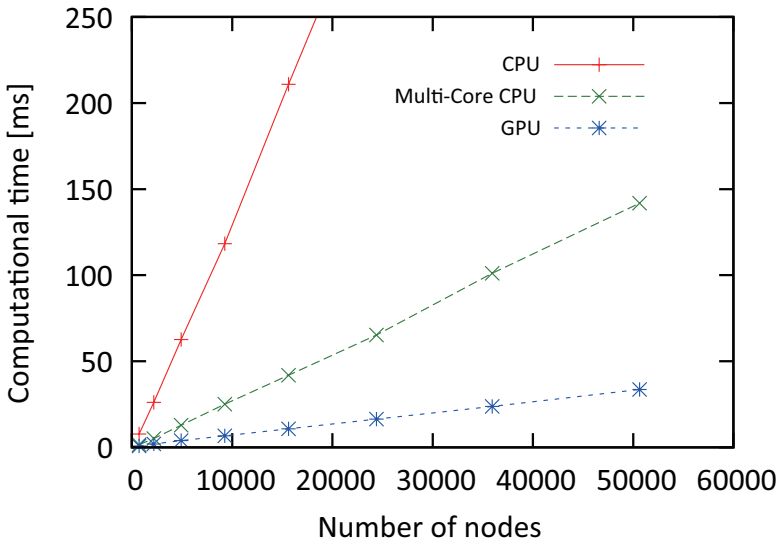



Figure 3.13: Computational time of matrix assembly. Reproduced from [1].

the three implementations in both evaluations. Let us compare the performance for the FE mesh with 15,625 nodes and 69,120 elements. For the matrix assembly, the GPU parallelization (10.7 ms) was 19.7 times faster than the single-CPU implementation (210.9 ms) and 3.9 times faster than the six-core CPU implementation (41.9 ms). For the matrix rearrangement, the GPU parallelization (11.0 ms) was 7.1 times faster than the single-CPU implementation (78.4 ms) and 5.1 times faster than the six-core CPU implementation (56.3 ms]).

### 3.6.2 Blunt Dissection Simulation

In surgery, an operation for separating tissues without cutting is called blunt dissection. For example, blunt dissection is performed to breaking very soft connective tissues that covers cerebral vessels in neurosurgery.

An cube-shaped FE mesh with a groove filled with soft material was used in the simulation. This model is a simple model that imitating the structure of a cerebral fissure filled with connective tissue. The number of nodes was 4,807 and the number of tetrahedrons was 19,600. The Young's modulus and Poisson's ratio of the connective tissue were set to 100 [Pa] and 0.4, respectively. The Young's modulus and Poisson's ratio of the main body were set to 1,000 [Pa] and 0.4, respectively. The threshold stresses (absolute value of the maximum

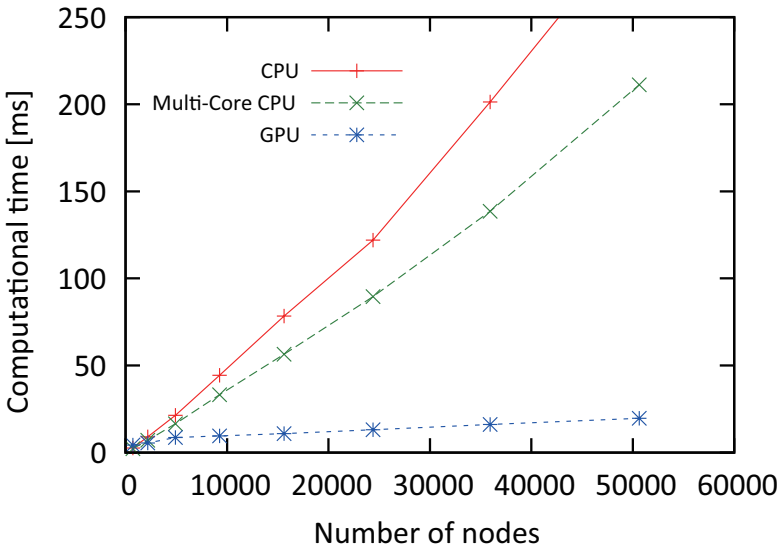



Figure 3.14: Computational time of matrix rearrangement. Reproduced from [1].

principal stress) were set to the same values with their Young's moduli. Note that these material parameters are intended to distinguish the relative stiffness of the materials. They are not reliable parameters for simulating the brain tissue and connective tissue.

In the simulation, two spatulas were inserted into the groove, and they were moved to dissect the connective tissues filled in the groove at a velocity of 5.0 mm/s. The time step was set to 20 ms.

Fig. 3.15(a) show the snapshots of the simulation. Fig. 3.15(b) show the stress visualizations of the same simulation. As seen in Fig. 3.15(b), the connective tissue was deformed larger than the main body because of the difference of the stiffness. The connective tissues were removed owing to the stress concentration because it was specified to be softer than the main body. In the simulation, no oscillation or divergent behavior was observed.

Fig. 3.16 and Fig. 3.17 show the computational time and the removed elements count at each time step. In Fig. 3.16, the jitter of the computational time was observed. One of the cause of the jitter is the difference in the convergence of the conjugate gradient method in each time step. Another cause is the change in the region where the geometrical constraints (boundary condition) were imposed . When the region of the constraints changed, the matrix rearrangement proce-

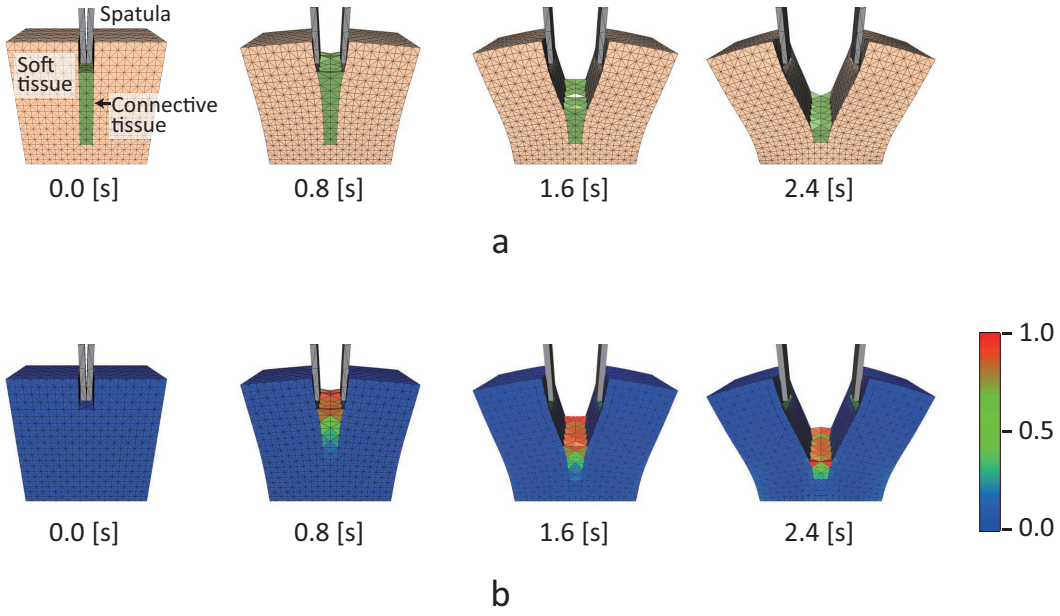



Figure 3.15: Results of the simulation of blunt dissection. (a) Snapshots. (b) Stress visualization (maximum principal stress). Reproduced from [1].

dure is performed and takes additional computations. The average computational times of the three implementations, a CPU with no parallelization, a six-core CPU with multithreaded parallelization, and a GPU implementation, were 103, 41, and 17 ms, respectively. The speed-up of the GPU against the CPU was 6.1. Only the GPU realized sufficient smooth animation with a refresh rate greater than 30 Hz. From Fig. 3.17, the fracture started at 25-th time step, and the peak number of removed elements was 33 at 60-th step. This result shows that the number of removed elements did not affect the computational time. This is preferable for real-time interactive simulation because the simulation can be continued at the constant refresh rate throughout.

### 3.6.3 Brain Retraction Simulation

Brain retraction is a pushing manipulation for creating surgical working space. In most approach in neurosurgery, brain retraction is inevitable and the imposed pressure can injure the brain tissue. Therefore, the haptic sense is important for performing the retraction. We focuses on the retraction of the brain tissues of

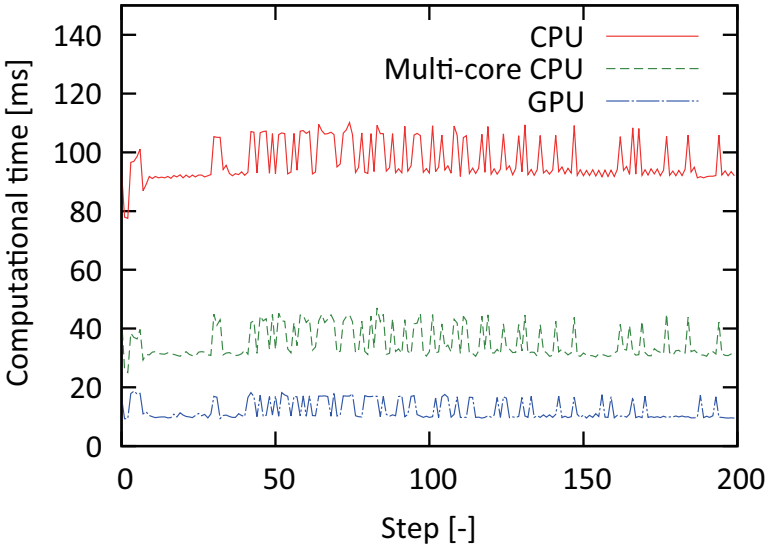



Figure 3.16: Computational time spent for each time step of the blunt dissection simulation. Reproduced from [1].

the Sylvian fissure. The Sylivian fissure is a cerebral fissure that separates lobus parietalis and lobus temporalis. The Sylvian fissure is filled with the arachnoid manbraine, which required to be dissected by surgeons to approach the deep part of the brain. We conducted a brain retraction simulation using a Sensable Phantom Omni haptic device. The calculated reaction forces that acting to the brain spatula model was fed back to the user through the haptic device. In this simulation, we assumed that the arachnoid menbrain had been dissected beforehand. The user was given the task objective to retract the brain tissues and expose the brain tumor existing at the bottom of the Sylvian fissure. We used a brain hemisphere mesh model in this simulation. The number of nodes was 8,647 and the number of the tetrahedral elements was 32,639. This model was constructed from scan data of an anatomical model of the human brain, Brain Model C20 (3B Scientific GmbH) with modifications using 3D modeling software. The nodes located on the bottom of the model were fixed, i.e., the displacements were set to zero.

Fig. 3.18 shows the snapshots of the simulation. By manipulating a haptic device, an operator moves a sphere-shaped pointer displayed in the virtual space. The operator picked up and manipulated the spatulas in the virtual space. Collision detection was performed between spatulas and the brain model, and the reaction

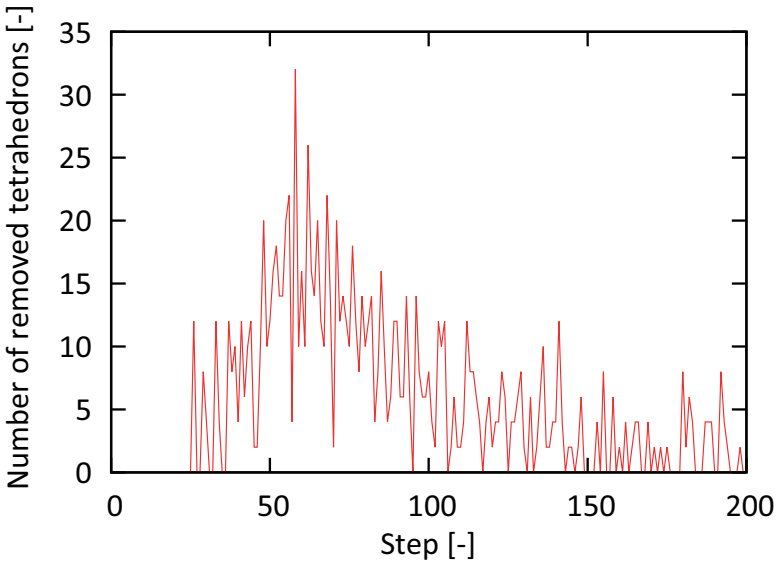



Figure 3.17: Number of removed elements at each time step. Reproduced from [1].

force acting to the spatula was fed back to the user. As seen in Fig. 3.18, the brain tissues were pressed, and the Sylvian fissure was opened using two spatulas. As a result, the tumor was exposed. Fig. 3.19 shows the computational time of each time step. The computational times spent for assembling a matrix, rearranging a matrix, and solving a linear system of equations are plotted. A stress analysis and the fracture processing were not performed in this simulation. Fig. 4.7 shows the history of reaction force to be rendered to the user. We filtered the reference forces by a first-order low-pass filter (cut-off frequency 1.0 Hz) for smooth force feed back.

Although the simulation was performed with good visual plausibility, the calculation speed was not sufficient for stable force feed back. The range of computational time for a simulation loop was 40–80 [ms]. This refresh rate is not sufficient for smooth animations and haptic rendering. Because of the discontinuous force update, the force display could oscillate without the low-pass filter. Although the low-pass filter smoothed the discontinuous force feedback, this is not a fundamental solution to display realistic reaction forces. From these results, further acceleration or stabilizing strategy are needed to achieve stable and visually acceptable simulation.

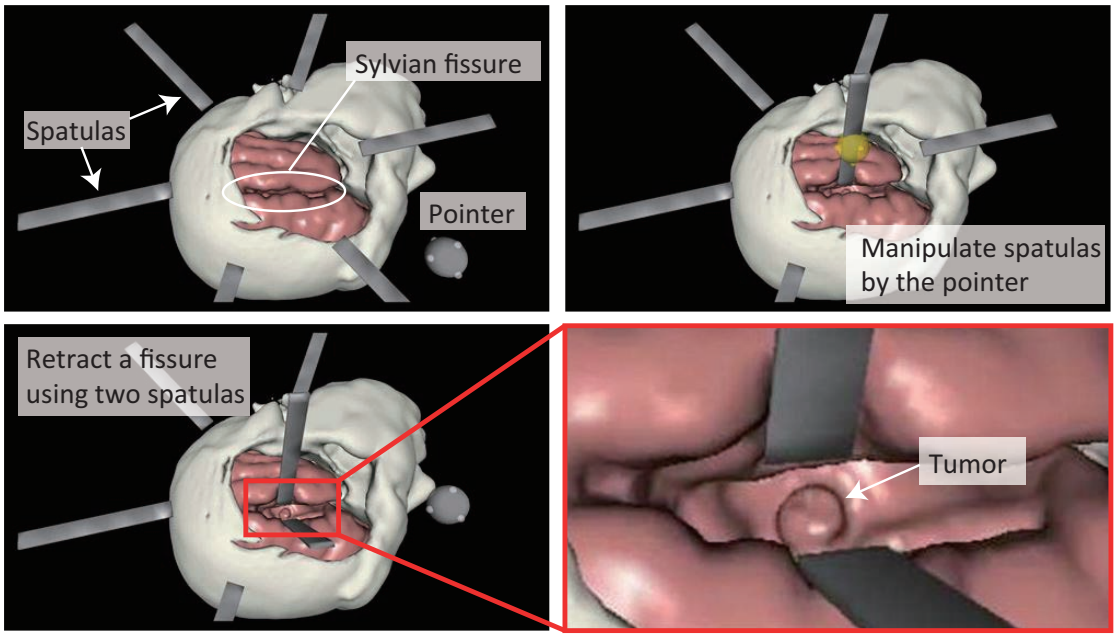



Figure 3.18: Result of the brain retraction simulation. Reproduced from [1].

## 3.7 Summary

In this chapter, a real-time simulation method for soft-tissue deformation and fracture is described. An formulation using corotational FEM and bounday-condition-based contact response was introduced. To accelerate the simulation, a GPU implementation was proposed for matrix assembly, matrix rearrangement, and solving linear equations. To stabilize the fracture simulation, an approach based on topological-singularity avoidance was proposed. To evaluate these methods, blunt dissection simulation and brain retraction simulation were conducted. Both simulations were conducted in (nearly) real time. Although the proposed method worked well with good visual plausibility, stabilization method for haptic rendering need to be developed.

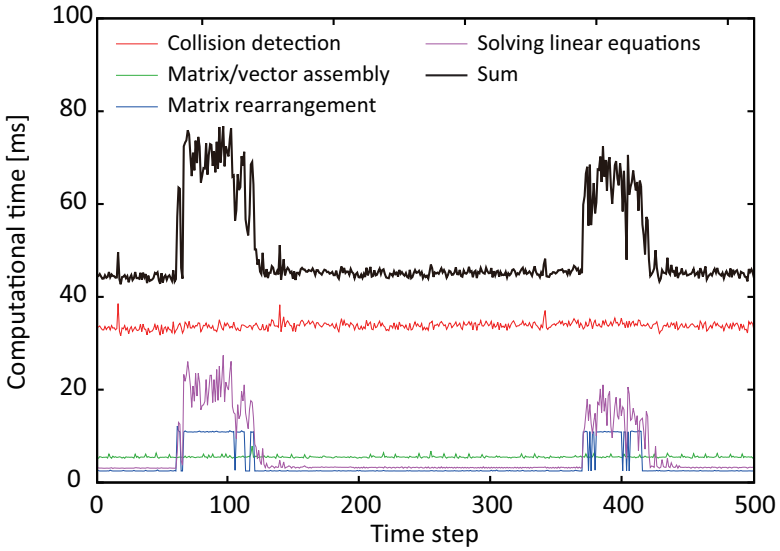



Figure 3.19: Computational time of the brain retraction simulation. Reproduced from [1].

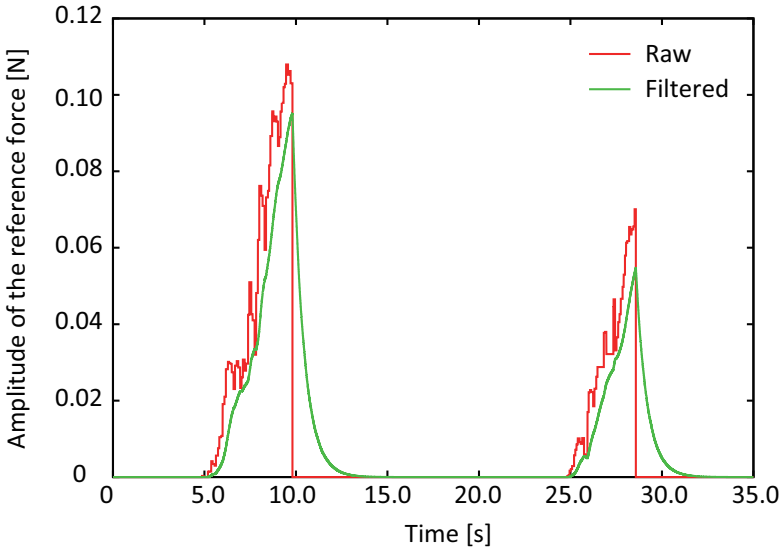



Figure 3.20: Time history of reaction force to be rendered to the user. The raw data is calculated by the FEM solver and the filtered data is the force used for the force feedback. In this simulation, a first-order low-pass filter (cut-off frequency, 1.0 Hz) was applied. Reproduced from [1].

# Chapter 4. Haptic Rendering Based on Virtual Coupling

## 4.1 Introduction

This chapter describes a stabilized haptic rendering method for enabling brain retraction simulation with a large simulation time step. In Chapter 3, haptic rendering was performed by a boundary-condition-based method. Such approaches are called direct rendering. However, a system that involves force feedback to a human hand can be unstable when the sampling time or the displaying stiffness are large. To address this instability issue, we developed a stabilizing method based on virtual coupling (VC). In addition, the contact response of a deformable object is formulated using a penalty method to allow a real-time simulation with high-resolution models.

## 4.2 Related Works

Six-DoF haptic rendering has been an important problem since the haptic interface was developed [63], In order to realize a responsive haptic interaction with virtual environments, many methods have been proposed. However, 6-DOF haptic rendering of the deformable objects is a still difficult problem because it involves high computational burdens.

In a recent study on haptic rendering, a constraint-based approach was adopted. In this approach, the collisions between virtual objects are resolved precisely using using equality-based formulations [64] and inequality-based formulations [65]. Although the approach makes it possible to resolve the collisions precisely, the

computational cost is too high to calculate the dynamics of soft objects in real-time. Thus, the resolution of the deformable object is limited.

The penalty method has also been used in haptic rendering of soft objects [66, 47]. The computational cost of the penalty method is less than that of the constraint-based method and the implementation is simple. A disadvantage of the penalty methods proposed by [66, 47] is the numerical instability.

The past haptic rendering methods for deformable objects based on the penalty method [66, 47] formulated the contact forces using explicit time integration. In general, explicit time integration involves numerical instability. To stabilize the simulation, a very small time step should be used. In the method proposed by Barbič and Doug [66], the physics simulation was executed at an extremely fast rate (1 kHz). They accelerated the simulation using a model reduction method and an adaptive proximity query strategy.

On the other hand, we focused on adoption of implicit time integration for stabilizing the contact response. The implicit time integration is an effective method for enhancement of the numerical stability even with a large time step (e.g., 30 ms) [67].

In this chapter, a handling method of contact between rigid and deformable objects using the implicit time integration is described. In our formulation, the dynamics of rigid models and deformable models are described in a single large system. To apply implicit time integration, the Jacobians of the contact forces were introduced.

Although our method is expected to enhance the numerical stability of the physics simulation, some stability problems can occur because our method applies the linearization to nonlinear terms (e.g., rotation of a rigid body). Although such drawbacks of linearization are concern, this problem has not been discussed well in the literature (e.g., in [68]). Therefore, we performed some numerical experiments of the step response of rigid body rotation under the constraint of a visco-elastic torsional spring (virtual coupling). By this experiment, we attempted to determine stable VC parameters for typical large time steps. Our method is evaluated using a simple model (cube model) and complex environments (bunny model).

## 4.3 Formulation of Contact Problem

### 4.3.1 Dynamics of Tool Object

In our study, the tool object is modeled as a rigid body. Let $\mathbf{y}$ be the state vector of a rigid body. The vector $\mathbf{y}$ consists of the position of the center of mass (CoM) $\mathbf{x}_{\text{com}}$, the orientation $\mathbf{q}$ (in quaternion form), the momentum $\mathbf{P}$, and the angular momentum $\mathbf{L}$. The time derivative of $\mathbf{y}$ is written as

$$\dot{\mathbf{y}}(t) = \begin{bmatrix} \dot{\mathbf{x}}_{\text{com}} \\ \dot{\mathbf{q}} \\ \dot{\mathbf{P}} \\ \dot{\mathbf{L}} \end{bmatrix} = \begin{bmatrix} \mathbf{P}/m_{\text{rb}} \\ \frac{1}{2}\boldsymbol{\omega}_{\text{q}}\mathbf{q} \\ \mathbf{F} \\ \mathbf{T} \end{bmatrix} = \mathbf{g}(\mathbf{y}, t), \tag{4.1}$$

where $m_{\text{rb}}$, $\mathbf{F}$, and $\mathbf{T}$ are the mass, the external force, and the external torque, respectively. The variable $\boldsymbol{\omega}_{\text{q}}$ is the angular velocity formed as a quaternion. The term $\boldsymbol{\omega}_{\text{q}}\mathbf{q}$ represents the multiplication of two quaternions. The angular velocity $\boldsymbol{\omega}$ is derived using angular momentum, as

$$\boldsymbol{\omega} = \boldsymbol{R}\boldsymbol{I}_{\text{body}}^{-1}\boldsymbol{R}^{T}\mathbf{L}, \tag{4.2}$$

where $\boldsymbol{R}$ is the rotation matrix equivalent to $\mathbf{q}$, and $\boldsymbol{I}_{\text{body}}$ is the moment of inertia defined in the body space. Further details of the rigid body dynamics for the numerical simulation can be found in [69].

In this study, we adopted implicit time integration using the Backward Euler method. The increment of the state vector at a time step $\Delta\mathbf{y}(= \mathbf{y}_{n+1} - \mathbf{y}_n)$ is written as $\Delta\mathbf{y} = \Delta t\ \mathbf{g}_{n+1}(\Delta\mathbf{y})$. Note that this is a nonlinear equation because of the rigid body rotation. We linearize $\mathbf{g}_{n+1}$ around $\mathbf{y}_n$ for reduction of the computational cost because a nonlinear solver such as Newton's method is computationally demanding. The linearized equation is written as

$$\mathbf{g}_{n+1} = \mathbf{g}_n + \frac{\partial \mathbf{g}}{\partial \mathbf{y}}\Delta\mathbf{y}, \tag{4.3}$$

where $\partial\mathbf{g}/\partial\mathbf{y}$ is the Jacobian described in the following equation.

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} = \begin{bmatrix} 0 & 0 & \frac{1}{m_{\text{rb}}}\mathbf{I} & 0 \\ 0 & \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{q}} & 0 & \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{L}} \\ \frac{\partial \mathbf{F}}{\partial \mathbf{x}} & \frac{\partial \mathbf{F}}{\partial \mathbf{q}} & \frac{\partial \mathbf{F}}{\partial \mathbf{P}} & \frac{\partial \mathbf{T}}{\partial \mathbf{L}} \\ \frac{\partial \mathbf{T}}{\partial \mathbf{x}} & \frac{\partial \mathbf{T}}{\partial \mathbf{q}} & \frac{\partial \mathbf{T}}{\partial \mathbf{P}} & \frac{\partial \mathbf{T}}{\partial \mathbf{L}} \end{bmatrix} \tag{4.4}$$

See [68] for the complete descriptions of the Jacobian. Using Eq. (4.3), the update equation of the state variable yields a linear simultaneous equation.

### 4.3.2 Dynamics of Deformable Object

A deformable object is modeled by a deformable tetrahedral mesh. The dynamic deformation of a deformable model is represented by the movements of the particles (nodes of the mesh) interacting with each other. The interactions of particles can be formulated by the potential energy of elasticity or the energy dissipation due to damping effects. The forces applied to particles are numerically calculated, e.g., by a spring-mass model or FEM. In this study, we adopted an FEM as described in Chapter 3. Let $\mathbf{x}$ be the position vector and $\boldsymbol{M}\mathbf{v}$ be the momentum vector, where $\boldsymbol{M}$ and $\mathbf{v}$ are the mass matrix and the velocity vector, respectively. These vectors consist of the position/momentum of all of the particles, i.e., $\mathbf{x} = [\mathbf{x}_1^T \ \mathbf{x}_2^T \ \dots \mathbf{x}_n^T]^T$, where $\mathbf{x}_i$ is the position vector of $i$-th particle and $n$ is the number of particles. The state of a deformable object is described by the combination of $\mathbf{x}$ and $\boldsymbol{M}\mathbf{v}$. The dynamics of the deformable object can be expressed as

$$\begin{bmatrix} \dot{\mathbf{x}}(t) \\ \boldsymbol{M}\dot{\mathbf{v}}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{v}(t) \\ \mathbf{f}(\mathbf{x}, \mathbf{v}, t) \end{bmatrix}, \tag{4.5}$$

where $\mathbf{f}$ is the external force vector that is composed of forces applied to particles. Note that, in this study, the internal forces originated from the dynamics of deformable objects are treated as external forces. Typically, the motion of equation of a deformable object is described as

$$\boldsymbol{M}\dot{\mathbf{v}} + \boldsymbol{C}\mathbf{v} + \mathbf{f}_{\text{els}}(\mathbf{x}) = \mathbf{f}_{\text{ex}}, \tag{4.6}$$

where $\boldsymbol{C}$ is the damping matrix, $\mathbf{f}_{\text{els}}$ is the internal force vector (elastic forces), and $\mathbf{f}_{\text{ex}}$ is an external force (e.g., gravity forces). In our formulation, however, the internal forces ($\boldsymbol{C}\mathbf{v}$ and $\mathbf{f}_{\text{els}}$) are considered as external forces acting on the particles, and the equation of motion is described by rearranging the terms as

$$\boldsymbol{M}\dot{\mathbf{v}} = \mathbf{f}_{\text{defo}} + \mathbf{f}_{\text{ex}}, \tag{4.7}$$

where $\mathbf{f}_{\text{defo}} = -\boldsymbol{C}\mathbf{v} - \mathbf{f}_{\text{els}}(\mathbf{x})$ is the force vector defined by the dynamics of a deformable object. Using this representation, we can combine the internal force

with the external forces to the same term in Eq. (4.5).

The increments of the state vector formulated in the Backward Euler method is introduced by the same manner as for rigid body dynamics, as

$$\begin{bmatrix} \Delta\mathbf{x} \\ \boldsymbol{M}\Delta\mathbf{v} \end{bmatrix} = \Delta t \begin{bmatrix} \mathbf{v}_{n+1} \\ \mathbf{f}_{n+1} \end{bmatrix}, \tag{4.8}$$

where $\Delta\mathbf{x} = \mathbf{x}_{n+1} - \mathbf{x}_n$ and $\Delta\mathbf{v} = \mathbf{v}_{n+1} - \mathbf{v}_n$. When we consider the nonlinear constitutive law of the material, Eq. (4.8) becomes a nonlinear equation. To reduce the computational burden, we linearize $\mathbf{f}_{n+1}$ around $\mathbf{x}_n$ and $\mathbf{v}_n$ as

$$\mathbf{f}_{n+1} = \mathbf{f}_n + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Delta\mathbf{x} + \frac{\partial \mathbf{f}}{\partial \mathbf{v}}\Delta\mathbf{v}. \tag{4.9}$$

Substituting Eq. (4.9) into (4.8) yields a system of linear equations.

As described in Chapter 3, a corotational formulation of FEM is adopted [1]. We show an example of the above-mentioned formulation for the corotational FEM. The element stiffness equation is expressed as

$$\mathbf{f}_e = \boldsymbol{R}_e\boldsymbol{K}_0\boldsymbol{R}_e^T\mathbf{x}_e - \boldsymbol{R}_e\boldsymbol{K}_0\mathbf{x}_{e0}, \tag{4.10}$$

where $\mathbf{f}_e$ is the internal force vector by elastic energy, $\boldsymbol{R}_e$ is the rotation of the element of the deformed configuration, $\boldsymbol{K}_0$ is the stiffness matrix of the linear FEM in the initial configuration, $\mathbf{x}_e$ and $\mathbf{x}_{e0}$ are current and rest position vectors of the element vertices, respectively. All $\mathbf{f}_e$ are summed to a global vector as

$$\mathbf{f}_{\text{els}} = \boldsymbol{K}\mathbf{x} + \mathbf{f}_0, \tag{4.11}$$

where $\boldsymbol{K}$ is the tangent stiffness matrix and $\mathbf{f}_0$ is a vector obtained by assembling the term $-\boldsymbol{R}_e\boldsymbol{K}_0\mathbf{x}_{e0}$. For the damping effect, we utilize Rayleigh damping for simplicity of implementation, as

$$\boldsymbol{C} = \alpha\boldsymbol{M} + \beta\boldsymbol{K}. \tag{4.12}$$

Finally, the contribution of the internal forces as external forces are described as

$$\mathbf{f}_{\text{defo}} = -(\alpha\boldsymbol{M} + \beta\boldsymbol{K})\mathbf{v} - \boldsymbol{K}\mathbf{x} - \mathbf{f}_0. \tag{4.13}$$

From this equation, the Jacobians are calculated as

$$\frac{\partial \mathbf{f}_{\text{defo}}}{\partial \mathbf{x}} = -\boldsymbol{K}, \quad \frac{\partial \mathbf{f}_{\text{defo}}}{\partial \mathbf{v}} = -\boldsymbol{\alpha}\boldsymbol{M} - \boldsymbol{\beta}\boldsymbol{K}. \tag{4.14}$$

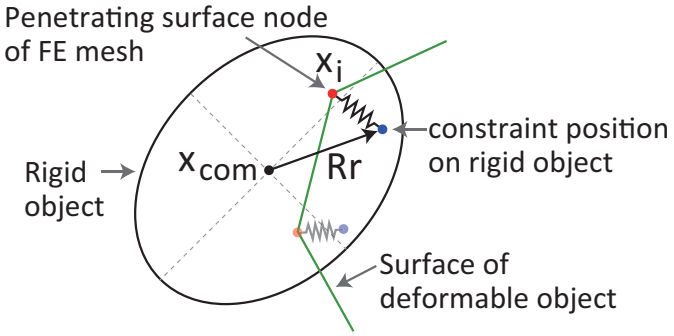



Figure 4.1: Spring connecting the surface node $i$ of an FE mesh and the rigid object. Reproduced from [11].

### 4.3.3 Contact Force

In this study, contact forces are formulated using a penalty method. In the penalty method, a potential energy is defined based on a vertex penetration [70]. Let $d$ be a penetration depth of a vertex into an obstacle and $k$ be a penalty coefficient. The potential energy can be defined as $E = \frac{1}{2}kd^2$. The contact force is derived from the negative gradient of the potential energy as $-\nabla E$. Apparently, this contact force can be considered as springs that are inserted between colliding objects. Therefore, the formulation of the contact forces is explained using spring elements in the following descriptions.

these penalty-based springs are generated according to object intersections. Fig. 4.1 shows the schematic of a spring that connects a surface node $i$ of an FE mesh and a point on the surface of the rigid object. The constraint position $\mathbf{r}$ is the relative position from the CoM defined in the fixed coordinate of the rigid body. The connecting position of the spring on the rigid body in global coordinates is represented as, $\boldsymbol{R}\mathbf{r} + \mathbf{x}_{\text{com}}$. The force acting on the surface node $i$ is defined as

$$\mathbf{f}_{\text{p},i} = k_{\text{p}}\boldsymbol{N}(\mathbf{x}_{\text{com}} + \boldsymbol{R}\mathbf{r} - \mathbf{x}_i) + b_{\text{p}}\boldsymbol{N}(\mathbf{v}_{\text{com}} + \boldsymbol{\omega} \times (\boldsymbol{R}\mathbf{r}) - \mathbf{v}_i), \tag{4.15}$$

where $k_{\text{p}}$ and $b_{\text{p}}$ are penalty parameters corresponding to the stiffness and damping of the spring. $\boldsymbol{N}$ is a matrix representing the anisotropic stiffness, defined as

$$\boldsymbol{N} = a\boldsymbol{I} + (1-a)\mathbf{n}\mathbf{n}^T, \tag{4.16}$$

where $\mathbf{n}$ is the normal vector of the surface, and $a$ is the parameter defining the anisotropic stiffness, e.g., when $a = 1$, the spring has an isotropic stiffness, and

when $a = 0$, the spring has a completely anisotropic stiffness, which allows to slide in the tangent direction [71]. We use this parameter to approximate friction behavior between a rigid object and a deformable object. The force/torque applied to the rigid body is described as

$$\mathbf{F}_{\mathrm{p}} = -\mathbf{f}_{\mathrm{p},i}, \quad \mathbf{T}_{\mathrm{p}} = \boldsymbol{R}\mathbf{r} \times \mathbf{F}_{\mathrm{p}}. \tag{4.17}$$

In the above paragraph, we explained the contribution of a single spring. The total contribution of multiple penalty-based springs are obtained by the sum of $\mathbf{F}_{\mathrm{p}}$ and $\mathbf{T}_{\mathrm{p}}$ in Eq. (4.17). They are accumulated in $\mathbf{F}$ and $\mathbf{T}$ in Eq. (4.1), respectively.

### 4.3.4 Implicit Time Integration

The penalty-based contact forces depend on both the state of a rigid object and of deformable objects as described in Sec. 4.3.3. In order to formulate the contact forces by a manner of implicit time integration, a large system involving both of the rigid and deformable objects are constructed:

$$\begin{bmatrix} \dot{\mathbf{x}}(t) \\ \boldsymbol{M}\dot{\mathbf{v}}(t) \\ \dot{\mathbf{y}}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{v}(t) \\ \mathbf{f}(\mathbf{x}, \mathbf{v}, \mathbf{y}, t) \\ \mathbf{g}(\mathbf{x}, \mathbf{v}, \mathbf{y}, t) \end{bmatrix}. \tag{4.18}$$

Note that here $\mathbf{f}$ and $\mathbf{g}$ become the functions depending on both the state variables of a rigid object ($\mathbf{y}$ in Eq. (4.1)) and those of a deformable object ($\mathbf{x}$ and $\mathbf{v}$ in Eq. (4.5)). The increments of the state variables at a simulation time step are written as

$$\begin{bmatrix} \Delta\mathbf{x} \\ \boldsymbol{M}\Delta\mathbf{v} \\ \Delta\mathbf{y} \end{bmatrix} = \Delta t \begin{bmatrix} \mathbf{v}_{n+1} \\ \mathbf{f}_{n+1} \\ \mathbf{g}_{n+1} \end{bmatrix}. \tag{4.19}$$

In the same manner as in Eq. (4.3) and (4.9), the linearization of the right-hand side of Eq. (4.19) around $\mathbf{x}_n$, $\mathbf{v}_n$, and $\mathbf{y}_n$ yields the following simultaneous linear equations:

$$\left( \boldsymbol{I} - \Delta t \begin{bmatrix} 0 & \boldsymbol{M}^{-1} & 0 \\ \frac{\partial \mathbf{f}}{\partial \mathbf{x}} & \frac{\partial \mathbf{f}}{\partial \mathbf{v}}\boldsymbol{M}^{-1} & \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \\ \frac{\partial \mathbf{g}}{\partial \mathbf{x}} & \frac{\partial \mathbf{g}}{\partial \mathbf{v}}\boldsymbol{M}^{-1} & \frac{\partial \mathbf{g}}{\partial \mathbf{y}} \end{bmatrix} \right) \begin{bmatrix} \Delta\mathbf{x} \\ \boldsymbol{M}\Delta\mathbf{v} \\ \Delta\mathbf{y} \end{bmatrix} = \Delta t \begin{bmatrix} \mathbf{v}_n \\ \mathbf{f}_n \\ \mathbf{g}_n \end{bmatrix}. \tag{4.20}$$

Note that, in this equation, all of the Jacobians of the contact forces are added to the coefficient matrix.

From Eq. (4.15) and (4.17), the Jacobians of the forces are derived. In the following equations, a matrix $\mathbf{u}^*$ given by a vector $\mathbf{u} = [u_x, u_y, u_z]^T$ is the skew-symmetric matrix used as a matrix-vector cross product, i.e., $\mathbf{u}^*\mathbf{a} = \mathbf{u} \times \mathbf{a}$, where $\mathbf{a}$ is an arbitrary vector applied to the cross product. The derived Jacobians are

described using the following equations:

$$\frac{\partial \mathbf{f}_{\mathrm{p},i}}{\partial \mathbf{x}_i} = -k_{\mathrm{p}}\boldsymbol{N}, \tag{4.21}$$

$$\frac{\partial \mathbf{f}_{\mathrm{p},i}}{\partial \mathbf{v}_i} = -b_{\mathrm{p}}\boldsymbol{N}, \tag{4.22}$$

$$\frac{\partial \mathbf{f}_{\mathrm{p},i}}{\partial \mathbf{x}_{\mathrm{com}}} = k_{\mathrm{p}}\boldsymbol{N}, \tag{4.23}$$

$$\frac{\partial \mathbf{f}_{\mathrm{p},i}}{\partial q_j} = k_{\mathrm{p}}\boldsymbol{N}\frac{\partial \boldsymbol{R}}{\partial q_j}\mathbf{r} + b_{\mathrm{p}}\boldsymbol{N}\boldsymbol{\omega}^*\frac{\partial \boldsymbol{R}}{\partial q_j}\mathbf{r} - b_{\mathrm{p}}\boldsymbol{N}(\boldsymbol{R}\mathbf{r})^*\frac{\partial \boldsymbol{\omega}}{\partial q_j}, \tag{4.24}$$

$$\frac{\partial \mathbf{f}_{\mathrm{p},i}}{\partial \mathbf{P}} = \frac{b_{\mathrm{p}}}{m_{\mathrm{rb}}}\boldsymbol{N}, \tag{4.25}$$

$$\frac{\partial \mathbf{f}_{\mathrm{p},i}}{\partial \mathbf{L}} = -b_{\mathrm{p}}\boldsymbol{N}(\boldsymbol{R}\mathbf{r})^*\boldsymbol{I}_{\mathrm{body}}^{-1}, \tag{4.26}$$

$$\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{x}_i} = k_{\mathrm{p}}\boldsymbol{N}, \tag{4.27}$$

$$\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{v}_i} = b_{\mathrm{p}}\boldsymbol{N}, \tag{4.28}$$

$$\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{x}_{\mathrm{com}}} = -k_{\mathrm{p}}\boldsymbol{N}, \tag{4.29}$$

$$\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial q_j} = -\frac{\partial \mathbf{f}_{\mathrm{p},i}}{\partial q_j}, \tag{4.30}$$

$$\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{P}} = -\frac{b_{\mathrm{p}}}{m_{\mathrm{rb}}}\boldsymbol{N}, \tag{4.31}$$

$$\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{L}} = b_{\mathrm{p}}\boldsymbol{N}(\boldsymbol{R}\mathbf{r})^*\boldsymbol{I}_{\mathrm{body}}^{-1}, \tag{4.32}$$

$$\frac{\partial \mathbf{T}_{\mathrm{p}}}{\partial \mathbf{x}_i} = (\boldsymbol{R}\mathbf{r})^*\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{x}_i}, \tag{4.33}$$

$$\frac{\partial \mathbf{T}_{\mathrm{p}}}{\partial \mathbf{v}_i} = (\boldsymbol{R}\mathbf{r})^*\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{v}_i}, \tag{4.34}$$

$$\frac{\partial \mathbf{T}_{\mathrm{p}}}{\partial \mathbf{x}_{\mathrm{com}}} = (\boldsymbol{R}\mathbf{r})^*\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{x}_{\mathrm{com}}}, \tag{4.35}$$

$$\frac{\partial \mathbf{T}_{\mathrm{p}}}{\partial q_j} = (\boldsymbol{R}\mathbf{r})^*\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial q_j} - \mathbf{F}_{\mathrm{p}}\frac{\partial \boldsymbol{R}}{\partial q_j}\mathbf{r}, \tag{4.36}$$

$$\frac{\partial \mathbf{T}_{\mathrm{p}}}{\partial \mathbf{P}} = (\boldsymbol{R}\mathbf{r})^*\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{P}}, \tag{4.37}$$

$$\frac{\partial \mathbf{T}_{\mathrm{p}}}{\partial \mathbf{L}} = (\boldsymbol{R}\mathbf{r})^*\frac{\partial \mathbf{F}_{\mathrm{p}}}{\partial \mathbf{L}}. \tag{4.38}$$

The terms $\frac{\partial \boldsymbol{R}}{\partial q_j}$ and $\frac{\partial \boldsymbol{\omega}}{\partial q_j}$ were derived by Otaduy and Lin [68].

Although Eq. (4.20) represents the entire system that requires solving, this equation includes large sparse submatrices in the coefficient matrix. To handle such sparse matrices with a sparse matrix format (as discussed in Chapter 3),

separation of the equation should be considered. First, we separate the first row of Eq. (4.20). The variable $\Delta\mathbf{x}$ can be represented using $\Delta\mathbf{v}$, as

$$\Delta\mathbf{x} = \Delta t(\mathbf{v}_n + \Delta\mathbf{v}). \tag{4.39}$$

Substituting Eq. (4.39) into the second and third rows of Eq. (4.20) yields a smaller system of linear equations, as

$$\begin{bmatrix} \boldsymbol{A}_{\mathrm{vv}} & \boldsymbol{A}_{\mathrm{vy}} \\ \boldsymbol{A}_{\mathrm{yv}} & \boldsymbol{A}_{\mathrm{yy}} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{v} \\ \Delta\mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{\mathrm{v}} \\ \mathbf{b}_{\mathrm{y}} \end{bmatrix}, \tag{4.40}$$

where

$$\boldsymbol{A}_{\mathrm{vv}} = \boldsymbol{M} - \Delta t\frac{\partial\mathbf{f}}{\partial\mathbf{v}} - \Delta t^2\frac{\partial\mathbf{f}}{\partial\mathbf{x}}, \tag{4.41}$$

$$\boldsymbol{A}_{\mathrm{vy}} = -\Delta t\frac{\partial\mathbf{f}}{\partial\mathbf{y}}, \tag{4.42}$$

$$\boldsymbol{A}_{\mathrm{yv}} = -\Delta t\frac{\partial\mathbf{g}}{\partial\mathbf{v}} - \Delta t^2\frac{\partial\mathbf{g}}{\partial\mathbf{x}}, \tag{4.43}$$

$$\boldsymbol{A}_{\mathrm{yy}} = \boldsymbol{I} - \Delta t\frac{\partial\mathbf{g}}{\partial\mathbf{y}}, \tag{4.44}$$

$$\mathbf{b}_{\mathrm{v}} = \Delta t\left(\mathbf{f}_n + \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\Delta t\mathbf{v}_n\right), \tag{4.45}$$

$$\mathbf{b}_{\mathrm{y}} = \Delta t\left(\mathbf{g}_n + \frac{\partial\mathbf{g}}{\partial\mathbf{x}}\Delta t\mathbf{v}_n\right). \tag{4.46}$$

Eq. (4.40) still involves a small dense submatrix ($\boldsymbol{A}_{\mathrm{yy}}$). The matrix $\boldsymbol{A}_{\mathrm{yy}}$ is separated from the large system by calculating the Schur complement as a similar to [72]. From the second row of Eq. (4.40), $\Delta\mathbf{y}$ can be represented by the following equation:

$$\Delta\mathbf{y} = \boldsymbol{A}_{\mathrm{yy}}^{-1}(\mathbf{b}_{\mathrm{y}} - \boldsymbol{A}_{\mathrm{yv}}\Delta\mathbf{v}). \tag{4.47}$$

By substituting Eq. (4.47) into the first row of the Eq. (4.40), we obtain the following equation:

$$(\boldsymbol{A}_{\mathrm{vv}} - \boldsymbol{A}_{\mathrm{vy}}\boldsymbol{A}_{\mathrm{yy}}^{-1}\boldsymbol{A}_{\mathrm{yv}})\Delta\mathbf{v} = \mathbf{b}_{\mathrm{v}} - \boldsymbol{A}_{\mathrm{vy}}\boldsymbol{A}_{\mathrm{yy}}^{-1}\mathbf{b}_{\mathrm{y}}. \tag{4.48}$$

Here, $\boldsymbol{A}_{\mathrm{vv}} - \boldsymbol{A}_{\mathrm{vy}}\boldsymbol{A}_{\mathrm{yy}}^{-1}\boldsymbol{A}_{\mathrm{yv}}$ is the Schur complement of $\boldsymbol{A}_{\mathrm{yy}}$. We calculate the increments as follows: (1) Calculate $\boldsymbol{A}_{\mathrm{yy}}^{-1}$ using a direct solver designed for a dense matrix (we use an LU decomposition-based implementation in Eigen C++ library[1]),

[1] `http://eigen.tuxfamily.org/index.php`

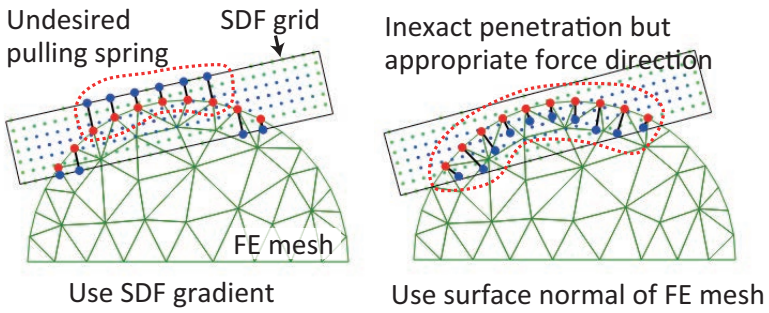



Figure 4.2: Penalty-based springs generated according to penetration of a rigid object into a deformable object using an SDF. The red circles are the penetrating surface nodes of the FE mesh. The blue circles are the contact positions on the rigid object. Reproduced from [11].

(2) solve Eq. (4.48) for $\Delta\mathbf{v}$ by an iterative solver designed for a sparse matrix (we used our own implementation of conjugate gradient (CG) method), (3) calculate $\Delta\mathbf{y}$ and $\Delta\mathbf{x}$ using Eq. (4.47) and (4.39), respectively. Finally (4) calculate $\mathbf{x}_{n+1}$, $\mathbf{v}_{n+1}$, and $\mathbf{y}_{n+1}$ by adding the calculated increments to their values from the previous time step.

## 4.4 Collision Detection

In order to detect the penetration of a rigid object into a deformable object, we compute the penetration depth against the rigid object at each surface node of the FE mesh (Fig. 4.2). The measurements of the penetration depth are executed using signed distance field (SDF) generated on the fixed coordinate of the rigid object as in the study by Barbič and Doug [66]. An SDF $\phi$ is a function that transforms a 3D spatial coordinate to a scalar value representing the distance to the nearest surface of the object. If $\phi(\mathbf{x}) = 0$, the point is on the surface of the object. If $\phi(\mathbf{x}) > 0$ or $\phi(\mathbf{x}) < 0$, the point is outside or inside of the object, respectively. In general, SDF values are stored on the points of a uniform grid for the numerical analysis. The SDF value between the points are trilinearly interpolated. The SDF is calculated before the simulation (preprocessing). A depth query using the SDF takes a constant time. This property is effective for real-time simulations.

The relative vector to the nearest surface from a point is obtained by the gra-

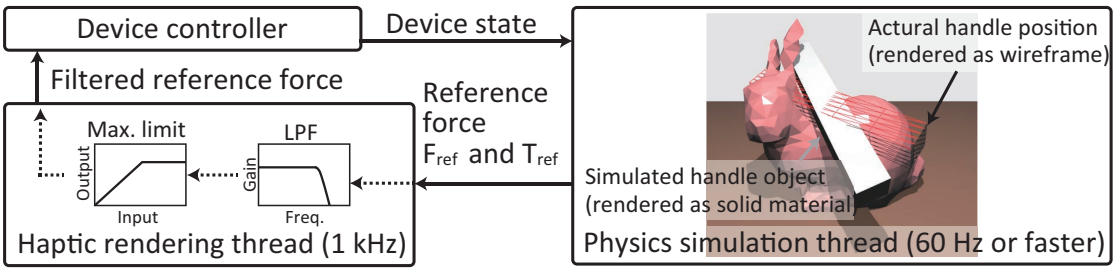



Figure 4.3: Haptic rendering pipeline of our method. Reproduced from [11].

dient of the SDF. Using this property, the penetration normal can be determined. However, as pointed out in [66], the normal vector calculated using the gradient of the SDF can point to the counterpart of contact faces when the rigid object has a thin structure (left part of Fig. 4.1). To address this problem, we adopted the approach proposed in [66]. In their method, the normal vector of a surface node of the FE mesh is used for the penetration vector instead of the gradient of the SDF. The calculated penetration depth is not exact, but it always directs to the outside of the deformable object. This property enhances the robustness of interactive simulations.

Although we explained the method using an SDF, the collision detection method can be replaced with other methods, e.g., the continuous collision detection method [64], another collision detection method using implicit surface representation by meta-ball [73], and so on.

In addition, the SDF can be generated in fixed coordinates on the deformable model. This approach is effective for a nonconforming deformable model or simulation with a thin tool (such as a spatula). In Chapter 5, such approaches are described for resolving the nonconformity between the surface of the graphics model and the FE mesh.

## 4.5 Haptic Rendering

### 4.5.1 Virtual Coupling

Fig. 4.3 shows the haptic rendering pipeline. As mentioned in the previous section, virtual coupling (VC) [63] is adopted for 6-DoF haptic rendering. In the

VC, the stylus of the haptic device and the simulated tool object are connected by a 6-DoF visco-elastic spring. Let $\mathbf{F}_\mathrm{c}$ and $\mathbf{T}_\mathrm{c}$ be the force and torque applied to the virtual tool object by the VC. The vectors $\mathbf{F}_\mathrm{c}$ and $\mathbf{T}_\mathrm{c}$ are calculated as:

$$\mathbf{F}_\mathrm{c} = k_\mathrm{c}(\mathbf{x}_\mathrm{h} - \mathbf{x}_\mathrm{com}) - b_\mathrm{c}(\mathbf{v}_\mathrm{h} - \mathbf{v}_\mathrm{com}), \tag{4.49}$$

$$\mathbf{T}_\mathrm{c} = k_\theta \mathbf{u}_\mathrm{c} + b_\theta(\boldsymbol{\omega}_\mathrm{h} - \boldsymbol{\omega}), \tag{4.50}$$

where $k_\mathrm{c}$, $b_\mathrm{c}$, $k_\theta$, and $b_\theta$ are the stiffness, damping, torsional stiffness, and torsional damping of the visco-elastic spring of VC, respectively. The vectors $\mathbf{x}_\mathrm{h}$, $\mathbf{v}_\mathrm{h}$, and $\omega_\mathrm{h}$ are the position, velocity, and angular velocity of the stylus of the haptic device, respectively. The vector $\mathbf{u}_\mathrm{c}$ is the rotation axis of the torsional stiffness [68]. The magnitude of $\mathbf{u}_\mathrm{c}$ is the angle between the stylus and the virtual tool object. The vector $\mathbf{F}_\mathrm{c}$ and $\mathbf{T}_\mathrm{c}$ contribute the term $\mathbf{g}$ in Eq. (4.1).

The rotation axis $\mathbf{u}_\mathrm{c}$ is represented by the orientations of the tool object and haptic device. Let $\mathbf{q}$ and $\mathbf{q}_\mathrm{h}$ be the orientation of the tool and the haptic device, respectively. The rotation from the frame of the tool to that of the haptic device is

$$\Delta\mathbf{q} = \mathbf{q}_\mathrm{h}\mathbf{q}^{-1}. \tag{4.51}$$

Following Otaduy *et al.* [68], we describe the vector part and scalar part of $\Delta\mathbf{q}$ as $\Delta\mathbf{q}_\mathrm{xyz}$ and $\Delta q_\mathrm{s}$;

$$\Delta\mathbf{q} = (\Delta\mathbf{q}_\mathrm{xyz},\ \Delta q_\mathrm{s}) \tag{4.52}$$

From the definition of the quaternion, $\Delta\mathbf{q}$ can also be expressed as

$$\Delta\mathbf{q} = \left(\sin\left(\frac{|\mathbf{u}_\mathrm{c}|}{2}\right)\frac{\mathbf{u}_\mathrm{c}}{|\mathbf{u}_\mathrm{c}|},\ \cos\left(\frac{|\mathbf{u}_\mathrm{c}|}{2}\right)\right). \tag{4.53}$$

From Eq. (4.52) and (4.53), the rotation angle are formed using $\Delta q_\mathrm{s}$ as

$$|\mathbf{u}_\mathrm{c}| = 2\cos^{-1}(\Delta q_\mathrm{s}), \tag{4.54}$$

$$\mathbf{u}_\mathrm{c} = \frac{|\mathbf{u}_\mathrm{c}|}{\sin\left(|\mathbf{u}_\mathrm{c}|/2\right)}\Delta\mathbf{q}_\mathrm{xyz}. \tag{4.55}$$

In Eq. (4.55), it seems that Otaduy *et al.* missed the term $\frac{1}{\sin(|\mathbf{u}_\mathrm{c}|/2)}$. In our formulation, this term is considered, and we believe that this consideration would strengthen the robustness and responsiveness of the method in [68]. In our implementation, we the use `std :: acos` function in C++11. The range of the argument

is [-1, 1], and the output range is $[0 : \pi]$. Considering the continuity around 0, the output range is converted from $[0 : \pi]$ to $[-\pi/2 : \pi/2]$. Therefore, in our mathematical formulation, we use a symbol $\theta$ ($[-\pi/2 : \pi/2]$) instead of $|\mathbf{u}_\mathrm{c}|$ ($[0 : \pi]$). Thus, we express the $\mathbf{u}_\mathrm{c}$ as

$$\theta = 2\cos^{-1}(\Delta q_\mathrm{s}), \tag{4.56}$$

$$\mathbf{u}_\mathrm{c} = \frac{\theta}{\sin(\theta/2)}\Delta\mathbf{q}_\mathrm{xyz}. \tag{4.57}$$

In Eq. (4.1), the Jacobians of $\mathbf{F}_\mathrm{c}$ and $\mathbf{T}_\mathrm{c}$ are required. The detailed description of the Jacobians are found in [68]. However, because we corrected the formulation of $\mathbf{u}_\mathrm{c}$ as shown Eq. (4.55), we also introduced the corrected Jacobian of $\mathbf{u}_\mathrm{c}$ ($\partial\mathbf{u}_\mathrm{c}/\partial\mathbf{q}$). Following [68], Eq. (4.51) is expressed with matrix-vector multiplication as

$$\Delta\mathbf{q} = \boldsymbol{C}\mathbf{q}, \tag{4.58}$$

where

$$\boldsymbol{C} = \begin{bmatrix} q_{hs} & -q_{hx} & -q_{hy} & -q_{hz} \\ q_{hx} & q_{hs} & -q_{hz} & q_{hy} \\ q_{hy} & q_{hz} & q_{hs} & -q_{hx} \\ q_{hz} & -q_{hy} & q_{hx} & q_{hs} \end{bmatrix}. \tag{4.59}$$

We divide the $\boldsymbol{C}$ into $\boldsymbol{C}_1 \in R^{1\times4}$ and $\boldsymbol{C}_{234} \in R^{3\times4}$ as

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C}_{234} \end{bmatrix}. \tag{4.60}$$

Using these matrices, $\Delta q_s$ and $\Delta\mathbf{q}_\mathrm{xyz}$ are expressed as $\Delta q_s = \boldsymbol{C}_1\mathbf{q}$ and $\Delta\mathbf{q}_\mathrm{xyz} = \mathbf{C}_{234}\mathbf{q}$, respectively. Thus, Eq. (4.57) and (4.57) can be described as follows:

$$\theta = 2\cos^{-1}(\boldsymbol{C}_1\mathbf{q}), \tag{4.61}$$

$$\mathbf{u}_\mathrm{c} = \frac{\theta}{\sin(\theta/2)}\boldsymbol{C}_{234}\mathbf{q}. \tag{4.62}$$

From these equations, the Jacobian is obtained as

$$\begin{aligned} \frac{\partial\mathbf{u}_\mathrm{c}}{\partial\mathbf{q}} &= \frac{1}{\sin(\theta/2)}\boldsymbol{C}_{234}\mathbf{q}\,\frac{\partial\theta}{\partial\mathbf{q}} - \frac{\theta}{2\sin^2(\theta/2)}\boldsymbol{C}_{234}\mathbf{q}\,\frac{\partial\theta}{\partial\mathbf{q}} + \frac{\theta}{\sin(\theta/2)}\boldsymbol{C}_{234} \\ &= \left(\frac{1}{\sin(\theta/2)} - \frac{\theta}{2\sin^2(\theta/2)}\right)\boldsymbol{C}_{234}\mathbf{q}\,\frac{\partial\theta}{\partial\mathbf{q}} + \frac{\theta}{\sin(\theta/2)}\boldsymbol{C}_{234}, \end{aligned} \tag{4.63}$$

where

$$\frac{\partial \theta}{\partial \mathbf{q}} = -\frac{2}{\sqrt{1-(\boldsymbol{C}_1\mathbf{q})^2}}\boldsymbol{C}_1. \tag{4.64}$$

The force and torque to be displayed to the user ($\mathbf{F}_{\text{ref}}$, $\mathbf{T}_{\text{ref}}$) are calculated as $\mathbf{F}_{\text{ref}} = -\mathbf{F}_c$, and $\mathbf{T}_{\text{ref}} = -\mathbf{T}_c$. However, in most case, the physics simulation cannot be refreshed at a sufficient rate (1 kHz for stiff material). If the simulation is performed at a slow rate, the displayed force becomes a stepping signal in the haptic rendering thread executed at 1 kHz. In this study, we decided to smooth the force by a first-order low-pass filter (LPF). However, an LPF enlarges the delay of the response and this is only a symptomatic treatment. To address this problem, a contact model that can be used in multi-rate haptic rendering should be developed as methods proposed in [64, 65].

### 4.5.2 Implementation

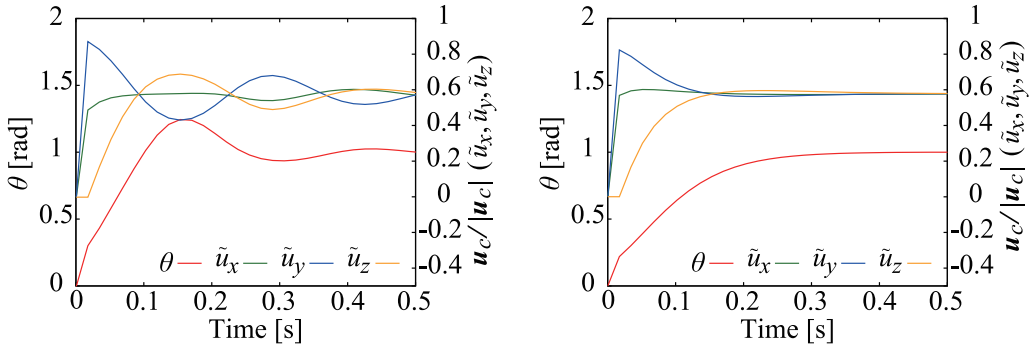
We used the Sensable Phantom Omni haptic device (input: 6-DoF position/orientation, output: 3-DoF force). The CHAI3D C++ library designed for haptic applications[2] was utilized for the communication between the haptic device and the PC with a 1-kHz refresh rate. The physics simulation was run with a constant refresh rate using a `clock_nanosleep` system call of the standard LINUX OS (without a real-time patch). The workstation mounted Intel Core i7-4790K CPU (4 cores, overclocked to 4.5 GHz), 16 GB of RAM, and an NVIDIA GTX TITAN GPU. The sparse matrix operations are parallelized on multi-cores using OpenMP. The GPU was used only for graphics rendering in the evaluations in this chapter. The interface of the haptic device was IEEE 1394.

## 4.6 Results and Discussion

### 4.6.1 Effects of Virtual Coupling Parameters

Our method updates both the state of a rigid object and of a deformable object at the same time and it enables two-way interaction between the objects with good

[2] `http://www.chai3d.org/`




$\Delta t = 0.017$, $k_\theta = 0.3$, $b_\theta = 0.01$ (a)

$\Delta t = 0.017$, $k_\theta = 0.3$, $b_\theta = 0.03$ (b)

Figure 4.4: Observed step responses of the rigid body rotation with VC. Reproduced from [11].

Table 4.1: Determined parameters by the rotational step response experiment with $k_\theta = 0.3$ N m/rad. Reproduced from [11].

| $\Delta t$ [s] | 0.017 | 0.033 | 0.10 |
|---|---|---|---|
| $b_\theta$ [N m s/rad] | 0.03 | 0.04 | 0.11 |
| $T_c$ [s] | 0.35 | 0.50 | 0.11 |

stability. However, when we take a large time step, the linearization error can be generated especially from the rigid body rotation.

To determine appropriate torsional parameters of the VC, we conducted numerical experiments of step response of torsional stiffness using different time steps, viz., 0.1 s (10 FPS), 0.033 s (30 FPS), and 0.017 s (60 FPS). The mass of the tool object is set to an as small as possible value to minimize the undesirable inertial effect: mass $m_{\mathrm{rb}} = 0.1$ kg and momentum of inertia $\boldsymbol{I}_{\mathrm{body}} = 5.0 \times 10^{-4}\boldsymbol{I}_3$, where $\boldsymbol{I}_3$ is the $3 \times 3$ identity matrix. For the step response experiment, the step input (torsion) was set to $\mathbf{u}_c = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$. This means that 1 rad is imposed around the axis $(1, 1, 1)$. The torsional stiffness $k_\theta$ was set to 0.3 N m/rad in this evaluation. Note that $k_\theta$ should be adjusted with consideration of the specification of haptic devices, and thus, this condition is an example for our specific setup.

Fig. 4.4 shows typical results of the step response experiments. In Fig. 4.4(a), the response shows a damping oscillation behavior. In Fig. 4.4(b), suppressed oscillation and overshoot are observed. Because the oscillation and overshoot in the

Table 4.2: Parameter sets of the stiffness and damping of the penalty-based springs. Reproduced from [11].

| Parameter set ID | A | B | C |
|---|---|---|---|
| $k_{\mathrm{p}}$ [N/m] | 50 | 50 | 1000 |
| $b_{\mathrm{p}}$ [N s/m] | 0.1 | 0.1 | 0.1 |
| Implicit integration | Yes | No | Yes |

transition phase decrease the quality of haptic rendering, the viscous parameter $b_\theta$ was adjusted to suppress the oscillation and overshoot. In Table 4.1, the parameters determined for typical time steps $\Delta t$ are summarized. In the table, $T_{\mathrm{c}}$ denotes the convergence time on which the error of $\theta$ reaches less than 0.001 rad. We needed to increase $b_\theta$ for larger $\Delta t$ to obtain responses without oscillations and overshoots. Therefore, when $\Delta t$ becomes larger, $T_{\mathrm{c}}$ also needs to be larger. In our experience, when $T_{\mathrm{c}}$ exceeds 0.5 s, the response is too slow to interact with virtual environments.

In the following experiments, given the responsiveness and the practicality at the computational speed of the current implementation, we decided to adopt the following parameters: $\Delta t = 0.017$ s, $k_\theta = 0.3$ N m/rad, $b_\theta = 0.03$ N m s/rad. In addition, the translational stiffness of the VC spring is determined from the specification of the Sensable Phantom Omni as $k_{\mathrm{c}} = 1000$ N/m. We determined the damping parameter for reducing the translation vibration as $b_{\mathrm{c}} = 10$ N s/m.

### 4.6.2 Evaluation Using a Simple Cube Model

An evaluation was conducted using a simple cube model was conducted (Fig. 4.6). To compare the different parameter settings, a trajectory of the haptic device was recorded and the trajectory was used repeatedly. Fig .4.5 shows the recorded trajectory of the haptic device. A tool object was modeled by a sphere-shaped rigid body. The radius of the sphere was 0.03 m. The properties of the rigid body was set to the parameters determined in Sec. 4.6.1. The deformable object was modeled by a cube-shaped tetrahedral mesh. The mesh had 1331 nodes and 5000 tetrahedral elements. The lengths of the cube edges were 0.1 m. The material

properties were set to the values of typical soft materials (such as biological soft tissues) as the follows: density 1000 kg/m$^3$, Young's modulus 2000 Pa, Poisson's ratio 0.45, and Rayleigh's damping parameters $\alpha = 0$ and $\beta = 0.1$. The nodes of the bottom of the mesh was constrained by springs that connect to each initial nodal position. The stiffness and damping parameters of the springs were set to 50.0 N/m and 0.1 N s/m. The time step $\Delta t$ was 0.017 s (60 FPS) as mentioned in Sec. 4.6.1. The cutoff frequency of the LPF for smoothing the displaying force was set to 10 Hz.

Table 4.2 shows three parameter sets of the stiffness and damping of each spring for the calculation of the penalty-based contact forces between the rigid object and the deformable objects. In addition, in order to compare our method with an explicit time integration approach (i.e. [47]), we performed simulations with and without implicit time integration of the contact force. All of the Jacobians related to contact forces, described in Eq. (4.21)–(4.38), are set to zero, in the simulation in which the implicit time integration is disabled.

Fig. 4.7 shows the effect of the LPF applied to the force output of the simulation using the parameter set A. Although the delay caused from LPF was observed, the LPF worked to smooth the stepping force output.

Figs. 4.8, 4.9, and 4.10 are the comparisons of the results of the three parameter sets. Fig. 4.8 shows the trajectories of the virtual tool object and the stylus of the haptic device. Fig. 4.9 shows the histories of the reaction forces to be displayed to the user. Fig. 4.10 shows the histories of the maximum penetration depth between the tool object and the deformable object. The maximum penetration depth was obtained by measuring the penetration depth of contacting surface nodes of the FE mesh.

From Fig. 4.8, the positions of the virtual tool object were kept at a distance from the actual stylus position. This is because of the visco-erastic spring of the VC and should be allowed to ensure stability.

From Fig. 4.8, the vibration is seen in the trajectory of the parameter set B. This is also seen in the force history (Fig. 4.9) and the maximum penetration-depth history (Fig. 4.10). Comparing the result of A with that of B, it is observed that implicit time integration obtained stable behavior, but explicit time integration did not. To suppress the vibrating behavior in the method employing explicit time

integration, it would be necessary to decrease $k_{\mathrm{p}}$, which leads to larger penetrations and less responsive haptic feed backs.

Comparing the result of A with that of C in Fig. 4.10, it could be seen that a large $k_{\mathrm{p}}$ reduces the maximum penetration depth. However, from the force history of C (Fig. 4.9), a sticking force is observed at the moving-back motion (6 s). This is because our method assumes that the penalty-based springs continue to connect the objects during a time step (from the beginning of a time step to its end). Thus, we cannot asign too large a value to $k_{\mathrm{p}}$. The sticking force can be canceled by observing the contact normals. However, such an approach might influence the dynamic behavior of a multi-contact and the visual sticking behavior cannot be resolved.

From these investigations, the parameters of penalty methods need to be adjusted for each specific simulation environment depending on whether the the penetration and sticking behavior are permitted.

### 4.6.3 Example of a Complex Environment Simulation

Fig. 4.11 shows the snapshots of the simulation using a complex environment. The deformable model was constructed from The Stanford 3D Scanning Repository[3]. It consists of 1229 nodes and 4584 tetrahedral elements. The tool object is modeled as a cube-shaped rigid body. The user manipulated the tool object through the haptic device. The torus and the floor are static rigid objects. In the simulation, the user is able to push the deformable object to pass through the hole of the torus. The simulation was conducted with good stability and no vibration was observed.

## 4.7 Summary

In this paper, we described a method for handle the contact between a rigid object and a deformable object. The contact response is formulated using the penalty method and implicit time integration. Collision detection using an SDF is

[3] `http://graphics.stanford.edu/data/3Dscanrep/`

combined with our penalty method. An efficient numerical computation method considering the sparse matrix was described. The parameters of the VC were determined by step response experiments. Our method was evaluated using a simple cube model and a complex bunny model. Although our method proved to have a drawback in terms of the difficulty in parameter determination, our method was able to handle complicated contacts using an appropriate parameter.

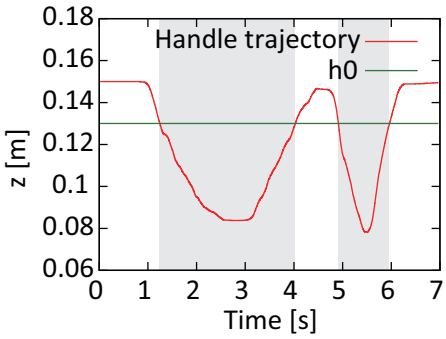



Figure 4.5: Recorded trajectory of the stylus of the haptic device. Reproduced from [11].

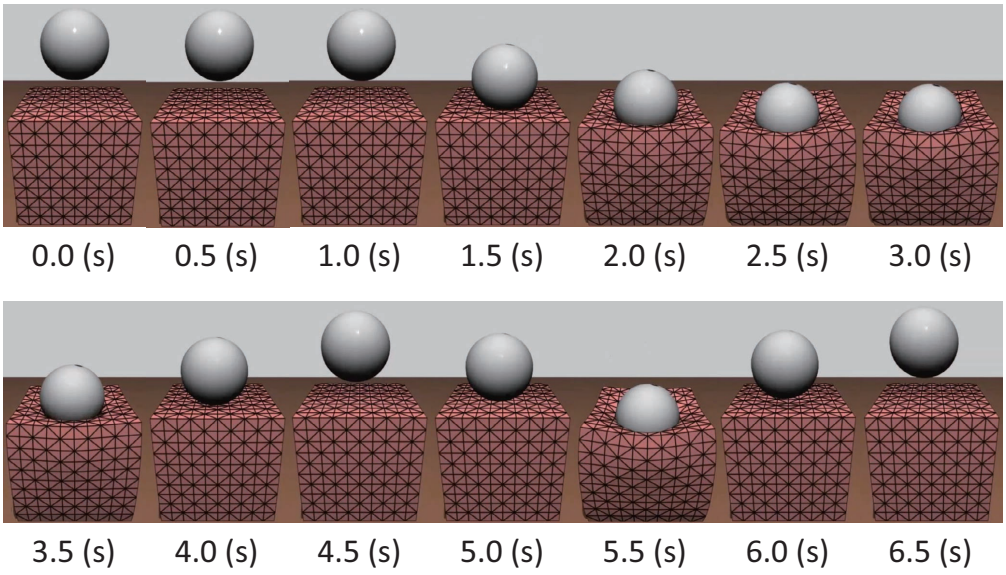



Figure 4.6: Snapshots of the evaluation using the simple cube model. Reproduced from [11].

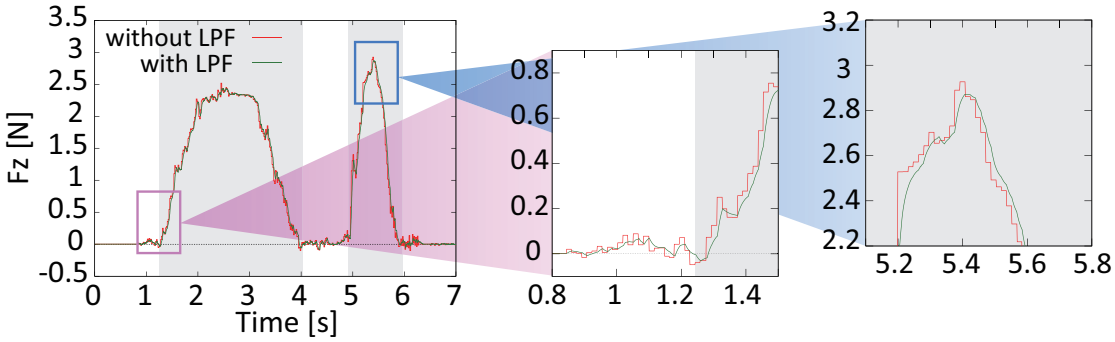



Figure 4.7: Force history with/without the LPF. Reproduced from [11].

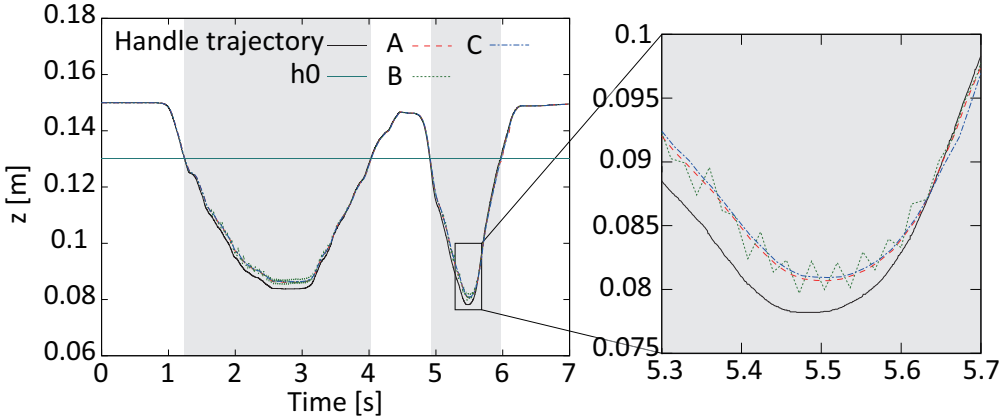



Figure 4.8: Trajectories of the virtual tool object in the evaluation using the simple cube model. Reproduced from [11].

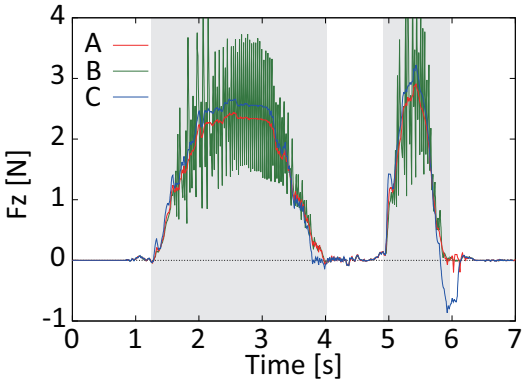



Figure 4.9: Force histories in the evaluation using the simple cube model. Reproduced from [11].

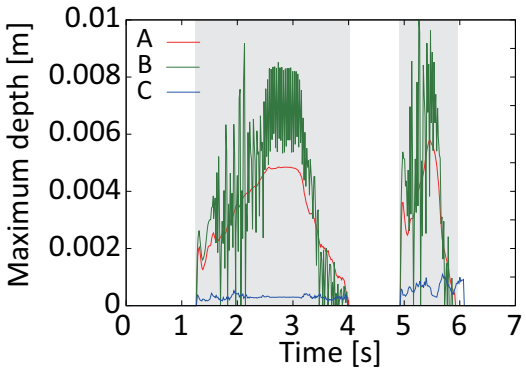



Figure 4.10: Histories of the maximum penetration depth in the evaluation using the simple cube model. Reproduced from [11].

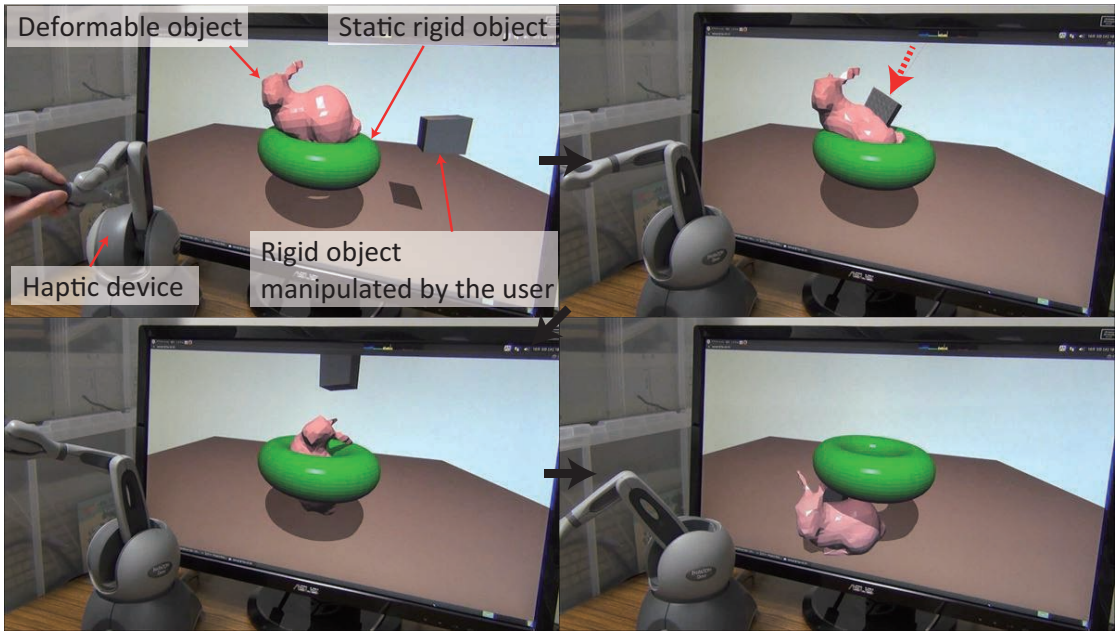



Figure 4.11: Snapshots of the simulation of a complex environment. Reproduced from [11].

# Chapter 5. Haptic Rendering for Embedded Volume

## 5.1 Introduction

The method of the patient-specific model generation was described in Chapter 2. The method adopted embedding approach in which a volume data is embedded in a coarse simple FE mesh. One of the drawbacks of the method is the nonconformity of the boundary between the surface of the FE mesh and that of the volume data. In the haptic surgery simulator of brain retraction, the contact handling is an essential problem to enable the force feed back. Therefore, collision detection and contact response computation need to be designed considering the nonconformity. This chapter describes the contact handling method for embedded volume (patient-specific model obtained by the method of Chapter 2).

A specific issue in our volume embedding approach is the existence of superimposed elements. We considered the issue and developed a method to detect correct collisions. Additionally, the collision detection method was designed not to increase the manual tasks. Our patient-specific model does not requires surface mesh. Off course the surface mesh can be generated from the volume data. But appropriate surface mesh also affects the computational cost, and thus, it might involve trials and errors. Therefore, we develop a method that does not requires explicit surface mesh representation. A collision detection method using an signed distance field (SDF) [74] is introduced in this chapter.

## 5.2 Related Works

Physics-based volume embedding methods combined with volume rendering techniques have been proposed in this two decades. Masutani *et al.* developed a

volume deformation method using a coarse regular tetrahedral grid [75]. FEM was utilized for the calculation of the deformation. They enabled haptic feedback using a haptic device. However, the haptic interaction is limited to the displacement impositions on nodes of the grid. Nakao *et al.* developed a volume manipulation method that embeds a medical volume data in an unstructured tetrahedral mesh [76]. Although they enabled various surgical manipulation such as grasping and cutting, their method required the tetrahedral meshes generation that conform to the volume domain, which is difficult to be automated for highly irregular organs, e.g., for brain . Torres *et al.* proposed a volume manipulation method to embed high-resolution heterogeneous volume data in a coarse regular grid [77]. They utilized a homogenization technique to construct stiffness matrices considering the spatial distribution of voxels inside each hexahedral element. In their report, some results of contact simulations were shown. They adopted contact handling method [78] that relies on the polygonal surface representation in their simulation, which involves surface mesh generation. To our best knowledge, previous studies on physics-based volume manipulation methods required explicit surface representation, i.e., polygonal surface, for resolving the contact handling. The polygonal surface can be generated using the marching cube method [79] or, e.g., Delaunay-triangulation-based method [4]. However, their methods tend to generate a significantly large number of surface polygons. And they increase the computational burden of contact handling procedures. One can apply a simplification techniques for the surface mesh to obtain a reasonable mesh. However, such operations increase the error between the obtained polygonal surface and the surface of volume data. Therefore, human decisions with the knowledge of computational burdens are needed to generate an appropriate surface mesh, which increases the tasks involved in patient-specific model construction.

In the computer graphics community, some studies have proposed contact handling methods that do not depend on the vertices of the mesh but on a precomputed SDF. Wu *et al.* introduced a collision detection method for a real-teim cutting simulation using an SDF calculated on the material coordinate of a deformable model [80]. Similarly, there are some studies that adopted the SDF on a deformable model for self-collisions for nearly real-time character skinning animation [71] and offline contact simulation for a deformable model represented using a

level set method [81]. These methods are based on the approximated SDF stored on the material coordinate first introduced in [82]. The effectiveness of the method was validated in contact simulation with FEM [83]. In this study, we adopt this approximated SDF for collision handling in our volume embedding strategy. The use of the SDF enables to preserve the full geometry described in volume data preserved without surface mesh generation. This chapter describes a automated SDF generation method as well as an efficient penetration-depth measuring method using the SDF.

## 5.3 Contact Handling

### 5.3.1 Overview

In this study, a tool object is modeled as a rigid body and the shape is represented by a polygonal surface. The collision detection between the tool object and a deformable object is computed by searching the vertices of the tool object that penetrate into the deformable object. This collision detection is performed at the beginning of the simulation time step. A projected point on the surface of the deformable object is determined for each penetrating vertex. The projected points are used for definition of the contact forces.

### 5.3.2 SDF Generation

Before the execution of a real-time simulation, the SDF is generated in advance with respect to the material coordinate (Fig. 5.1). The material coordinate is a coordinate defined on the rest shape of a deformable model. As explained in Chapter 4, SDF $\Phi(\mathbf{x})$ is a scalar field constructed in 3-D space and the abusolute value $|\Phi(\mathbf{x})|$ represents the minimum distance to the object surface from the point $\mathbf{x}$. The sign of $\Phi$ is positive if the position is outside the volume and negative otherwise. In this study, the SDF values are stored on the points aligned at regular intervals. The SDF value is interpolated by those on the discrete points using trilinear interpolation. From the length of the intervals, the resolution of the surface is determined. In our method, the values are stored on the positions

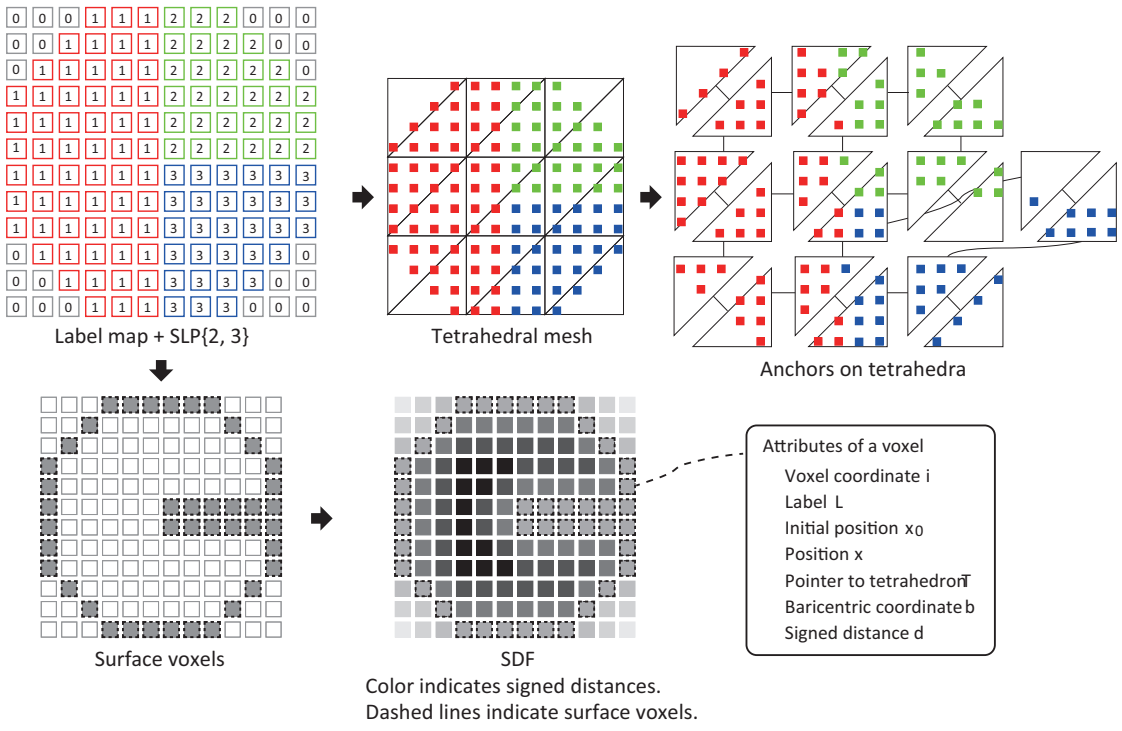



Figure 5.1: Overview of the SDF generation with 2-dimensional illustration.

of voxels. In other words, the SDF value is treated as an attribute of a voxel.

In our volume embedding, the deformable object does not have surface meshes. To define the surface, the surface points are first extracted from the label map (see Fig. 5.1). Let $V$ be the set of all voxels of the label masp. A voxel $v \in V$ is on the surface of the volume if $v$ satisfies the following condition:

- at least one neighbor voxel has an empty label, or
- at least one neighbor voxel is registered as an SLP.

We denote the set of the voxels that satisfy these conditions by $V_s$. The positions of the voxels $v \in V_s$ are considered as the representative points of the surface of volume, and thus, we call $V_s$ the surface voxels.

The SDF is generated using $V_s$ (see Fig. 5.1). At each voxel $v \in V$, the nearest voxel $v_{\text{nearest}}$ is searched from $V_s$. The absolute SDF value at the position of $v$ is determined to be the distance between $v$ and $v_{\text{nearest}}$. The sign of the SDF value at the sample point is positive if $v$ has an empty label (outside the volume), otherwise it is negative (inside the volume). Because this generation is a computationally

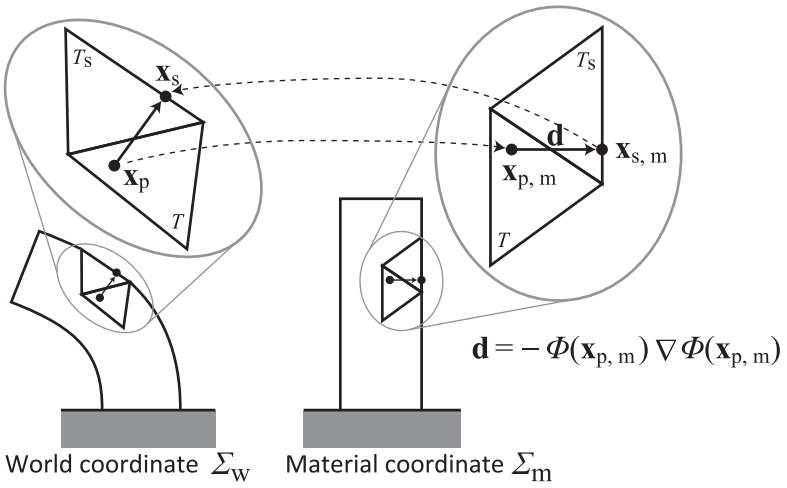



Figure 5.2: Projection of vertex using SDF in 2D. $\mathbf{x}_p$ and $\mathbf{x}_s$ are a vertex of a tool object and the projected point to the surface of a deformable object. The penetration depth is determined using an SDF calculated in the material coordinate.

expensive procedure, the SDF generation is performed once as the precomputation. The SDF values on the voxels are not changed through real-time simulation.

### 5.3.3 Determination of Projection Points

For collision detection and contact point estimation, we adopt the deformed distance field proposed in [82]. Let $\mathbf{x}_p$ be the position of a rigid body vertex penetrating into the deformable object. A typical algorithm of penetration-depth

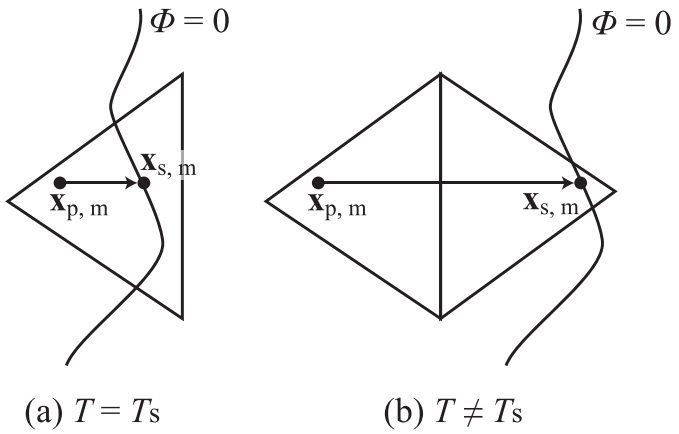



Figure 5.3: SDF sampling for implicit surface.

measurement using the deformed distance field is as the following (Fig. 5.2):

- In the world coordinate $\Sigma_{\mathrm{w}}$, the tetrahedron $T$ that includes $\mathbf{x}_{\mathrm{p}}$ is determined (*intersection test A*).
- The barycentric coordinate $\mathbf{b}$ of $\mathbf{x}_{\mathrm{p}}$ inside $T$ is computed.
- By using $\mathbf{b}$, $\mathbf{x}_{\mathrm{p}}$ is transformed into the position $\mathbf{x}_{\mathrm{p,m}}$ in the material coordinate $\Sigma_{\mathrm{m}}$.
- The nearest surface position from the sampling point is computed as $\mathbf{x}_{\mathrm{s,m}} = \mathbf{x}_{\mathrm{p,m}} - \Phi(\mathbf{x}_{\mathrm{p,m}})\nabla\Phi(\mathbf{x}_{\mathrm{p,m}})$.
- In $\Sigma_{\mathrm{m}}$, the tetrahedron $T_{\mathrm{s}}$ that includes $\mathbf{x}_{\mathrm{s,m}}$ is determined (*intersection test B*).
- The barycentric coordinate $\mathbf{b}_{\mathrm{s}}$ of $\mathbf{x}_{\mathrm{s,m}}$ inside $T_{\mathrm{s}}$ is calculated.
- Using $\mathbf{b}_{\mathrm{s}}$, $\mathbf{x}_{\mathrm{s,m}}$ is transformed to the position $\mathbf{x}_{\mathrm{s}}$ in $\Sigma_{\mathrm{w}}$.

The difficulty in the application of the above-mentioned algorithm to our volume embedding arises from the existence of the superimposed tetrahedral elements. Even in the material coordinate, some tetrahedral elements overlap. Hence, the *intersection tests A* and *B* return multiple tetrahedra for a single sample point. Therefore, a determination method of the correct tetrahedron is required.

The determination of the correct tetrahedron for *intersection test A* can be resolved by using the anchored position of the voxels. Let $V_T$ be the set of voxels inside a tetrahedron $T$. The correct tetrahedron can be detected as follows:

- Multiple tetrahedra $T_1$, $T_2$, ..., $T_n$ are detected.
- For each tetrahedron $T_i$, the nearest voxel from the sample point is detected in $V_{T_i}$.
- The correct tetrahedron is determined to be the tetrahedron that has the nearest voxel among the detected tetrahedra.

The tetrahedron determination for *intersection test B* is more difficult than for *intersection test A* because the SDF cannot always construct the complete surface

boundary using interpolation of discrete sample points. Thus, we simply consider that the tetrahedron detected by *intersection test B* is the same with that chosen in *intersection test A*; i.e., $T_{\mathrm{s}} = T$. Fig. 5.3 shows that this assumption is not correct for all cases ($T_{\mathrm{s}}$ is not always identical to $T$). However, the assumption helps simple and fast implementation. We confirmed that the contacts can be well handled with this assumption.

We discuss on the computational cost of the collision handling below. If the brute-force algorithm is adopted, the collision detection requires $O(N_{\mathrm{tet}} \times N_{\mathrm{vert}})$, where $N_{\mathrm{tet}}$ is the number of tetrahedral elements of the FE mesh, and $N_{\mathrm{vert}}$ is the number of vertices of the rigid body's surface mesh. To reduce the computational cost, an approach based on spatial partitioning is adopted [84]. A uniform grid is generated inside the axis-aligned boundary box of the mesh of the deformable model before the collision detection. The intersection test with cells of the grid is performed for each tetrahedron. When an intersection is detected, the reference to the tetrahedron is stored in the array that is associated with the intersected cell. Collision detection is divided into two phases, the broad phase and the narrow phase. In the broad phase, the vertices of the rigid body's are tested for intersection with the axis-aligned cells. When an intersection between a vertex and a cell is detected, the references to the tetrahedra stored in the list is extracted. In the narrow phase, the intersection test between the tetrahedra and the vertex is performed. This approach improves the effectiveness of the collision detection.

### 5.3.4 Contact Response

The contact forces are formulated based on the penalty method described in Chapter 4. The contact forces acting to the deformable model at position $\mathbf{x}_s$ are expressed as the following equation;

$$\mathbf{f}_s = k_p \boldsymbol{N}(\mathbf{x}_p - \mathbf{x}_s) + b_p \boldsymbol{N}(\mathbf{v}_p - \mathbf{v}_s), \tag{5.1}$$

where $\mathbf{v}_p$ and $\mathbf{v}_s$ are velocities of the points $\mathbf{x}_p$ and $\mathbf{x}_s$, respectively. The matrix $\boldsymbol{N}$ is a matrix that defines the anisotropy of the contact stiffness (See Chapter 4). The contact force is distributed by using the barycentric coordinate $\mathbf{b}$. The distributed

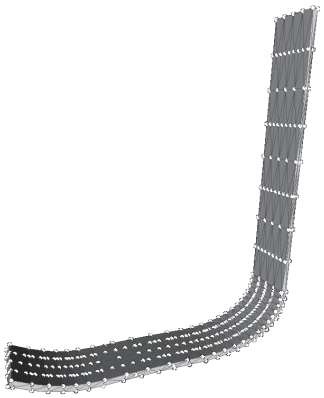


Figure 5.4: Retractor model. The white spheres are the vertices of the surface polygons, which used to detect the penetration with deformable models. The number of vertices and faces are 712 and 1352, respectively.

contact force at the $i$-th vertex of the tetrahedral element is calculated as

$$\mathbf{f}_i = b_i \mathbf{f}_s. \tag{5.2}$$

On the other hand, the force and torque applied to the tool object are written using the Newton's second law as

$$\mathbf{f}_{rb} = \mathbf{f}_s, \quad \mathbf{T}_{rb} = \mathbf{r} \times \mathbf{f}_{rb}, \tag{5.3}$$

where $\mathbf{r}$ is a vector from the CoM of the rigid body to the contact point $\mathbf{x}_p$, which is defined w.r.t the rigid body coordinate. The following relationship is satisfied with the position of the CoM $\mathbf{x}_{\text{com}}$ and the rotation matrix $\boldsymbol{R}$ as follows:

$$\mathbf{x}_p = \boldsymbol{R}\mathbf{r} + \mathbf{x}_{\text{com}}. \tag{5.4}$$

The contact forces are computed for all penetrating vertices of the tool object and incorporated in the dynamic system of the tool object and the deformable object described in Chapter 4.

### 5.3.5 Experimental Conditions

We utilized a haptic device Phantom Omni haptic device with the IEEE 1394 interface provided by Sensable Inc.. The physics simulation and control of the

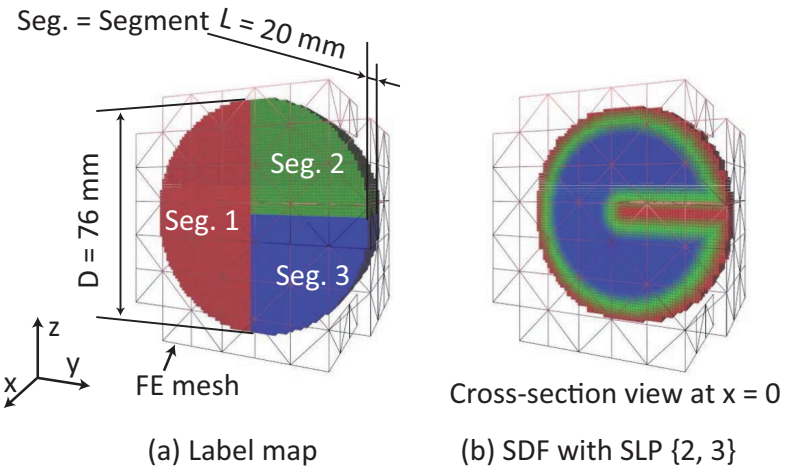



Figure 5.5: Cylinder model. Spacing is 1 mm. $D$ and $L$ are the diameter and the length, respectively. In (a), the color of the voxels indicate the labels. In (b) the color of the voxels indicate the signed distance. The value is close to 0 (near the surface) if the color is red.

haptic device were processed on a workstation with an six-cores CPU (Intel Core i7-3960 overclocked to 4.5 GHz), 64 GB of RAM, and NVIDIA Quadro K5000 GPU. In the current implementation, the GPU was used only for graphics rendering. The OS of the computer was a standard Linux for a desktop configuration without real-time patches (Ubuntu 14.04 64 bit). The algorithms was parallelized on the multi-core CPU using OpenMP. The time step of the physics simulation was set to 33 ms (30 FPS) and synchronized using the clock_nanosleep system call.

In Fig. 5.4, the geometrical model of a tool (retractor) to be manipulated by a user though the haptic device is shown. The polygonal mesh consists of 712 vertices. The mass and moment of inertia were 0.1 kg and 0.0005 $I_3$, where $I_3$ is a $3 \times 3$ identity matrix. The VC parameters $k_c$, $k_\theta$, $b_c$, and $b_\theta$ were 1000.0 N/m, 0.3 Nm/rad, 10.0 Ns/m, and 0.03 Nms/rad, respectively.

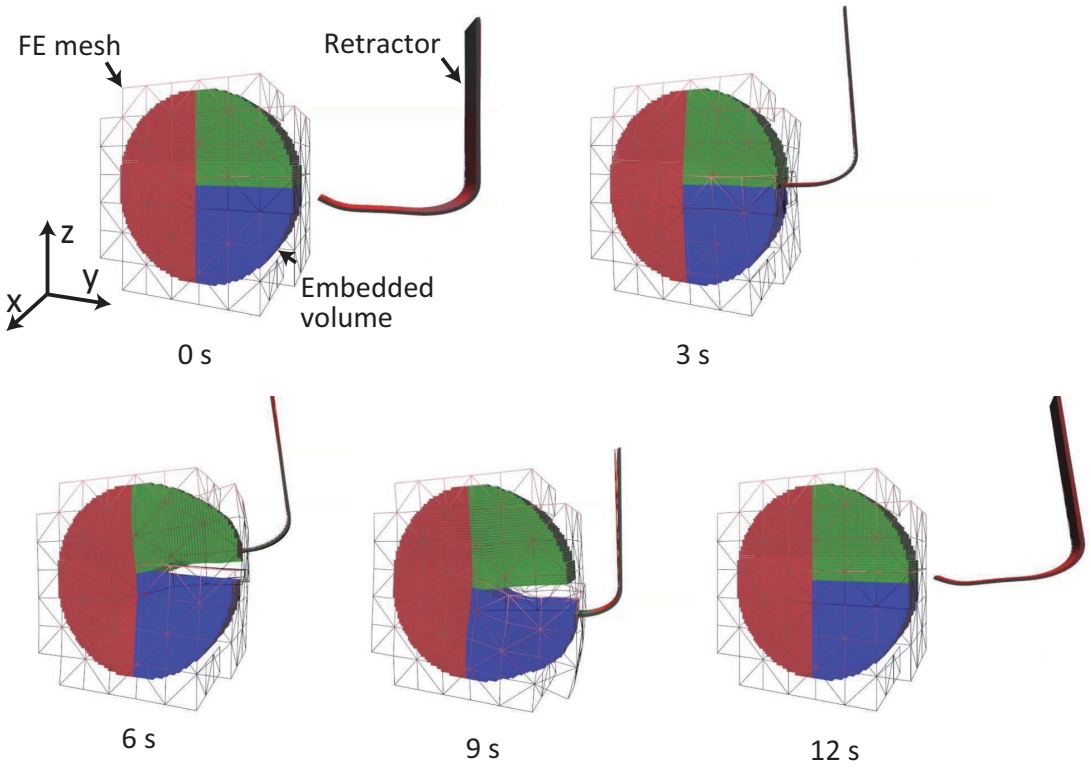



Figure 5.6: Snapshots of the evaluation using cylinder model.

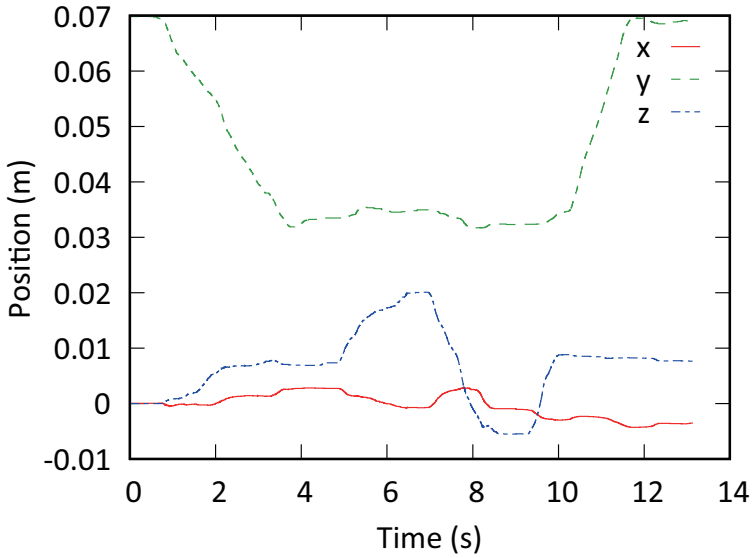



Figure 5.7: Tool trajectory of the evaluation using cylinder model.

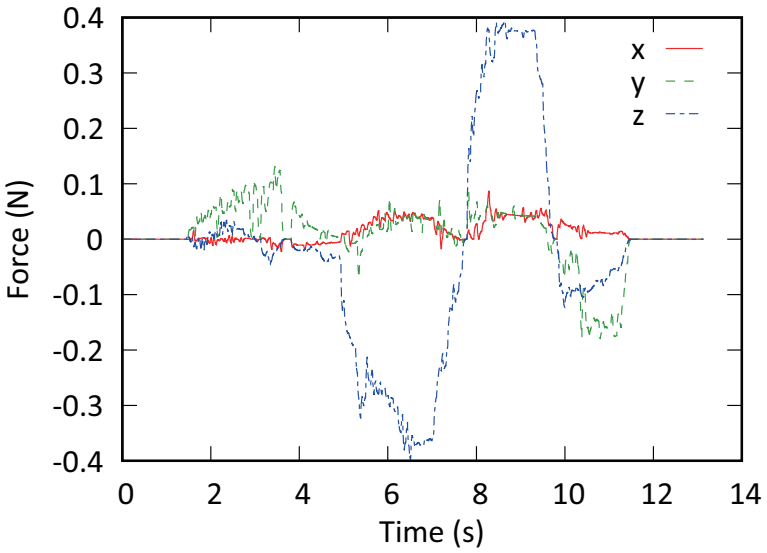



Figure 5.8: Force history of the evaluation using cylinder model.

## 5.4 Results and Discussion

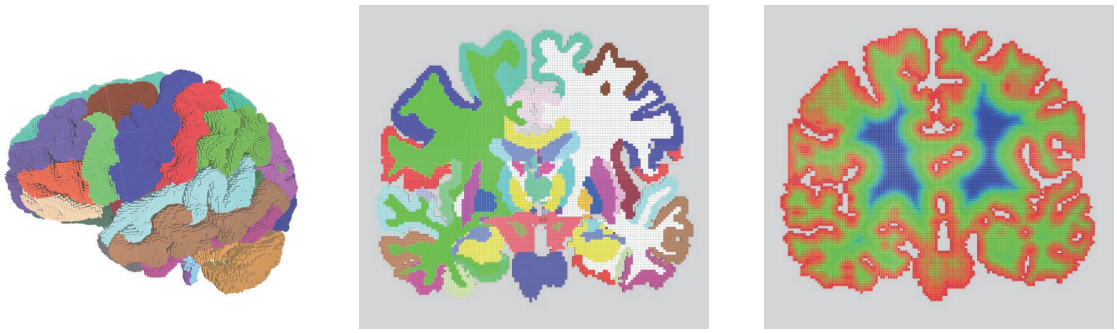
### 5.4.1 Evaluation Using a Cylinder Model

The proposed method was evaluated using a simple cylinder model (Fig. 5.5(a)). The voxel spacing of the cylinder model was 1 mm. The diameter and length of the cylinder were 76 mm (76 voxels) and 20 mm (20 voxels), respectively. The volume was segmented to 3 parts that are attached each other without any gap. An SLP {2, 3} was input to our algorithm for separating the boundary between segment 2 and 3. The FE mesh was generated with 12 mm of mesh size. The number of nodes and tetrahedral elements of the FE mesh were 264 and 729, respectively.

Fig. 5.5(b) shows the SDF generated on the cross-section surface at $x = 0$ inside the volume. This figure shows that the SDF reflects the separation between segment 2 and 3 according to the SLP.

Using the SDF, we conducted the evaluation of haptic rendering. The material properties of the deformable model was set to those of a typical soft tissue. The Young's modulus, Poisson's ratio, and density of the deformable model were set to 1000.0 Pa, 0.45, and 1000 kg/m$^3$, respectively. The Rayleigh's damping parameters $\alpha$ and $\beta$ were set to 0 and 0.1 respectively. The parameters of penalty-based contact forces were set as followings: $k_p = 1.0$ N/m, $b_p = 0.1$ Ns/m, and $a = 1.0$. The vertices on the top and bottom of the FE mesh were fixed using springs with 10 N/m of stiffness and 0.1 of damping.

The results of the simulation with haptic feedback are shown in Fig. 5.6, 5.7, and 5.8. Fig. 5.6 shows the snapshots of the simulation. Fig. 5.7 shows the logged trajectory of haptic device handle. From these figures, we can see that the user moved the tool to left direction for touching the FE model (the negative direction of $y$ axis), and then, moved it up and down to open the separated boundary, and finally moved it back to initial position. The simulation was executed with no visible vibration. Fig. 5.8 shows the contact force history sent to the haptic device controller to feedback the reaction forces to the user. Average calculation times for a time step were as followings: 1.0 ms for the collision detection, 7.9 ms for dynamics solver, 1.4 ms for voxel transformations, and 3.2 ms for graphics rendering (13.6 ms for a time step).



(a) Label map (b) Cross-section view of label map (c) Cross-section view of SDF

Figure 5.9: Brain model generated from a brain atlas [2]. In (a) and (b) the color of the voxels indicate the labels. In (c) the color of the voxels indicate the signed distance. The value is close to 0 (near the surface) if the color is red.

these results show that the collision handling in superimposed cells is successfully resolved. In addition, we can see that the direction and magnitude were qualitatively correct, i.e. the force resisted the penetration and the magnitude became larger when the larger pushing displacement was applied. The calculation time spent for the collision detection was smaller than that of dynamics solver. These results show that the proposed collision handling method worked as expected, and can be practically used for haptic rendering applications. However, we recognize that the implementation is not optimized well especially for the graphics rendering. For example, in our current implementation, all of the voxel transformations were computed on the CPU, and the results were sent to the GPU later. These computations and communications can be reduced if they are computed on GPU. Thus, the optimization of the implementation is left as an issue to be addressed for the future improvements.

### 5.4.2 Evaluation Using a Brain Model

The proposed method was evaluated using a brain atlas model [2] ($256 \times 256 \times 256$ voxels with spacing of 1 mm). The SLPs are set for separating Sylvian fissure as described in Appendix A. Fig. 5.9 shows the label map and the generated SDF according to the SLPs. The generated FE mesh consist of 3343 nodes and 9180 tetrahedral elements.

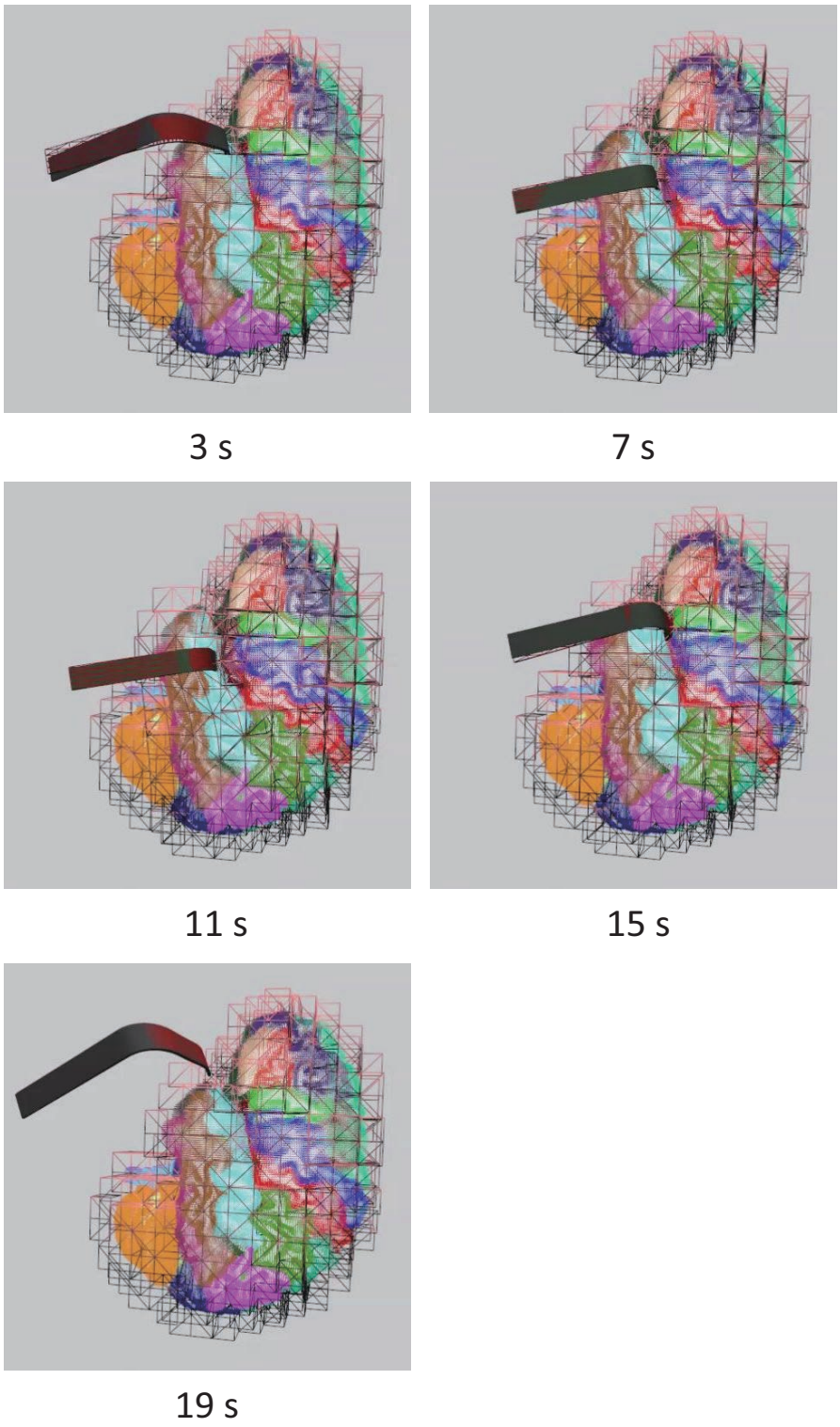



Figure 5.10: Snapshots of the evaluation using the brain model.

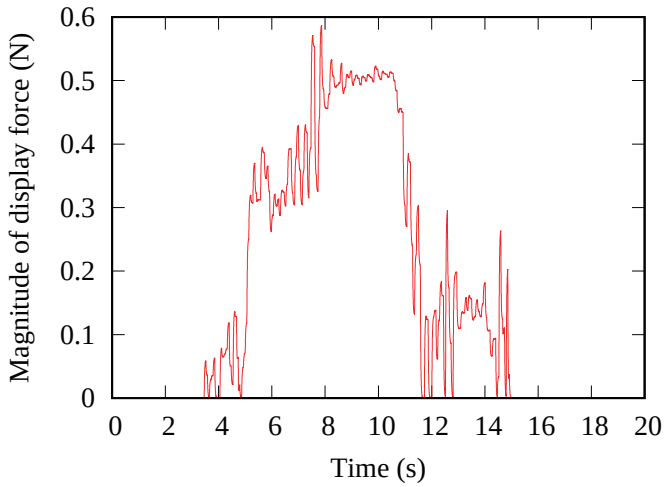



Figure 5.11: History of the magnitude of the force to be displayed to the user.

Some parameters were modified from those of Section 5.4.1 because of the complexity of the model geometry. Because the computational time became larger, the stiffness of VC visco-elastic spring was needed to be reduced for stable interaction as $k_\theta = 0.1$ Nm/rad and $b_\theta = 0.02$ Nms/rad. Additionally, because the brain model locally has thin structures, the parameters of penalty-based springs were modified to larger values as $k_p = 10.0$ N/m, $b_p = 0.01$ N/ms. Further, the friction effect was disabled by setting $a = 0$.

A result of the evaluation using the brain model is shown in Fig. 5.10. The force history to be displayed to a user is shown in Fig. 5.11. Average calculation times for a time step were as followings: 17.2 ms for the collision detection, 22.4 ms for dynamics solver, 9.4 ms for voxel transformations, and 14.0 ms for graphics rendering (62.9 ms for a time step).

In the simulation, the user suceeded in opening the Sylvian fissure with force feedback. However, because of the time delay due to the large calculation time, the stability was not maintained when the user quickly moved the tool with large velocity.

Finally, we show a visualization example on the postprocess of the simulation. Fig. 5.12 shows the visualization of the deformed medical image. The visualization is performed on 3D Slicer [6]. The deformed medical image was exported from the real-time simulation at the time 11 s (the third figure from the left in Fig. 5.10) by resampling the volume data. Because our method stores explicit relation between


Figure 5.12: Visualization on 3D Slicer [6].

the tetrahedral mesh and voxels, the deformation of the medical image can be realized straight-forwardly. Additionally, the result of the stress analysis can be reflect on the same image with the deformed volumes. For example, in Fig. 5.12, the region where the maximum principal stress exceed a threshold is colored on the images. This visualization will be useful for the quantitative evaluation of the surgery skills for learning safe surgical operations.

## 5.5 Summary

In this chapter, a haptic rendering method for embedded volume was described. The penetration depth estimation is efficiently performed using deformed distance field. The contact forces are distributed on vertices of nonconforming FE mesh. This method achieved a stable brain retraction simulation with force feedback.

# Chapter 6. Conclusion

In this thesis, a brain retraction simulator that enables force display to a user has been reported. This chapter includes a summary and future directions of this work.

## 6.1 Summary

Brain retraction is an inevitable basic operation in most approaches of craniotomy. The education and training of brain retraction are important because inappropriate retractions can induce injuries including blood flow blocking and mechanical damage on vessels or nerves. However, training opportunities are limited and new training methods have been expected.

For this problems, this thesis proposed the use of a virtual reality surgery simulator that enables a user to touch a virtual organs with force feedback. The development of a brain retraction simulator involves some difficulties: the construction of brain model, especially of patient-specific model, real-time computation of soft-tissue deformation and fracture, and stable contact simulation with force feedback. In this study, these issues were addressed by the following approaches.

**Brain model construction**

An automated patient-specific mesh generation method was developed. A generated FE mesh is nonconforming orthogonal mesh with duplicated elements. The duplicated elements are generated to preserve the topology of input fine structures. The input fine structures are deformed according to the FE mesh by interpolation of displacement field in a master-slave manner. To preserve the topology of a brain fissure, we also proposed the consideration of separation label pairs. Using this idea, we achieved the separation of a complex boundary on the Sylvian fissure.

**Real-time computation of the deformation and fracture of brain tissue**

In general, real-time computing requires to trade off the accuracy and speed. In this study, for simplicity and computation efficiency, brain tissues are modeled as a uniform linear elastic material. The geometrical nonlinearity was considered by adopting corotational formulation for strain measure. Using GPU, the global matrix assembly, the boundary condition enforcement, and iterative linear solver were accelerated. This enhanced the capacity of the simulator for simulation using a large high-resolution brain model.

**Stable contact simulation with force feedback**

The contact problem was formulated using a penalty method. To enhance the numerical stability, implicit time integration was applied. This approach enabled to simulate the contact between a rigid object and a deformable object in large time step, e.g., up to 30 ms. The stable simulation was combined with virtual coupling method for haptic rendering. Furthermore, an efficient contact handling method for nonconforming volumetric mesh was proposed. In this method, overlap between objects are estimated using deformed distance field and the overlap is resolved by a penalty method. This approach enabled retraction simulation with force feedback in good stability.

Our method has also enabled to visualization of stress field. Because proposed patient-specific model generation stores the explicit relationship between the voxels of medical images and FE mesh, the stress value on each voxel can be calculated straightforwardly. This feature is expected for useful to evaluate the risk of brain tissue damages.

As mentioned above, a haptic simulator that enables brain retraction in virtual environment has been developed. The contributions of this thesis are expected to help the clinical application of surgery simulators in the field of neurosurgery.

## 6.2 Future Directions

In this section, some possible future directions of this research are described.

**Consideration of heterogeneity in an elements**

In the proposed embedding method, the material property was assumed as uniform in an element. Obviously, this assumption causes error in mechanical modeling. This problem can be improved by adopting homogenization method.

**Accurate mechanical modeling of brain tissues**

In this thesis, the material property of a brain tissues are modeled by linear elasticity. However, brain tissues are known to have nonlinear viscoelasticity and anisotropy [85]. The measurement of material properties have been worked by many researchers including our laboratory. Such works will benefit in the future study of a brain retraction simulators. To consider such complex properties, we might need to develop a new efficient implementation of a nonlinear FEM solver.

**Consideration of important microstructures**

In this thesis, vessels, membranes and nerves were ignored. However, these structures are considerably important for neurosurgery. For practical use of the surgery simulator, we must address this problem. This problem requires more complex model construction method. One possible approach is the use of multimodal medical images. By combining multiple medical images, we could obtain more detailed structures including vessel and nerves.

**Modeling of frictions**

In this thesis, friction is represented by a anisotropic stiffness of penalty springs. However, this is a fake effect. When the effect of frictions becomes dominant in any context, one should adopt more reliable friction model such as Coulomb friction. This requires unilateral constraints in the dynamic system and leads a linear complementarity problem. A real-time solver should be implemented for such extensions.

# Acknowledgements

I would like to thank my adviser, Professor Konno for his continued support. He taught me lots of things about research and engineering as well as how a researcher should be. I also would like to express my gratitude to Professor Hajime Igarashi and Professor Satoshi Kanai for their useful advices and patient readings.

I greatly thankful to the members of the Intelligent Robots and Systems Laboratory. I would like to thank Dr. Shunsuke Komizunai for discussions on the direction of my research. I also would like to thank the members of the surgery simulator group, Ms. Xiaoshuai Chen, Mr. Akito Ema, and Mr. Noriyuki Shido, for their dairy discussions and cooperation in my experiments. I have enjoyed productive and delightful times with all of the members of our Lab.

I would like to thank Dr. Teppei Tsujita. He has supported me from my undergraduate project to my PhD thesis. I have learned a lot of useful things through the collaborative work with him. I also appreciate Mr. Akira Fukuhara for the collaborative work on surgery planning and productive discussions during my master project. I would like to thank Dr. Meng-Hung Wu and Mr. Shuhei Ogawa for advising the implementation of my algorithms and sharing useful information on current technologies.

I also appreciate medical doctors who gave useful advices to my research. Dr. Atsuhiro Nakagawa has advised me through TV meetings as well as showed his experiment rooms in his hospital. Dr. Atsuhiro Nakagwa and Dr. Tomohiro Kawaguchi kindly evaluated our simulator and gave us meaningful comments. I would like to thank Projessor Teiji Tominaga for his kind cooperation of the collaborative work with our research group. I would like to thank Professor Masaru Uchiayama for supervising my master thesis at Tohoku University. My PhD project is an extension of the master thesis and knowledge obtained in the master project was the base of my PhD project. I also would like to thank Dr. Satoko Abiko,

Dr. Xin Jiang, and Dr. Koyu Abe for their productive discussions during my master project at Tohoku University.

Finally, I would like to thank my parents, sisters, and friends for supporting me. Without their encouragement, this work would have been impossible.

# References


[1] K. Sase, A. Fukuhara, T. Tsujita, and A. Konno, "GPU-accelerated surgery simulation for opening a brain fissure," *ROBOMECH Journal*, vol. 2, no. 1, pp. 1–16, 2015.

[2] M. Halle, I.-F. Talos, M. Jakab, N. Makris, D. Meier, L. Wald, B. Fischl, and R. Kikinis, "Multi-modality MRI-based atlas of the brain," 2015, https://www.spl.harvard.edu/publications/item/view/2037.

[3] D. Boltcheva, M. Yvinec, and J.-D. Boissonnat, "Mesh generation from 3d multi-material images," in *MICCAI 2009, Part II. LNCS*, G.-Z. Yang, D. Hawkes, D. Rueckert, A. Noble, and C. Taylor, Eds. Heidelberg: Springer, 2009, vol. 5762, pp. 283–290.

[4] J. P. Pons, F. Ségonne, J. D. Boissonnat, L. Rineau, M. Yvinec, and R. Keriven, "High-quality consistent meshing of multi-label datasets," in *IPMI 2007, LNCS*, N. Karssemeijer and B. Lelieveldt, Eds. Heidelberg: Springer, 2007, vol. 4584, pp. 198–210.

[5] K. Sase, T. Tsujita, and A. Konno, "Embedding segmented volume in finite element mesh with topology preservation," in *Proceedings of the 19th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2016), Part III*, 2016, pp. 116–123.

[6] A. Fedorov, R. Beichel, J. Kalpathy-Cramer, J. Finet, J.-C. Fillion-Robin, S. Pujol, C. Bauer, D. Jennings, F. Fennessy, M. Sonka, J. Buatti, S. Aylward, J. V. Miller, S. Pieper, and R. Kikinis, "3D Slicer as an image computing platform for the quantitative imaging network," *Magnetic Resonance Imaging*, vol. 30, no. 9, pp. 1323–1341, 2012.

[7] "Natinal Alliance for Medical Image Computing (NAMIC): Brain Mutlimodality," http://insight-journal.org/midas/collection/view/190.

[8] K. Sase, T. Tsujita, and A. Konno, "Topology presavation method for embedding a segmented medical image (in Japanese)," *Proceedings of the 34th Annual Conference of the Robotics Society of Japan*, pp. RSJ2016AC1W1–06, 2016.

[9] K. Sase, T. Tsujita, and A. Konno, "Automatic generation of finite element mesh with brain-fissure structure based on medical-image segmentation (in Japanese)," *IEICE Technical Report*, vol. 115, no. 401, pp. 7–11, 2016.

[10] K. Sase, A. Konno, T. Tsujita, A. Fukuhara, X. Chen, and S. Komizunai, "Stable fracture model of soft materials for a simulation of brain tumor resection (in Japanese)," in *Proceedings of the 2014 JSME Conference on Robotics and Mechatronics*, 2014, pp. 3A1–B03.

[11] K. Sase, T. Tsujita, and A. Konno, "Haptic rendering of contact between rigid and deformable objects based on penalty method with implicit time integration," *Proceedings of the 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO 2016)*, pp. 1594–1600, 2016.

[12] M. Dujovny, O. Ibe, A. Perlin, and T. Ryder, "Brain retractor systems," *Neurological Research*, vol. 32, no. 7, pp. 675–683, 2010.

[13] R. J. Andrews and J. R. Bringas, "A review of brain retraction and recommendations for minimizing intraoperative brain injury," vol. 33, no. December 1993, pp. 1–19, 1993.

[14] J. Zhong, M. Dujovny, A. R. Perlin, E. Perez-Arjona, H. K. Park, and F. G. Diaz, "Brain retraction injury," *Neurological Research*, vol. 25, no. 8, pp. 831–838, 2003.

[15] M. G. Yasargil, *Microneurosurgery*. Thieme, 1995.

[16] K. Hino, R. Tanikawa, T. Sugimura, M. Iwasaki, N. Izumi, A. Hashidume, T. Fujita, M. Hashimoto, and H. Kamiyama, "Microsurgical technique with-

out pial injury for transsylvian approach," *Surg. Cereb. Stroke (in Japanese)*, vol. 34, pp. 96–100, 2006.

[17] T. Mashiko, T. Konno, N. Kaneko, and E. Watanabe, "Training in brain retraction using a self-made three-dimensional model," *World Neurosurgery*, vol. 84, no. 2, pp. 585–590, 2015.

[18] A. G. Gallagher and E. Matt Ritter, *Virtual Reality: Objective Assessment, Education, and Training*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 27–33.

[19] T. R. Coles, D. Meglan, and N. W. John, "The role of haptics in medical training simulators: A survey of the state of the art," *IEEE Transactions on Haptics*, vol. 4, no. 1, pp. 51–66, 2011.

[20] S. Kapoor, P. Arora, V. Kapoor, M. Jayachandran, and M. Tiwari, "Haptics - touchfeedback technology widening the horizon of medicine," *Journal of Clinical and Diagnostic Research*, vol. 8, no. 3, pp. 294–299, 2014.

[21] LapSim, http://www.surgical-science.com/lapsim-the-proven-training-system.

[22] LapVR, http://caehealthcare.com/eng/interventional-simulators/lapvr.

[23] D. Escobar-Castillejos, J. Noguez, L. Neri, A. Magana, and B. Benes, "A review of simulators with haptic devices for medical training," *Journal of Medical Systems*, vol. 40, no. 4, pp. 1–22, 2016.

[24] H. R. Malone, O. N. Syed, M. S. Downes, A. L. D'Ambrosio, D. O. Quest, and M. G. Kaiser, "Simulation in neurosurgery: A review of computer-based simulation environments and their surgical applications," *Neurosurgery*, vol. 67, pp. 1105–1116, 2010.

[25] G. M. Lemole, P. P. Banerjee, C. Luciano, S. Neckrysh, and F. T. Charbel, "Virtual reality in neurosurgical education: Part-task ventriculostomy simulation with dynamic visual and haptic feedback," *Neurosurgery*, vol. 61, pp. 142–149, 2007.

[26] M. A. Spicer, M. van Velsen, J. P. Caffrey, and M. L. J. Apuzzo, "Virtual reality neurosurgery: a simulator blueprint," *Neurosurgery*, vol. 54, pp. 783–798, 2004.

[27] S. Delorme, D. Laroche, R. DiRaddo, and R. F. Del Maestro, "Neurotouch: a physics-based virtual simulator for cranial microneurosurgery training," *Neurosurgery*, vol. 71, pp. 32–42, 2012.

[28] V. Mora, D. Jiang, R. Brooks, and S. Delorme, "A computer model of soft tissue interaction with a surgical aspirator," in *Proceedings of the 12th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2009), Part I*, 2009, pp. 51–58.

[29] NeuroVR, http://caeneurovr.com.

[30] A. Alaraj, C. J. Luciano, D. P. Bailey, A. Elsenousi, B. Z. Roitberg, A. Bernardo, P. P. Banerjee, and F. T. Charbel, "Virtual reality cerebral aneurysm clipping simulation with real-time haptic feedback," *Neurosurgery*, vol. 11, no. 1, pp. 52–58, 2015.

[31] T. Koyama, H. Okudera, and S. Kobayashi, "Computer-generated surgical simulation of morphological changes in microstructures: concepts of "virtual retractor"," *Journal of Neurosurgery*, vol. 90, no. 4, pp. 780–785, 1999.

[32] K. V. Hansen, L. Brix, C. F. Pedersen, J. P. Haase, and O. V. Larsen, "Modelling of interaction between a spatula and a human brain," *Medical Image Analysis*, vol. 8, no. 1, pp. 23–33, 2004.

[33] Y. Hasegawa, K. Adachi, Y. Azuma, A. Fujita, E. Kohmura, and H. Kanki, "A study on cerebellar retraction simulation for developing neurosurgical training system," *Jounal of Japan Society of Computer Aided Surgery*, vol. 12, no. 4, pp. 533–543, 2010.

[34] S. Misra, K. T. Ramesh, and A. M. Okamura, "Modeling of tool-tissue interactions for computer-based surgical simulation: A literature review," *Presence: Teleoper. Virtual Environ.*, vol. 17, no. 5, pp. 463–491, Oct. 2008.

[35] A. Wittek, N. M. Grosland, G. R. Joldes, V. Magnotta, and K. Miller, "From finite element meshes to clouds of points: A review of methods for generation of computational biomechanics models for patient-specific applications," *Annals of Biomedical Engineering*, vol. 44, no. 1, pp. 3–15, 2016.

[36] "CGAL: Computational Geometry Algorithms Library," http://www.cgal.org/.

[37] "GiD," www.gidhome.com/.

[38] J. R. Nieto and A. Susín, *Cage Based Deformations: A Survey.* Dordrecht: Springer Netherlands, 2013, pp. 75–99.

[39] M. Muller, M. Teschner, and M. Gross, "Physically-based simulation of objects represented by surface meshes," in *Proceedings of the Computer Graphics International*, ser. CGI '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 26–33.

[40] M. Nesme, P. G. Kry, L. Jeřábková, and F. Faure, "Preserving topology and elasticity for embedded deformable models," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 52:1–52:9, 2009.

[41] M. Müller and M. Gross, "Interactive virtual materials," in *Proceedings of Graphics Interface 2004*, 2004, pp. 239–246.

[42] I.-F. Talos, M. Jakab, and R. Kikinis, "SPL abdominal atlas," 2008, http://www.spl.harvard.edu/publications/item/view/1266.

[43] C. Forest, H. Delingette, and N. Ayache, "Removing tetrahedra from manifold tetrahedralisation: application to real-time surgical simulation," *Medical Image Analysis*, vol. 9, no. 2, pp. 113–122, 2005.

[44] M. Nakayama, S. Abiko, X. Jiang, A. Konno, and M. Uchiyama, "Stable soft-tissue fracture simulation for surgery simulator," *Journal of Robotics and Mechatronics*, vol. 23, no. 4, pp. 589–597, 2011.

[45] H. Courtecuisse, H. Jung, J. Allard, C. Duriez, D. Y. Lee, and S. Cotin, "Gpu-based real-time soft tissue deformation with cutting and haptic feedback," *Progress in Biophysics and Molecular Biology*, vol. 103, pp. 159–168, 2010.

[46] H. Courtecuisse, J. Allard, P. Kerfriden, S. P. Bordas, S. Cotin, and C. Duriez, "Real-time simulation of contact and cutting of heterogeneous soft-tissues," *Medical Image Analysis*, vol. 18, no. 2, pp. 394–410, 2014.

[47] N. Galoppo, S. Tekin, M. A. Otaduy, M. Gross, and M. C. Lin, "Interactive haptic rendering of high-resolution deformable objects," in *Proceedings of the 2nd International Conference on Virtual Reality*, 2007, pp. 215–233.

[48] S. Cotin, H. Delingette, and N. Ayache, "Real-time elastic deformations of soft tissues for surgery simulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 62–73, Jan. 1999.

[49] K. Hirota and T. Kaneko, "Haptic representation of elastic objects," *Presence: Teleoperators Virtual Environments*, vol. 10, no. 5, pp. 525–536, Oct. 2001.

[50] J. F. O'Brien, A. W. Bargteil, and J. K. Hodgins, "Graphical modeling and animation of ductile fracture," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 291–294, 2002.

[51] C. Wojtan, N. Thürey, M. Gross, and G. Turk, "Deforming meshes that split and merge," *ACM Transactions on Graphics*, vol. 28, no. 3, pp. 76:1–76:10, 2009.

[52] J. Hegemann, C. Jiang, C. Schroeder, and J. M. Teran, "A level set method for ductile fracture," *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 193–201, 2013.

[53] A. B. Mor and T. Kanade, "Modifying soft tissue models: progressive cutting with minimal new element creation," in *Proceedings of the Third International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2000, pp. 598–607.

[54] L. Jeřábková and T. Kuhlen, "Stable cutting of deformable objects in virtual environments using xfem," *IEEE Computer Graphics and Applications*, vol. 29, no. 2, pp. 61–71, 2009.

[55] H. Delingette, S. Cotin, and N. Ayache, "A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation," in *Proceedings of Computer Animation 1999*, 1999, pp. 70–81.

[56] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler, "Stable real-time deformations," *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 49–55, 2002.

[57] G. Irving, J. Teran, and R. Fedkiw, "Invertible finite elements for robust simulation of large deformation," *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 131–140, 2004.

[58] A. Myronenko and X. Song, "On the closed-form solution of the rotation matrix arising in computer vision problems," *Technical Report arXiv:0904.1613v1 [cs.CV]*, 2009.

[59] S. Lahabar and P. Narayanan, "Singular value decomposition on gpu using cuda," in *Proceedings of 23rd IEEE International Parallel and Distributed Processing Symposium*, May 2009, pp. 1–10.

[60] J. Bedkowski and A. Maslowski, "GPGPU computation in mobile robot applications," *International Journal on Electrical Engineering and Informatics*, vol. 4, no. 1, pp. 15–26, 2011.

[61] A. Fukuhara, T. Tsujita, K. Sase, A. Konno, X. Jiang, S. Abiko, and M. Uchiyama, "Proposition and evaluation of a collision detection method for real time surgery simulation of opening a brain fissure," *ROBOMECH Journal*, vol. 1, no. 1, p. 6, 2014.

[62] C. Cecka, A. Lew, and E. Darve, "Application of assembly of finite element methods on graphics processors for real-time elastodynamics," in *GPU Computing Gems Jade Edition*, W.-m. W. Hwu, Ed., Boston, 2012, pp. 187–205.

[63] J. E. Colgate, M. C. Stanley, and J. M. Brown, "Issues in the haptic display of tool use," in *Proceedings. 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots'*, vol. 3, Aug 1995, pp. 140–145.

[64] M. A. Otaduy and M. Gross, "Transparent rendering of tool contact with compliant environments," in *Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WHC'07)*, March 2007, pp. 225–230.

[65] C. Duriez, C. Andriot, and A. Kheddar, "A multi-threaded approach for deformable/rigid contacts with haptic feedback," in *12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, March 2004, pp. 272–279.

[66] J. Barbič and D. L. James, "Six-DoF haptic rendering of contact between geometrically complex reduced deformable models," *IEEE Transaction on Haptics*, vol. 1, no. 1, pp. 39–52, Jan. 2008.

[67] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '98. New York, NY, USA: ACM, 1998, pp. 43–54.

[68] M. A. Otaduy and M. C. Lin, "A modular haptic rendering algorithm for stable and transparent 6-DOF manipulation," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 751–762, Aug 2006.

[69] D. Baraff, "Physically based modeling: Rigid body simulation," *ACM SIGGRAPH Course Notes*, 2001.

[70] M. A. Otaduy, C. Garre, and M. C. Lin, "Representations and algorithms for force-feedback display," *Proceedings of the IEEE*, vol. 101, no. 9, pp. 2068–2080, 2013.

[71] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis, "Efficient elasticity for character skinning with contact and collisions," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 37:1–37:12, 2011.

[72] C. Garre and M. A. Otaduy, "Haptic rendering of objects with rigid and deformable parts," *Computers & Graphics*, vol. 34, no. 6, pp. 689–697, 2010.

[73] K. Hirota and K. Tagawa, "Interaction with virtual object using deformable hand," in *2016 IEEE Virtual Reality (VR)*, March 2016, pp. 49–56.

[74] M. W. Jones, J. A. Baerentzen, and M. Sramek, "3D distance fields: A survey of techniques and applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 581–599, Jul. 2006.

[75] Y. Masutani, Y. Inoue, K. Ishii, N. Kumai, F. Kimura, and I. Sakuma, "Development of surgical simulator based on fem and deformable volume-rendering," in *Proc. SPIE*, vol. 5367, 2004, pp. 500–507.

[76] M. Nakao and K. Minato, "Physics-based interactive volume manipulation for sharing surgical process," *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, no. 3, pp. 809–816, May 2010.

[77] R. Torres, A. Rodríguez, J. M. Espadero, and M. A. Otaduy, "High-resolution interaction with corotational coarsening models," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 211:1–211:11, Nov. 2016.

[78] J. Allard, F. Faure, H. Courtecuisse, F. Falipou, C. Duriez, and P. G. Kry, "Volume Contact Constraints at Arbitrary Resolution," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 82:1—-82:10, 2010.

[79] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 163–169.

[80] J. Wu, C. Dick, and R. Westermann, "Efficient collision detection for composite finite element simulation of cuts in deformable bodies," *Vis. Comput.*, vol. 29, no. 6-8, pp. 739–749, Jun. 2013.

[81] N. Mitchell, M. Aanjaneya, R. Setaluri, and E. Sifakis, "Non-manifold Level Sets : A multivalued implicit surface representation with applications to self-collision processing," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 247:1–247:9, 2015.

[82] S. Fisher and L. C. Ming, "Deformed distance fields for simulation of non-penetrating flexible bodies," *Computer Animation and Simulation 2001*, pp. 99–111, 2001.

[83] G. Hirota, S. Fisher, and A. State, "An improved finite-element contact model for anatomical simulations," *Visual Computer*, vol. 19, no. 5, pp. 291–309, aug 2003.

[84] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," in *Proceedings of the Vision, Modeling, and Visualization Conference 2003 (VMV 2003)*, 2003, pp. 47–54.

[85] D. Sahoo, C. Deck, and R. Willinger, "Development and validation of an advanced anisotropic visco-hyperelastic human brain FE model," *Journal of the Mechanical Behavior of Biomedical Materials*, vol. 33, no. 1, pp. 24–42, 2014.

# List of Publications

## Journal Publications

[1] X. Chen, **K. Sase**, A. Konno, T. Tsujita, S. Komizunai, "A simple damage and fracture model of brain parenchyma for haptic brain surgery simulations," *Journal of Biomechanical Science and Engineering (JBSE)*, vol. 11, no. 4, 2016.

[2] A. Fukuhara, T. Tsujita, **K. Sase**, A. Konno, Atsuhiro Nakagawa, Toshiki Endo, Teiji Tominaga, X. Jiang, S. Abiko and M. Uchiyama, "Securing an Optimum Operating Field without Undesired Tissue Damage in Neurosurgery," *Advanced Robotics (AR)*, Taylor & Francis, vol. 30, no. 19, pp. 1245–1259, 2016.

[3] **K. Sase**, A. Fukuhara, T. Tsujita, A. Konno, "GPU-accelerated surgery simulation for opening a brain fissure," *ROBOMECH Journal*, vol. 2, no. 1, Article 17, 2015.

[4] A. Fukuhara, T. Tsujita, **K. Sase**, A. Konno, X. Jiang, S. Abiko, M. Uchiyama, "Proposition and evaluation of a collision detection method for real time surgery simulation of opening a brain fissure," *ROBOMECH Journal*, vol. 1, no. 1, 2014.

[5] T. Tsujita, **K. Sase**, A. Konno, M. Nakayama, X. Chen, K. Abe, and M. Uchiyama, "Design and Evaluation of an Encountered-type Haptic Interface Using MR Fluid for Surgical Simulators," *Advanced Robotics (AR)*, Taylor & Francis, vol. 27, no. 7, pp. 525–540, 2013.

[6] Y. Hayashi, Y. Tamura, **K. Sase**, K. Sugawara, Y. Sawada, "Intermittently-

visual Tracking Experiments Reveal the Roles of Error-correction and Predictive Mechanisms in the Human Visual-motor Control System", *Transactions of the Society of Instrument and Control Engineers*, vol. 46, no. 7, pp. 391–400, 2010 (in Japanese).

## International Conferences (Full Review)

[1] X. Chen, **K. Sase**, A. Konno, T. Tsujita, "A Viscoelastic Model of Brain Parenchyma for Haptic Brain Surgery Simulations," *Proceedings of the 2016 IEEE/SICE International Symposium on System Integration (SII)*, Sapporo, Japan , December 14, 2016.

[2] **K. Sase**, T. Tsujita, A. Konno, "Haptic Rendering of Contact Between Rigid and Deformable Objects Based on Penalty Method with Implicit Time Integration," *Proceedings of the 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Qingdao, China, December 6, 2016.

[3] X. Chen, **K. Sase**, A. Konno, T. Tsujita, "Experimental and Numerical Analysis of Damage Fracture Mechanics of Brain Parenchyma," *Proceedings of the 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Qingdao, China, December 4, 2016.

[4] **K. Sase**, T. Tsujita, A. Konno, "Embedding Segmented Volume in Finite Element Mesh with Topology Preservation," *Proceedings of the 19th Internatinal Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 116–123, October 17-21, 2016, Athens, Greek.

[5] A. Fukuhara, T. Tsujita, **K. Sase**, A. Konno, X. Jiang, S. Abiko, M. Uchiyama, "Optimization of retraction in neurosurgery to avoid damage caused by deformation of brain tissues," *Proceedings of the 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, December 5-10, 2014, Bali, Indonesia.

[6] X. Chen, **K. Sase**, A. Konno and T. Tsujita, "Identification of Mechanical Properties of Brain Parenchyma for Brain Surgery Haptic Simulation,"

*Proceedings of the 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, December 5-10, 2014, Bali, Indonesia.

[7] T. Tsujita, M. Ohara, **K. Sase**, A. Konno, M. Nakayama, K. Abe, and M. Uchiyama, "Development of a haptic interface using MR fluid for displaying cutting forces of soft tissues," *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA 2012)*, pp. 1044-1049, Saint Paul, 2012.

## International Conferences (No Review)

[1] A. Konno, M. Nakayama, X. Chen, A. Fukuhara, **K. Sase**, T. Tsujita and S. Abiko, "Development of a Brain Surgery Simulator," in *Proceedings of the International Symposium on Interdisciplinary Research and Education on Medical Device Developments (IREMD)*, I-6, Hirosaki, Japan, 13 September, 2013.

## Domestic Conferences

[1] **K. Sase**, T. Tsujita, A. Konno, "Topology Presavation Method for Embedding a Segmented Medical Image (in Japanese)," *Proceedings of the 34th Annual Conference of the Robotics Society of Japan*, RSJ2016AC1W1-06, 2016 (in Japanese).

[2] A. Ema, **K. Sase**, X. Chen, T. Tsujita, A. Konno, "GPU Acceleration Method of Fluid-Structure Coupled Analysis for Intraoperative Brain-sift Estimation," *Proceedings of the 34th Annual Conference of the Robotics Society of Japan*, RSJ2016AC1W1-08, 2016 (in Japanese).

[3] X. Chen, **K. Sase**, A. Konno, T. Tsujita, "Damage Model of Brain Parenchyma for Neurosurgery Simulation," *Proceedings of the 34th Annual Conference of the Robotics Society of Japan*, RSJ2016AC1W1-01, 2016 (in Japanese).

[4] A. Ema, **K. Sase**, T. Tsujita, A. Konno, "Fluid-Structure Coupled Analysis Using Particle Method and Finite Element Method for Brain-shift Estimation," *Proceedings of JSME Conference on Robotics and Mechatronics 2016*, 1A1-02b7, 2016 (in Japanese).

[5] **K. Sase**, T. Tsujita, A. Konno, "Automatic Generation of Finite Element Mesh with Brain-Fissure Structure Based on Medical-Image Segmentation," *IEICE Technical Report*, vol. 115, no. 401, pp. 7–11, 2016 (in Japanese).

[6] A. Ema, **K. Sase**, T. Tsujita, A. Konno, "A Fluid-Structure Coupled Analysis Method for Intraoperative Brain-shift Estimation," *Proceedings of the 33th Annual Conference of the Robotics Society of Japan*, RSJ2015AC1J2-02,2015 (in Japanese).

[7] **K. Sase**, A. Fukuhara, T. Tsujita, A. Konno, "Haptic Interaction Using a Robust Soft-tissue Deformation and Fracture Model for Real-time Neurosurgery Simulator," *Proceedings of the 32th Annual Conference of the Robotics Society of Japan*, RSJ2014AC3H2-03, 2014 (in Japanese).

[8] A. Fukuhara, T. Tsujita, **K. Sase**, A. Konno, X. Jiang, S. Abiko, M. Uchiyama "Optimization of Surgical Path for Avoidance of Brain Retraction Injury," *Proceedings of the 32th Annual Conference of the Robotics Society of Japan*, RSJ2014AC3H2-01, 2014 (in Japanese).

[9] N. Nagayasu, A. Konno, T. Tsujita, **K. Sase**, S. Komizunai, "Physical and Contact Modeling of Surgical Thread for Suture Simulation," *Proceedings of the 32th Annual Conference of the Robotics Society of Japan*, RSJ2014AC3H2-02, 2014 (in Japanese).

[10] **K. Sase**, A. Konno, T. Tsujita, A. Fukuhara, X. Chen, S. Komizunai, "Real-time Simulation of Brain Tumor Resection Using GPGPU," *Proceedings of JSME Conference on Robotics and Mechatronics 2014*, 3A1-B04, 2014 (in Japanese).

[11] **K. Sase**, A. Konno, T. Tsujita, A. Fukuhara, X. Chen, S. Komizunai, "Stable Fracture Model of Soft Materials for a Simulation of Brain Tumor Resection," *Proceedings of JSME Conference on Robotics and Mechatronics 2014*, 3A1-B03, 2014 (in Japanese).

[12] N. Nagayasu, A. Konno, T. Tsujita, **K. Sase**, S. Komizunai, "Real-time Suture Simulation of Soft Tissue," *Proceedings of JSME Conference on Robotics and Mechatronics 2014*, 3P1-A04, 2014 (in Japanese).

[13] A. Fukuhara, T. Tsujita, **K. Sase**, A. Konno, X. Jiang, S. Abiko, M. Uchiyama, "Collision Detection for Real-time Surgery Simulation of Opening a Brain Fissure", *Proceedings of JSME Conference on Robotics and Mechatronics 2013*, 2A1-L02, 2013 (in Japanese).

[14] Y. Inoue, T. Kameyama, **K. Sase**, T. Tsujita, X. Jiang, S. Abiko, M. Uchiyama, "Cutting Force Display Using a MR-fluid-based Encountered-type Haptic Interface for Surgical Simulators," *Proceedings of the 13th SICE System Integration Division Annual Conference (SI2012)*, 2D1-2, pp. 1307–1310, 2012 (in Japanese).

[15] T. Kameyama, T. Tsujita, **K. Sase**, X. Jiang, S. Abiko, M. Uchiyama, "Analysis of Flow characteristics of MR Fluid for Displaying Cutting Force of Soft Tissue," *Proceedings of the 13th SICE System Integration Division Annual Conference (SI2012)*, 2D1-5, pp. 1318–1321, 2012 (in Japanese).

[16] **K. Sase**, M. Nakayama, Y. Satake, S. Abiko, X. Jiang, T. Tsujita, A. Konno, M. Uchiyama, "Evaluation of Real-time Performance of the Blunt Dissection Simulator Using a Real Brain Model," *Proceedings of JSME Conference on Robotics and Mechatronics 2012*, 2P1-U03, 2012 (in Japanese).

[17] **K. Sase**, X. Chen, M. Tomita, T. Tsujita, A. Konno, M. Nakayama, K. Abe, M. Uchiyama, "Development of a Haptic Surgery Simulator Using MR Fluid", *Proceedings of the 12th SICE System Integration Division Annual Conference (SI2011)*, pp. 1540–1543, 2011 (in Japanese).

[18] T. Tsujita, **K. Sase**, M. Ohara, A. Konno, M. Nakayama, K. Abe, M. Uchiyama, "Development of Haptic Interface for Force Display of Soft-tissue Cutting Using MR Fluid," *Proceedings of the 29th Annual Conference of the Robotics Society of Japan*, 3L1 − 6, 2011 (in Japanese).

[19] T. Tsujita, M. Nakano, **K. Sase**, K. Yoshida "Small Braille Display System Using Diaphragm Actuator Controlled by ER Fluid Microvalve," *Proceedings of the Annual Spring Conference of the Japan Fluid Power System Society*

*2010*, pp. 109-111, 2010 (in Japanese).

[20] Y. Hayashi, Y. Tamura, **K. Sase**, K. Sugawara, Y. Sawada, "Generation of Rhythm and Occurrence of Precedence in Human Visual Tracking Motion", *Proceedings of the 24th Symposium on Biological and Physiological Engineering*, pp. 273–274, 2009 (in Japanese).

[21] **K. Sase**, K. Yaegashi, M. Yuki, Y. Hayashi, "Experiment Education Program Starting with the Manufacture of Measuring Instruments", *Proceedings of the Annual Conference of the Physical Society of Japan*, vol. 64, no. 2, p. 305, 2009 (in Japanese).

[22] K. Yaegashi, **K. Sase**, M. Yuki, Y. Hayashi, "Development and Implementation of an 80-hour Educational Program for the MONODUKURI Course," *Proceedings of the Annual Conference of the Physical Society of Japan*, vol. 64, no. 2, p. 304, 2009 (in Japanese).

# Appendix A Examples of SLPs

The SLPs (Separation Label Pairs, defined in Chapter 2) for preserving topology of the brain fissures were manually specified. The task of the construction of this SLPs required several hours. Although this is a labor-intensive work, SLPs are repeatedly used if the application is specified.

The specified SLPs are as the following. Note that the integers are labels used in the atlas-based segmentation tool of 3D Slicer ver. 4.5.0. The definition of the labels are found in the file located on the following path: `(Slicer_installed_directory)/share/Slicer-4.5/ColorFiles/SPL-BrainAtlas-2012-ColorFile.txt`.

```
# Sylvian fissure
{1000, 1018}, {1000, 1022}, {1000, 1030}, {1000, 1034}, {1030, 1031}, {1031, 1034}, {1022, 1030},
{1022, 1034}, {1024, 1030}, {2000, 2024}, {2000, 2018}, {2000, 2022}, {2000, 2030}, {2000, 2031},
{2000, 2034}, {2018, 2030}, {2022, 2024}, {2022, 2030}, {2022, 2034}, {2030, 2018}, {2030, 2022},
{2030, 2024}, {2030, 2031}, {2031, 2034},
# Left cerebellum
{8, 2}, {8, 1099}, {8, 1011}, {8, 1013}, {8, 1016}, {8, 1007}, {8, 1009}, {7, 1099}, {7, 1011},
{7, 1013}, {7, 1016}, {7, 1007}, {7, 1009},
# Right cerebellum
{47, 2}, {47, 2013}, {47, 2007}, {47, 2011}, {47, 41}, {47, 2009}, {47, 1013}, {46, 2013},
{46, 2007}, {46, 2011}, {46, 2009},
# Longitudial fissure
{2, 54}, {2, 58}, {2, 508}, {2, 2005}, {2, 2013}, {2, 2014}, {2, 2017}, {2, 2023}, {2, 2021},
{2, 2028}, {2, 3002}, {26, 2014}, {41, 26}, {41, 509}, {41, 1005}, {41, 1013}, {41, 1014},
{41, 1017}, {41, 1021}, {41, 1025}, {41, 1028}, {41, 1029}, {54, 3003}, {508, 509}, {1002, 2002},
{1002,2023}, {1002, 2026}, {1005, 2005}, {1005, 2021}, {1010, 2010}, {1010, 2013}, {1010, 2025},
{1011, 2005}, {1011, 2011}, {1011, 2021}, {1012, 2012}, {1013, 2005}, {1013, 2013}, {1013, 2021},
{1014, 2012}, {1014, 2014}, {1014, 2032}, {1017, 2017}, {1017, 2023}, {1017, 2025}, {1017, 2029},
{1021, 2005}, {1021, 2021}, {1022, 2029}, {1023, 2010}, {1023, 2017}, {1023, 2023}, {1023, 2025},
{1025, 2005}, {1025, 2010}, {1025, 2013}, {1025, 2021}, {1025, 2025}, {1025, 2029}, {1026, 2014},
{1026, 2026}, {1028, 2002}, {1028, 2017}, {1028, 2023}, {1028, 2024}, {1028, 2026}, {1028, 2028},
{1029, 2005}, {1029, 2025}, {1029, 2029}, {1032, 2012}, {1032, 2014}, {1032, 2032},
# Mammillary body
{3002, 3003},
```