Interactive Volumetric Image Manipulation for Patient-specific Soft Tissue Deformation and Cutting

Abstract

This paper systematically advocates an interactive volumetric image manipulation framework, which enables physically plausible modeling of patient-specific soft tissues such as: large-scale deformation, arbitrary cutting, synchronized collision handling, and realistic anatomical visualization. We present a set of seamless technical elements that allow material-aware proxy structure generation, Non-linear Finite Element Method (NFEM) based simulation, fitting based up-sampling of sparse displacement field, and dynamic visualization of the evolved volume, which are all relevant to the instant utilization of clinical images in patient-specific and subject-centric virtual surgery. In particular, the entire framework is built upon CUDA, and thus can achieve interactive performance even on a common laptop. The detailed implementation, timing tests, and physical behavior measurements show its practicality, efficiency, and robustness.

Keywords: volumetric image manipulation, patient-specific medical simulation, physics based modeling, deformation and cutting, CUDA.

1 Introduction and motivation

The accurate and efficient deformation/cutting simulation of soft tissues plays a vital role in medical simulation. Many iso-surface related generic models, ranging from simple spring-mass structure to complex finite element representations, have been employed to achieve this goal. However, although the interior anatomical structures can greatly affect the behavior and appearance of soft tissues, most of the state-of-the-art methods have little concern over the wide range variability of patient-specific datasets, which usually adopt common geometric model and do not afford instant data replacement. In sharp contrast, a great amount of patient-specific volumetric medical images have been routinely collected. Thus, the strong demand for medical simulation has been evolving rapidly from being solely operation-oriented to a higher level of being patient-specific and subject-centric [1]. Meanwhile, it also requires synchronous exploration of the real internal anatomical structures and interactive feedback. Therefore, there is still a large gap between the flexible manipulation of patient-specific volumetric images and the simulation with physical and physiological reality. Specifically, the technical challenges are concluded as follows.

First, since volumetric image in nature comprise a densely-sampled 3D scalar field and has no explicit inner geometric structures, surface appearance, and accompanying physical properties, state-of-the-art surgery simulators highly depend on time-consuming manual intermediate steps to reconstruct object-specific physical and geometric models from volumetric medical images. This is the major reason that prevents patient-specific simulations from being efficiently and extensively used in clinical setting [2].

Second, surface based models are not well suited for realistic visualization of arbitrary cutting surfaces, since it remains difficult to real-time synthesizing and mapping a plausible texture onto the dynamically-generated cutting surface, not to mention the representation of de-

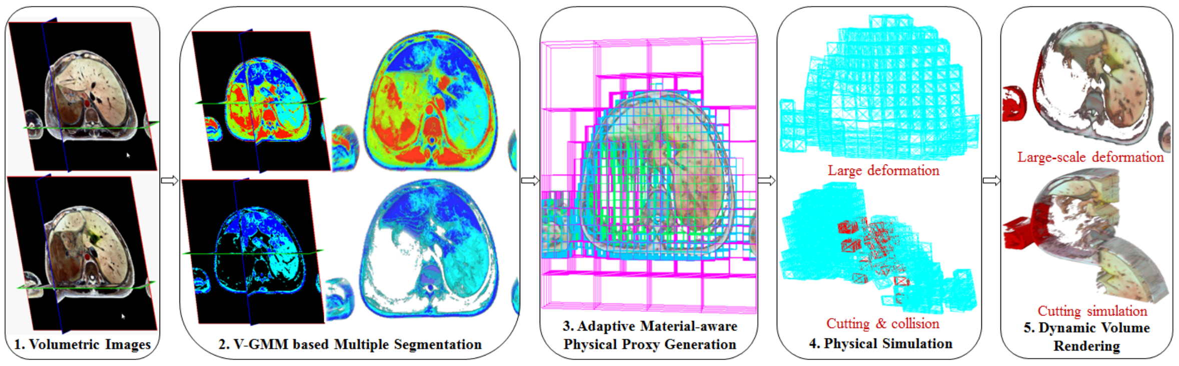

Figure 1: The pipeline of our framework.

formable anatomical structures. This unavoidably downgrades the role of patient-specific volumetric medical images and limits the realistic experience of clinicians during surgical rehearsal [3, 4].

Third, although voxel based volume editing approaches can produce simple transforming/sculpting/drilling results in local regions [5], they are usually susceptible to global aliasing artifacts. Besides, due to abundant computation time from massive voxels, it is not feasible to employ them to conduct large-scale deformation and arbitrary cutting simulation of soft tissues.

Fourth, the constitutive behavior of soft tissues should be rigorously modeled with nonlinear kinematic formulations [6], and the physical structure changes, accompanying collision, user intervention, and visualization must be dynamically handled during cutting simulation, which in all are very time-consuming. Thus, to achieve interactive efficiency, it requires sophisticated algorithms and unified parallel computation framework.

To tackle the aforementioned challenges, we focus on a unified CUDA-based framework to directly utilize clinical volumetric medical images for instant patient-specific simulation. Fig. 1 intuitively illustrates the pipeline of our framework. It contributes to the state of the knowledge in volumetric image based animation, physics-based modeling, and medical simulation. Specifically, the salient contributions can be summarized as follows:

(1) We systematically articulate a versatile framework for direct volumetric image manipulation while avoiding rebarbative manual geometry processing, which seamlessly covers the dominating processes of patient-specific images based surgical simulation.

(2) We generate an adaptive material-aware physical proxy from volumetric images to drive the nonlinear deformation and cutting simulation of soft tissues, and design a fitting based sparse displacement field up-sampling algorithm for dynamic volume rendering of deformed/cut anatomical structures.

(3) We synchronously integrate cutting simulation, collision detection, and collision response into a NFEM-based deformable model, and design a parallel algorithm for the explicit solving of the physical equations.

(4) We implement the framework within the parallel computing architecture of CUDA to guarantee the interactive efficiency. At the technical fronts, all the CUDA-based algorithms can contribute to other relevant applications.

2 Related work

Closely relevant to the central theme of this paper, we briefly review previous work in the following categories.

Volumetric Image Manipulation. Direct volumetric image manipulation has the advantage of faithfully preserving contents. Most of the earlier studies address this issue as geometry-based image morphing. Driven by the demand of arbitrary exploration of internal structures, some more complex geometric manipulation strategies were successively proposed. Masutani et al. [7] achieved interactive deformation plausible manipulation by simply controlling the vertices of textured proxy objects. Correa et al. [8] presented an efficient volume deformation method by moving the proxy points placed on the cross sections to desirable positions. However, such methods are only suitable for geometry based volume-editing deformation, and thus lack physical reality and efficiency. Most recently, Nakao et al. [2] extended the proxy points [8] to automatically-generated proxy tetrahedral mesh according to the original image curvature, and then they further refined the proxy mesh by taking adaptive optimization algorithm into account [1] in the pre-processing stage. However, it can't produce material-aware mesh elements, ignores the element topological changes due to cutting and collision.

Fem-based Deformation And Cutting. Most of real-time FEM approaches are based on linear formulations. To enable large deformation, Christian [9] extended the co-rotated strain formulation to hexahedral elements and achieved linear time complexity by introducing a multi-grid solver. However, its constitutive law remains linear. The TLED method proposed by Miller et al. [10] rigorously formulated the non-linear constitutive law, which was also employed as a deformable model to conduct non-rigid registration in [11]. By integrating the visco-hyper-elastic model, TLED was extended to handle anisotropic viscoelastic deformation [12]. Despite of the success of TLED in large deformation simulation, it has not been generalized to handle cutting simulation. Most recently, similar to diffusive regularization in image registration, Fortmeier et al. [13] proposed to conduct volumetric soft tissue deformation by relaxing ChainMail algorithm on the inverse of the displacement field; however, it can only simulate local deformations caused by needle insertion.

FEM-based cutting simulation is still challenge in dynamic topological update and realistic cutting surface visualization [14]. Finite element subdivision based method [15] is usually used to handle topological update, but it can create severely ill-conditioned simulation elements. Wicke et al. [16] improved it by using arbitrarily-convex finite elements, but it is even more involved in numerical integration. In sharp contrast, Molino et al. [17] proposed a virtual node algorithm to avoid element splitting by duplicating the cut elements and redistributing material components. However, since each fragment is required to contain at least one FEM node, arbitrary cutting is strictly limited. To avoid re-meshing, Jin et al. [18] propose an image based meshless algorithm for soft tissue cutting, but currently it can only afford 2D cutting simulation. Most recently, adaptive regular hexahedron approximation is used to simulate the cut in linear elastic deformable bodies [9], and Jerabkova et al. [14] achieved interactive FEM-based cutting simulation by removing the finest level voxels from multi-resolution volumetric images; however, both of them ignore the synchronized collision handling and realistic cutting surface visualization.

Brief Summary. From the view point of the physical fidelity and visual reality for direct image manipulation, although most of the current algorithms are competitive, but they are lonesome, and still require a trade-off between physical (or visual) plausibility and speed. Therefore, a unified and efficient framework that affords instant image utilization, physics based large-scale deformation, arbitrary cutting simulation, synchronized collision handling, realistic cutting surface texturing is urgently needed.

3 Framework overview

As shown in Fig. 1, we first focus on the framework overview as follows:

Multi-label tissue segmentation: It designs Volumetric Gaussian Mixture Model (V-GMM) algorithm to automatically conduct multi-label segmentation, and chooses the target tissues with little user interaction.

Physical proxy generation: In a user-transparent way, it succeeds to build a material-aware tetrahedral mesh according to the multi-label segmentation, which will serve as a proxy to encode the geometry structure and physical properties based on NFEM.

Physical simulation: In each simulation cycle, based on CUDA, it computes the internal and external forces according to user interaction and self-collision, updates the proxy structure and its accompanying properties such as displacement, velocity, position, etc.

Dynamic volume rendering: Based on CU-

DA, the cut/deformed volume is driven to evolve by resorting to super-resolution of the displacement field resulted from proxy structure and serves as a dynamic 3D texture for the interactive volume rendering.

In the following sections, we will describe the algorithms involved in our framework together with their CUDA implementation in detail.

4 Physical proxy generation

4.1 Multi-label segmentation of images

To analyze the physical structure of an organ and its surrounding tissues, we should first perform multi-label segmentation on the original volumetric image, which is represented by a stack of 2D image slices in 3D space, and where the material variations of tissues give rise to varying colors. To achieve the goal of instant utility of patient-specific 3D images, it is necessary to integrate an automatic image segmentation method in our integrated framework. To the best of our knowledge, Gaussian Mixture Model (GMM) serves this purpose very well for 2D images.

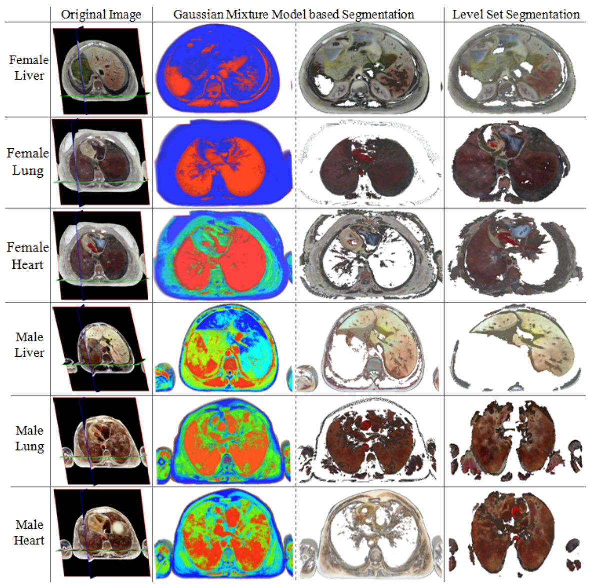

Figure 2: Multi-label segmentation.

Given patient-specific volumetric images, we extend GMM to V-GMM to conduct automatic multi-label segmentation. Then, we further pick the target organs by manually indicating the color-coded labels. It should be noted that, this is the only required manual input in our framework. As shown in Fig. 2, the first column shows the original volumetric images, the second column shows the multi-label segmentation results by using V-GMM, and the third column shows the segmentation results of the target tissues. Meanwhile, we also document the 3D Level Set based segmentation results in the fourth column for comparison. The results state that V-GMM method is more accurate than 3D level set for images with low color contrast such female heart and male heart. As a preprocessing step, our experiments state that V-GMM can usually obtain the volumetric segmentation results within tens of seconds.

4.2 Material-aware proxy generation

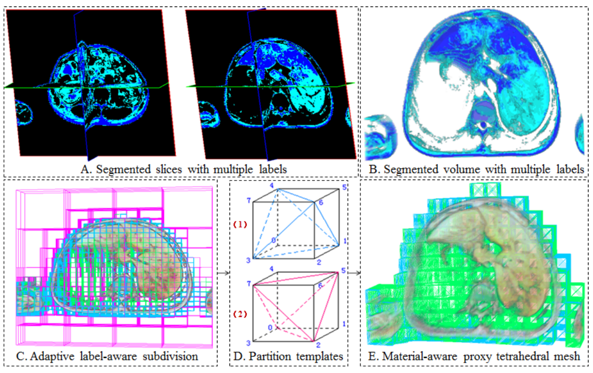
A. Segmented slices with multiple labels
B. Segmented volume with multiple labels
C. Adaptive label-aware subdivision
(1)
(2)
D. Partition templates
E. Material-aware proxy tetrahedral mesh

Figure 3: Material-aware proxy generation.

Based on the multi-label segmentation results (Fig. 3(A, B)), we continue to generate a proxy mesh to encode the physical topology and properties in a user transparent way. As shown in Fig. 3(C), we first hierarchically divide the working space by way of an adaptive octree. For any sub-cuboid at the current level, whether to perform further space partitioning is determined by the voxel colors contained in the sub-space. And the material type (encoded with different colors in Fig. 3(C)) of the sub-cuboid can be determined according to the voxel labels. We then decompose each leaf cuboid into five tetrahedra. To avoid edge intersection between neighboring tetrahedra, we define two different partitioning schemes (Fig. 3(D)). By taking the 3D adjacency relations of the leaf cuboid into account, we can easily determine the partitioning type of each cuboid. Fig. 3(E) demonstrates the generated material-aware tetrahedral mesh over a segmented liver dataset, where each tetrahedron will serve as a material-aware finite element for the subsequent physical modeling.

4.3 NFEM-based Physical Modeling

Given material parameters of the target tissue such as density ρ, Young's modulus E, and Poisson's ration ν, we first compute the mass of each tetrahedron according to its volume, and distribute each tetrahedron's mass to its four vertices. Meanwhile, we calculate the time step $\triangle t$ that can guarantee the stability of the explicit time integration by

$$\begin{cases} \triangle t = \frac{L_e}{c} \\ c = \sqrt{\frac{E(1-\nu)}{\rho(1+\nu)(1-2\nu)}} \end{cases}, \tag{1}$$

where L_e is the smallest edge length among the tetrahedra and c is defined as the dilatational wave speed of the material.

Similar to TLED, we measure the deformation at time t with respect to its un-deformed configuration at time 0, which is defined by deformation gradient tensor ${}^t_0\mathbf{X}$

$${}^t_0\mathbf{X}_{ij} = \frac{\partial^t x_i}{\partial^0 x_j}. \tag{2}$$

Then the second Piola-Kirchhoff stress ${}^t_0\mathbf{S}$ is employed to measure the stress, which is defined as

$${}^t_0\mathbf{S} = \frac{{}^0\rho}{{}^t\rho}({}^0_t\mathbf{X})({}^t\mathbf{T})({}^0_t\mathbf{X}^T), \tag{3}$$

where ${}^t\mathbf{T}$ represents the *Cauchy stress* per unit area in the deformed geometry, ${}^0_t\mathbf{X} = ({}^t_0\mathbf{X})^{-1}$, and the mass density ratio ${}^0\rho/{}^t\rho = det({}^t_0\mathbf{X})$.

With the help of the discrete tetrahedral proxy mesh, we use the shape functions to interpolate relevant physical quantities.

$$\begin{cases} h_1 = (1-r)(1-s)(1-t)/8 \\ h_2 = (1+r)(1-s)(1-t)/8 \\ h_3 = (1+s)(1-t)/4 \\ h_4 = (1+t)/2 \end{cases} \tag{4}$$

of which h_1, h_2, h_3, and h_4 are respectively the shape functions of the four vertices in each tetrahedron. Thus the stiffness matrix needs not to be

Algorithm 1: Physical proxy generation.

```
input  : segmented image $f_s$, $\rho$, $E$, $\nu$,
         and the maximal octree depth $n_d$.
output: void.

1: Create adaptive octree based on $f_s$,$n_d$;
2: Generate tetrahedral proxy mesh;
3: Compute $L_e$, $\triangle t$, and perform mass
distribution;
4: Allocate memory for $A_i$, $B_i$, $C_i$, ${}^t\mathbf{U}_i$,
${}^{t+\triangle t}\mathbf{U}_i$, and the shape function derivative
lists on GPU;
5: Invoke a CUDA kernel to precompute
$A_i$, $B_i$, $C_i$ ;
6: Invoke a CUDA kernel to precompute
the derivatives of shape functions;
```

dynamically assembled anymore, which can be directly computed at the element level by

$$K(U)U = \sum_e {}^t\mathbf{F}(e), \tag{5}$$

$${}^t\mathbf{F} = \int_{{}^0V} ({}^t_0\mathbf{B}_L^T)({}^t_0\hat{\mathbf{S}})d^0V, \tag{6}$$

where ${}^t_0\hat{\mathbf{S}}$ is the vector form of the *second Piola-Kirchhoff stress*, ${}^t_0\mathbf{B}_L$ is the strain-displacement matrix, whose i-th sub-matrix can be obtained by transforming ${}_0\mathbf{B}_{L0}$ using the deformation gradient ${}^t_0\mathbf{X}$ through${}^t_0\mathbf{B}_L^{(i)} = {}_0\mathbf{B}_{L0}^{(i)}{}^t_0\mathbf{X}^T$. And ${}_0\mathbf{B}_{L0}^{(i)}$ can be computed by means of the spatial derivatives of the shape functions:

$${}_0\mathbf{B}_{L0}^{(i)} = \begin{bmatrix} \frac{\partial h_i}{\partial^0 x} & 0 & 0 \\ 0 & \frac{\partial h_i}{\partial^0 y} & 0 \\ 0 & 0 & \frac{\partial h_i}{\partial^0 z} \\ \frac{\partial h_i}{\partial^0 y} & \frac{\partial h_i}{\partial^0 x} & 0 \\ 0 & \frac{\partial h_i}{\partial^0 z} & \frac{\partial h_i}{\partial^0 y} \\ \frac{\partial h_i}{\partial^0 z} & 0 & \frac{\partial h_i}{\partial^0 x} \end{bmatrix} \quad (i = 1,\ldots,4). \tag{7}$$

Suppose that all the nodal forces ${}^t\mathbf{F}$ and the external forces ${}^t\mathbf{R}$ have been obtained, we can further iteratively update the displacement for each vertex in the proxy mesh by

$${}^{t+\triangle t}\mathbf{U}_i = A_i({}^t\mathbf{R}_i - {}^t\mathbf{F}_i) + B_i{}^t\mathbf{U}_i + C_i{}^{t-\triangle t}\mathbf{U}_i. \tag{8}$$

And A_i, B_i, C_i can be pre-computed in a vertex-wise fashion as

$$\begin{cases} A_i = \frac{1}{\mathbf{D}_{ii}/(2\triangle t)+\mathbf{M}_{ii}/\triangle t^2} \\ B_i = \frac{2\mathbf{M}_{ii}/\triangle t^2}{\mathbf{D}_{ii}/(2\triangle t)+\mathbf{M}_{ii}/\triangle t^2} = \frac{2\mathbf{M}_{ii}}{\triangle t^2}A_i \\ C_i = \frac{\mathbf{D}_{ii}/2\triangle t-\mathbf{M}_{ii}/\triangle t^2}{\mathbf{D}_{ii}/(2\triangle t)+\mathbf{M}_{ii}/\triangle t^2} = \frac{\mathbf{D}_{ii}}{\triangle t}A_i - \frac{B_i}{2} \end{cases} \tag{9}$$

To sum up, the pseudo-code of physical proxy generation is documented in Algorithm 1.

5 CUDA based simulation

5.1 *Deformation*

Since the external force ${}^t\mathbf{R}_i$ can be directly obtained, at each time step we only need to recompute ${}^t\mathbf{F}_i$ according to Eq.(6). In fact, ${}^t\mathbf{F}_i$ transitively depends on ${}^t_0\hat{\mathbf{S}}$ and ${}^t_0\mathbf{X}$. For ${}^t_0\hat{\mathbf{S}}$, we compute it using a *hyperelastic neo-Hookean* model given by

$${}^t_0\hat{\mathbf{S}} = \mu(\delta_{ij} - {}^t_0\mathbf{C}^{-1}_{ij}) + \lambda({}^tJ)({}^tJ-1)({}^t_0\mathbf{C}^{-1}_{ij}), \tag{10}$$

where μ and λ are the *Lamé* constants, δ_{ij} is the *Kronecker's delta*. As for ${}^t_0\mathbf{X}$, it can be computed by way of shape-function-based interpolation:

$${}^t_0\mathbf{X} = {}^t\mathbf{u}_e^T \partial\mathbf{H} + \mathbf{I}, \tag{11}$$

${}^t\mathbf{u}_e$ is a 4×3 matrix consisting of the vertex displacements of each tetrahedron, $\partial\mathbf{H}$ is a 4×3 matrix consisting of the pre-computed shape function.

Then, by summing all the corresponding forces in the tetrahedra that share vertex i, we can finally get the internal force ${}^t\mathbf{F}_i$ for the vertex i in the proxy mesh. And the vertex-wise displacement updating is well suitable for CUDA-based parallel computation, which is detailed in Algorithm 2.

5.2 *Cutting and collision*

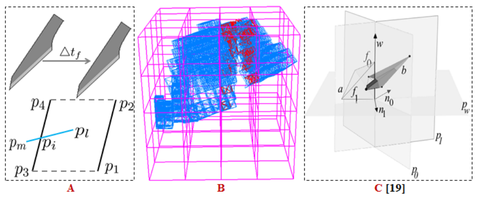

Figure 4: Cutting and collision handling.

Algorithm 2: CUDA based simulation.

```
input : user interaction.
output: void.

for each cycle of the simulation loop do
    if cutting mode is switched on then
        1:Invoke a CUDA kernel for collision detection;
        2:Invoke a CUDA kernel to update ${}^0V, m, A_i, B_i, C_i$;
    for i=1:iteration number do
        1:Compute ${}^t\mathbf{R}_i$;
        2:Invoke a CUDA kernel for ${}^t\mathbf{F}_i$;
        3:Invoke a CUDA kernel for ${}^t\mathbf{U}_i$;
    end
    1:Invoke a CUDA kernel to construct grid cell indices;
    2:Invoke a CUDA kernel to collect the nearby element indices and assign them to grid cells;
    3:Invoke a CUDA kernel to conduct collision detection and label the affected vertices to be constrained;
    4:Invoke a CUDA kernel to add constraints to ${}^t\mathbf{F}_i$.
end
```

As shown in Fig. 4(A), during the cutting procedure, since the inter-frame time interval $\triangle t_f$ is usually very short, the moving direction of the operating tool (e.g., knife) can be considered to be constant, and the surface swept by the knife can be approximated as a plane with a parallelogram $\diamondsuit_{p_1p_2p_3p_4}$, where the edges p_1p_2, p_3p_4 respectively represent the current and the last positions of the knife. Thus, we can easily determine the number of cut edges in each tetrahedral element, and then we extend the tetrahedra topological updating method [15] on CUDA to refine the cut tetrahedral mesh. Furthermore, according to the refined mesh, we need to update the volume ${}^0V$, the mass distribution, A_i, B_i, C_i, and the shape function derivatives for all the affected tetrahedra. Meanwhile, the integral time step $\triangle t$ should also be updated according to the new smallest edge length L_e. Since the cut process can be handled per tetrahedron, its CUDA-based parallel computation is natural.

As shown in Fig. 4(B), we employ a regular grid to assist CUDA-based collision detection. In each cycle of simulation loop, to rapidly exclude a large majority of tetrahedron-pairs that are impossible to collide with each other in advance, we first assign each deformed tetrahedron to some grid cells according to the current positions of its four vertices. As shown in Fig. 4(C), we extend the fast tetrahedron overlap testing algorithm [19] to CUDA, whose central idea is: let e be an edge shared by faces f_0, f_1, p_0, p_1 be half-planes extending those faces, and W be the resulting wedge containing the tetrahedron, then if the other tetrahedra intersect with W there can not be a separating plane containing e. Thus, taking the tetrahedra allocated in the same grid cells as a group, we can parallelly conduct collision detection in an element-wise fashion. The collided tetrahedra are shown in red in Fig. 4(B). If a tetrahedron collides with others, we add a punishment force to its four vertices in the opposite direction of their own movement. And the CUDA-based implementtation of cutting and collision handling is documented in Algorithm 2.

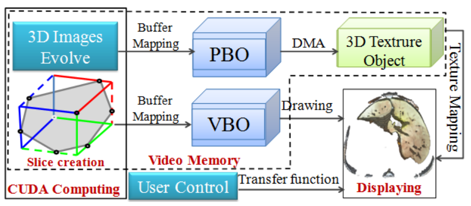

Figure 5: GPU data flow of rendering pipeline.

Algorithm 3: CUDA based rendering.

```
input  : slice number $n_s$, 3D texture
         cache $tex$, transfer function $T_f$.
output: visualization.

for each cycle of the simulation loop do
    if the first cycle then
        1:Initiate a $pbo$;
        2:Initiate a $vbo$ according to $n_s$;
        3:Allocate memory for primitive
        vertex array $A_r$;
    1:Update $T_f$ with user control;
    2:Invoke a CUDA kernel to
    up-sample ${}^t\mathbf{U}$;
    3:Map $pbo$ to deformed image $I$;
    4:Invoke a CUDA kernel to compute
    $I(p)$;
    5:Transfer data from $pbo$ to $tex$;
    6:Map $vbo$ to $A_r$;
    7:Invoke a CUDA kernel to generate
    $A_r$;
    8:Invoke the rendering function;
end
```

5.3 *Dynamic volume rendering*

So far, in each simulation cycle, we can get the sparse vertex displacement of the proxy mesh. We first up-sample the sparse displacement field at each voxel position p of the evolving dense field. Similar to collision handling, the regular grid can also be used to assist the fitting-based up-sampling. Given the second-order monomial basis $\mathbf{b}$, we locally fit a polynomial function around each cell p by minimizing the following energy function:

$$\sum_{v \in N(p)} (\mathbf{b}^T \cdot \mathbf{c} - {}^t\mathbf{U}(v))^2 \omega_p(v), \qquad (12)$$

$N(p)$ represents the vertex set in the supporting domain of the spatial Gaussian kernel function ω_p. Then, the evolved voxel color should be further indexed based on its deformed position from the original volumetric images.

Although all the runtime computation are executed on CUDA, in each cycle, conventionally it needs to transfer the evolved 3D images to host memory first and then copy them to GPU texture cache through graphics API, where the greatest bottleneck is the bandwidth and texture cache coherency. As shown in Fig. 5, our strategy is to employ a pixel buffer object (PBO) as a bridge to transfer image data from CUDA buffer to texture cache, which can perform fast pixel data transfer through direct memory access (DMA) without involving any CPU cycle.

Besides, since image-order approaches such as GPU-based ray casting, usually need many iterations to fetch 3D texture and blend the color for a single pixel, it cannot be afforded here due to the already-existed expensive NFEM simulation. Thus, 3D texture is not enough, we should create object-order primitives by decomposing the volume into slice stacks of textured polygons, and we always create a 6-vertex polygon for each slice in order to improve the uniformity of

CUDA-based data structure. Then, the generated polygon data is directly mapped to a vertex buffer object (VBO) from CUDA buffer, which can drastically increase the rendering efficiency. Finally, with the transfer function controlled by the user, we can achieve the interactive volume rendering during physics based image manipulation, and CUDA-based rendering implementation is detailed in Algorithm 3.

6 Experimental Results

We implement a prototype system using C++ and CUDA. All the experiments run on a laptop with NVIDIA GeForce 330M GPU, Intel Core (TM) i7 CPU (1.6GHz, 4 cores) and 4G RAM, where all the volumetric images are from our cooperative hospitals.

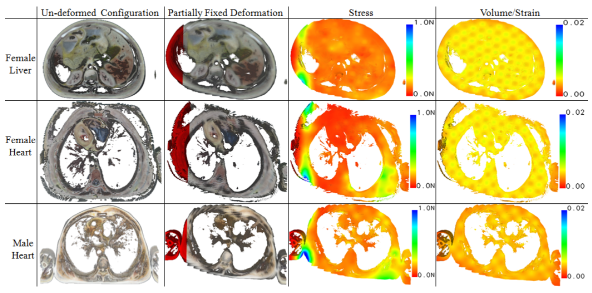

Figure 6: Partially fixed deformation.

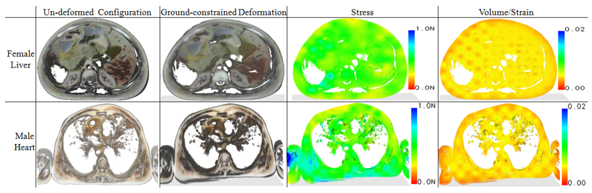

Figure 7: Deformation due to free falls.

Fig. 6 shows the deformation results of partially fixed organs (marked in red) under the influence of gravity. Here and in all the following experiments, we set the density parameter to $1000kg/m^3$, and the parameters μ and λ are respectively set to be 400 and 49000. It shows that our method can support large deformation while keeping accurate anatomical structures during deformation. Since strain in nature is a tensor, we employ $volume/strain$ to measure the deformation change of each element as

$$M_e = \frac{V_e}{{}^tJ} = \frac{V_e}{det({}^t_0\mathbf{X})}, \qquad (13)$$

M_e describes each element's mass change ratio with respect to its un-deformed configuration. We scale each tissue into a cube of 8 cubic dm, the volume of each element is very small and thus the value of M_e correspondingly appears in a small scale. The measurement results are color-coded in the fourth column, while the stress distribution is shown in the third column. It can be seen that tissues nearby the fixed parts usually have bigger stress, which coincides well with the fact. Besides, Fig. 7 shows the deformation results under the influences of gravity and ground constraints. Due to the collision with the constrained plane, the stress should spread quickly and cause large deformation. The deformed 3D images correctly present such visual effects.

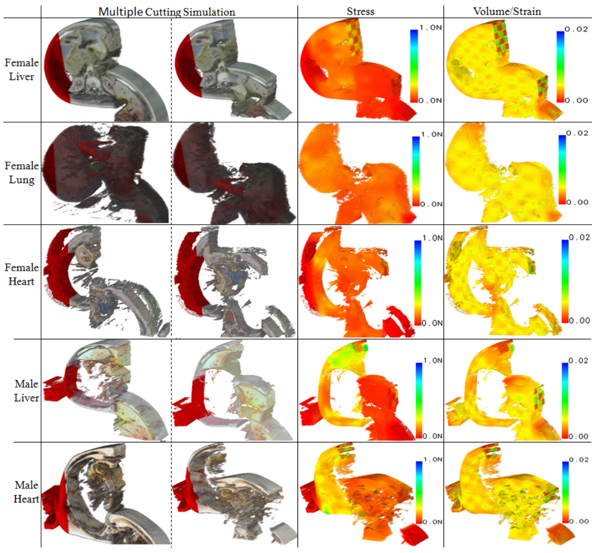

Figure 8: Partially fixed Cutting simulation.

Fig. 8 demonstrate the cutting simulation of partially fixed tissues, where the cut parts will have a free fall due to gravity. The results state that our method can accurately demonstrate the patient-specific anatomical structures on the deformed cutting surfaces, which are currently hard to achieve in most of the existing surgery simulators. Meanwhile, it shows that our method is stable and robust to sustain arbitrary multiple cuts. During the cutting process, since the cut parts are not rigid bodies and tend to undergo deformation, collision may occur and induce local deformation nearby the cut surfaces. The integration of collision handling can effectively simulate these cases, which can be seen from the color-coded quantitative element-wise stress and strain measurements. In the interest of space, for more vivid results, please refer to our supplementary video.

Table 1: Time performance(in millisecond)

Images	NFEM#	PG	FEM	TU	CH	DVR
M-liver	6775	578	14	14	1	48
M-heart	7515	577	15	16	2	49
F-liver	11940	890	17	17	2	74
F-lung	10095	795	17	17	2	71
F-heart	8630	655	15	17	2	53

To fully analyze the time performance of our method, Table 1 documents the time testing for most of the dominating steps, including proxy generation (PG), NFEM simulation (N-FEM), topological updating (TU), collision handling (CH), and dynamic volume rendering (D-VR), wherein PG is only executed in the preprocessing stage. It shows that DVR is a little time-consuming. However, even on a common laptop, we can still achieve interactive performance. Therefore, with the help of more powerful computer or multi-GPU scheme, our framework has great potentials to afford even more finite elements and achieve higher physical fidelity, real-time performance efficiency.

7 Conclusion and Discussion

We have detailed a comprehensive and fully CUDA-accelerated volumetric image manipulation framework to address a suite of research challenges in image based interactive surgical simulation. Our framework supports instant utilization of patient-specific clinical images, physics based large-scale non-linear deformation, arbitrary cutting simulation, synchronized collision handling, and realistic cutting surface visualization, which collectively have a broader appeal to both image based animation and physics based simulation. Extensive experiments on various medical images together with their quantitative physical behavior measurements demonstrated its superior performance.

Meanwhile, although the paper's current foci are on the seamless image manipulation framework for physically plausible soft tissue deformation and cutting simulation, our technical vision is to achieve material-aware and even physiological property coupled surgical simulation directly over clinical images in the near future. However, there are still some open problems yet to be solved. First, it is difficult to accurately capture and register material parameters of heterogeneous tissues. Second, given volumetric images, it remains hard to automatically get accurate multi-label segmentation. Besides, our immediate efforts are geared towards incorporating blooding and suturing simulation into our current framework, and then propelling its demonstrated applications in clinical setting.

References

[1] K. W. C. Hung, M. Nakao, K. Yoshimura, and K. Minato. Background-incorporated volumetric model for patient-specific surgical simulation: A segmentation-free, modeling-free framework. *International Journal of Computer Assisted Radiology and Surgery*, 6(1):35–45, 2011.

[2] M. Nakao, K. W. C. Hung, S. Yano, K. Yoshimura, and K. Minato. Adaptive proxy geometry for direct volume manipulation. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 161–168, 2010.

[3] R. M. Satava. Historical review of surgical simulation: A personal perspective. *World Journal of Surgery*, 32(2):141–148, 2008.

[4] A. Plass, H. Scheffel, H. Alkadhi, P. Kaufmann, M. Genoni, F. Volkmar, and J. Grüenfelder. Aortic valve replacement through a minimally invasive approach: Preoperative planning, surgical technique, and outcome. *Annals of Thoracic Surgery*, 88(6):1851–1856, 2009.

[5] K. Burger, J. Kruger, and R. Westermann. Direct volume editing. *IEEE Trans. on Visualization and Computer Graphics*, 14(6):1388–1395, 2008.

[6] Z. A. Taylor, M. Cheng, and S. Ourselin. High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. *IEEE Transactions on Medical Imaging*, 27(5):650–663, 2008.

[7] Y. Masutani, Y. Inoue, F. Kimura, and I. Sakuma. Development of surgical simulator based on fem and deformable volume rendering. In *Proceedings of the SPIE*, pages 500–507, 2004.

[8] C. D. Correa, D. Silver, and M. Chen. Volume deformation via scattered data interpolation. In *Proceedings of IEEE VGTC Workshop on Volume Graphics*, pages 91–98, 2007.

[9] C. Dick, J. Georgii, and R. Westermann. A hexahedral multigrid approach for simulating cuts in deformable objects. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1663–1675, 2011.

[10] K. S. Miller, G. R. Joldes, D. Lance, and A. Wittek. Total lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. *Communications in Numerical Methods in Engineering*, 23(2):121–134, 2007.

[11] K. O. Noe and T.S. Sorensen. Solid mesh registration for radiotherapy treatment planning. *Lecture Notes in Computer Science*, 5958:59–70, 2010.

[12] Z. A. Taylor, O. Comas, M. Cheng, J. Passenger, D. J. Hawkes, D. Atkinson, and S. Ourselin. On modelling of anisotropic viscoelasticity for soft tissue simulation: Numerical solution and gpu execution. *Medical Image Analysis*, 13(2):234–244, 2009.

[13] D. Fortmeier, A. Mastmeyer, and H. Handels. Gpu-based visualization of deformable volumetric soft-tissue for real-time simulation of haptic needle insertion. *Bildverarbeitung fr die Medizin 2012*, pages 117–122, 2012.

[14] L. Jerabkova, G. Bousquet, S. Barbier, F. Faure, and J. Allard. Volumetric modeling and interactive cutting of deformable bodies. *Progress of Biophysics and Molecular Biology*, 103(2-3):217–224, 2010.

[15] C. Forest, H. Delingette, and N. Ayache. Cutting simulation of manifold volumetric meshes. In *Proceedings of Medical Image Computing and Computer-Assisted Intervention*, pages 235–244, 2002.

[16] M. Wicke, M. Botsch, and M. Gross. A finite element method on convex polyhedra. *Comput. Graph. Forum*, 26(3):355–364, 2007.

[17] E. Sifakis, K. G. Der, and R. Fedkiw. Arbitrary cutting of deformable tetrahedralized objects. In *Proceedings of Eurographics Symposium on Computer Animation*, pages 73–80, 2007.

[18] X. Jin, G. R. Joldes, K. Miller, K.H. Yang, and A. Wittek. Meshless algorithm for soft tissue cutting in surgical simulation. *Computer Methods in Biomechanics and Biomedical Engineering*, PMID: 22974246, 2012.

[19] F. Ganovelli, F. Ponchio, and C. Rocchini. Fast tetrahedron-tetrahedron overlap algorithm. *ACM Journal of Graphics Tools*, 7(2):17–26, 2002.