Summer 2014

Meshless Mechanics and Point-Based Visualization Methods for Surgical Simulations

Rifat Aras
Old Dominion University

MESHLESS MECHANICS AND POINT-BASED VISUALIZATION METHODS FOR SURGICAL SIMULATIONS

by

Rifat Aras
B.S. May 2005, Bilkent University, Turkey
M.Sc. December 2008, Bilkent University, Turkey

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

MODELING AND SIMULATION

OLD DOMINION UNIVERSITY
August 2014

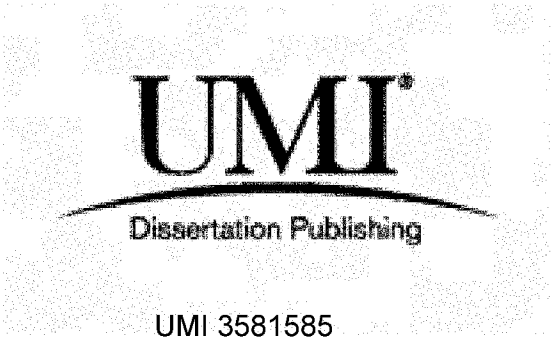
Approved by:

Yuzhong Shen (Director)

Michel Audette (Member)

Frederic McKenzie (Member)

Duc T. Nguyen (Member)



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

MESHLESS MECHANICS AND POINT-BASED VISUALIZATION METHODS FOR SURGICAL SIMULATIONS

Rifat Aras
Old Dominion University, 2014
Director: Yuzhong Shen

Computer-based modeling and simulation practices have become an integral part of the medical education field. For surgical simulation applications, realistic constitutive modeling of soft tissue is considered to be one of the most challenging aspects of the problem, because biomechanical soft-tissue models need to reflect the correct elastic response, have to be efficient in order to run at interactive simulation rates, and be able to support operations such as cuts and sutures.

Mesh-based solutions, where the connections between the individual degrees of freedom (DoF) are defined explicitly, have been the traditional choice to approach these problems. However, when the problem under investigation contains a discontinuity that disrupts the connectivity between the DoFs, the underlying mesh structure has to be reconfigured in order to handle the newly introduced discontinuity correctly. This reconfiguration for mesh-based techniques is typically called dynamic remeshing, and most of the time it causes the performance bottleneck in the simulation.

In this dissertation, the efficiency of point-based meshless methods is investigated for both constitutive modeling of elastic soft tissues and visualization of simulation objects, where arbitrary discontinuities/cuts are applied to the objects in the context of surgical simulation. The point-based deformable object modeling problem is examined in three functional aspects: modeling continuous elastic deformations with, handling

discontinuities in, and visualizing a point-based object. Algorithmic and implementation details of the presented techniques are discussed in the dissertation. The presented point-based techniques are implemented as separate components and integrated into the open-source software framework SOFA.

The presented meshless continuum mechanics model of elastic tissue were verified by comparing it to the Hertzian non-adhesive frictionless contact theory. Virtual experiments were setup with a point-based deformable block and a rigid indenter, and force-displacement curves obtained from the virtual experiments were compared to the theoretical solutions.

The meshless mechanics model of soft tissue and the integrated novel discontinuity treatment technique discussed in this dissertation allows handling cuts of arbitrary shape. The implemented enrichment technique not only modifies the internal mechanics of the soft tissue model, but also updates the point-based visual representation in an efficient way preventing the use of costly dynamic remeshing operations.

This dissertation is dedicated to my family
for their endless support and their faith in me.

ACKNOWLEDGMENTS

Over the course of my doctoral studies, I have received support and encouragement from a great number of individuals. Dr. Yuzhong Shen has been a mentor and a guide. His guidance helped me a lot in this journey. Another individual I would like to give credit is Dr. Ahmed Noor, as he was an inspiration both intellectually and spiritually. I would like to express my gratitude to my committee of Dr. Michel Audette, Dr. Frederic McKenzie, and Dr. Duc T. Nguyen for their support and helpful feedback over the years.

In addition, I would like to thank my friends and colleagues, who were with me throughout my curricular and extra-curricular activities.

Finally, I would like to give my deepest thanks to my family and my soon-to-be family for their endless support and their faith. You are the real MVPs!

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

Medical education involves the concept of apprenticeship [1], where novices directly learn from experienced doctors and their interactions with or operations on the patients in hospitals to increase their level of expertise. Although classical apprenticeship program constitutes the basis of the medical education field, computer-based modeling and simulation practices have begun to make an impact as well. Compared to other modeling and simulation (M&S) domains though, researchers are still facing several challenges in computer-based medical M&S. For example, flight simulators have reached a point where military and commercial pilots can be easily trained to fly a new aircraft as flight simulators can provide experiences that are almost indistinguishable from the reality. For medical simulators however, we are still far away from this level as stated by Loftin [2] as it is quite difficult to capture and reflect correct mechanical properties of soft tissue. Measuring the mechanical properties has several parameters on its own: in vivo vs. ex vivo, tissue temperature, amount of blood circulation and storage conditions are some of these. Recently, successful medical simulators that are targeted for very specialized procedures are seen in the field [3], however a general solution is yet to be found due to the aforementioned wide array of parameters[1].

Medical modeling and simulation features both anatomical models and therapy models, both of which represent significant challenges. According to Ota et al. [4], the greatest challenge in building complex therapy models is to capture the accurate response of soft-tissue. For surgical simulator applications, biomechanical models of human soft-tissues have to be accurate, efficient enough to be computed in real time, and able to handle topology altering operations

[1] IEEE Transactions and Journals Style is used in this dissertation for formatting figures, tables, and references.

such as cuts and sutures [5]. These requirements lead to simulation interactivity vs. model fidelity trade-offs that need to be examined in the context in which they are applied. Some attempts to address these requirements are using mass-spring networks as an alternative to finite element methods [6] and incorporating models of linear materials instead of more complex non-linear materials.

1.1 Problem Statement

Deformable modeling of soft tissue is a continuum elasticity problem, whose numeric solution involves discretization of the continuous domain into discrete elements. Numerous non-physical and physically-based models have been utilized in order to approximate this solution. Free-form deformation lattices, mass-spring networks, and finite element models (FEM) that are composed of tetrahedral/hexahedral elements are all examples of such models. The discretized models result in systems with many degrees of freedom (DoFs) that essentially define the total kinematic state of the modeled object. The aforementioned model examples have one property in common, they all define the connectivity information between the DoFs explicitly. When there is a situation that disrupts this connectivity, such as an introduced discontinuity in the form of a cut, the discretization of the continuum needs to be redefined to handle the changes in the connectivity. Various approaches have been proposed to handle these changes caused by cuts.

Courtecuisse et al. [7] presented an FEM-based soft tissue deformation methodology that also supports real-time virtual cutting. In the presence of a cut, the topology of the finite elements that build up the simulation object changes along with the simulation-specific matrices. The topology changes in this work were encoded in three types of topology operations: element

removal, element subdivision, and element addition. This work benefitted from a GPU-based parallel implementation in order to ensure interactive operation rates.

In the work of Wu et al. [8], the authors discretized the simulation object by using a semi-regular hexahedral finite element grid. The volume was partitioned using an octree, and the face-adjacent cells of the octree were linked together. The advantage of this discretization is the ability to update the topology of the elements in an efficient way, when a cut is being introduced to the simulation domain by marking the links between the affected elements as disconnected. The octree was refined dynamically along the cut surface in order to retain fine detailed cuts. The authors employed several approximations of the deformable model, such as the concept of Composite Finite Elements (CFEs), in which smaller neighboring hexahedral elements are grouped together to form larger elements, thus decreasing the number of DoFs significantly. With this CFE-based approximation and a multigrid implicit solver, the authors were able to achieve simulation rates of 15 frames per second during the cutting operation.

Wu et al. [8] visualized the simulation object through a Dual Contouring-based surface extraction algorithm, where the dual grid was defined by the element-connecting links, and the optimal surface vertices were placed using the cut information such as the intersection point of the cut with the link and the normal of the cut surface. The Dual Contouring algorithm is discussed in detail in Chapter 4.

Steinemann et al. [9] followed a different approach to visualize split deformable objects. Instead of extracting the iso-surface of the object and triangulating the surface with a grid structure, the authors proposed to represent the surface of the object explicitly with an unstructured triangle mesh. When a cut is applied to the deformable object, the topology of the triangular surface is updated by a series of topological tests and operations. The intersection

points between the cut surface defined by the propagating blade and the triangle surface of the object being cut were detected and these intersection points were used to triangulate the newly cut surfaces of the object. With a physical model with fewer DoFs and a coarse triangular surface, this work was able to achieve simulation rates of about 21 frames per second.

The cuts that modify the elastic response of the deformable objects are called strong discontinuities in the continuum elasticity mechanics field, and one way to treat them is through the so-called enrichment functions, which themselves are also discontinuous functions. Enrichment functions have been introduced to the classical FEM approach by Moës et al. [10] and incorporated into several studies afterwards. Kaufmann et al. [11] presented a method that employed harmonic enrichment functions for simulating detailed cutting of thin shells. These functions were stored in 2D texture elements in higher resolutions than the underlying finite element discretization, and computed by solving a Laplace equation that is subject to appropriate boundary conditions. Although being a 2D enrichment technique, the time-consuming solution of the Laplace equation prevented this work to achieve interactive rates.

In contrast with the previous approach, Barbieri et al. [12] proposed an analytical enrichment function in 2D that is easy to compute and extend into 3D. Designed for elastic isotropic deformable bodies modeled with meshless methods, this technique handled multiple cut segments by simply multiplying their enrichment functions consecutively. Although the multiplicative application is easy to implement, it can result in incorrect modification of meshless weight functions, therefore decreasing the stability of the simulation.

1.2 Aims and Objectives

The thesis in this dissertation is: *Using meshless mechanics for representing the elastic response and point-based visualization method for the visual aspect of a deformable object facilitates the efficient handling of cuts in a surgical simulation setting.*

In the context of surgical simulations, a deformable object has a minimum of two representations, one that represents the elastic response of the object, and another that represents the visual aspect of the object. When these representations are modeled with methods that are composed of tightly coupled elements, such as finite elements for elastic response and explicit triangle meshes for the visualization, an introduced discontinuity such as a cut is typically challenging to handle as it requires first to de-couple the affected elements, followed by establishing new connections, and updating simulation related values. Therefore, deformable object models and accompanying visual models that do not define an explicit connectivity between the DoFs would be more advantageous in terms of efficiency for handling strong discontinuities such as cuts.

In light of the problems states above, the objectives and contributions of the work contained in this dissertation are:

1. To develop a point-based meshless mechanics framework for modeling the behavior of the deformable objects and investigate the feasibility of such a framework for interactive operations.
2. To build a point-based visualization approach without the need of explicit connectivity information of the vertices.

3. To create a methodology to implement a cutting simulation that correctly modifies the tissue topology to update the elastic response as well as the visualization of the evolving point-based tissue model.
4. To provide a complete suite of point-based methods that can be integrated into a flexible software framework and used seamlessly along with the other components of the framework.

1.3 Dissertation Outline

The work contained in this dissertation is presented in seven chapters. In Chapter 2, the theoretical foundation for point-based modeling of deformable behavior is presented. Details of the modeling approach such as point-based discretization of the simulation object and approximation of simulation variables through Moving Least Squares-based reconstruction are discussed. Chapter 3, after discussing several previous approaches to treat discontinuities in point-based/meshless methods, describes the developed approach first in 2D, and then extends it into 3D. In Chapter 4, previous techniques for rendering objects represented with point primitives are discussed. This discussion is followed by the detailed description of the adopted point-based rendering pipeline. This pipeline is constructed from ground-up, where each step and the resulting improvement on the visual quality are shown in details. Chapter 5 describes the software environment into which the point-based techniques are being integrated. The algorithmic and implementation details of the presented techniques along with the Hertzian non-adhesive contact-based verification methodology are also discussed in this chapter. In Chapter 6, the results for the point-based object representation with several examples of cutting operation are shown, which is followed by Chapter 7, where conclusions and future work are presented.

CHAPTER 2

POINT-BASED MESHLESS MECHANICS FOR DEFORMABLE OBJECT MODELING

The ability to model and simulate the manipulation of deformable objects is essential to many application areas. In order to fulfill the requirements of different use-cases, deformable object modeling has been studied across a range of paradigms. This chapter will first present several previous approaches related to this field, then a brief theoretical foundation about continuum mechanics will be laid, followed by discussions of point-based discretization, Moving Least Squares-based approximation of field variables, and meshless mechanics-based modeling of deformable objects.

2.1 Overview

Approaches for modeling object deformation range from non-physical methods to physically realistic methods based on continuum mechanics. The former category of methods makes use of one or more control points or shape parameter values. These are typically adjusted in a user-friendly manner to obtain the desired deformation of the object. Physically-based methods on the other hand, account for the effects of external and internal forces, material properties, and environmental boundary conditions on object deformation.

The early work of Sederberg and Parry [13] presented a technique for deforming solid geometric models in an intuitive free-form manner. In this work, the deformations were based on interpolating trivariate Bernstein polynomials, and could be applied either globally or locally with volume preservation by encapsulating the target object in a lattice of control points. The control points of the enclosing lattice were manipulated in an intuitive way to achieve the desired deformation of the object (Fig. 1).

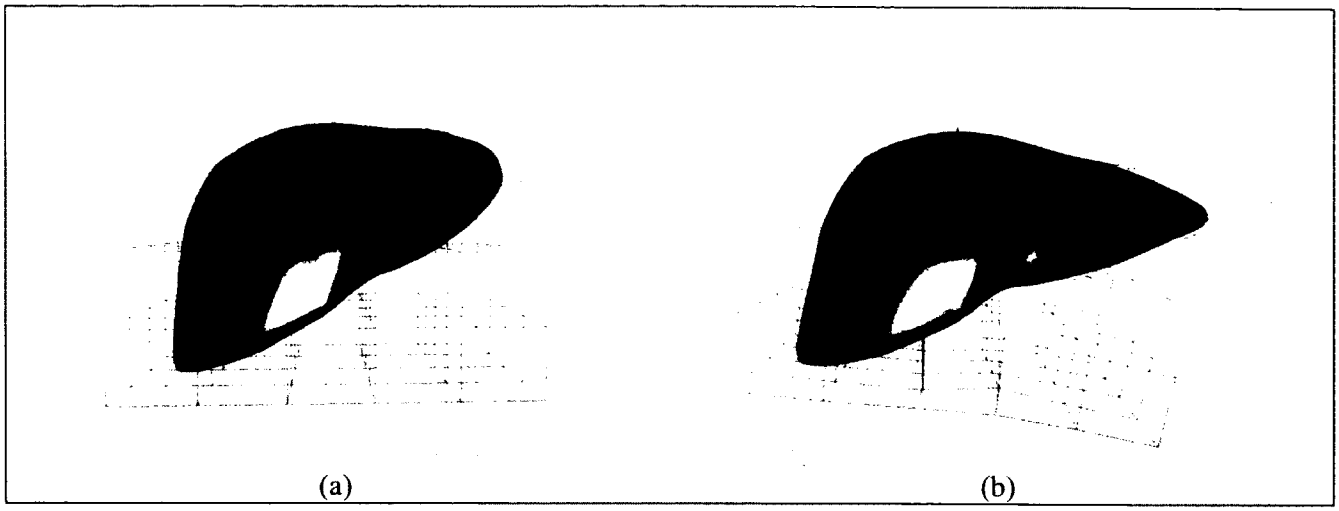


Fig. 1. In free-form deformation techniques, the object is enclosed in a lattice of control points. These control points (a) are manipulated freely to apply the desired deformation to the object (b).

Another type of non-physical methods is Shape Matching/Position-Based methods. These are geometrically motivated, and as opposed to physically-based methods, they resolve the dynamics of a deformable object through geometric constraints and distances from current to target positions instead of energies and forces. In the work of Müller et al. [14], an object, which is composed of individual particles, kept track of two configurations: the original shape and the current shape. At each time step, target positions for each particle were calculated by matching the original shape of the object to the current shape. Then, inter-point distances between the corresponding particle locations of the matched shape and the current shape were used to pull the particles towards their target positions (Fig. 2). Position-based methods are not as accurate as physically-based methods, but they provide visually plausible results, making them a good choice for virtual reality applications and games. Similarly, the work of Frisken-Gibson [15] modified the traditional voxel-based representation of the objects and proposed a linked volume representation that was capable of handling interactive object manipulations such as carving, cutting, tearing, and joining, but still unable to produce physically realistic results.

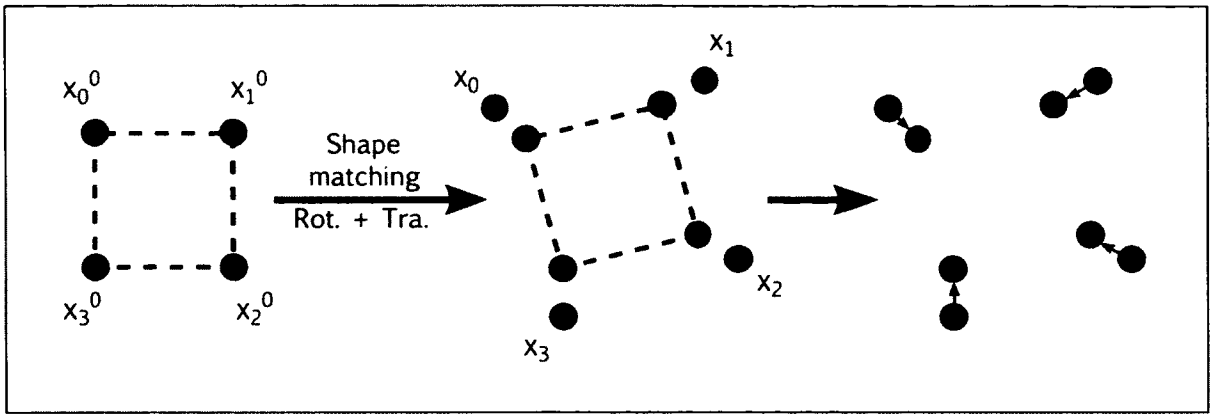


Fig. 2. Shape matching methods first match the original shape of the object to its deformed configuration by defining a rigid-body transformation. Next, the points at deformed configuration are driven towards their homologs in the rigidly transformed matched shape.

Free-form deformation and the Shape Matching technique are approximating and simple methods for deforming solid objects. Although they produce plausible results for certain application areas, the motion of their DoFs are not controlled by constitutive equations, and this is grounds for excluding them as options for most of the realistic simulations with medical focus.

An alternative to non-physical approximations is to use mass-spring models [16] and membrane based approximations that utilize spring networks [17]. A mass-spring network is composed of vertices and edges, in which each edge is realized as a spring that connects vertices pair-wise, and each vertex is idealized as a point mass. Although these constructs employ physical equations like Hooke's law, it is difficult to reproduce specific elastic material properties even with very careful distribution of spring stiffness throughout the network. Moreover, mass-spring networks are stiff systems, meaning the numerical solutions to these systems fluctuate rapidly after each time step, whereas the exact solutions vary slowly. Therefore, mass-spring networks are prone to suffer from poor numerical stability, unless they are simulated with small-enough time steps.

A continuum model typically relies on an underlying mesh structure either in 2D or 3D depending on the nature and the requirements of the problem. A breadth-first classification of mesh-based continuum models includes mass-spring networks [18], finite element methods [19], finite volume methods [20], and finite difference methods [21]. Among these, the finite element method has received particular interest in the biomechanical modeling community.

The early work of Bro-Nielsen discussed a fast adaptation of finite element modeling to satisfy speed and robustness requirements in a surgical simulation setting [22]. The body part was modeled as a 3D linear elastic solid that consisted of particles, which were deformed into a new shape when forces were applied to the elastic solid. In this framework, the author incorporated a technique called condensation. In the finite element modeling context, condensation translates into obtaining a more compact version of the system model by rearranging or eliminating terms of the matrix equations by simplifying a volume into a system of boundary elements. For example, for a single element, the displacement degrees of freedom at a node in the internal region of the element can be condensed out because they are not used in the inter-element continuity definition [19]. At a macro level, for a volumetric finite element system, masses of the internal nodes can be lumped to the surface nodes, and the equations can be arranged accordingly to only consider finite element nodes on the surface of the model. Accuracy of the condensation procedure largely depends on the redistribution quality of the masses, in case of a non-optimal distribution, solution accuracy can be adversely affected [19]. Moreover, this type of simplification is generally incompatible with a cutting simulation.

A number of recent techniques have addressed the fidelity versus efficiency trade-off. One important approach in the area is the finite element model based on Total Lagrangian Explicit Dynamics (TLED) by Miller et al. [23]. The difference between the TLED based finite

element model and other approaches is the former's use of the original reference configuration of the object to calculate the stress and strain tensors during a simulation step. As a result of expressing computations in the reference coordinates, the authors were able to pre-compute spatial derivatives. The pre-computation of the spatial derivatives leads to efficiency in terms of computational resources, while being capable of handling geometric and material non-linearities. The authors employed central differences-based explicit integration rather than the implicit integration scheme. With this choice, they were able to avoid solving the set of non-linear algebraic equations that are required by the implicit integration at each time step. However, the use of explicit integration brings limitation on the time step size in order to ensure the stability of the system. The authors justified their implementation choice by stating that the relatively lower stiffness (Young's modulus) value of the soft tissue relaxes the time step limitation considerably compared to the typical simulations involving more stiff material like steel or concrete.

Another attempt to increase the computational efficiency of the elastic model in the context of interactive simulation was discussed in the method proposed by Marchesseau et al. [24]. The authors presented a new discretization method called Multiplicative Jacobian Energy Decomposition (MJED), which allows the simulation to assemble the stiffness matrix of the system faster than the traditional Galerkin FEM formulation. The authors reported computation accelerations of up to five times for the St. Venant Kirchoff materials.

TLED and MJED methods both rely on pre-computation of simulation variables in order to achieve faster solutions at each time step. Although being useful for reflecting the elastic response of the deformable body that does not involve topological changes, these pre-computations at the initial configuration of the simulation object would be invalidated when a

topology-changing cut is introduced to the system. In other words, TLED and MJED are undermined by interactive cutting requirements.

2.2 Continuum Elasticity Theory

Continuum elasticity theory describes the mechanical behavior of the material that is modeled as a continuous medium rather than as discrete particles. The matter is assumed to be continuously distributed and fills the entire region of space that it occupies. Compared to simpler methods, continuum-based approaches offer a significant advantage, which is the convergence of the discrete solution to the continuous solution as the granularity of the discretization goes to zero. In addition to this consideration, the material properties are represented with well-established rheological properties such as Young's modulus and Poisson's ratio for linear material models, in contrast to the ad-hoc fine-tuning requirement of stiffness values of the mass-spring networks.

In continuum elasticity, deformation of a body is analyzed by studying the three major quantities *displacement*, *strain*, and *stress*. These quantities can be formulated either with the Lagrangian formulation or the Eulerian formulation. Eulerian formulation is commonly used in the analysis of fluid mechanics problems, where the attention is mostly focused on the motion of the material through a regular grid of volume [19]. Lagrangian formulation on the other hand, is typically used in solid mechanics problems, and describes the position and physical properties of the particles in terms of reference coordinates. Following the Lagrangian formulation, we can define configurations as sets containing the positions of all particles of the continuum. When the object undergoes deformation, the *current* configuration of the body is changing continuously. It is natural to keep the original positions of the particles, called the *reference* configuration, and

define the deformation as the transformation of a body from its *reference* configuration to its *current* configuration (Fig. 3). Strain, in the Lagrangian formulation, is a unitless measure of relative deformation representing the displacement between the particles of the current configuration and the reference configuration. Any strain of a solid material generates an internal elastic stress, which expresses the restoring internal forces that the neighboring particles exert on each other. The relation between stress and strain quantities is defined by the so-called constitutive equations as described below. In purely elastic materials the deformation of the object is recovered to its reference configuration after the stress is removed.

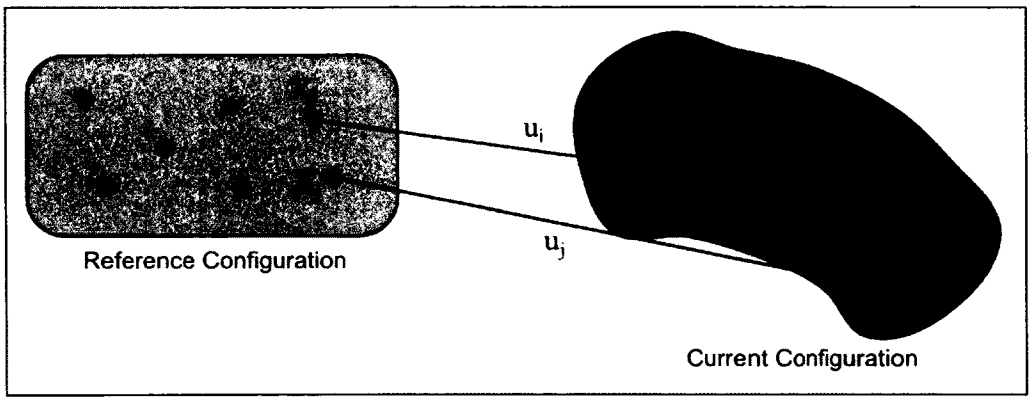


Fig. 3. In the Lagrangian formulation of continuum elasticity, a deformable body is represented in two configurations.

In the continuum elasticity modeling scheme, a deformable object is defined in its reference configuration with additional material parameters that are part of the underlying constitutive equations. The points inside the domain of the deformable body at its reference configuration are located by *material coordinates* $\mathbf{X} = (X, Y, Z)$, whereas the points at the current configuration are located by *world coordinates* $\mathbf{x} = (x, y, z)$. The displacement vector field is therefore defined as

$$\mathbf{u}(\mathbf{X}) = \begin{bmatrix} u_X \\ u_Y \\ u_Z \end{bmatrix} = \begin{bmatrix} x - X \\ y - Y \\ z - Z \end{bmatrix} = \mathbf{x} - \mathbf{X}. \tag{1}$$

The elastic strain tensor ϵ is computed from the spatial derivatives of the displacement field $\mathbf{u}(\mathbf{X})$. Tensors are generalizations of vectors that describe linear mappings between scalars, vectors, and other tensors. Tensors are defined by their orders/ranks, which also give the dimension of the array that represent the tensor. For example, a 0^{th}-rank tensor is a scalar value, whereas a 1^{st}-rank tensor is represented with a vector [25]. ϵ is a 2^{nd}-rank tensor in three dimensions, therefore having 3×3 symmetric elements

$$\epsilon = \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{xy} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{xz} & \epsilon_{yz} & \epsilon_{zz} \end{bmatrix}. \tag{2}$$

The components of ϵ are computed by using Green's nonlinear strain tensor equation under the assumption of small strain

$$\epsilon = \nabla\mathbf{u} + \nabla\mathbf{u}^{\mathrm{T}} + \nabla\mathbf{u}^{\mathrm{T}}\nabla\mathbf{u}, \tag{3}$$

where the gradient of the continuous displacement vector field $\nabla\mathbf{u}$ is essentially the derivatives of (u_X, u_Y, u_Z) with respect to (X, Y, Z) arranged in Jacobian format

$$\nabla\mathbf{u}^{\mathrm{T}} = \begin{bmatrix} \frac{\partial u_X}{\partial X} & \frac{\partial u_X}{\partial Y} & \frac{\partial u_X}{\partial Z} \\ \frac{\partial u_Y}{\partial X} & \frac{\partial u_Y}{\partial Y} & \frac{\partial u_Y}{\partial Z} \\ \frac{\partial u_Z}{\partial X} & \frac{\partial u_Z}{\partial Y} & \frac{\partial u_Z}{\partial Z} \end{bmatrix}. \tag{4}$$

The stress quantity is the directional force per unit area applied to a plane. Like the strain measurement, stress is also represented by a 3×3 tensor, relating the three-dimensional force vector with the normal of the plane

$$\sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix}. \tag{5}$$

The force per area applied to a plane with normal **n** is then obtained by multiplying the stress tensor σ by **n**.

Both the strain and stress tensors are symmetric with six independent components each. For the selection of the stress tensor, the concept of being work-conjugate is important. A strain tensor and a stress tensor are said to be work-conjugate when both of them are either defined with respect to material coordinates or world coordinates at the previous point in time. Work-conjugate property is a requirement for stability and correct solutions for certain types of problems [26]. For the Green-Lagrangian strain tensor, an appropriate stress tensor is the second Piola-Kirchoff stress tensor [19]. For an isotropic linear-elastic material, the strain tensor is mapped to the stress tensor by the 6×6 **C** matrix that approximates the material properties with elements that are composed of two independent coefficients, Young's Modulus (E) and Poisson's Ratio (v)

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{bmatrix} = \frac{E}{(1+v)(1-2v)} \begin{bmatrix} 1-v & v & v & 0 & 0 & 0 \\ v & 1-v & v & 0 & 0 & 0 \\ v & v & 1-v & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-2v & 0 & 0 \\ 0 & 0 & 0 & 0 & 1-2v & 0 \\ 0 & 0 & 0 & 0 & 0 & 1-2v \end{bmatrix} \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \epsilon_{xy} \\ \epsilon_{yz} \\ \epsilon_{zx} \end{bmatrix}. \tag{6}$$

Young's Modulus is a quantity that describes the elastic stiffness of a material, and defined as the ratio of the stress (units of pressure – *Pa*) over the strain (dimensionless) along the axis, where the stress is applied. For isotropic linear-elastic material models, the Young's Modulus is accurately defined only in the range of stress values, where there is a linear relation between the stress and strain values, i.e. where the Hooke's law holds [27]. Although linear material models reflect acceptable elastic response for small deformations of soft-tissues, experimental rheology studies show non-linear stress-strain relationships for actual soft-tissue samples. In addition to this non-linear complexity, the experiments conducted by Miller and

Chinzei also suggest that this relationship is also dependent on strain rates, i.e. loading speeds [28].

When an elastic object is compressed in one direction, it usually expands along the perpendicular directions. Poisson's Ratio is the linear parameter for the material model that describes the relationship between the compression and expansion amounts. An isotropic linear elastic incompressible material has a Poisson's Ratio of 0.5 for small deformations [29], which is assumed to be a valid value for soft-tissues as well [28].

The internal elastic forces can be derived through the so-called *strain energy formulation*. When the continuum is undergoing deformation, the energy stored in the system is called strain energy. For isotropic linear materials this energy term can be written as

$$U = \frac{1}{2}\sigma \cdot \epsilon, \tag{7}$$

where $\sigma \cdot \epsilon$ is defined as

$$\sigma \cdot \epsilon = \sum_{i=1}^{3}\sum_{j=1}^{3} \sigma_{ij} \cdot \epsilon_{ij}. \tag{8}$$

Strain energy density is a function of the displacement field. The elastic force per unit volume can be obtained by taking the negative directional derivative of the strain energy density with respect to the displacement field as

$$\mathbf{f} = -\nabla_{\mathbf{u}} U = -\frac{1}{2}\nabla_{\mathbf{u}}(\epsilon \cdot C\epsilon) = -\sigma\nabla_{\mathbf{u}}\epsilon. \tag{9}$$

After the force vectors are accumulated for each of the particles, the corresponding acceleration values are obtained through Newton's second law of motion. The acceleration

values are then used to update the velocity and position vectors of the individual particles with an appropriate time integration scheme.

2.3 Point-Based Discretization

Mesh-based discretization techniques such as FEM have dominated the field of computational mechanics in the past several decades. FEM is characterized by three fundamental steps:

1. A geometrically complex problem domain is decomposed into a collection of geometrically simpler subdomains called finite elements. The finite elements are tightly connected together, and they are collectively called the finite element mesh.
2. The governing differential equation is approximated over each finite element.
3. The approximated element equations are assembled using the element connectivity information, which leads to a large system of equations.

These methods have been widely used for modeling physical phenomena such as elasticity, heat transfer, and electromagnetism and they heavily rely on the assumption of a continuous domain. However, FEM is not well suited to problems involving extreme mesh distortions that result in degenerate element shapes, moving discontinuities that do not align with the element edges such as propagating cracks, and advanced material transformations such as melting of a solid or freezing. To address these issues, significant interest has been developed towards a different class of methods for solving PDEs, namely meshless or mesh-free methods [25, 30]. Mesh-based methods divide the deformable body into tightly connected finite-sized

elements. Meshless methods, on the other hand, represent a deformable object by a set of points, whose influence is distributed around them by a weight function as illustrated in Fig. 4.

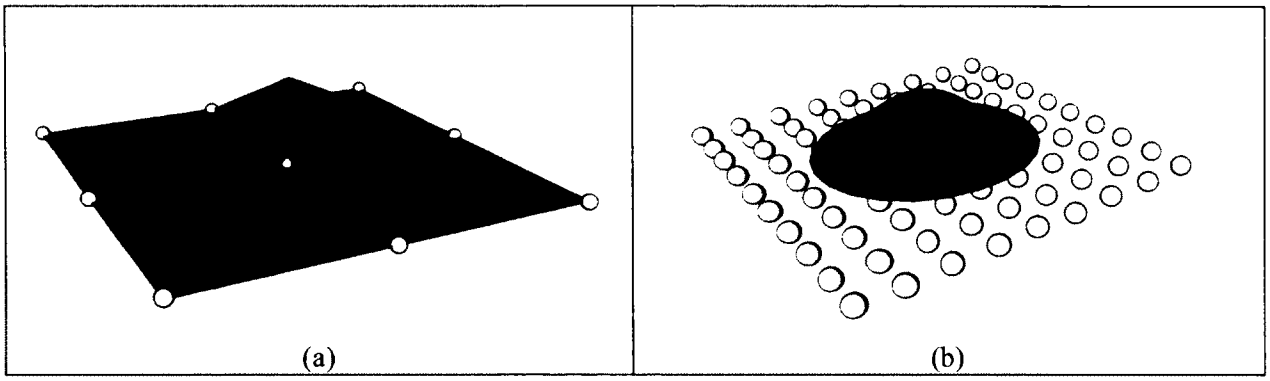


Fig. 4. Influence of a node to its neighbors in (a) mesh-based techniques, and in (b) meshless techniques.

Meshless methods are characterized by several fundamental steps [31]:

1. Meshless nodes are distributed throughout the computational domain and the boundaries. The radii of the support domains are set.
2. For each node, field variables such as displacement are approximated using the neighboring nodes that fall in the support domain. Shape functions, described later in this chapter, are used in this approximation.
3. The governing differential equation is discretized in the domain to obtain the linear algebraic equations at the nodes, either following the strong-form approach, which uses the differential equations directly and discretizes them at the meshless nodes, or following the weak-form approach, which first converts the differential equations to their integral-form and then numerically integrates them over a sub-domain.

4. The linear algebraic equations obtained from the previous step are assembled to get the global stiffness matrix and the force vector. This final system of equations is solved by a direct or iterative solver.

The first meshless approach dated back to 1977 [32] and proposed a smoothed particle hydrodynamics (SPH) method that was used to model theoretical astrophysical phenomena such as galaxy formation, star formation, stellar collisions, and dust clouds. Its Lagrangian formulation allowed diverse usage areas besides astrophysics such as fluid flow, ballistics, volcanology, and oceanography [33]. The discrete SPH form can be written as

$$u(x) = \sum_{I} w_I(x)\, u_I \Delta V_I, \tag{10}$$

where $u(x)$ is the approximation of the field variable at the independent variable location x, I is the set of nodes that has the location x inside their domain of influence, $w_I(x)$ is the weight of the node I at x, u_I is the value of the field variable at the node I, and ΔV_I is the size of the domain of the node I.

Although the SPH method eliminates the necessity of a mesh structure and allows the solution of unbounded problems, it also has its limitations. Because of its approximation scheme is based only on the weight function, it fails to reproduce first-order polynomials and even constant fields, resulting in severe consistency problems [25] as depicted in Fig. 5.

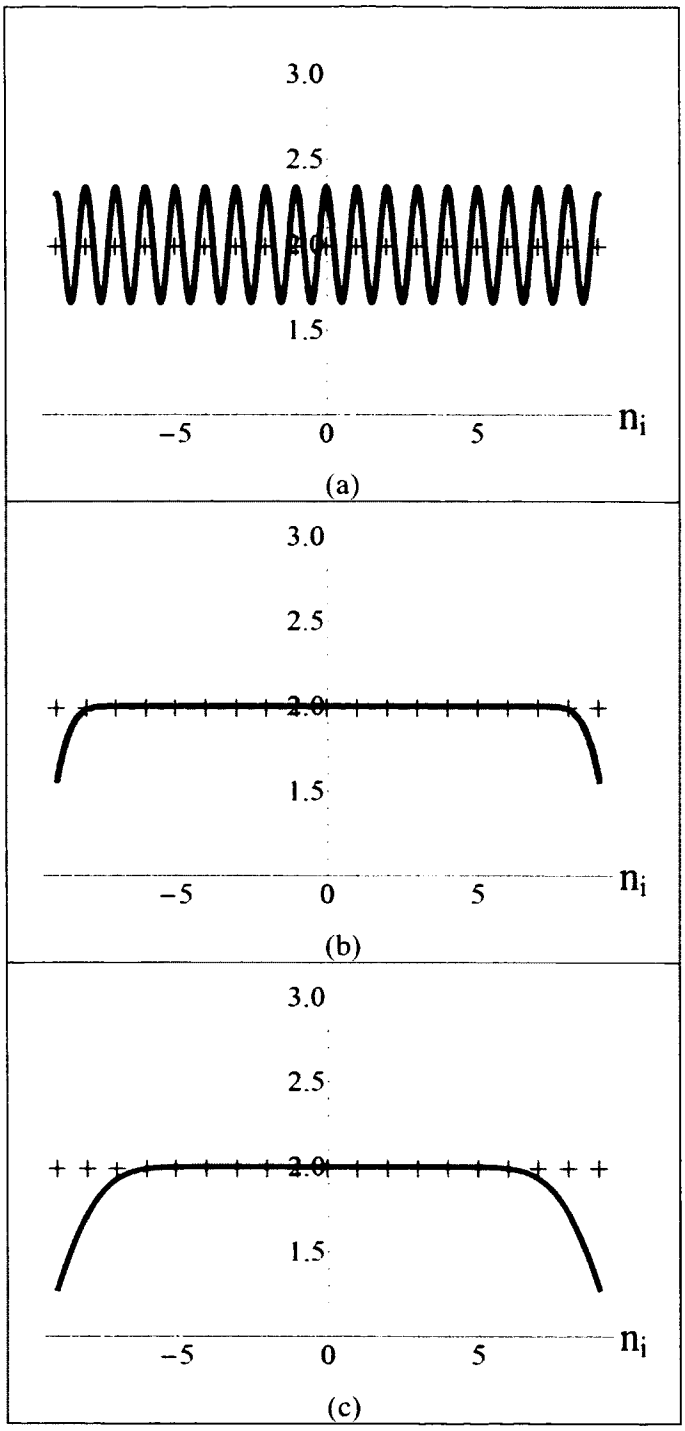


Fig. 5. SPH-based approximation fails reconstructing constant fields (plus signs). Reconstruction results are presented for different support radii with the order (a) < (b) < (c).

Although the SPH method performs relatively well in reconstructing constant fields when the nodes are distributed uniformly, its approximation power degrades quickly under the assumption of non-uniform node distribution as shown in Fig. 6.

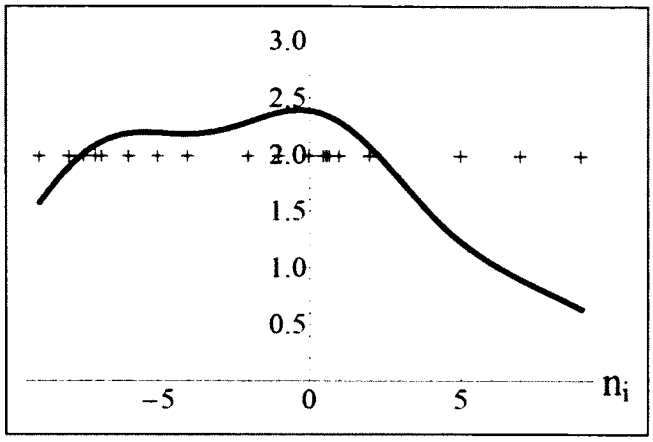


Fig. 6. Approximation power of the SPH method degrades as the uniformity of the nodes vanishes.

To alleviate this problem, methods that utilize *Moving Least Squares* (MLS) approximations have been developed. The first work that used MLS approximations in a Galerkin method is the work of Nayroles et al. [34], which was refined by Belytschko et al. [35] and named Element-Free Galerkin (EFG) method. This class of methods, different from the SPH method, uses shape functions in approximations that are essentially corrected versions of compact supported weight functions

$$u(x) = \sum_I \phi_I(x)\, u_I, \tag{11}$$

here, the sum of shape functions $\phi_I(x)$ for a given approximation equals to 1, which is known as the *partition of unity* paradigm [30] and important to improve the consistency of an approximation as described below.

The shape functions are obtained by first representing the approximation as a product of a polynomial basis and a vector of unknown coefficients. Then, a functional is created by taking the weighted sum of square of the approximation error. By taking the derivative of this functional with respect to the unknown coefficients and setting it to zero for minimizing the approximation error, we obtain a set of equations that are reorganized to solve for the MLS shape functions. The consistency – a solution needs to be consistent in order to ensure stability – of the MLS approximation scheme depends on the order and completeness of the chosen basis function. If the basis function used in the approximation is a complete polynomial of order k, then the MLS approximation is said to be k-th order consistent. In other words, an approximation that is k-th order consistent can reproduce a k-th order polynomial exactly. This characteristic makes the MLS based approximations more consistent than the SPH method [30] (Fig. 7). Detailed description of complete polynomials and a brief analysis of MLS approximations are discussed in the next section.

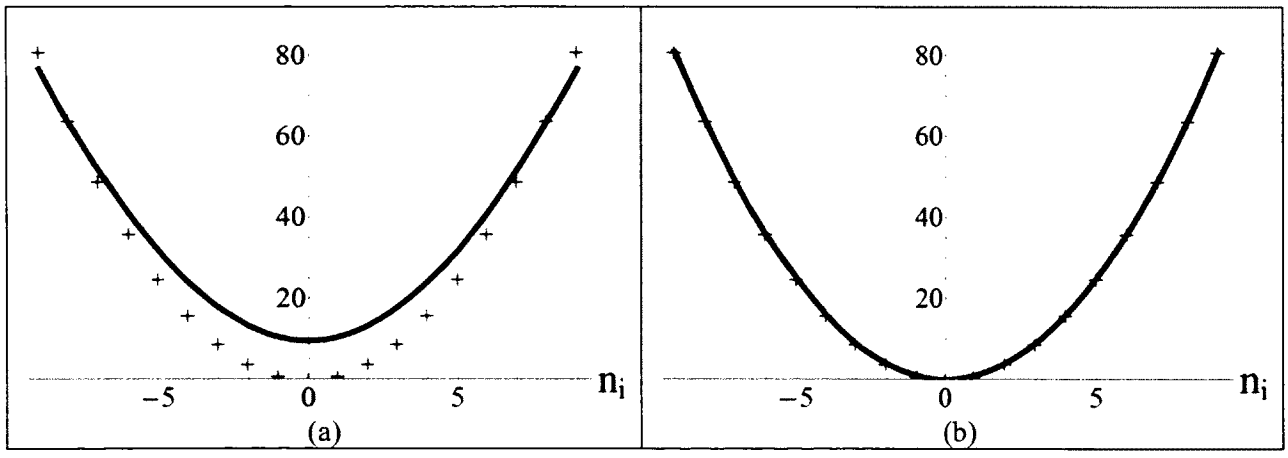


Fig. 7. Comparison of different order reconstructions. The data points corresponding to the function $y = x^2$ cannot be reconstructed exactly via (a) MLS-based approximation with the linear basis $[1\ x]$. On the other hand, by using (b) the quadratic basis $[1\ x\ x^2]$ they can be reconstructed exactly.

Another technique that has used the MLS approximation is the work of Müller et al. [36], which forms the basis of the point-based method discussed in this dissertation. In their presented framework, the authors calculated the spatial derivatives of the deformation gradient only at the particle locations. This approach is similar to the meshless point collocation methods [37] that discretize the differential equations only at the meshless nodes. Typical characteristic of meshless point collocation methods is their truly meshless nature as they do not require an underlying mesh structure for field variable approximation or spatial integration. As described above in the meshless method steps, instead of converting the governing differential equations into their weak form and integrate over a sub-domain, point collocation methods directly discretize the strong-form of the governing differential equation at the meshless nodes. The advantage of the point collocation methods is the computational efficiency as the shape functions do not need to be evaluated at the integration points, with the expense of difficulty in imposing natural boundary conditions, where the field variables take the specified values. The technique described by Mueller et al. is capable of simulating a wide range of material properties from very stiff materials to soft ones, while also being able to handle plastic deformations as well.

Horton et al. [38] proposed a new kind of meshless method named meshless total Lagrangian explicit dynamics method. The authors extended their previous TLED algorithm [23] to the meshless discretization framework by pre-computing the MLS shape function derivative matrices for each of the integration point. Through this pre-computation, they did not have to compute the costly shape functions at each time step at an expense of the additional memory consumption. Their method is a fully explicit method, meaning not requiring the costly solution of large system of equations that one need to solve for the implicit schemes. The additional

computation burden of an implicit solution can be exemplified with a simple system with the following governing equation

$$\frac{d\mathbf{x}}{dt} = -\mathbf{K}\mathbf{x}, \tag{12}$$

where $\mathbf{x}$ is a vector of field variables and $\mathbf{K}$ is a coefficient matrix. This equation can be written in its discrete form using the implicit (backward) Euler scheme as

$$\mathbf{x}^{t+1} = \mathbf{x}^{t} - \Delta t\mathbf{K}\mathbf{x}^{t+1}, \tag{13}$$

and the terms of this equation can be arranged to obtain the system of equations that one needs to solve for an implicit integration

$$\mathbf{x}^{t+1}(\mathbf{I} + \Delta t\mathbf{K}) = \mathbf{x}^{t}, \tag{14}$$

where $\mathbf{I}$ is the identity matrix.

Horton et al.'s proposed algorithm integrates the weak-form of the governing equations over a regular background grid, where each cell has a single integration point. By having a single integration point per each cell rather than the traditional multiple integration points, Horton et al. were able to increase the computational throughput of their algorithm.

Node distribution is the first step in the presented point-based discretization algorithm, which supports both regular and hierarchical distribution of the nodes through the simulation domain. In the case of a simulation domain with a regular geometric shape, a regular uniform distribution of the nodes is the natural choice. On the other hand, if the simulation domain has a complex geometry, which is the general case, a uniform distribution becomes inapplicable. In this case, meshless nodes have to be distributed throughout the domain bounded by the complex boundary surface. In the meshless framework presented by Pauly et al. [39], the authors used a balanced octree data structure to partition the volume of the object as illustrated in Fig. 8.

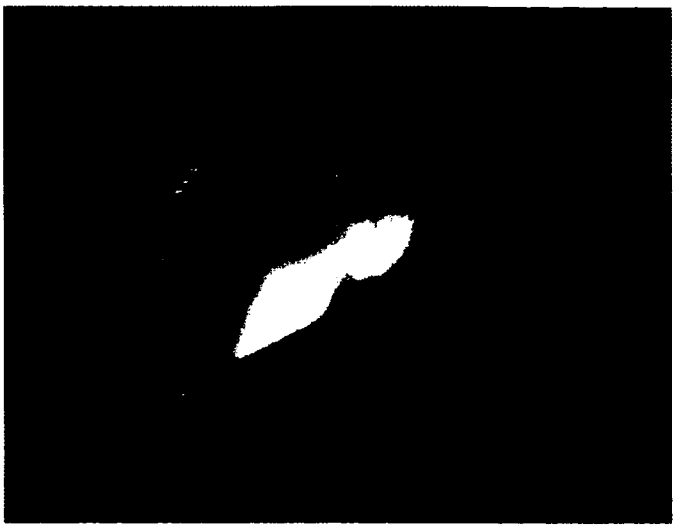


Fig. 8. Starting from the surface's bounding box, each octree cell is refined if it has a part of the surface until the desired octree depth is reached.

After the octree partition was completed, Pauly et al. created a meshless node at each octree cell center that is located inside the bounding surface. A similar approach to the octree-based node sampling algorithm proposed by Pauly et al. [39] was followed and the object represented by a boundary surface was converted to a set of meshless nodes through the tetrahedralization algorithm. For this purpose, well-established computational geometry libraries like TetGen [40] and CGAL [41] were utilized as an offline process. The output tetrahedra meshes obtained from these were later post-processed, and the vertex positions were extracted to be used as the initial meshless node locations (Fig. 9).

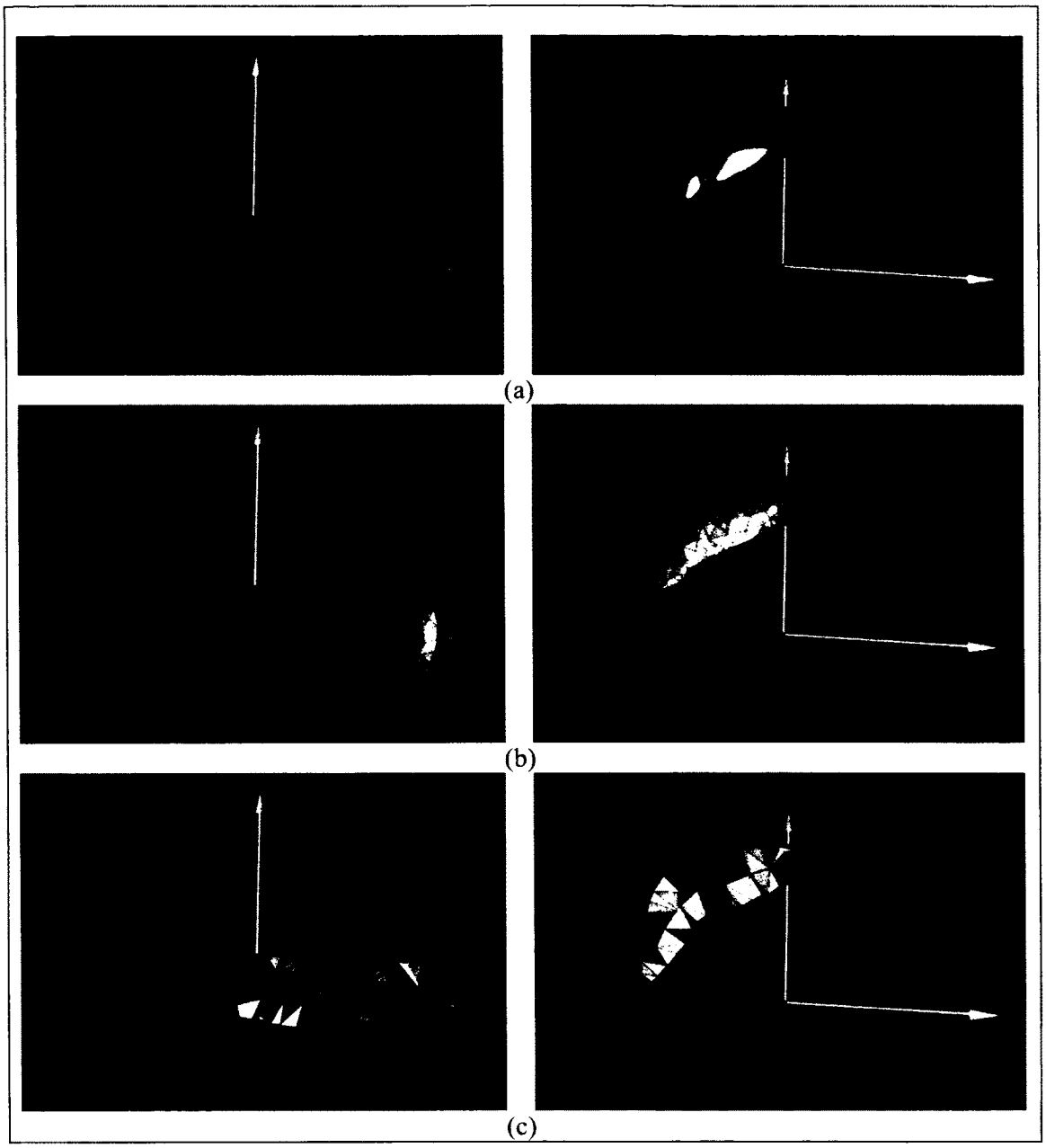


Fig. 9. Volumetric discretization steps of sample objects. (a) Triangular surface mesh of the rectangular block and liver objects, (b) the surface meshes are tetrahedralized with the QTetraMesher tool [42], (c) display of the internal tetrahedra.

Meshless methods represent a deformable body by a *cloud of particles*, or *nodes* with domains of overlapping support. Quantities such as mass, volume, support size, strain, and stress are stored and updated per particle for the duration of the simulation. In this work, the support

domains of the particles are spherical and their radii are computed by finding the average distance of the central node to its k-nearest neighbors (Fig. 10). For efficient neighborhood search purposes, a k-d tree data structure is used. Spherical support domains are chosen over rectangular support domains, as the former type requires the evaluation of the weight function only once with respect to the distance to the center of the sphere whereas the latter type requires evaluating the weight function three times in 3D with respect to the distance to the support center along each axis.

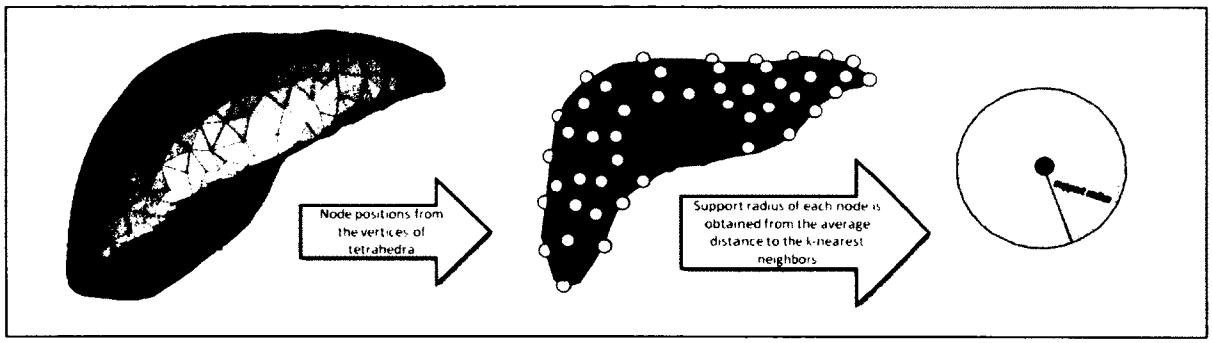


Fig. 10. The workflow of the point-based discretization pipeline takes node positions from the vertices of tetrahedral. For each node, the support radius is calculated from the average distance to the k-nearest neighbors, which is used to define the nodal neighborhood.

The weight (kernel) function in the meshless method context describes the way meshless nodes affect each other and how the material values of the continuum such as mass, volume, and density are distributed among the nodes as detailed below. The neighboring particles that fall inside the support domain of a central particle are weighted using the polynomial weight function (Fig. 11)

$$w(r_{ij}) = \begin{cases} 1 - 3r_{ij}^2 + 3r_{ij}^4 - r_{ij}^6, & r_{ij} \leq 1 \\ 0, & r_{ij} > 1' \end{cases} \tag{15}$$

with $r_{ij} = \frac{\|\mathbf{x}_j - \mathbf{x}_i\|}{h_j}$, where $\mathbf{x}_j$ and $\mathbf{x}_i$ are the current locations of the neighboring and central particles respectively and h_j is the support radius of the neighboring particle j (Fig. 12). This function satisfies every requirement of a meshless weight function; in practice, it can also be simplified as

$$w(r_{ij}) = (1 - r_{ij}^2)^3. \tag{16}$$

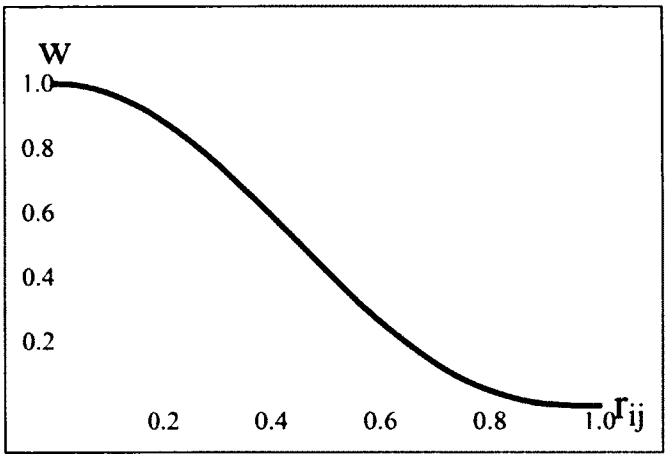


Fig. 11. The polynomial weight function (kernel) used in the MLS approximations. In meshless methods, weight functions have to be continuous and positive in their support and they are critical to solution accuracy and stability.

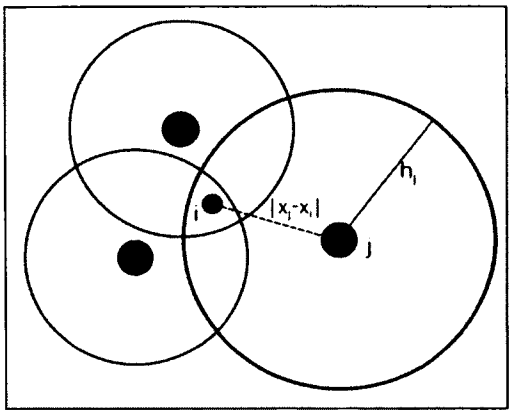


Fig. 12. The central (blue) and neighboring (green) particles and the variables that define the influence of one on another.

The mass and density of a meshless node are assigned at the beginning and kept fixed throughout the simulation. The mass values for each node are initialized with

$$m_i = s\bar{r}_i^3 \rho \,, \tag{17}$$

where ρ is the material density value, $\bar{r}_i$ is the average distance of the i^{th} node to its k-nearest neighbors, and s is a scaling factor that is chosen so that the average of the assigned densities is close to the actual material density, which is assumed to be constant throughout the object. The assigned mass value of a meshless node is distributed around the node with the weight function. Therefore the density of a meshless node is calculated after the mass allocation step by taking the weighted average of the masses of the neighboring nodes

$$\rho_i = \sum_j m_j \, w(r_{ij}) \,. \tag{18}$$

In other words, local density value ρ_i, calculated at each meshless node, is a smoothed quantity with weighted contributions from all nodes within the neighborhood. After nodal densities are calculated, the volume of each node is obtained by dividing the node's mass m_i by its density ρ_i.

In the approach of this dissertation, instead of converting the governing differential equation to the weak-form and performing numerical integration, it was decided to directly discretize the strong-form of the governing equation at the meshless nodes to obtain the linear algebraic equations. Similar to the meshless point collocation methods, the spatial derivatives of the deformation gradient are calculated only at the meshless node locations (Fig. 13).

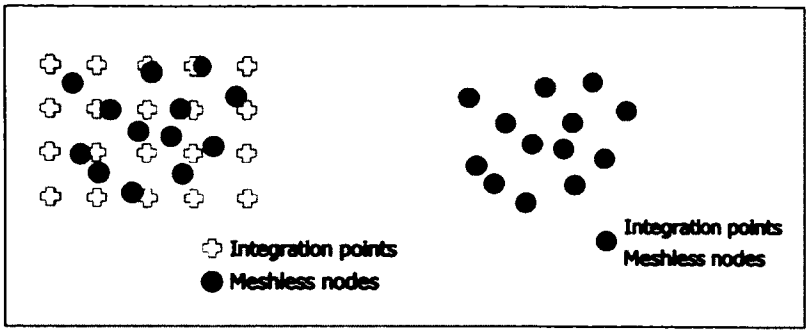


Fig. 13. Typically, spatial integration techniques utilize a background grid with multiple integration points per region (left), in nodal integration techniques (right) spatial derivatives are calculated only at the node locations.

2.4 Moving Least Squares Approximation

Moving Least Squares (MLS) is a method of approximating a continuous function for a given set of point samples [43]. In MLS-based approximation, the approximated function f^h at a given location x is defined as

$$f^h(x) = \sum_{I=1}^{N} \phi_I(x) f_I, \tag{19}$$

where $\phi_I(x)$ is the shape function of node-I evaluated at x, and f_I is the value of the function at node-I. The shape function is computed from the weight function $w(r_{ij})$ defined earlier in equation (15) and the basis function $\mathbf{p}(x)$, which is a complete polynomial of a given order such as the second-order polynomial in 1D

$$\mathbf{p}^{\mathrm{T}}(x) = [1\ x\ x^2], \tag{20}$$

and the first-order polynomial in 2D

$$\mathbf{p}^{\mathrm{T}}(\mathbf{x}) = [1\ x\ y]. \tag{21}$$

The reconstruction basis $\mathbf{p}^{\mathrm{T}}(\mathbf{x})$ and the support radius of the weight function $w(r_{ij})$ play important roles in the approximation accuracy. MLS-based approximation of random data at

equidistant locations shows the effects of the basis order and support radius on the approximation accuracy (Fig. 14-Fig. 17).

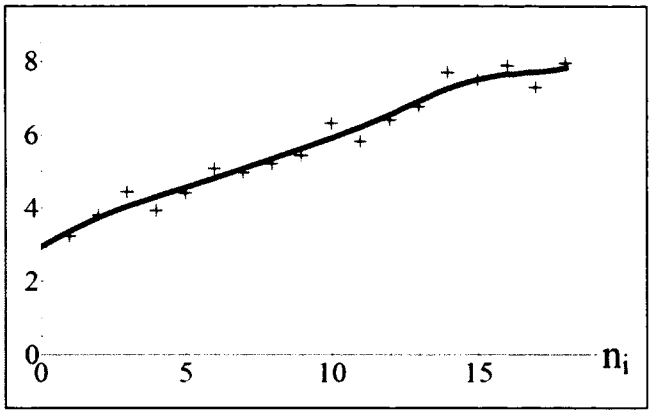


Fig. 14. A continuous function is approximated from random data points at equidistant locations using the linear basis [1 x]. The support radii of the weight functions of the nodes n_i are all set to 5 node distance.

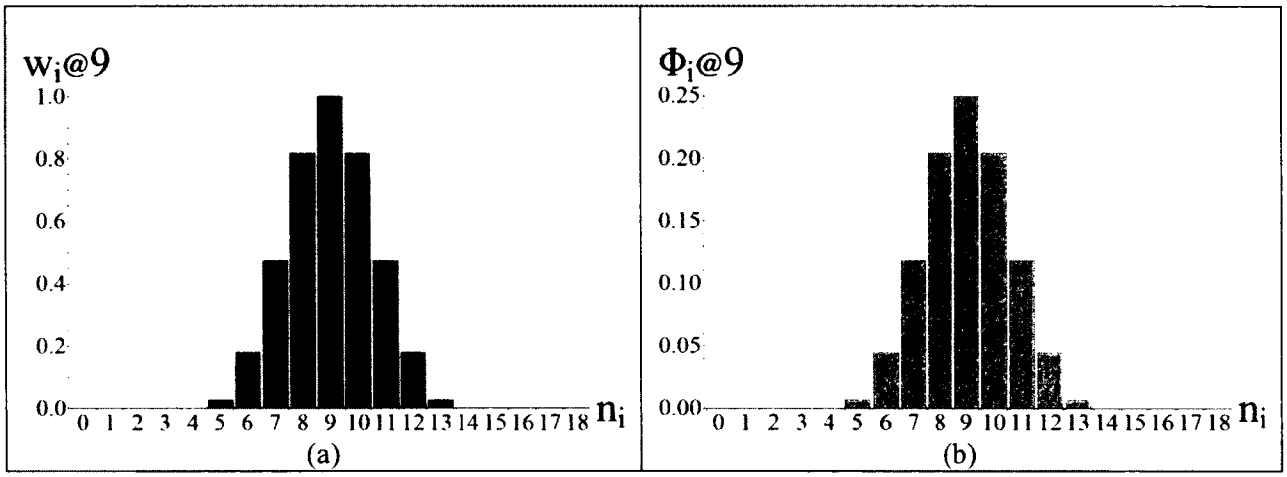


Fig. 15. Comparison of weight and shape functions of linear basis reconstruction. (a) The weight function of node i evaluated at the 9th node location, and (b) the shape function of node i evaluated at the 9th node location obtained from the linear basis. The shape values sum up to 1.

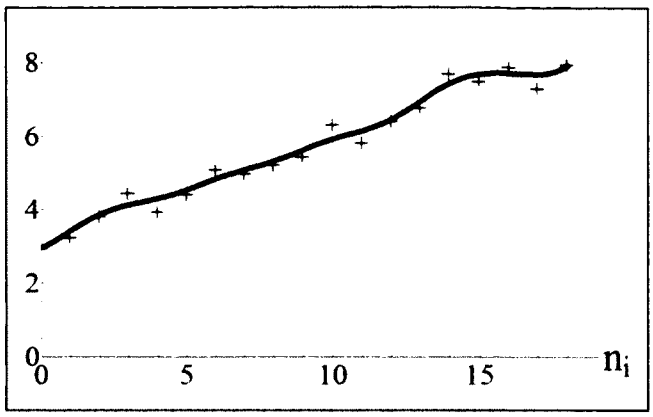


Fig. 16. A continuous function is approximated from random data points at equidistant locations using the quadratic basis $[1\ x\ x^2]$. The support radii of the weight functions of the nodes n_i are all set to 5 node distance.

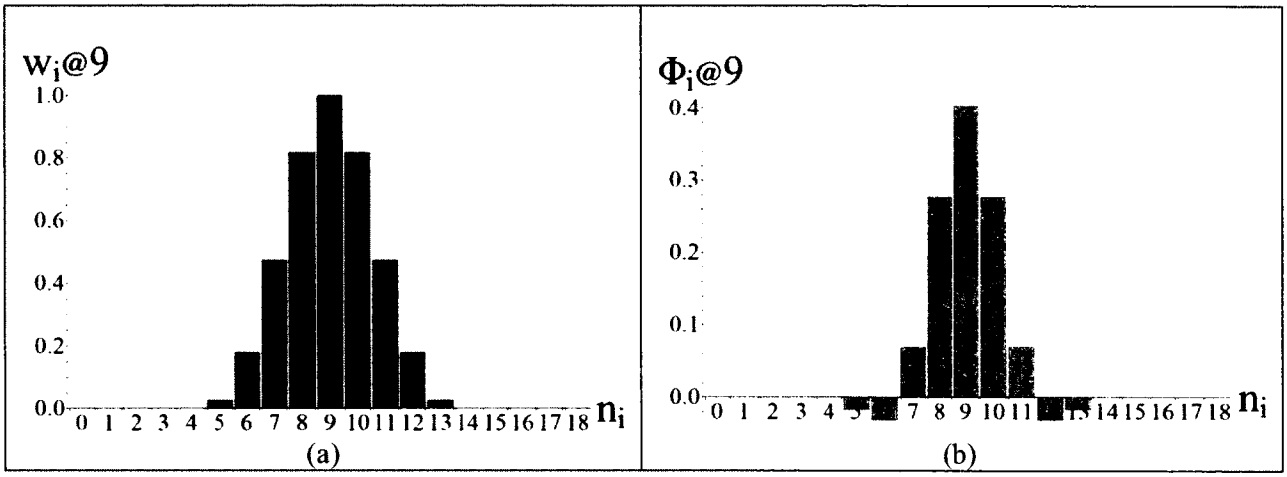


Fig. 17. Comparison of weight and shape functions of quadratic basis reconstruction. (a) The weight function of node i evaluated at the 9th node location, and (b) the shape function of node i evaluated at the 9th node location obtained from the quadratic basis. The shape values sum up to 1.

Several support radii configurations have been tried for both the linear basis and the quadratic basis reconstruction of the given random points. The root mean square error values for the approximations were obtained from the difference of f_I and their reconstructed values at the node locations. The results are shown in

Table 1.

Table 1. Root mean square error values of the approximations are compared for different MLS configurations.

Support Radius	RMSE at Linear Basis	RMSE at Quadratic Basis
1	Singular matrix	Singular matrix
2	0.148041	Singular matrix
3	0.212183	0.146464
4	0.241102	0.192909
5	0.258320	0.222548
6	0.267671	0.243705
7	0.273891	0.254194

The results show that approximations with the quadratic basis are generally better in terms of the root mean square error. Also, decreasing support radius values result in better approximations as well meaning that support radius in meshless approximations plays a similar to the element size in finite element methods [30].

For the problem outlined in this dissertation, the partial derivatives of the displacement vector field were needed in order to compute the strain, stress, and the internal elastic forces applied to the meshless particles. Therefore, a version of the MLS-based approximation is used to compute this gradient ($\nabla \mathbf{u}$).

Recalling from the Equation (1), for a central meshless node i and its neighbor j, the value of the x-component of the displacement, u_x, at the location of j can be approximated by the first-order Taylor expansion as

$$\tilde{u}_{x_j} = u_{x_i} + \left.\frac{\partial u_X}{\partial \mathbf{X}}\right|_i \cdot (\mathbf{X}_j - \mathbf{X}_i). \tag{22}$$

The weighted sum of squared differences between the displacement vector and its approximation obtained from Equation (22) gives the error measure of the MLS approximation

$$e = \sum_j \left(\tilde{u}_{x_j} - u_{x_j}\right)^2 w(r_{ij}). \tag{23}$$

By inserting Equation (22) into Equation (23), the expression for the error measure for the particle *i* becomes

$$e = \sum_j \left(u_{x_i} + \left(\frac{\partial u_X}{\partial X}\right) X_{ij} + \left(\frac{\partial u_X}{\partial Y}\right) Y_{ij} + \left(\frac{\partial u_X}{\partial Z}\right) Z_{ij} - u_{x_j}\right)^2 w(r_{ij}), \tag{24}$$

where X_{ij}, Y_{ij}, and Z_{ij} are the x, y, and z-components of the vector $\mathbf{X}_j - \mathbf{X}_i$. It is desirable to minimize this error measure for some values of $\frac{\partial u_X}{\partial X}$, $\frac{\partial u_X}{\partial Y}$, and $\frac{\partial u_X}{\partial Z}$, therefore, based on the calculus of variations [44], the derivative of the error measure e with respect to the partial derivatives of the displacement vector was set to zero, resulting in three equations for three unknowns

$$\left(\Sigma_j(\mathbf{X}_j - \mathbf{X}_i)(\mathbf{X}_j - \mathbf{X}_i)^T w(r_{ij})\right) \frac{\partial u_x}{\partial \mathbf{X}}\Big|_i = \Sigma_j \left(u_{x_j} - u_{x_i}\right)(\mathbf{X}_j - \mathbf{X}_i) w(r_{ij}). \tag{25}$$

This equality can be represented in a more compact format as

$$\mathbf{A} \frac{\partial u_x}{\partial \mathbf{X}}\Big|_i = \sum_j \left(u_{x_j} - u_{x_i}\right)(\mathbf{X_j} - \mathbf{X_i}) w(r_{ij}), \tag{26}$$

where **A**, the 3x3 moment matrix, is given by

$$\mathbf{A} = \left(\sum_j (\mathbf{X}_j - \mathbf{X}_i)(\mathbf{X}_j - \mathbf{X}_i)^T w(r_{ij})\right). \tag{27}$$

The moment matrix **A** can be inverted and pre-multiplied with both of the sides of the equation for computing the partial derivatives.

2.5 Computation of Forces via Strain Energy

The elastic body forces that are applied to the individual nodes in the meshless collocation method are calculated through the *strain energy density*, which is a function of the particle displacements. For a node i with volume v_i, strain ϵ_i, and stress σ_i, the nodal strain energy, which was defined in Equation (7), becomes

$$U_i = v_i \frac{1}{2}(\epsilon_i \sigma_i). \tag{28}$$

Recalling from the Equation (9), the elastic force applied to the volume at the meshless node is the negative directional derivative of the above strain energy with respect to this node's displacement. The forces applied to the particle i and its neighbors j are then

$$\begin{aligned} f_i &= -\nabla_{u_{x_i}} U_i = -v_i \sigma_i \nabla_{u_{x_i}} \epsilon_i \\ f_j &= -\nabla_{u_{x_j}} U_i = -v_i \sigma_i \nabla_{u_{x_j}} \epsilon_i \end{aligned} \tag{29}$$

Equation (29) is iterated over all meshless nodes in the domain and the total force applied to a given node is the sum of all the forces with the same index as that node. These force components are obtained by using the Green-Saint-Venant strain tensor, which measures the linear and shear elongation. In the case of a volume-inverting displacement however, this strain becomes zero. In order to introduce restoring body forces in the event of volume inversion, Muller et al. [36] added another energy term to the system that penalizes deviations from a volume-conserving transformation

$$\acute{U}_i = v_i \frac{1}{2} k_v (|\mathcal{J}_i| - 1)^2. \tag{30}$$

In this energy term, $\mathcal{J}_i$ is the Jacobian of the displacement vector field mapping, which is equal to $\mathcal{J}_i = \mathbf{I} + \nabla \mathbf{u}_\mathbf{i}^\mathrm{T}$, and k_v is the volume restoration constant. In the experiments, there was not a situation that required this additional force component; therefore it was not included in the implementation.

2.6 Summary

In this chapter, the approaches for modeling deformable objects were discussed. These approaches range from non-physical methods to physically-based techniques. The former category of methods is generally easier to implement and also gives the user the ability to control the resulting animation. The problem with these techniques is that they are not based on physical theories/calculations, which prevents them from being used in simulations with medical focus. Next, a basis for the approach was established by discussing the basics of the continuum elasticity theory. Finally the point-based discretization of the continuum and Moving Least Square (MLS) approximation-based meshless method were presented. Compared to the traditional mesh-based continuum elasticity methods, such as the FEM, meshless methods may be useful in problems with extreme deformations or spatial discontinuities such as cuts.

CHAPTER 3

HANDLING DISCONTINUITIES IN MESHLESS METHODS

In engineering problems, discontinuities are a common occurrence. In these cases, the continuum assumption of the elastic theory is undermined, which typically requires special treatment to ensure the correct solution to the system. Discontinuities may be caused when the continuum domain is composed of different material types or when there is a spatial gap in the continuum such as a cut. This chapter will first present several existing techniques to handle discontinuities, which will be followed by the approach used in this dissertation, where the 2D model is described and subsequently its extension into 3D.

3.1 Overview

In meshless methods, there are three main classes of techniques to treat discontinuity of the field variable (displacement). These techniques are: (1) modification of the weight function, (2) intrinsic enrichment of the basis of the approximation, and (3) techniques based on extrinsic enrichment. Discontinuity treatment in meshless methods has been studied within a wide range of approaches such as visibility criterion, diffraction/transparency methods, and H- and P-refinements [25].

The visibility method is an example of techniques that modify the weight function. In this method, the cut segment is treated as an opaque object and the influence of a node to a point in the domain is decided by drawing a line between the node and the point in question, and testing whether the line intersects with the cut segment or not. Although being simple in nature, this method can lead to incorrect discontinuity calculations along the lines that pass through the cut tips, as depicted in Fig. 18. Another disadvantage of this method is that it cannot be used to treat

non-convex boundaries. The diffraction method follows the same steps as the visibility criterion, but improves the technique by passing the ray around the cut tip and calculating the influence of a node on a point via the ray length. The diffraction method requires complex computations of the bending rays and its extension into three dimensions is even more complex [45].

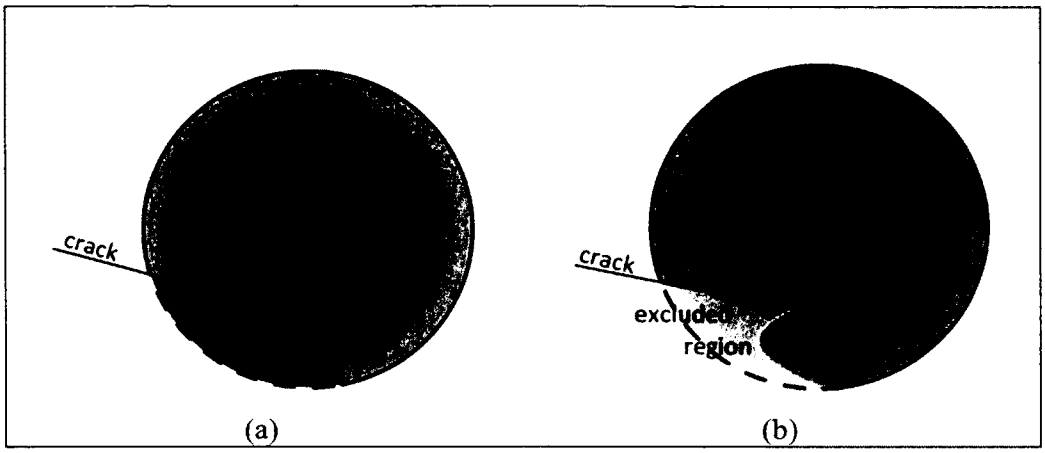


Fig. 18. Two examples of weight function-modifying discontinuity treatment techniques in meshless methods, (a) the visibility criterion and (b) the diffraction method.

Element Free Galerkin (EFG) method is a MLS-based meshless technique that uses intrinsic basis for approximating the field function. Given an approximation of the form

$$u(x) = \sum_I \phi_I(x)\, u_I, \tag{31}$$

MLS-based approximations compute the shape functions $\phi_I(x)$ from the weight function and the polynomial basis $p^T(\mathbf{x})$ as discussed in Chapter 2. To model strong discontinuities (discontinuities involving the field variable), the intrinsic basis can be modified by using the information from the cut geometry. For a 2D problem domain, the linear basis

$$p^T(\mathbf{x}) = [1, x, y] \tag{32}$$

can be extended to

$$p^T(\mathbf{x}) = [1, x, y, \sqrt{r}\sin\left(\frac{\theta}{2}\right), \sqrt{r}\cos\left(\frac{\theta}{2}\right), \sqrt{r}\sin\left(\frac{\theta}{2}\right)\sin(\theta), \sqrt{r}\cos\left(\frac{\theta}{2}\right)\sin(\theta)] \quad (33)$$

where r is the radial distance to the cut tip and θ is the incident angle to the cut [30]. A disadvantage of the intrinsic enrichment technique is the additional computational cost that comes from the increased size of the basis. Shape function computations use the polynomial basis, and also take place at each time step. This additional computation cost at each time step eliminates the suitability of this type of enrichment for interactive applications.

3.2 Modeling Discontinuities in 2D

Barbieri et al. [12] proposed an enrichment technique based on a distance function for handling discontinuities with multiple boundaries. Their method processed cuts as piecewise-linear segments and calculated the absolute distance of a meshless node to these segments. The enrichment function obtained from the distance field is then multiplied with the weight kernel of the node. Based on an analytic formulation, this approach not only required less computation compared to the competing techniques like the visibility criterion, but also was easier to extend into 3D by modifying the distance function to support the extra dimension.

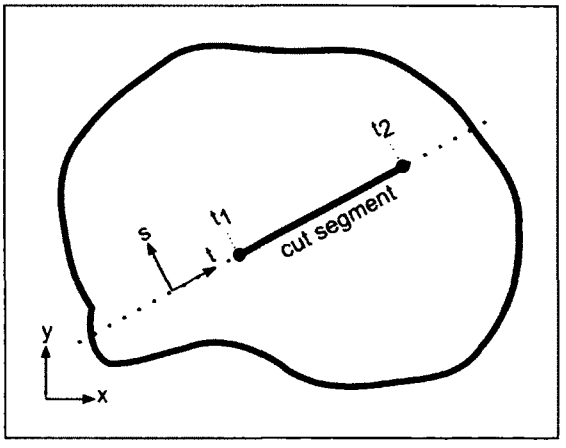


Fig. 19. The discontinuity caused by a cut segment is defined at the local coordinate frame of the cut segment.

As depicted in Fig. 19, the distance function is computed in the local coordinate system of the cut piece. The 2D distance function for a given point (x, y), can therefore be computed in terms of the local coordinates (t, s) as

$$d_2(x, y) = \sqrt{d_1^+(t)^2 + s^2} \tag{34}$$

where $d_1^+(t)$ is the positive part of the 1D signed distance function $d_s(t)$ for a 1D segment, in local coordinates, and defined as

$$d_s(t) = \left|t - \frac{t_1 + t_2}{2}\right| - \left|\frac{t_1 - t_2}{2}\right| \tag{35}$$

where t_1 and t_2 are the endpoints of the cut segment in the cut's local coordinate system (Fig. 20) and

$$d_1^+(t) = \frac{d_s(t) + |d_s(t)|}{2}. \tag{36}$$

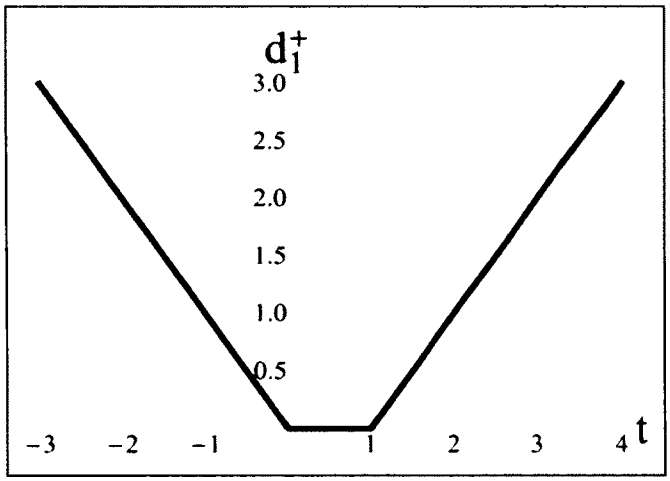


Fig. 20. The function $d_1^+(t)$ is calculated for a cut segment with $t_1 = 0$ and $t_2 = 1$.

In order to introduce a sharp discontinuity across the cut segment that smoothly varies from one side to the other side, we can take the partial derivative of this distance function with respect to the normal coordinate axis s,

$$\frac{\partial d_2}{\partial s} = \frac{s}{d_2} = \varphi, \tag{37}$$

and obtain the discontinuous function φ (Fig. 21) across the segment that is 1 on one side of the cut and -1 on the other side and varies smoothly around the cut (Fig. 22).

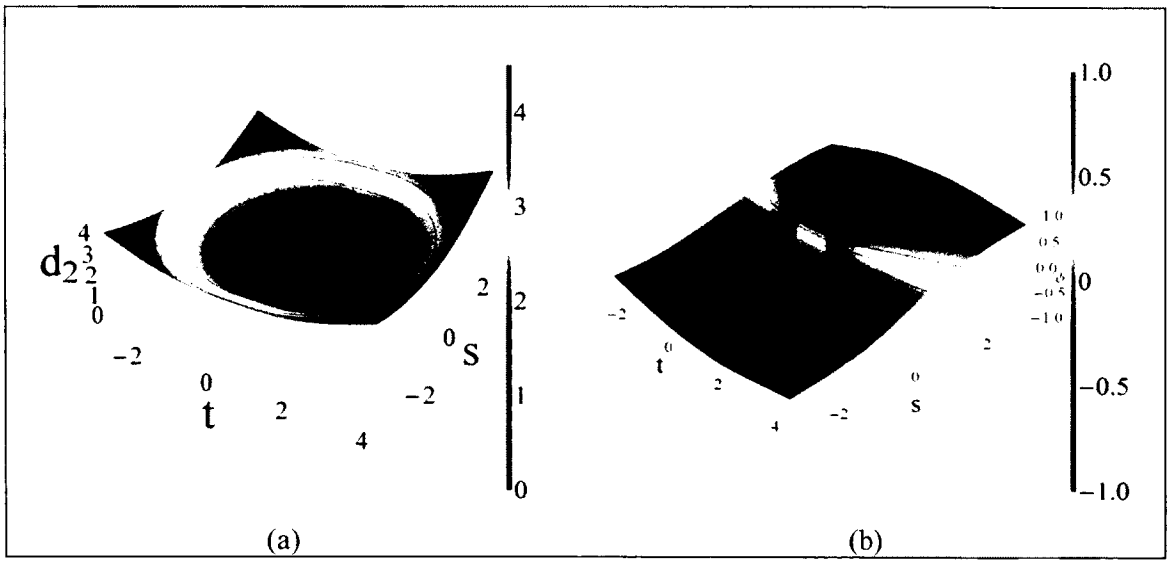


Fig. 21. Three dimensional plots of (a) the distance function d_2 and (b) the discontinuous function φ.

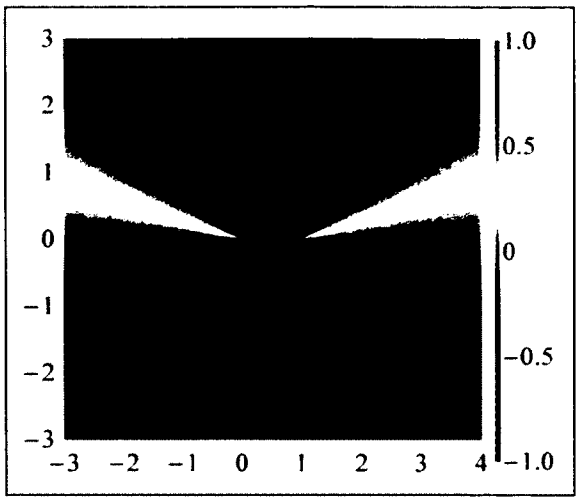
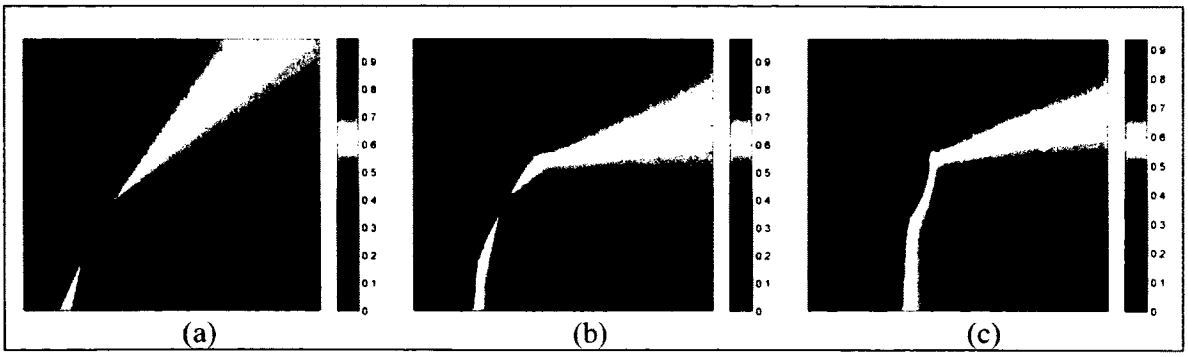


Fig. 22. Contour plot of the discontinuous φ function.

This technique is easy to implement and also applicable to the existing methods to extend their functionalities. One shortcoming of the technique is the approach it takes on handling multiple cuts. Let h_l be the enrichment value for the l-th cut on a node, the cumulative enrichment value for the node that is in the vicinity of/affected by n cuts is given by the product

$$h = \prod_{l=1}^{n} h_l. \tag{38}$$

Multiplicative application of consecutive enrichments is a practical approach and requires the least amount of processing, but it may also lead to incorrect weight function modifications and therefore fatal instabilities in the simulation as visualized in Fig. 23.



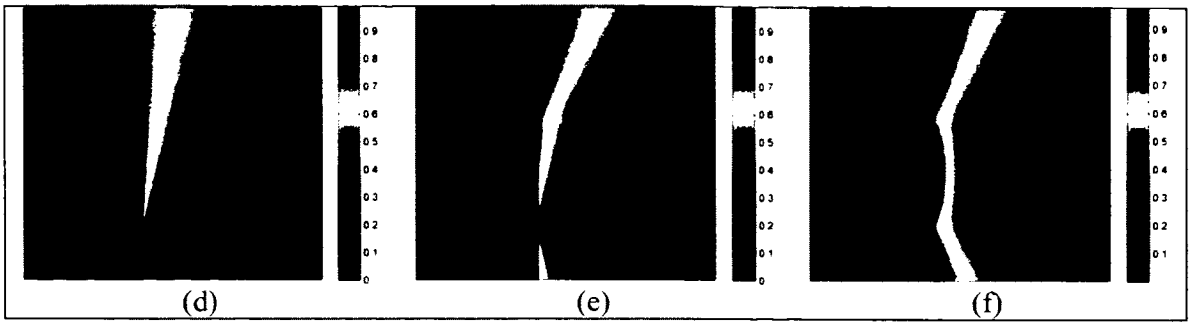


Fig. 23. The weight functions of the close meshless nodes are modified incorrectly when consecutive enrichments are applied by multiplying. (a-c) The three consecutive segments of a cut and the resulting enrichment function from multiplying. (d-f) The three consecutive segments of a cut that separates the domain completely and the resulting enrichment function.

The top row (a, b, c) of Fig. 23 shows the effect of a cut that is composed of three linear segments, when their enrichment values are multiplied consecutively. The bottom row (d, e, f) of the figure represents a cut that completely separates the domain into two parts. For this example, the correct values for the enrichment at the final configuration (f) would be completely 1 on one side and completely 0 on the other side.

In order to address these issues, an extension of the distance function-based enrichment technique is proposed to support consecutive discontinuity fronts in a correct way. In this extended technique, the enrichment values for multiple cuts are evaluated inside a grid structure, named the *enrichment grid.* For each grid point, the corresponding enrichment value is calculated similar to the original distance function-based technique, though, instead of a multiplicative approach, each grid point is assigned to a specific cut segment region and its enrichment value is calculated with respect to this specific cut segment.

The first step in the enrichment grid algorithm is to define a *common coordinate system* for calculating the regions of the grid points. In 2D, this common coordinate system is defined by the enrichment origin $\mathbf{p_0}$ with coordinates (x_0, y_0) and θ_0, which is the angle between the

horizontal axis of the common coordinate system and the positive x-axis of the world coordinate system. The coordinate system is updated with each propagating cut as

$$\mathbf{p_0} = \frac{\sum_{l=1}^{n} w_l \mathbf{p}_l}{\sum_{l=1}^{n} w_l} \tag{39}$$

and

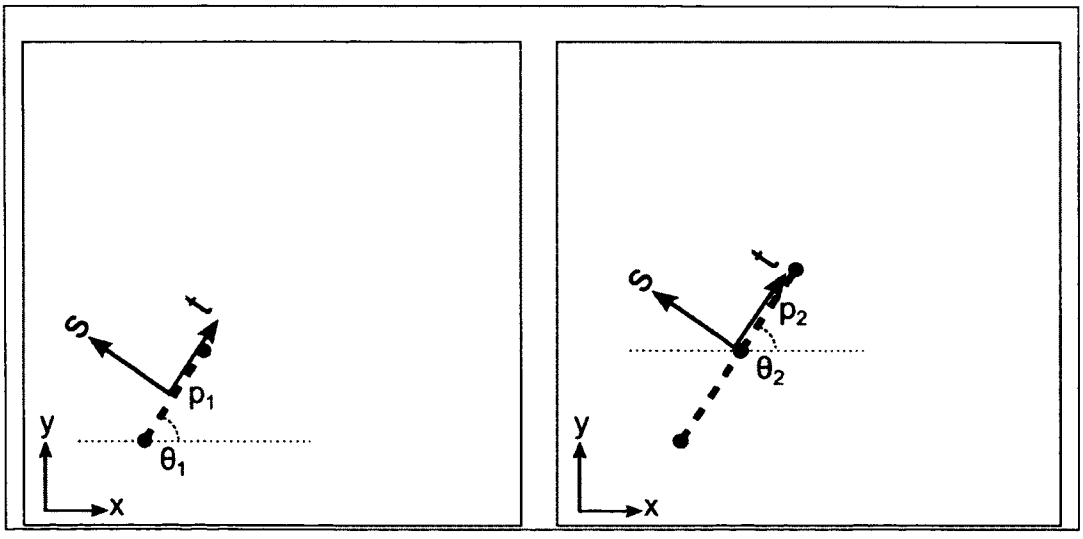
$$\theta_0 = \frac{\sum_{l=1}^{n} w_l \theta_l}{\sum_{l=1}^{n} w_l}, \tag{40}$$

where n is the number of segments, w_l is the associated weight with the cut segment l, which is typically the length of the segment in 2D problems, and $\mathbf{p}_l$ and θ_l are the center point and horizontal angle of the l-th cut segment respectively (Fig. 24). After setting the global coordinate system for the series of cut segments, each grid point with coordinates (x, y) as well as the endpoints of the cut segments (x_l, y_l) are translated into this new coordinate system to obtain new coordinates (t) and (t_l) by

$$t = \cos(\theta)\,(x - x_0) + \sin(\theta)\,(y - y_0) \tag{41}$$

and

$$t_l = \cos(\theta)\,(x_l - x_0) + \sin(\theta)\,(y_l - y_0). \tag{42}$$



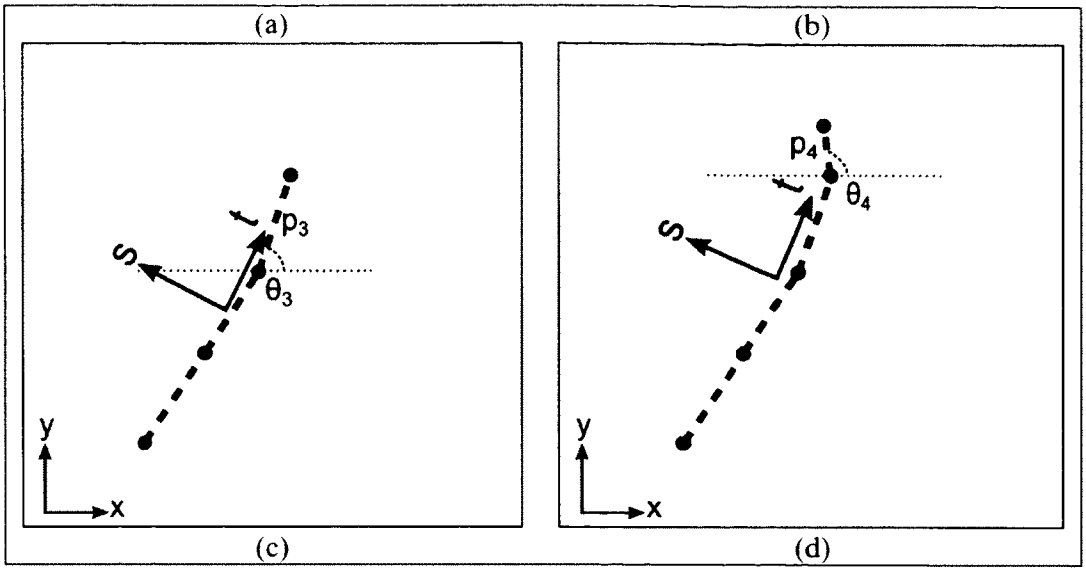
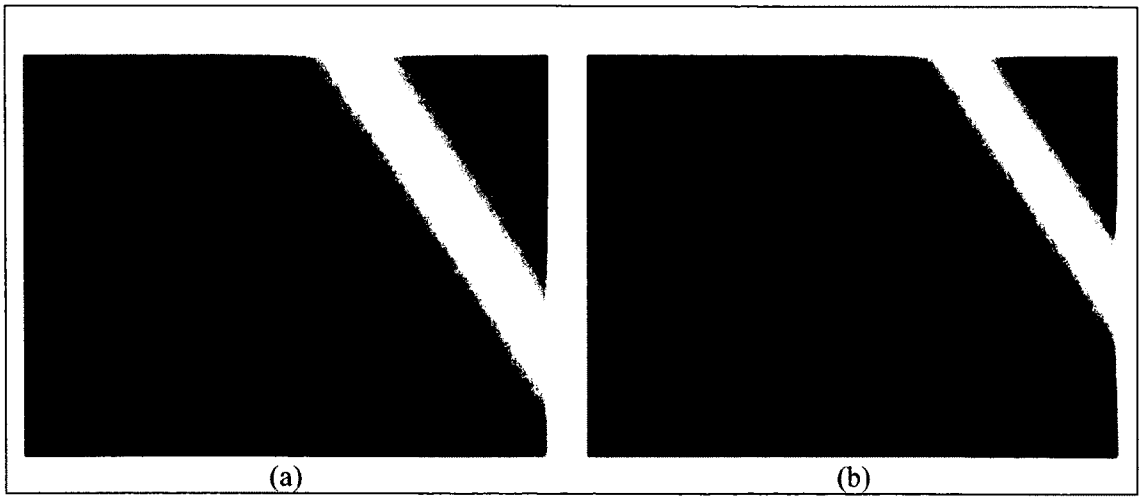


Fig. 24. The common coordinate system (s, t) is updated with each propagating cut segment (a-d).

With these translated points, the modified $d_1^+(t)$ function is now defined as

$$d_1^+(t) = (t_0 - t) \cdot H(t_0 - t) + (t - t_n) \cdot H(t - t_n), \tag{43}$$

where t_0 and t_n are the t-coordinates of the first and last points of the cut segment series, and H is the Heaviside step function (Fig. 25).



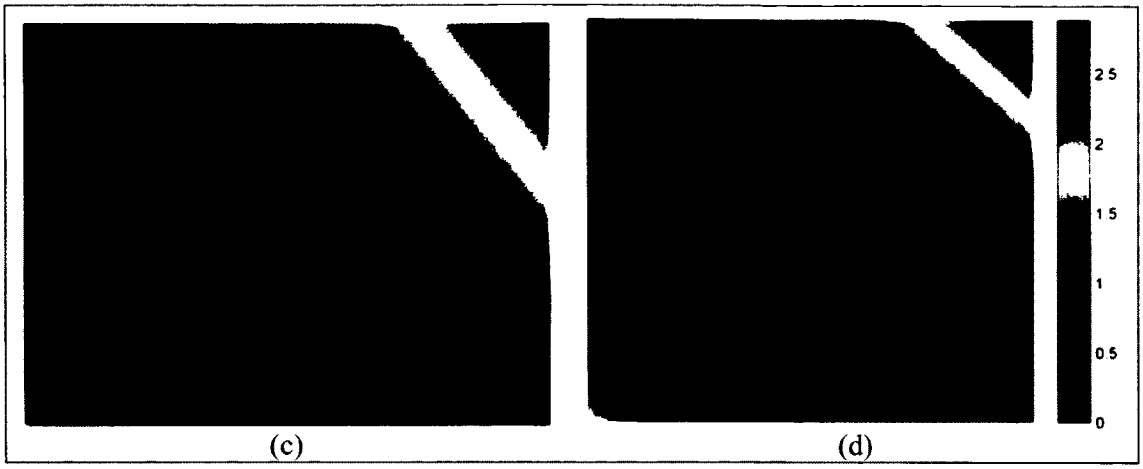


Fig. 25. Cut segments are processed as a series instead of individual processing. The series of cut segments (a-d) are used to derive a common coordinate system. The function d_1^+ is calculated with respect to this common coordinate system according to Equation (43).

The next step to calculate the distance function $d_2(x, y)$ is to set the s-coordinates of the grid points. This is achieved by assigning a cut segment region for each of the grid points by comparing their t-coordinates against the t-coordinates of the cut segment endpoints. A grid point with t-coordinate t' is set to be in the region l when $t' > t_{l-1}$ & $t' < t_l$. For grid points whose t-coordinates are smaller than t_0 and larger than t_n, their regions are set to the first and last regions respectively (Fig. 26).

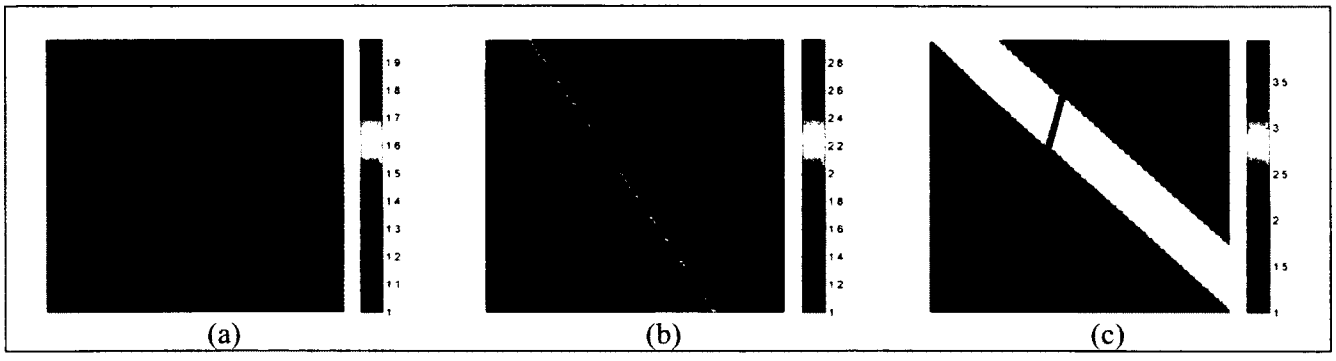


Fig. 26. The region values are assigned for individual cut segments with respect to the common coordinate system. (a-c) are the consecutive segments of the complete cut.

After assigning the region values for the grid points, the s-coordinates are calculated by finding the vertical distance of the grid point to the assigned cut segment. For a grid point with coordinates (x, y) and assigned region l, the s-coordinate (Fig. 27) is calculated as

$$s = -\sin(\theta_l)\,(x - x_l) + \cos(\theta_l)\,(y - y_l). \tag{44}$$

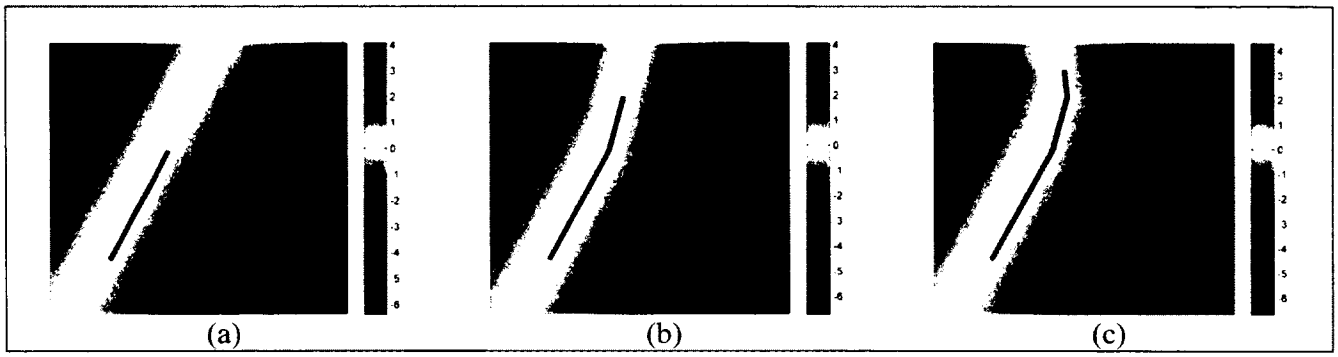
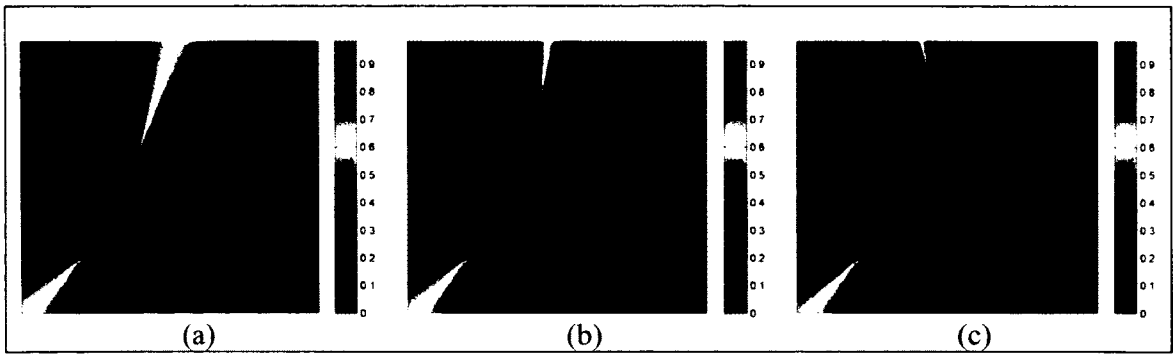


Fig. 27. For each grid location, the s-coordinates, which essentially represent the signed vertical distance of a point to the cut, are calculated according to the assigned region value.

The (t, s) coordinates of the grid points are enough to calculate the extended enrichment function. Using these values, $d_2(x, y)$ is calculated as in Equation (34) and its partial derivative with respect to the s-coordinates is taken to obtain the extended enrichment function (Fig. 28).



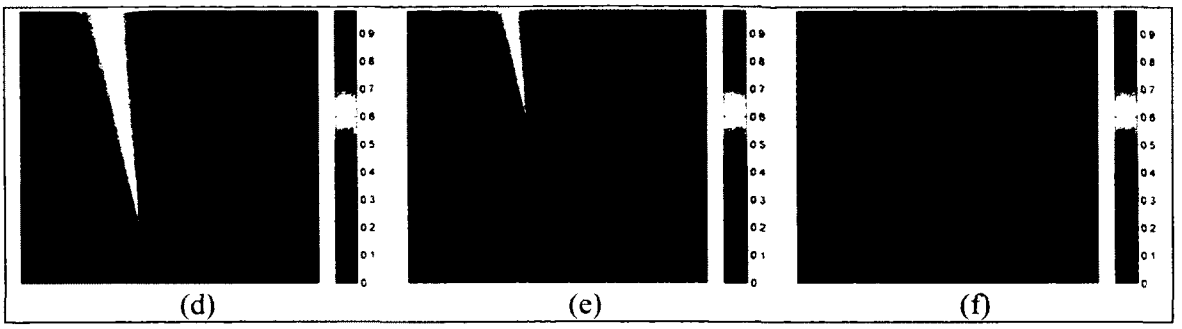


Fig. 28. The extended enrichment function for two cases of sequential cut segments. (a-c) Piecewise-linear cut segment and the corresponding updates to the enrichment grid. (d-f) Piecewise-linear cut segment that completely separates the domain. Unlike the original multiplicative approach, the enrichment function modifies the weight functions correctly to prevent instabilities.

The extended enrichment technique with enrichment grids provides computation cost-efficient way of handling multiple discontinuity fronts and unlike the multiplicative approach, it modifies the weight function of the affected meshless nodes in a way such that stability problems are avoided.

3.3 Modeling Discontinuities in 3D

In three dimensions, discontinuity-imposing cuts are represented by triangle strips. The enrichment grid is also extended into a 3D lattice structure with grid point coordinates (x, y, z). Similar to the 2D case, the first step is to derive a common plane for calculating the regions of the grid points. This plane is represented by the point $\mathbf{c_0}$ and the normal vector $\mathbf{n_0}$. For a cut front that is composed of n triangles, these values are calculated as

$$\mathbf{c_0} = \frac{\sum_{t=1}^{n} w_t \mathbf{c}_t}{\sum_{t=1}^{n} w_t} \tag{45}$$

and

$$\mathbf{n}_0 = \frac{\sum_{t=1}^{n} w_t \mathbf{n}_t}{\sum_{t=1}^{n} w_t} \tag{46}$$

where w_t is the associated weight of the triangle t, which is equal to the area of the triangle, and $\mathbf{c}_t$ and $\mathbf{n}_t$ are the centroid and normal of the t-th triangle respectively. Each grid point (x, y, z) and the vertices of the triangle strip are projected onto this common plane for further processing as illustrated in Fig. 29.

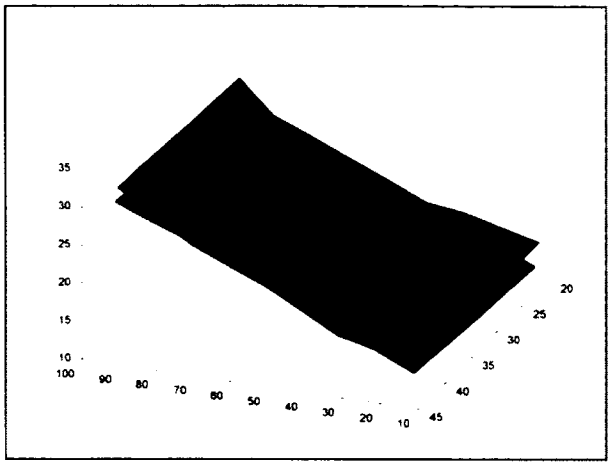


Fig. 29. The original triangle strip that defines the cut surface (red), and the projected triangle strip (blue). The triangles of the blue triangle strip are coplanar and on the common plane.

The two projected triangles that are adjacent to each other are grouped together to define their respective regions. *Separator vectors* $\mathbf{n}'_r$ are located at the boundary of the adjacent groups of triangles, and obtained by taking the cross product of the border vectors $\mathbf{b}'_r$ and the normal $\mathbf{n}_0$ of the plane that they are projected on, as shown in Fig. 30. Each projected grid point is then tested against the separator vectors $\mathbf{n}'_r$ to see whether the vector is facing towards the point. This test is implemented by checking the sign of the dot product of $\mathbf{n}'_r$ and the vector from the origin of $\mathbf{n}'_r$ to the tested point. A positive dot product means that the $\mathbf{n}'_r$ faces the projected grid point. If the vector $\mathbf{n}'_r$ faces towards to projected grid point, the region of the point is set to the index of

the corresponding separator vector $\mathbf{n}'_r$, which is r. The region assignment for each grid point is visualized in Fig. 31.

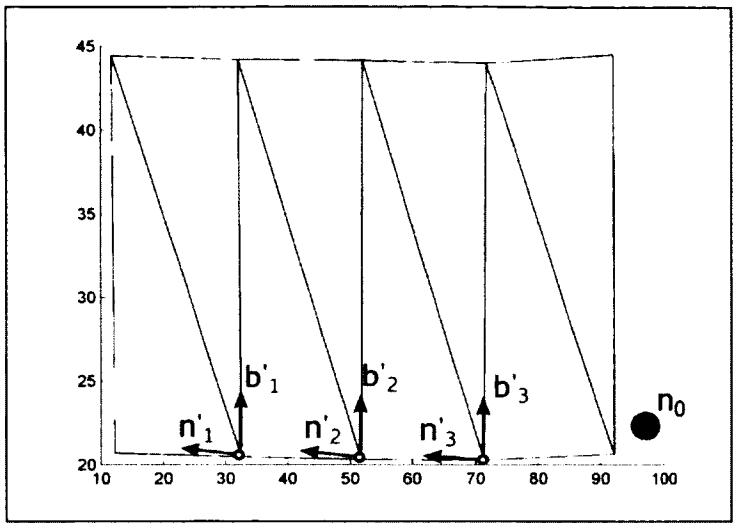


Fig. 30. Separator vectors $\mathbf{n}'_r$ are calculated at the boundaries of the triangle groups. Each projected grid point is then tested against these to find their corresponding regions.

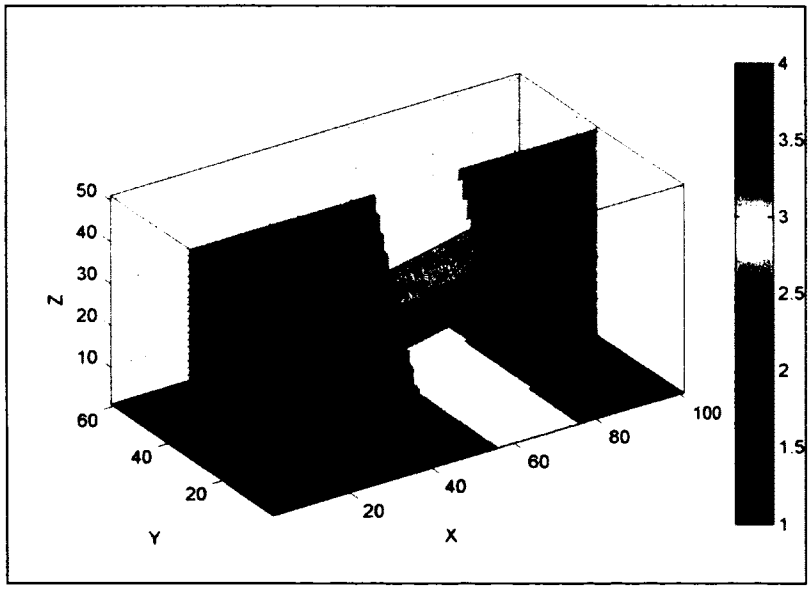


Fig. 31. The regions of 3D grid points are set by first projecting them onto the common plane and then testing them against the separator vectors.

Once the regions are set for the grid points, the next step is to calculate enrichment-related values. In three dimensions, the distance function that forms the basis of the enrichment technique is defined as

$$d_3(x,y,z) = \sqrt{d_2^{+2} + s^2}, \tag{47}$$

where d_2^+ is the absolute distance of the projected grid point to the projected triangle that returns positive values outside of the triangle and 0 inside the triangle, and s is the distance of the grid point to the triangle along its normal direction.

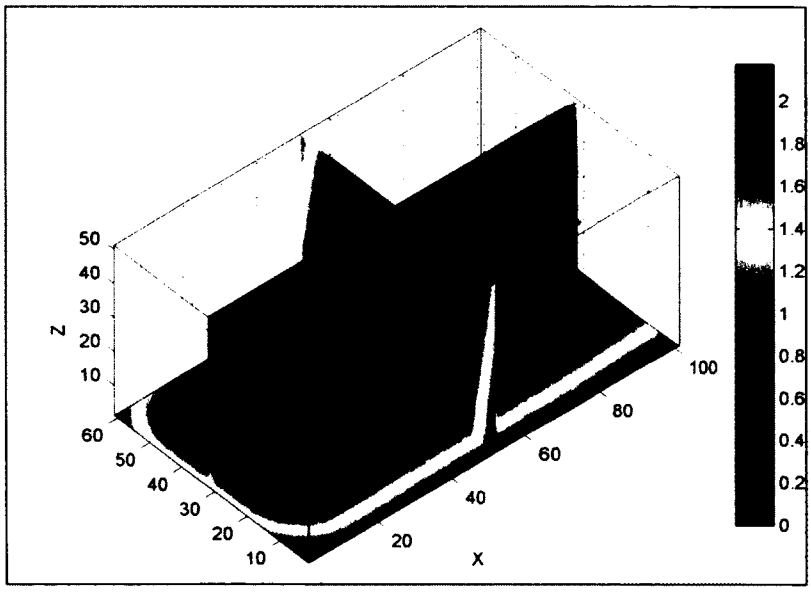


Fig. 32. The d_2^+ distance function computed for the cut surface.

Because, each region is associated with two triangles; for a projected grid point that is assigned to a region, there are two candidate triangles to calculate d_2^+ and s values. One of these triangles has to be chosen and designated as the master triangle in order to correctly calculate the necessary enrichment values. For a projected grid point inside the region r, this is achieved by comparing the absolute distance d_2^+ of this point to the two triangles that are associated with the

r-th region. The triangle with the smallest absolute distance is set as the master triangle and used to compute the d_2^+ (Fig. 32) and s values (Fig. 33).

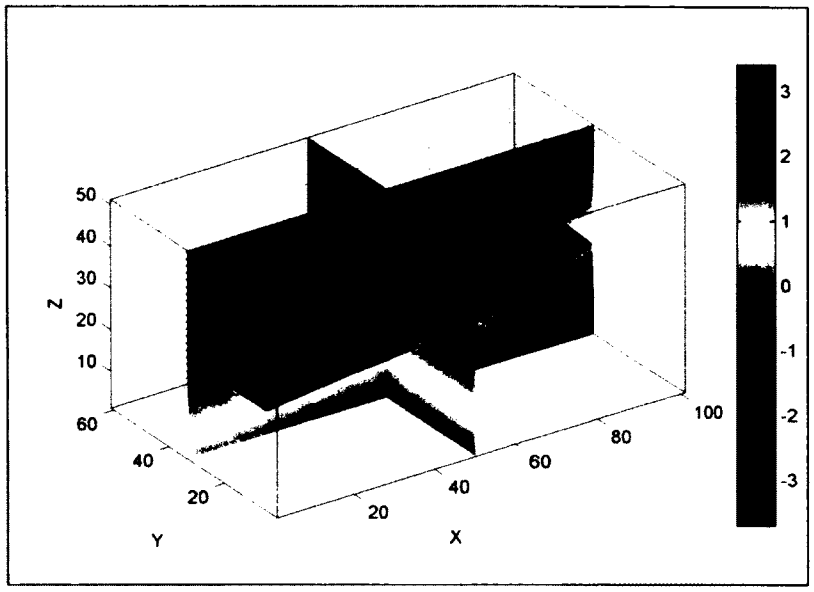


Fig. 33. The computed s values for the grid points with respect to their master triangles.

The computed d_2^+ and s values are used to obtain the resulting distance function d_3 as given in Equation (47). Similar to the 2D case, the partial derivative of d_3 with respect to the normal direction s is taken to get the discontinuous function (Fig. 34, Fig. 35, Fig. 36, Fig. 37)

$$\phi = \frac{\partial d_3}{\partial s} = \frac{s}{d_3}. \tag{48}$$

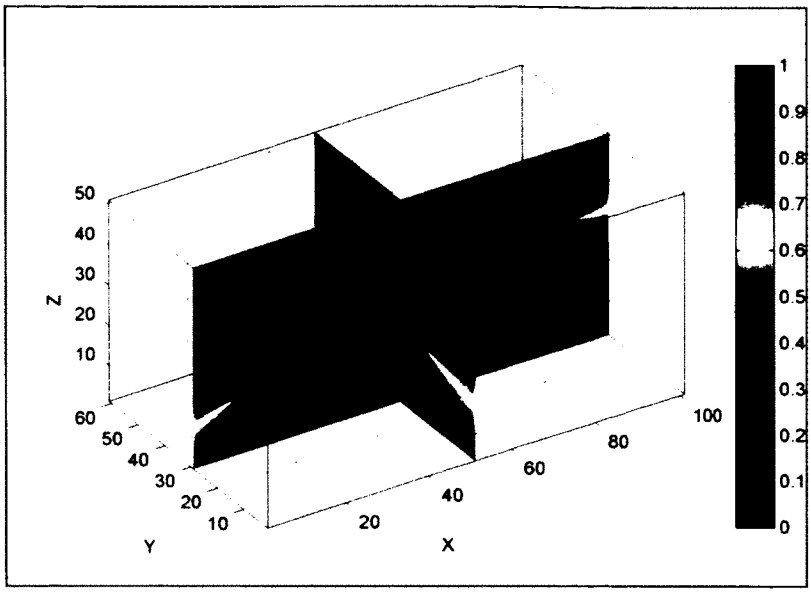


Fig. 34. The contour plot of the discontinuous function ϕ (view 1).

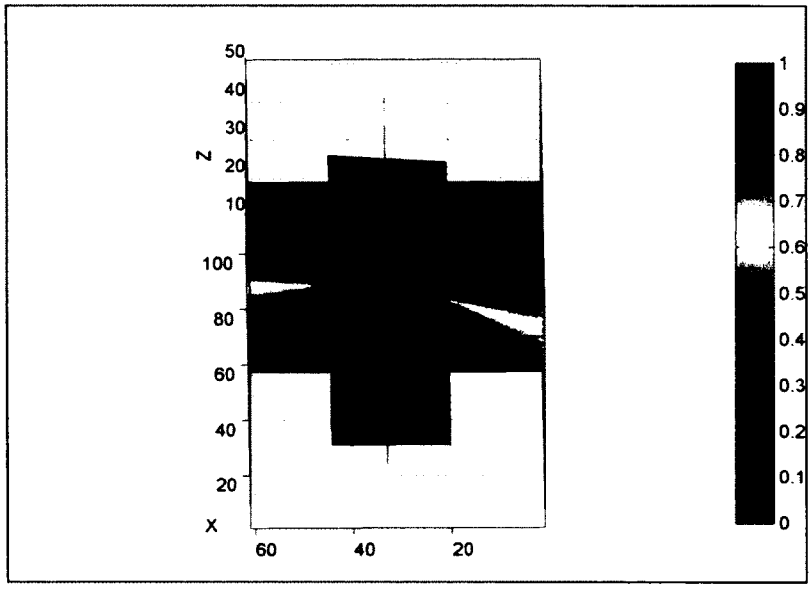


Fig. 35. The contour plot of the discontinuous function ϕ (view 2).

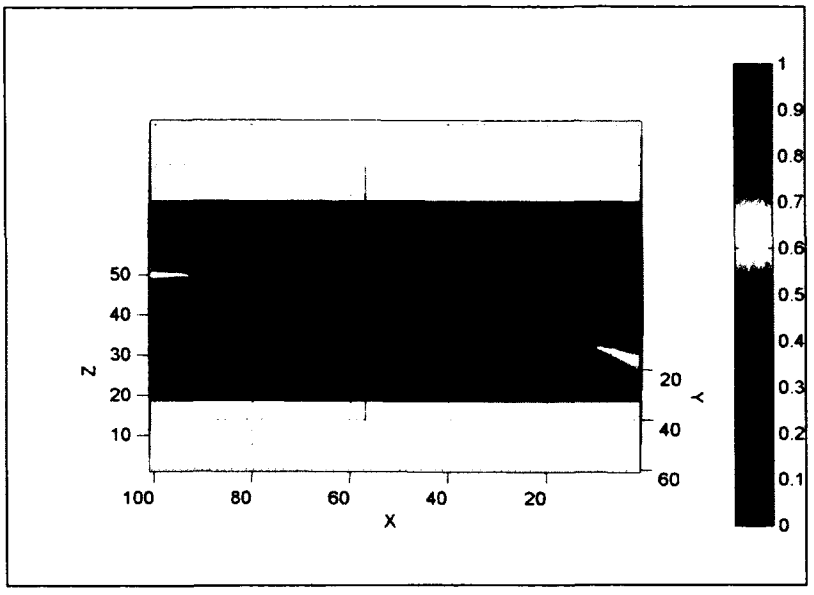


Fig. 36. The contour plot of the discontinuous function ϕ (view 3).

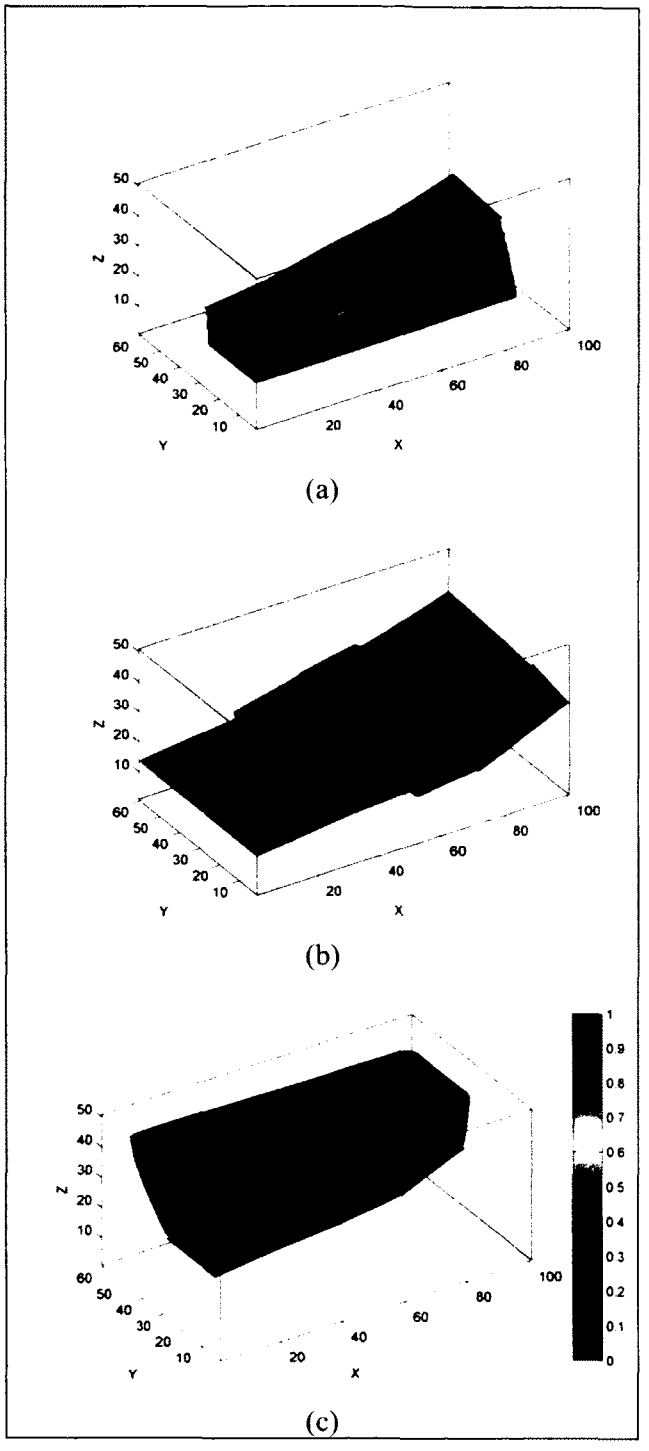


Fig. 37. The iso-surfaces of the calculated enrichment function for the grid, (a) 1-level-set, (b) 0.5-level-set, and (c) 0-level-set.

3.4 Summary

Discontinuities are frequently found in engineering problems and they have a variety of causes, such as different material types or stronger discontinuities like fissures or fractures in the domain. In a surgical simulation setting, discontinuities typically take the form of cuts that are introduced by the user through haptic interaction. In this chapter, the approach on handling the discontinuities that were built upon the meshless method that was discussed in the previous chapter was presented. A previous technique was extended and the enrichment that is caused by a cut in two and three dimensions was calculated. Cuts that are composed of multiple segments were handled in a novel way to avoid instabilities in the simulation. This approach is a completely analytic solution that can be efficiently computed and applied to the simulation in real time. Moreover, the proposed enrichment grid doubles as a spatial data structure that proves to be efficient for the intersection tests of the meshless nodes that are affected by the cuts. Implementation details of the enrichment grids are discussed in Chapter 5.

CHAPTER 4

VISUALIZATION OF POINT-BASED MODELS

Realistic and efficient rendering of simulation objects is essential in a surgical simulation application. For visualization purposes, simulation objects are usually represented as triangle meshes. In the point-based approach used in this dissertation however, the point primitives that represent the visualized surface of the object do not have explicit connectivity information. Without this knowledge of the connectivity of the vertices, or in other words a mesh, the traditional algorithms that work with visualization meshes become dysfunctional, which require the use of specialized algorithms. In this chapter, a discussion is provided on techniques in various categories to visualize point-based objects, and then the approach to this problem is presented and the steps that have taken to improve the visual quality of the rendered simulation object are explained.

4.1 Overview

Visualization of point-based object representations can be divided into two main categories. The first, the so-called *direct point rendering* methods, uses the point primitives directly without inferring any connectivity information, while the second includes techniques that reconstruct a surface from the discrete set of points and use that information to display the object. Surface reconstruction techniques can be further divided into two categories, which are direct triangulation of the point data through computational geometry, and triangulation of the zero-level-set of the fitted implicit function into a mesh.

4.2 Surface Reconstruction Techniques

The Ball-Pivoting Algorithm [46] is a simple algorithm to construct a triangle mesh for a given point cloud. It is a computational geometry algorithm that was originally developed for triangulating point clouds acquired from range scanners. The algorithm accepts as input a point cloud and a parameter d that denotes the radius of the pivoting ball. It first finds a seed triangle such that a ball with radius d touches the three vertices of the triangle but does not contain any other points in the point cloud. From this point, the algorithm chooses one edge of the triangle as the pivoting axis and rotates the ball around this axis until it touches another point (Fig. 38). The pivoting edge and the new point form a new triangle and the algorithm repeats until termination. The conceptual simplicity of the algorithm and the fact that it only requires a single parameter d, make this approach easy to use and implement, but, unless the single parameter is chosen carefully, some of the features of the object in the vicinity of high curvature may be lost. Another shortcoming of this technique is that if the point cloud has been composed of points with varying density, the algorithm would result in a mesh with holes, or would need to perform multiple passes with increasing ball radii.

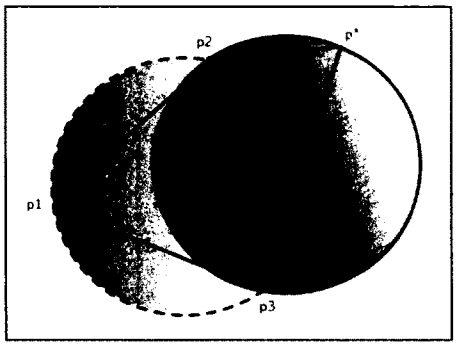


Fig. 38. The Ball-Pivoting algorithm starts from a seed triangle and a sphere that touches all three vertices. The algorithm advances by pivoting the sphere over a chosen edge and stop the sweep when the sphere touches another point.

Another algorithm that fits a triangle mesh to the input points is the Power Crust algorithm [47]. This algorithm guarantees a water-tight mesh that is produced from the approximated medial axis transform of the object represented by point-cloud. Power Crust makes use of a Voronoi partition of the point data, specifically the concept of Voronoi balls, because the object is approximated as the union of them. Power Crust is a robust algorithm in the sense that it is guaranteed to return the boundary of the object and no surface extraction or hole-filling steps are required. Although the reconstruction tolerates noise in the data, sampling should be sufficiently dense.

The other class of surface reconstruction techniques first processes the point-cloud to obtain an implicit function that best fits the data, and then extract and triangulate a certain iso-value of this function. Poisson surface reconstruction [48] is one such method that reconstructs a smooth watertight surface from a set of oriented points – points p_i with normal information n_i. This algorithm reconstructs the surface by solving for the indicator function of the shape, which is defined to take the value 1 inside the boundary of the object and the value 0 outside. The gradient of the indicator function is closely related to the vector field that is defined by the oriented point set and this relation can be transformed into a Poisson problem. The algorithm first approximates a vector field by sampling it at the cells of the generated octree. Next, the divergence operator is applied to the vector field, which gives the right hand side of the Poisson equation. The resulting equation is numerically solved using the multigrid method by utilizing the previously generated octree structure. The solution gives the gradient of the indicator function, whose zero-level-set is extracted to obtain the final surface. The surface reconstructed with this algorithm is robust to noise and can handle large sets of points, but it also over-smoothes the original shape. The over-smoothing problem of the technique is addressed in a later

extension, the so-called Screened Poisson surface reconstruction [49]. This algorithm interpolates the samples better than the original approach with the cost of increased sensitivity to noise.

There are other techniques with a conceptually similar idea i.e., build an implicit function from the given point-cloud and extract its zero-level-set [50-52]. Unfortunately, the level-sets of an implicit function are not directly usable by the graphics hardware as they need to be converted to primitives that can be processed in the graphics pipeline. This problem is essentially known as polygonising a scalar field, or 3D contouring. The seminal work of Lorensen and Cline [53], the Marching Cubes (MC), describes an algorithm for creating a polygonal surface of an isosurface of a 3D scalar field. The algorithm divides the domain into cubical regions, where the corners of each cube are assigned a scalar value from the field. If the cube contains two or more corners that are on the opposite sides of a user-defined iso-value, the cube contains a part of the surface. For each such cube, the intersection of the surface with the cube's edges are computed and the intersection points of the neighboring cubes are stitched together to obtain the triangulation of the level-set of the given scalar field (Fig. 39).

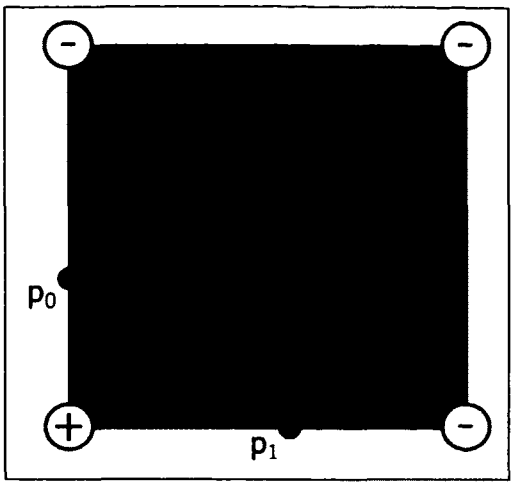


Fig. 39. The Marching Cubes algorithm finds the intersection points of the surface with the cubes and stitches a triangular surface out of these intersection points.

The MC algorithm is easy to implement and also efficient as it is based on table look-ups. One shortcoming of the algorithm is that it smoothes out sharp features. Dual Contouring (DC) [54], although being conceptually similar to the MC, is a triangulation algorithm designed to alleviate this problem. In addition to the scalar values, this algorithm also needs surface normal information or the gradient of the scalar function itself. Instead of just using edge-surface intersection points for triangulation as MC does, DC finds points inside the cubes by minimizing the error metric

$$E(\mathbf{d}) = \sum_{i=1}^{k} ((\mathbf{d} - \mathbf{p_i}) \cdot \mathbf{n_i})^2 \tag{49}$$

where $\mathbf{d}$ is the point inside the cube, k is the number of intersected edges, and $\mathbf{p_i}$ and $\mathbf{n_i}$ are the location and normal of the i-th edge intersection (Fig. 40).

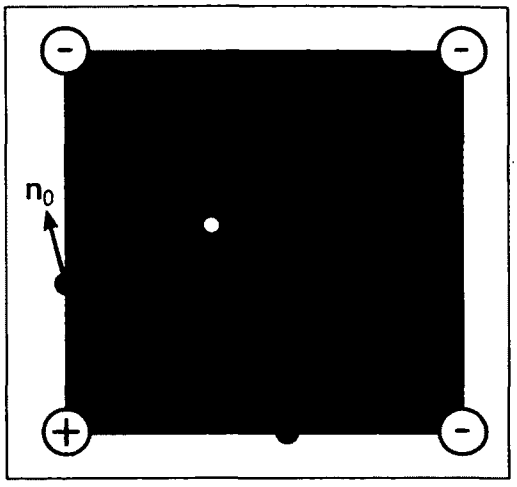


Fig. 40. The Dual Contouring algorithm uses the surface normal information to find a point inside the cubical volumes that minimizes a given error metric. These points are used to stitch triangular surfaces.

Another way of visualizing the level-sets of implicit functions is through ray casting/ray tracing, where individual rays are cast for each pixel of the image plane and these rays are traced throughout the scene that is visualized. Ray tracing and its variants generate highly-realistic images as the rays are reflected and refracted according to a physical basis. The application of ray tracing operation on simple implicit shapes such as spheres, blobs, and other general implicit surfaces are described in the work of Hart [55], and later on ported to massively parallel graphical processing unit (GPU) architecture by Singh and Narayanan [56]. The work of Hadwiger et al. [57] presented an interactive ray casting method for discrete isosurfaces defined by a regular volumetric grid. Deferred shading is applied in this approach, which means that the data required for the advanced shading computation is gathered into an intermediate render buffer and later on processed later on to obtain the final composition of the image. The implicit function is enclosed by two distinct grid structures with different refinement levels in order to efficiently skip the empty space during the ray casting operation and use as a brick caching mechanism in case the whole model does not fit into the graphical processing unit's texture

memory. The implicit function values are evaluated at the discrete locations along the casted ray to find the ray-surface intersection point. This approach can be problematic near fine-detailed surface features as the discrete evaluations can completely miss the surface. The authors addressed this issue by adopting an adaptive sampling approach to refine the intersection detection.

4.3 Direct Point Rendering Techniques

Instead of inferring the topology and reconstructing the surface that represents the point-cloud, direct point rendering techniques work on the raw point primitives, thus avoiding the time-consuming reconstruction processes. The use of points as display primitives was introduced by the seminal report of Levoy and Whitted [58]. Other research has followed such as the work of Schaufler and Jensen [59], in which the well-established ray tracing algorithm is extended to work directly on the point primitives. The ray-point intersection is handled by representing the ray as a cylinder and the points as oriented disks with same radii. The common radius r of the cylinder and the disks is chosen such that r is slightly larger than the radius of the largest hole in the point-cloud sampling. For a cast ray, the set of disks that intersect with the cylinder enclosing the ray are collected. The final intersection location and orientation between the ray and the point-cloud is approximated with a weighted average of the points in this set. The weighting coefficients of the average point are chosen to be the tangential distances of the points to the ray. For more advanced illumination models, it is possible to assign additional attributes to the individual points and interpolate these attributes using the same weighted averaging technique. Although this technique generates high-quality visualizations, the resulting surface

representations are view-dependent, which may be acceptable for single renderings of static scenes, but can be problematic for rendering sequential images of an animation.

Adamson and Alexa [60, 61] apply the ray tracing algorithm to a point cloud from a different approach. In a manner similar to the Moving Least Squares approximation method presented earlier, each point is augmented with a spherical domain of influence, which is used to define a neighborhood around the point. Then, for each point, the weighted average of the neighboring point locations and orientations are used to approximate a plane. The ray-surface intersection is handled in two steps; first a starting point is approximated using a bounded volume hierarchy, either spherical volumes or oriented bounding boxes, then the actual intersection point is calculated with an iterative procedure.

4.4 Point-Based Visualization with Surface Splatting

As opposed to the ray tracing algorithms discussed earlier, surface splatting is a forward rendering approach that uses the point depth order information to resolve the visibility of the object portions that share the same pixel on the image plane. As a result of this forward rendering approach, point splats need to be rasterized - mapped to the image plane - and shaded. There are several approaches to rastering point splats and they address different requirements of the task at hand.

In the OpenGL rendering pipeline [62], splats are not directly represented as a native primitive type. Therefore, they need to be approximated with the supported types such as points, triangles, or patches. A naïve approach is to represent the splats as OpenGL point primitives. Passing a single point primitive for each splat is efficient and compact. For each vertex, the vertex shader in the rendering pipeline can be set to output the size of the point as an optional

parameter. When set to s, the point size parameter results in $s \times s$ squares in the image plane. The point size parameter has to be carefully adjusted for the splats in order to cover the represented surface without leaving any holes (Fig. 41).

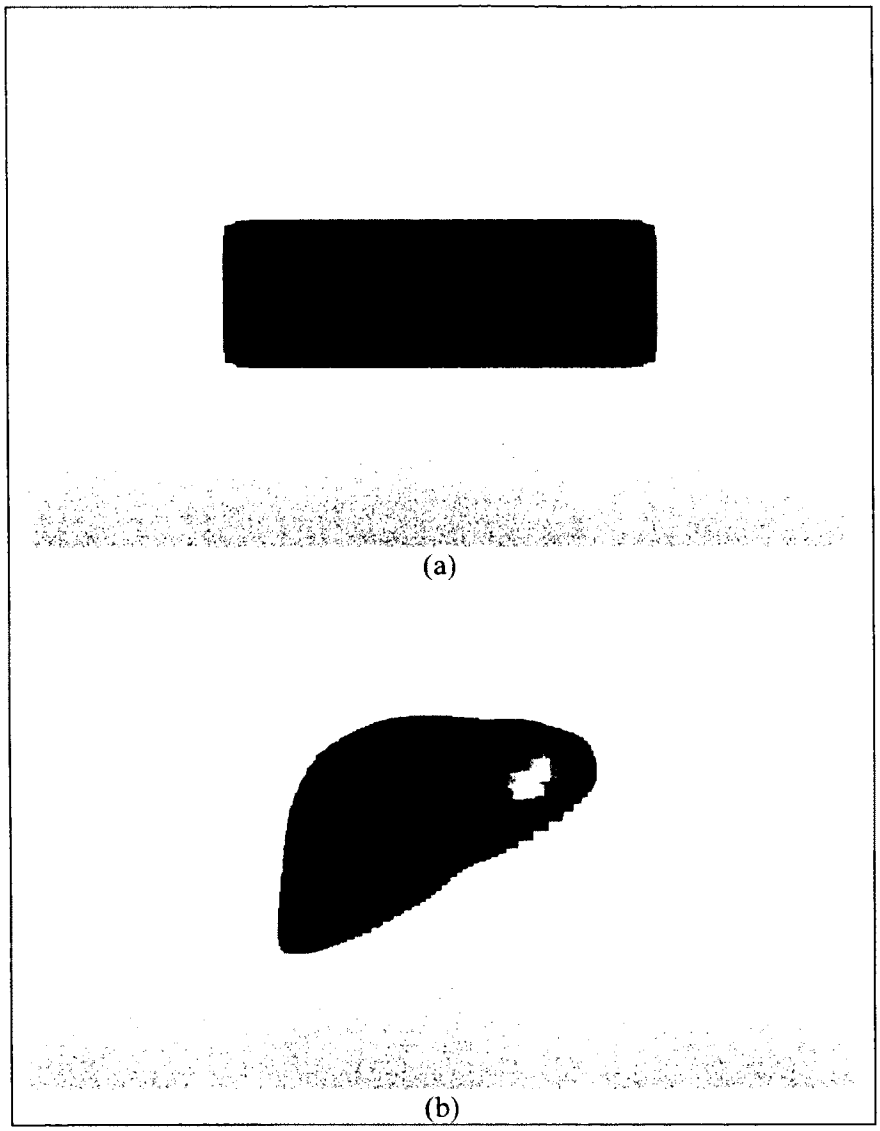


Fig. 41. Direct rendering of points with OpenGL point primitives as screen-aligned squares with adjustable size. (a) a rectangular block, and (b) a liver model rendered with point primitives.

In supporting graphics hardware, it is also possible to draw screen-aligned disks instead of squares by enabling the anti-aliasing filter through the OpenGL call (Fig. 42):

```
glEnable(GL_POINT_SMOOTH);
```

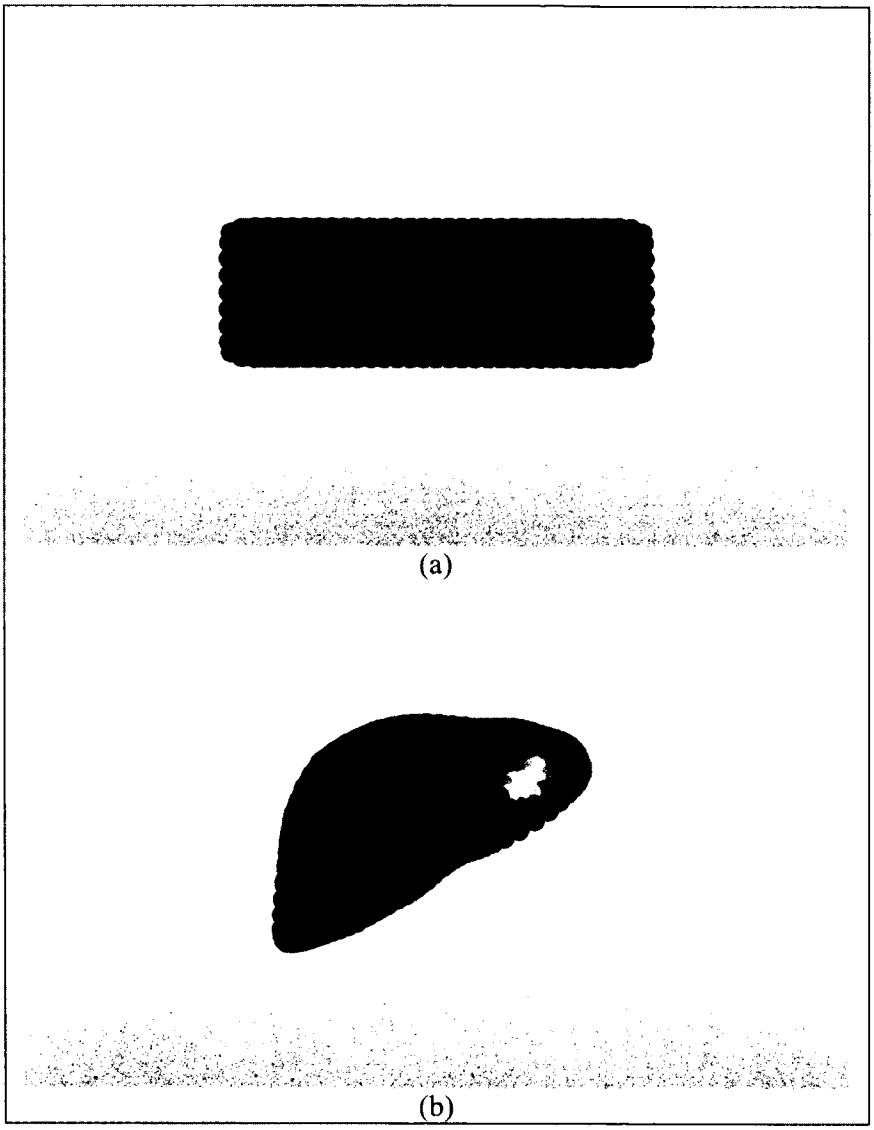


Fig. 42. On supporting graphics hardware, it is possible to render point primitives as screen-aligned disks.

Although rendering splats as point primitives is extremely efficient, they result in poor visual performance, especially near the object's silhouette. When using the specular lighting model, the overlapping point primitives also pose a problem near highlighted areas. In order to

address the aliased look near the object's contour, splats can be visualized as oriented disks. The orientation of the disks can either be taken directly from the normal information of the point cloud, or approximated by applying differential geometry techniques. The splats are then rendered on polygonal primitives by mapping an elliptical alpha texture onto them (Fig. 43).

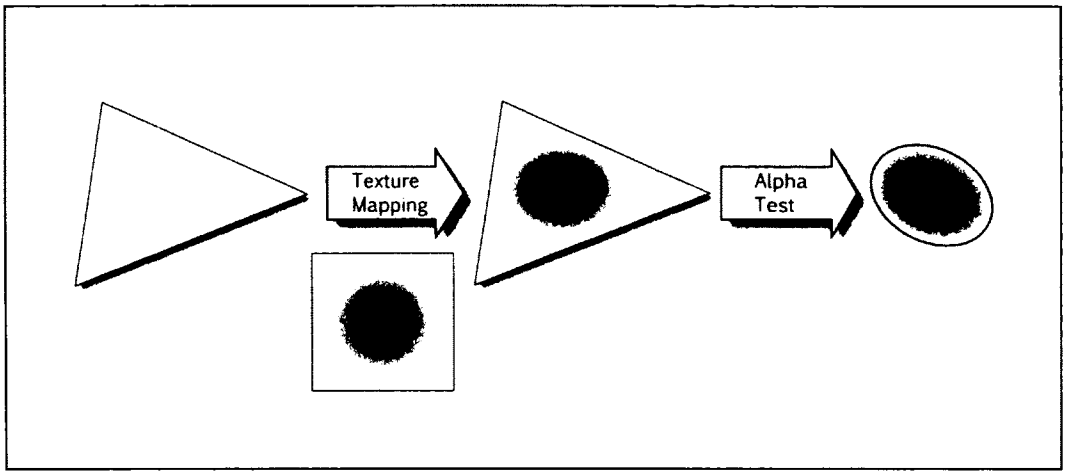


Fig. 43. Oriented point splats can be generated by applying an alpha texture on a polygon.

In conjunction with this alpha texture, the render pipeline alpha test is set to discard the pixels that are smaller than or equal to a cut-off alpha value, which is typically zero. This operation results in splats rendered as oriented and alpha-mapped disks. Here, the lighting is computed per splat and the colors are blended through the alpha values obtained from the alpha maps. A custom weight function is used that ranges from 0 to 1 to represent the splat alpha maps. This function is defined as

$$\mathrm{bf}(u,v) = \cos(\frac{\pi}{2}\sqrt{u^2+v^2}) \tag{50}$$

where u and v are the horizontal and vertical texture coordinates respectively and they both vary from 0 to 1. This function represents the weights of a splat's color contribution to its neighboring

pixels, and also defines how to obtain the final color of a pixel if there are more than one splat that land on that pixel (Fig. 44).

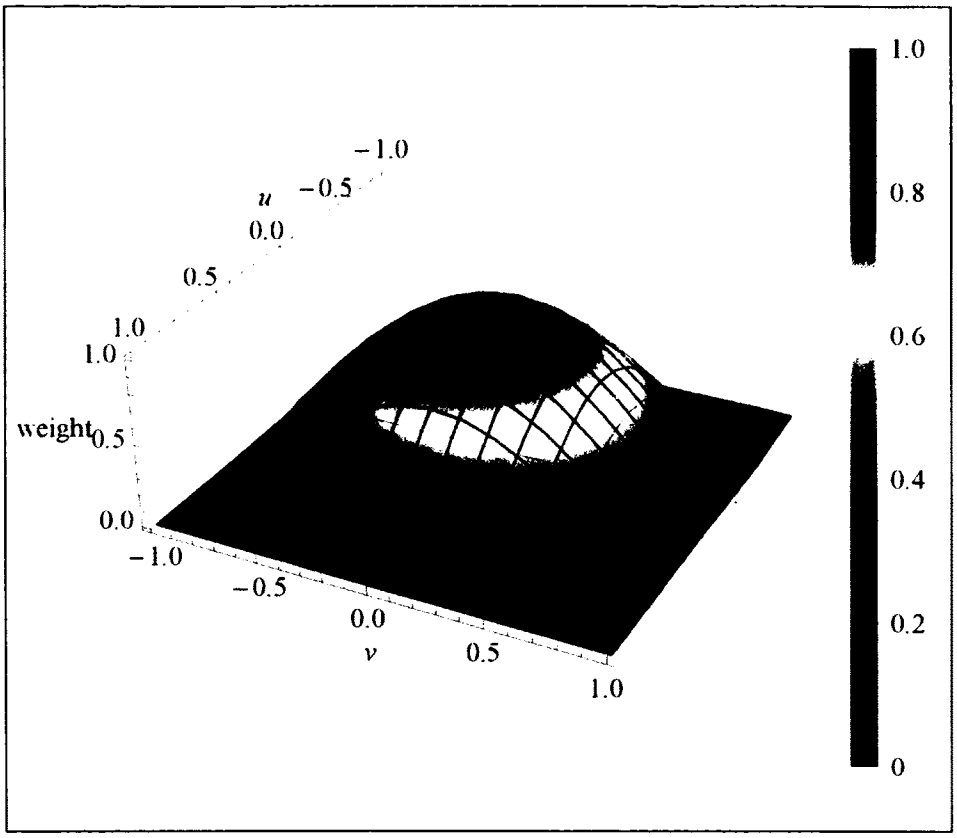


Fig. 44. The weight function is encoded as an alpha texture and applied to polygons to create oriented splats. The weight values obtained from this function are used to blend the colors that land on same pixel location.

In graphics libraries that support pixel blending, individual functions that specify the pixel arithmetic can be assigned to the incoming (source) pixels and the pixels that are already in the frame buffer (destination). OpenGL library's `glBlendFunc` function specifies how the pixels that land at the same location are blended together to obtain the final color at that location. The function accepts two parameters, one for the source factor (*sfactor*) and another one for the destination factor (*dfactor*). The incoming RGBA value is scaled with *sfactor* and the RGBA value in the frame buffer is scaled with *dfactor*. The final color of the pixel is then obtained by

adding these two together. In the implementation for this dissertation, *sfactor* was chosen to be the *alpha value of the incoming pixel* and *dfactor* to be (*1 - alpha value of the incoming pixel*) (Fig. 45 - Fig. 46).

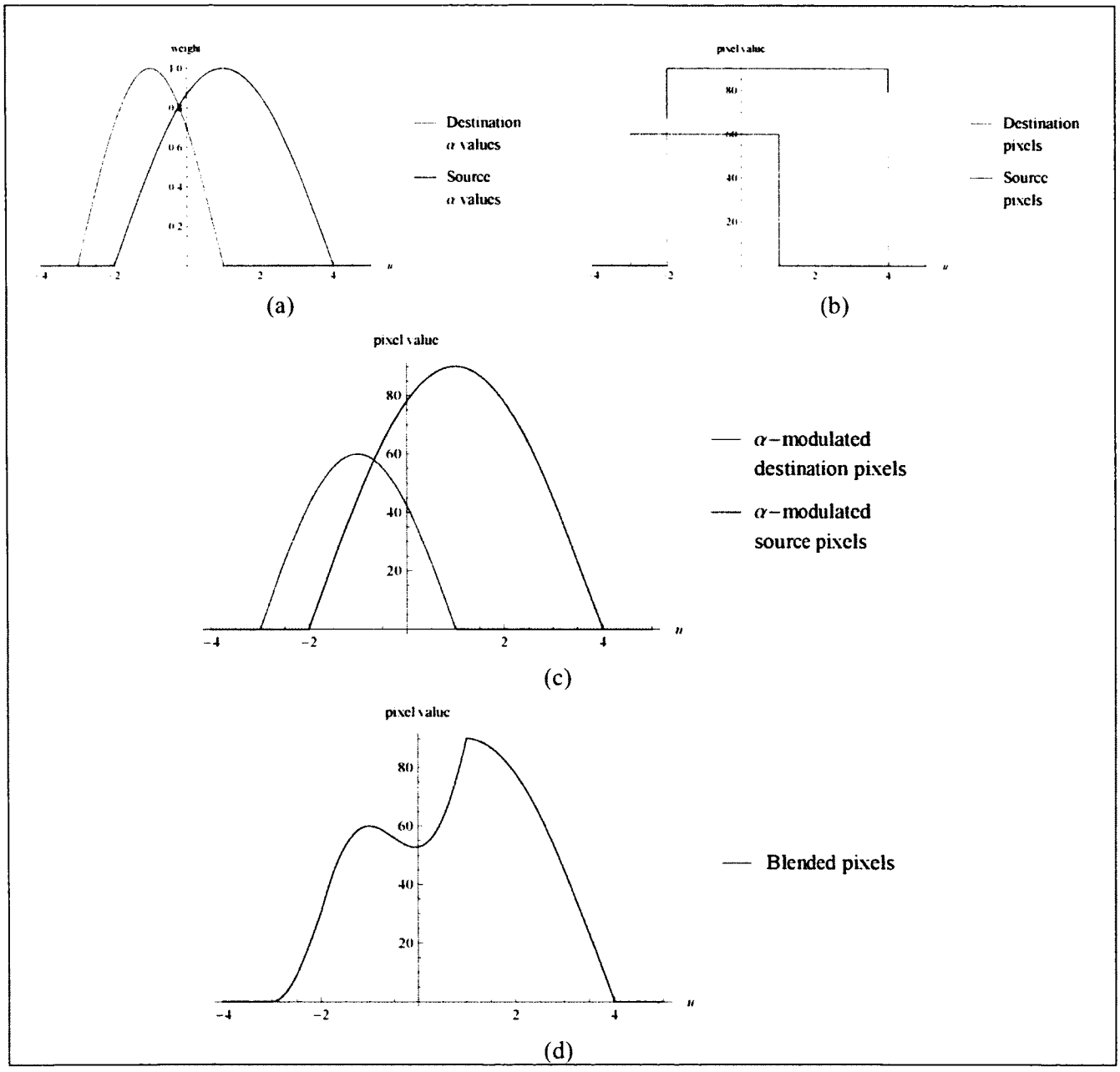


Fig. 45. Pixels of multiple splats can land on the same location in the frame buffer. In this case, pixels of the splat that are already in the frame buffer (destination pixels) and pixels of the incoming splat (source pixels) are blended.

(a) Each splat has an associated blend weight function in the form of an alpha texture. (b) The original and (c) alpha-modulated pixel values of the splats to be blended. (d) Blended pixel values.

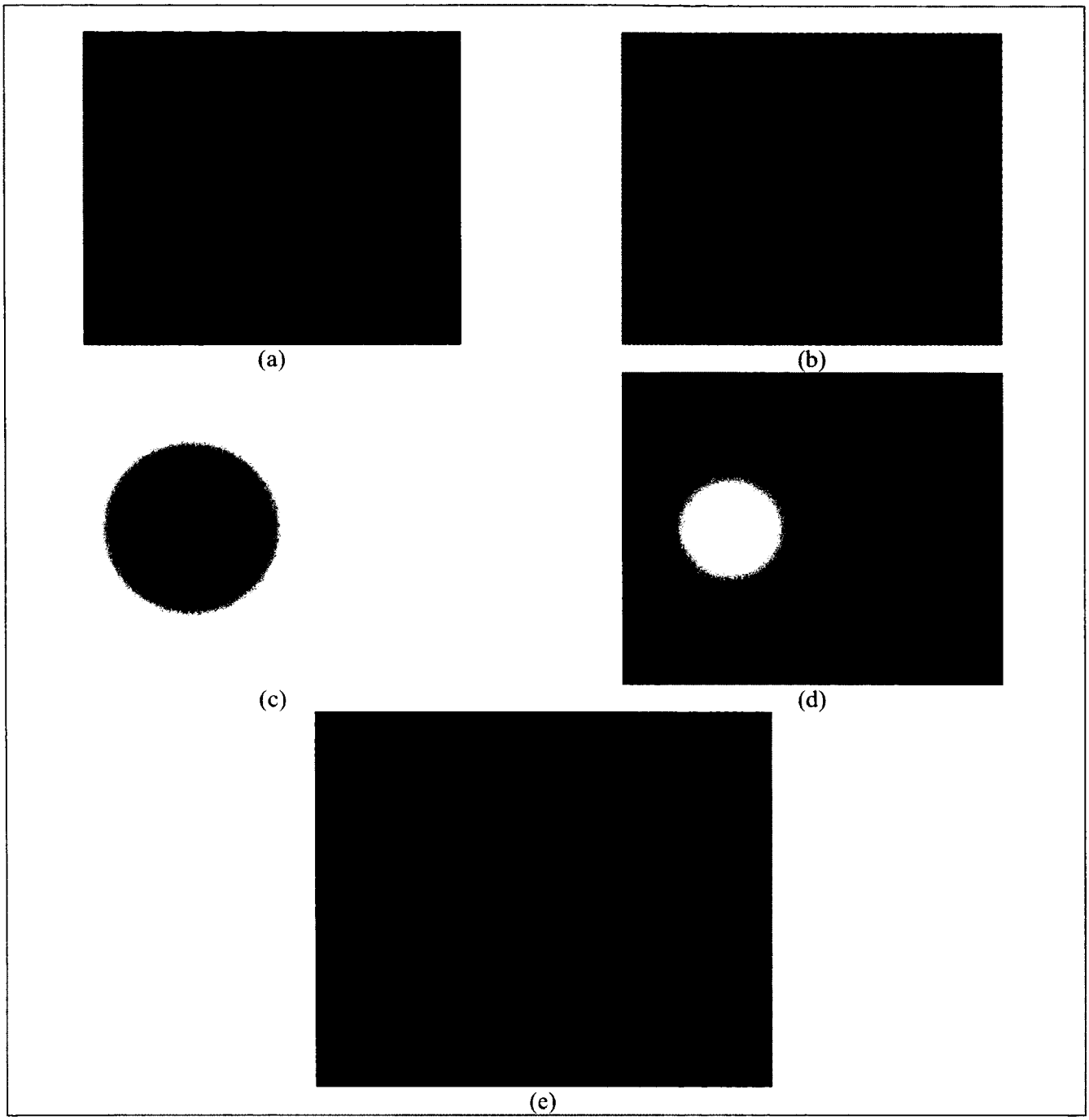


Fig. 46. The alpha value-based color blending is implemented in several steps. (a) The frame buffer contents initially, (b) the incoming color buffer from the new splat, (c) and (d), the (*1-source alpha*) and *source alpha* masks respectively, (e) the resulting frame buffer after the blending operation.

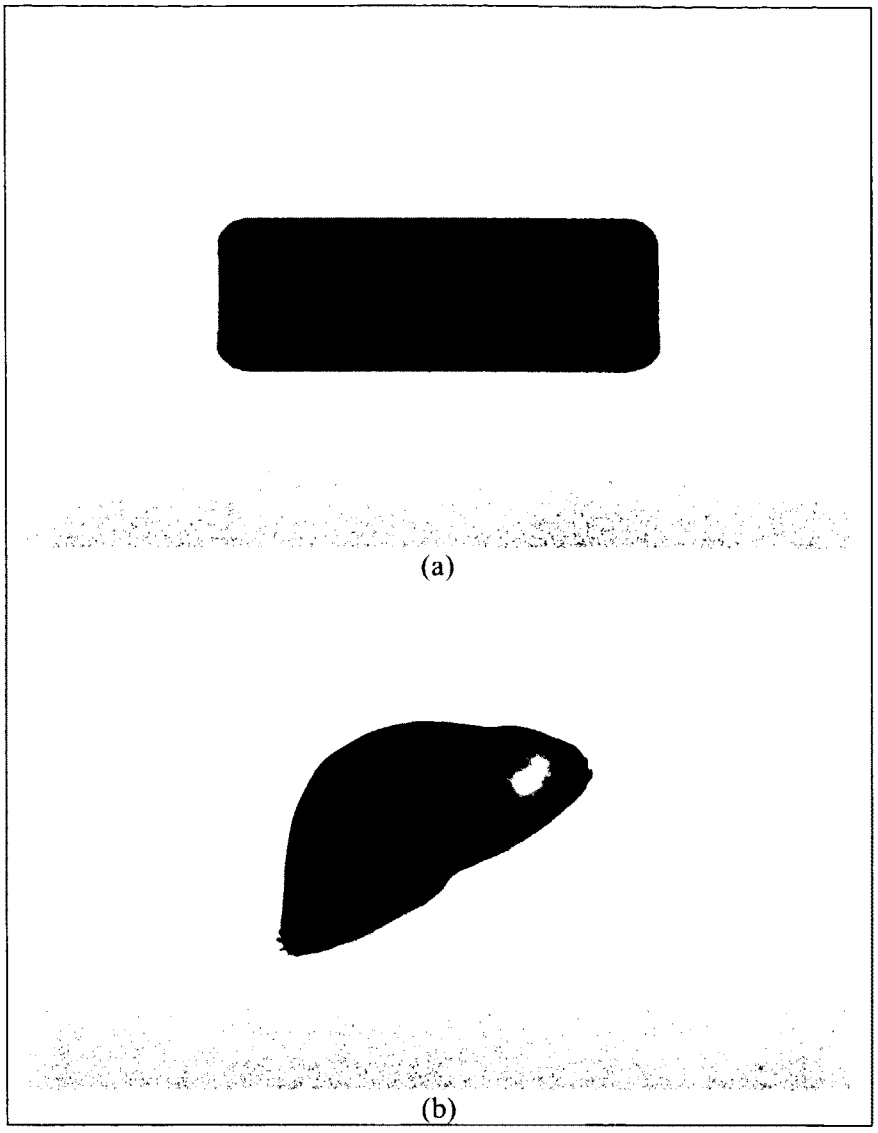


Fig. 47. Direct point rendering results with alpha-blended point splats that are oriented according to the given point normal information.

The surface splatting technique described here is an efficient single pass rendering technique. However efficient it is, the visualization quality can still be improved (Fig. 47). One approach would be to improve the visual quality through a multi-pass rendering scheme. The QSplat [63] algorithm implemented the surface splatting with 3 passes. First, the splats were rendered to the depth buffer, thus storing the depth information of the individual splats. In the next pass, the blending pass, lighting was computed for the splats and their colors were blended

through additive alpha blending as described above. Different from the standard rendering procedure, the blending pass did not update the depth buffer, instead, it added a small depth amount to splat depth values and compared the output of the visibility pass to these modified depth values. Finally, the RGB values were normalized with another render pass that essentially divided the RGB channels by the accumulated alpha values. In the technique used in this dissertation and in the QSplat algorithm, lighting equations were computed per splat and the resulting colors were blended as depicted in Fig. 46. This type of shading is typically equivalent to Gouraud shading model [64]. Gouraud shading, while being used to render a triangle mesh, computes the color values for the individual triangle vertices and then interpolates these color values through the triangle. Unless applied to a low-resolution/under-sampled geometry, Gouraud shading results in smooth and continuous renders of the object.

Although Gouraud shading is relatively a low-cost shading solution, it also produces excessively blurred visuals and faceted looks in case of low-resolution geometry. A significant improvement upon the Gouraud shading is the per-pixel Phong shading model [65], which, unlike Gouraud shading, interpolates normal vectors, not the colors, across the primitives. Botsch et al. came up with the idea of Phong splatting [66] that essentially assigned linear normal fields $\mathbf{n}_i(u, v)$ to the individual splats s_i instead of using their associated constant normal vectors. The linear normal field is defined as

$$\mathbf{n}_i(u, v) = \bar{\mathbf{n}}_i + u\alpha_i\mathbf{u}_i + v\beta_i\mathbf{v}_i \tag{51}$$

where u and v are the local splat parameters, $\mathbf{u}_i$ and $\mathbf{v}_i$ are tangent vectors of splat s_i, and $\bar{\mathbf{n}}_i$, α_i, and β_i are normal field descriptors that are obtained through a least square fitting operation. In this visualization scheme, normal fields for the individual splats have to be precomputed, which

makes it suitable for static geometry. In the case of a dynamically changing object however, this technique becomes inapplicable.

4.5 Curvature Adaptive Splat Radius Sampling

Oriented point splats improve the visuals of the point-based objects significantly compared to naïve point primitive based rendering. The improvements are more profound at object contours and locations with specular highlights. However, using the same radius value for every splat poses new unwanted visual artifacts at the regions with high curvature (Fig. 47 - Fig. 48).

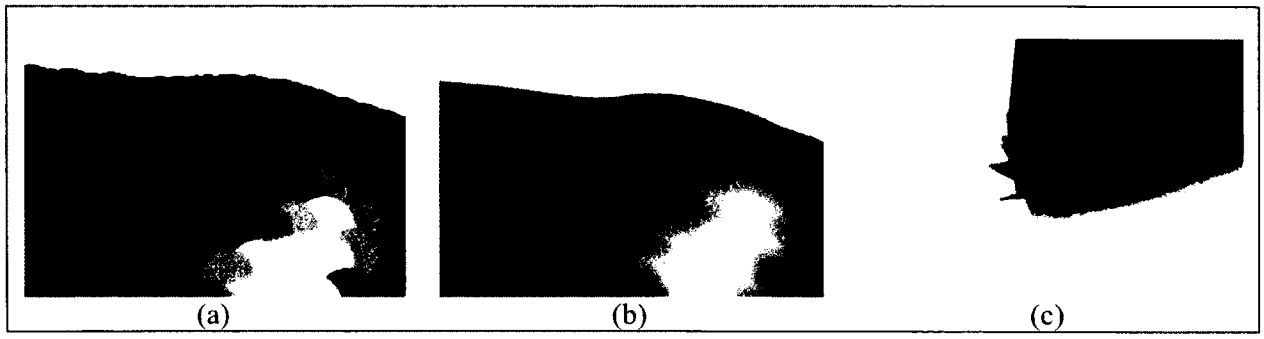


Fig. 48. Comparison of direct point rendering approaches. (a) Direct point rendering with screen-aligned disks produce poor visual quality at regions with specular highlight and object contours. (b) Oriented point splats with alpha-blending improve the visual quality significantly. (c) Using same radii for all splats result in visual artifact at high-curvature regions.

This problem can be addressed by adjusting the radii of the splats so that they are decreased at high-curvature regions and increased at relatively smooth regions. Because point-based representations of objects do not have explicit connectivity information, the curvature information has to be approximated using the point neighborhoods. Three different approaches were implemented to estimate the curvature information.

The first approach finds the local point density for each point, and uses this information to estimate the curvature, based on the assumption that high-curvature areas have greater number of points per area than smoother areas (Fig. 49). The technique accepts parameters k, which denotes the number of nearest neighbors to estimate the local point density, and r_{min} and r_{max} that are the lower and upper bounds of the radii of the splats respectively. The metric to estimate the local point density for the point p_i is the average distance to its k-nearest neighbors and defined as

$$\mu_i = \frac{1}{k}\sum_{j=1}^{k}|\mathbf{p}_i - \mathbf{p}_j| \tag{52}$$

where $\mathbf{p}_j$ are the nearest neighbors of the point $\mathbf{p}_i$.

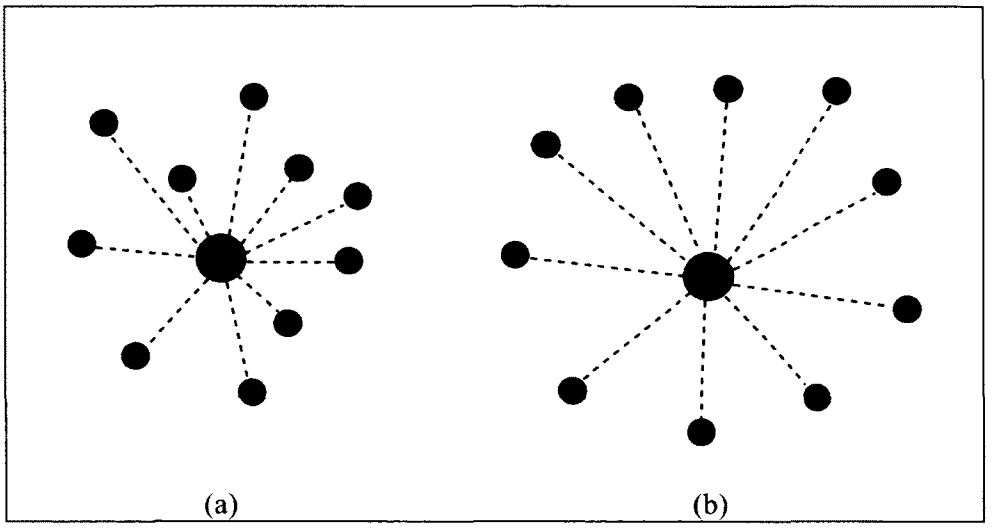


Fig. 49. The average distance to k-nearest neighbors is smaller in denser point samplings (a) compared to (b) sparse point samplings.

The minimum and maximum values of the average distances μ_{min} and μ_{max} are mapped to r_{min} and r_{max}, and the μ_i in-between are linearly mapped to r_i, where r_i are the radii of the individual splats (Fig. 50).

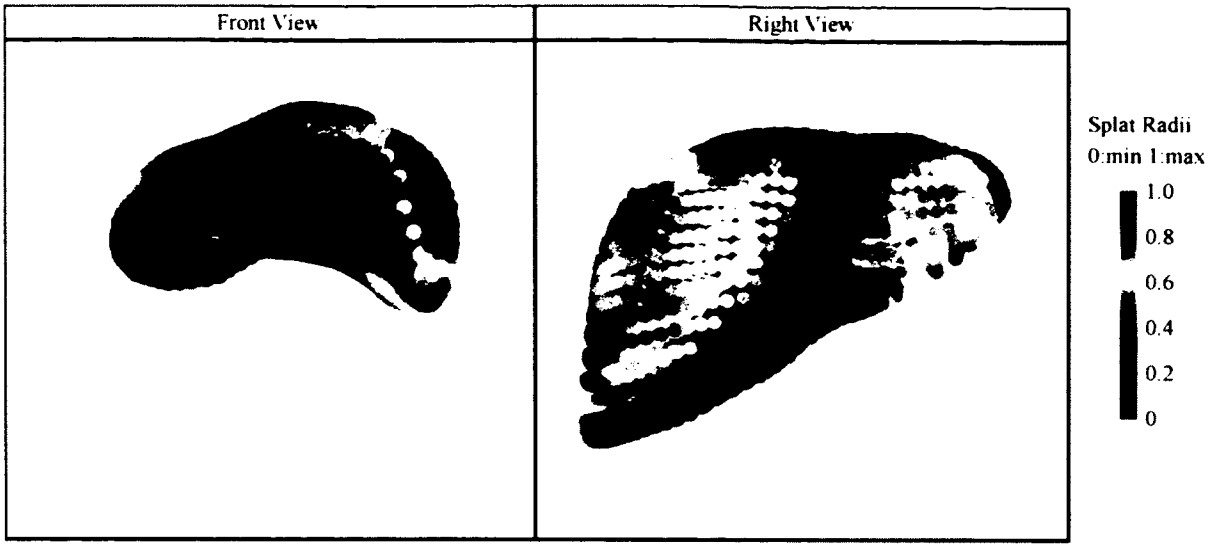


Fig. 50. The splat radii distribution with point density-based sampling.

The point density-based sampling of splat radii works well for a point cloud that is irregularly distributed according to the level of detail of the object. For a regularly distributed point cloud though, this approach would not perform very well. Therefore, other approaches were implemented which use the neighborhood information along with differential geometry concepts to estimate the curvature information for the individual points and sample the splat radii accordingly. The approach of Gumhold et al. [67] was used to extract the features of the point cloud. For each point $\mathbf{p}_i$ in the point cloud, we compute two quantities $\mathbf{c}_i$ and $\mathbf{C}_i$, which are the local centroid of the $\mathbf{p}_i$ neighborhood and the covariance matrix respectively,

$$\mathbf{c}_i = \frac{1}{k}\sum_{j=1}^{k} \mathbf{p}_j \tag{53}$$

and

$$\mathbf{C}_i = \frac{1}{k}\sum_{j=1}^{k} (\mathbf{p}_j - \mathbf{c}_i)(\mathbf{p}_j - \mathbf{c}_i)^{\mathrm{T}} \tag{54}$$

where $\mathbf{p}_j$ are the k-nearest neighbors of $\mathbf{p}_i$. The covariance matrix is a useful construct and widely used in statistics and applications as a measure of dispersion. The covariance matrix is a symmetric positive-semidefinite matrix, meaning its eigenvectors give an orthogonal basis for the plane that passes through the centroid of the points in the neighborhood, and the eigenvector that corresponds to the smallest eigenvalue is the estimate of the normal of that plane (Fig. 51).

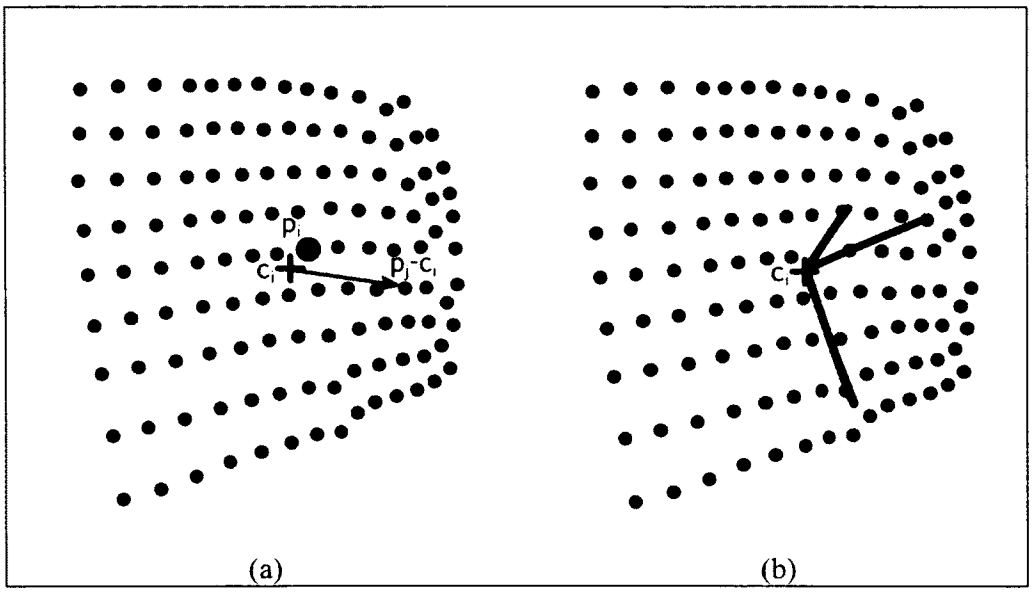


Fig. 51. Local plane fitting for curvature estimation. (a) For a local neighborhood of points around the central point $\mathbf{p}_i$, the centroid $\mathbf{c}_i$ of neighboring points $\mathbf{p}_j$ is computed. From these values, the covariance matrix $\mathbf{C}_i$ is computed, (b) which is used to estimate the least-square-fitted plane that passes through $\mathbf{c}_i$ with normal $\mathbf{e}_0$ (red vector).

After obtaining the eigenvectors $(\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2)$ and the corresponding eigenvalues $(\lambda_0, \lambda_1, \lambda_2 \mid \lambda_0 \leq \lambda_1 \leq \lambda_2)$ of the covariance matrix of the local neighborhood, this information is used in two different ways to estimate the curvature information.

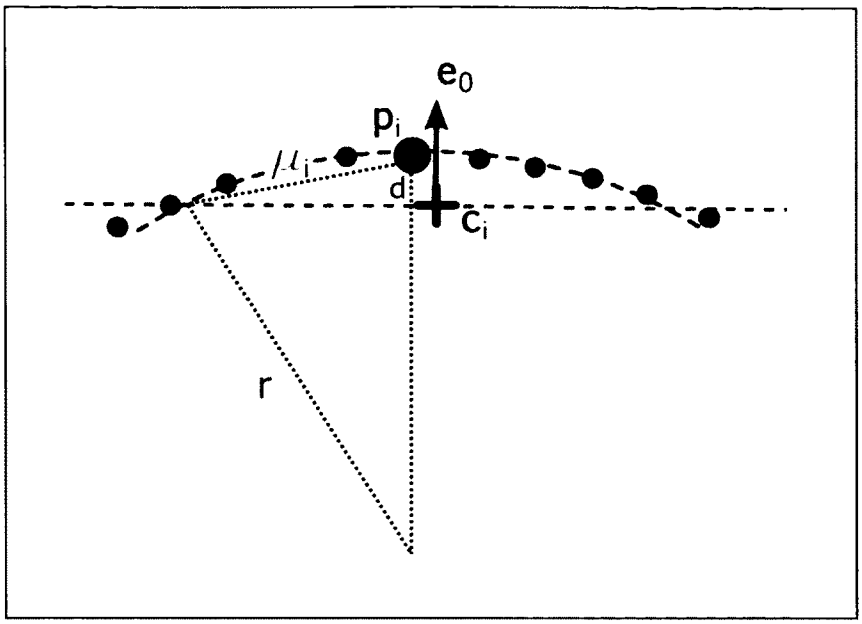


Fig. 52. Estimating the curvature of the point $\mathbf{p}_i$ with geometrical equalities.

The plane that passes through the centroid $\mathbf{c}_i$ with normal $\mathbf{e}_0$ is the least squares-fitted plane to the local point neighborhood. It is possible to estimate the average curvature κ of the point $\mathbf{p}_i$ from this fitted plane. The distance of $\mathbf{p}_i$ to the plane is $d = |\mathbf{e}_0 \cdot (\mathbf{p}_i - \mathbf{c}_i)|$. The curvature of the point $\mathbf{p}_i$ is represented by a circular arc with radius r. This arc intersects the fitted plane at a point, which has an approximate distance of μ_i to $\mathbf{p}_i$ (Fig. 52). From the equality

$$\mu_i{}^2 - d^2 = r^2 - (r - d)^2 \tag{55}$$

the estimated curvature $\kappa = \frac{1}{r}$ becomes

$$\kappa = \frac{2d}{\mu_i{}^2}. \tag{56}$$

Similar to the point density-based splat radii sampling, we find the minimum and maximum curvature values κ_{min} and κ_{max}, and use these to linearly map the curvature values to splat radii between r_{min} and r_{max} (Fig. 53).

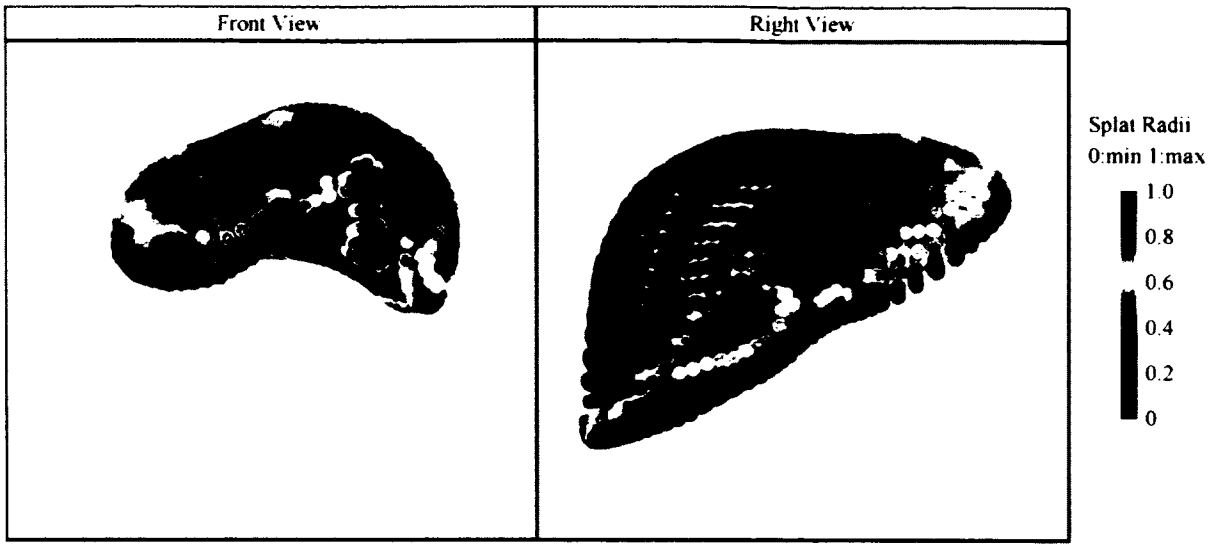


Fig. 53. The splat radii distribution with curvature-based sampling.

Curvature-based splat radius sampling provides good results in terms of setting as small as possible the splat radii at regions with high curvature and as large as possible the splat radii at smoother regions. Another technique to extract high-curvature features is the Surface Variation-based feature detection proposed by Pauly et al. [68]. Compared to the curvature-based technique discussed earlier, surface variation metric describes the variation of a point along the estimated surface normal without computing the distance of the point from the fitted plane. The deviation of the point $\mathbf{p}_i$ from the tangent plane $< \mathbf{e}_0, \mathbf{c}_i >$ is computed as

$$\sigma(\mathbf{p}_i) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{57}$$

where λ_0, λ_1, and λ_2 are the eigenvalues of the covariance matrix defined in Equation (54) such that $\lambda_0 \leq \lambda_1 \leq \lambda_2$. Surface variation varies from 0 to $1/3$, where a value of 0 means that all neighborhood points lie in the tangent plane and $1/3$ means that all points are distributed evenly at all directions [68]. Surface variation gives a good estimate for the curvature information without requiring the extra distance calculation, therefore providing an efficient tool to sample splat radii (Fig. 54).

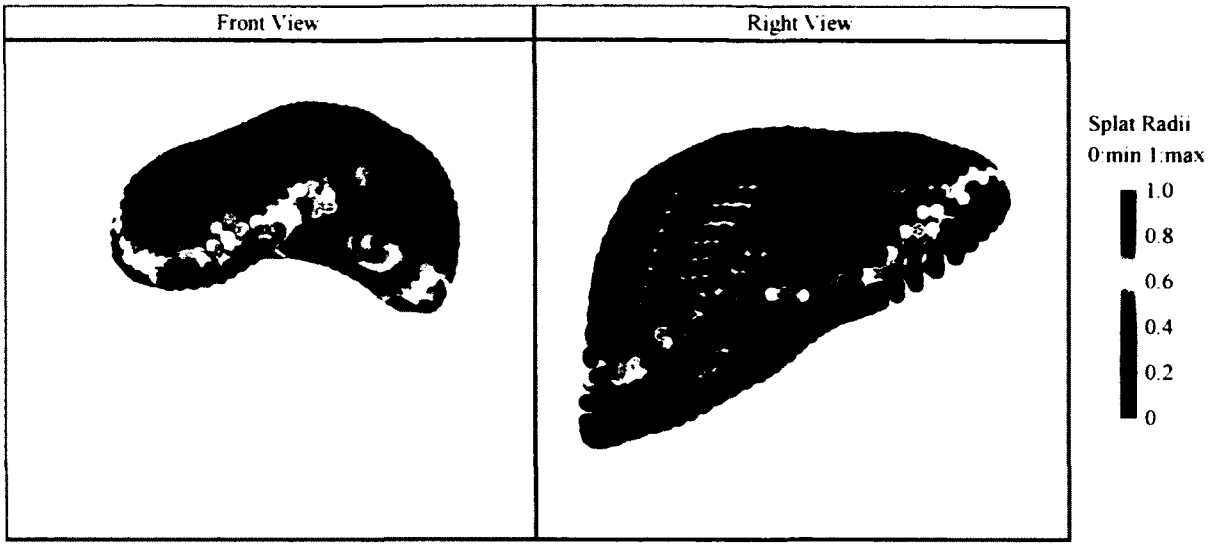


Fig. 54. The splat radius distribution with surface variation-based sampling.

The presented three approaches can be used to adjust the radii of the splats according to the curvature of the underlying surface that the splats represent. The three approaches can be used interchangeably in order to obtain the best visual quality for a given point-based simulation object (Fig. 55).

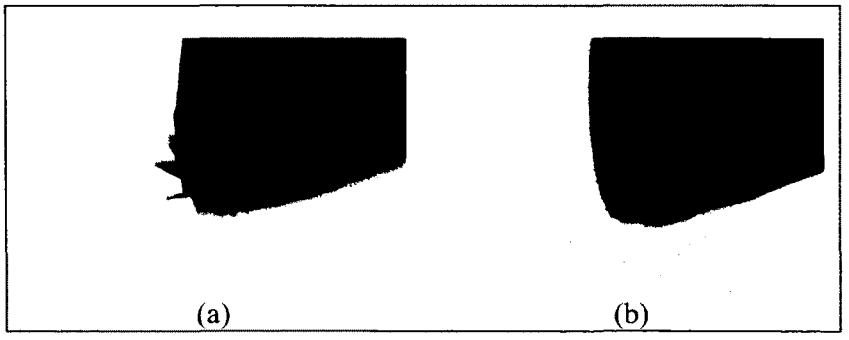


Fig. 55. Original point splat implementation vs. the adaptive implementation. (a) Original visualization of the simulation object with constant splat radii results in visual artifacts at regions with high curvature, (b) curvature adaptive splat radii distribution alleviates these artifacts by adjusting the splat radii at high-curvature areas.

4.6 Animation of the Point Splats

The visualization of the deformable body has to be controlled by the physically-based deformable behavior. In the presented visualization technique, an approach similar to the skeletal animation technique [69] has been adopted. Each surface point splat has been assigned a number of meshless nodes, whose influences on their corresponding surface point are measured with a weight function of their distances in-between (Fig. 56).

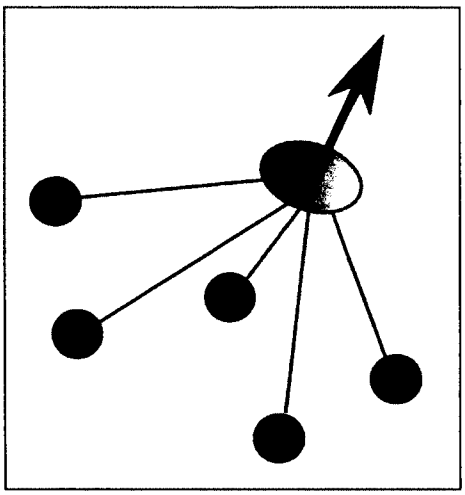


Fig. 56. The surface point splat (disk with normal vector) is controlled by a number of nearby meshless nodes.

For each surface point splat, k-nearest meshless nodes are found. The maximum distance from these k-nearest meshless nodes to the surface point splat is set as the splat's influence radius. This radius is used to calculate the influence of a meshless node i on the point splat as

$$w_i = e^{-\frac{d_i^2}{h^2}}, \tag{58}$$

where d_i is the distance between the meshless node i and the point splat, and h is the splat's radius of influence. These influence values are used to calculate the displacement of the point splat as

$$\mathbf{u}_s{}' = \frac{\mathbf{u}_i w_i}{\sum_{i=1}^{k} w_i}, \tag{59}$$

where $\mathbf{u}_s{}'$ is the point splat displacement that is desirable and $\mathbf{u}_i$ is the displacement vector of the meshless node i.

4.7 Summary

In this chapter, a visualization approach for point-based object representations was discussed. Several related studies were presented on surface reconstruction techniques as the current state of the art. Direct point rendering approaches, as opposed to surface reconstruction techniques, were favored as they are computationally more efficient and produce visuals with adequate quality. These properties make them good choices for the surgical simulation setting. Simple direct point rendering approaches were built such as naive point primitive rendering, and described the techniques that were employed in order to increase the visual quality such as oriented surface splats. The theoretical basis for the three approaches was also provided, namely a framework for adaptively sampling the splat radii in order to treat the unwanted visual artifacts. Finally, the methodology to animate the surface point splats was presented.

The point-based objects are shaded with Gouraud shading, where the lighting computations are based on the Phong reflection model. The presented approach is a single-pass, efficient rendering scheme. One current limitation is the lack of texture support, which can be implemented with multi-pass rendering schemes that remains to be analyzed for its feasibility for the surgical simulation setting.

CHAPTER 5

A COMPLETE POINT-BASED SURGICAL SIMULATION FRAMEWORK

In the previous chapters of this dissertation, a meshless method-based approach for representing the deformable object and the methodology used in this dissertation on how to handle discontinuities such as incisions/cuts, and algorithms for visualizing deformable objects with point-inspired primitives have been discussed. This chapter presents design decisions, functional modules, Hertzian Contact Theory-based code verification, and specialized algorithms developed for the point-based framework.

5.1 Overview

Medical education has embraced computer-based modeling and simulation. From the surgical training point of view, virtual environments provide a safe means to train medical professionals, where there is no risk to a real patient. As opposed to the conventional training procedures that involve text books and apprenticeships, they are interactive, multi-modal, with 3D visualizations, and also less expensive compared to cadaver training. A surgical simulation framework has many facets, and each of them poses various challenges for medical simulation research and development. These challenges are multi-disciplinary problems such as biomechanical modeling, visualization, haptic interaction, and collision/contact handling. This diverse problem set led the medical simulation community to seek the answers in open-source software toolkits. Montgomery et al. presented Spring [70] as a generalized framework for collaborative and real-time surgical simulation. Spring incorporated soft tissue modeling and also featured network-based collaboration in terms of multi-user and multi-instrument haptics interaction. Although the Spring framework was released to the public as an open-source project,

its development has ceased since 2007 and the authors have not provided the necessary means to easily extend the framework. Other examples include GiPSi [71], Virtual Reality Aided Surgical Simulation (VRASS) [72], and the Surgical Simulation and Training Markup Language (SSTML) [73], all of which lack the necessary flexibility and modularity to be widely adopted by the community.

5.2 Simulation Open Framework Architecture (SOFA)

SOFA [74] is an open-source object-oriented software library that is targeted towards interactive medical simulations. SOFA has a modular structure that allows users to quickly prototype simulation scenes with ready-to-use components. The architecture of SOFA is designed to be modular and flexible, which makes it feasible for developers to extend the functionalities of the library by deriving new components from the existing ones. With the object-oriented design principle, the components that implement new algorithms integrate seamlessly with the rest of the framework and interact with the core components within a common simulation. The two most important objectives of SOFA are to provide a software framework for the simulation community with an emphasis of medical simulation, and to enable component derivation, evolution, sharing, and exchange through a *modular plug-in* mechanism.

In order to provide this level of flexibility and extendibility, SOFA architecture relies on the notion of multi-model representation, which decomposes simulation objects into functional units such as visualization, deformation, and collision detection. These functional units, called *model*s, are then connected to each other through *mapping* objects (Fig. 57, Fig. 58).

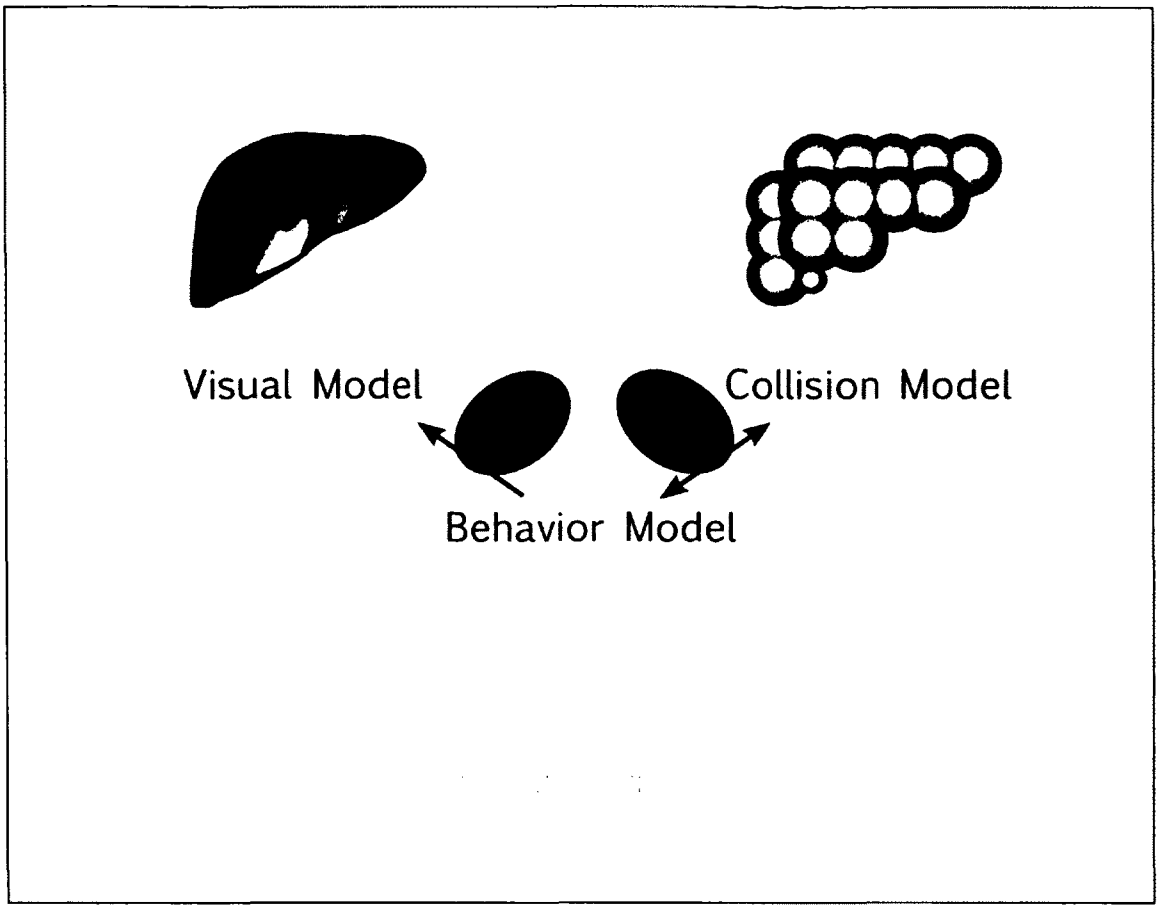


Fig. 57. In SOFA, a simulation object has multiple representations (models) each corresponding to a different functionality. These models are linked to each other with mapping objects.

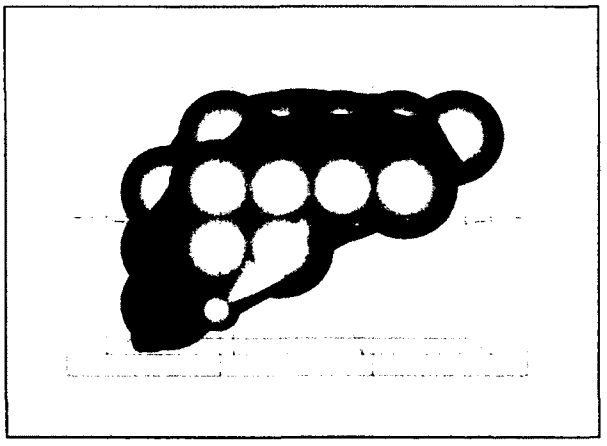


Fig. 58. All three representations of a deformable liver model are shown together.

Mapping objects essentially establishes a relationship or association between the sets of primitives of the connected models. A mapping can be defined either as a bidirectional relationship or a unidirectional relationship. The mapping between a behavior model and a collision model is an example for a bidirectional mapping, meaning an update of the behavior model is applied to the collision model and an update of the collision model is applied to the behavior model. On the contrary, the mapping between a behavior model and a visual model is a unidirectional mapping, meaning only the updates of the behavior model are applied to the visual model, not vice versa. The multi-model decomposition of simulation objects allows developers to extend the functionality of the framework by implementing new components, and integrating them to the existing components in a seamless way.

The behavior model is the underlying driving engine for a deformable simulation object. Therefore, SOFA introduces a deeper level of modularity for the behavior model. The behavior model itself is defined by a series of components such as *DoF*s in a *Mechanical Object*, *Mass*, *Force Field*, and *Solver*. These components work together and define the kinematic properties of the simulation object. For example, the *Force Field* type of components describes internal or external forces applied to the *DoF*s of the behavior model. Following this modular and flexible open-source approach, a developer can integrate his or her ideas and algorithms within the rest of the SOFA framework.

5.3 Point-Based Methods Plug-in for SOFA

For the implementation of the previously described point-based approach, the SOFA platform was selected because of its extensible architecture, large number of active users, continuing support, and up-to-date development effort. A point-based meshless behavior and

splat-based visualization algorithms were implemented as a set of SOFA components that are bundled together in the form of a plug-in (Fig. 59). The plug-in architecture allowed us to integrate the point-based components into SOFA without altering the core SOFA classes and directories.

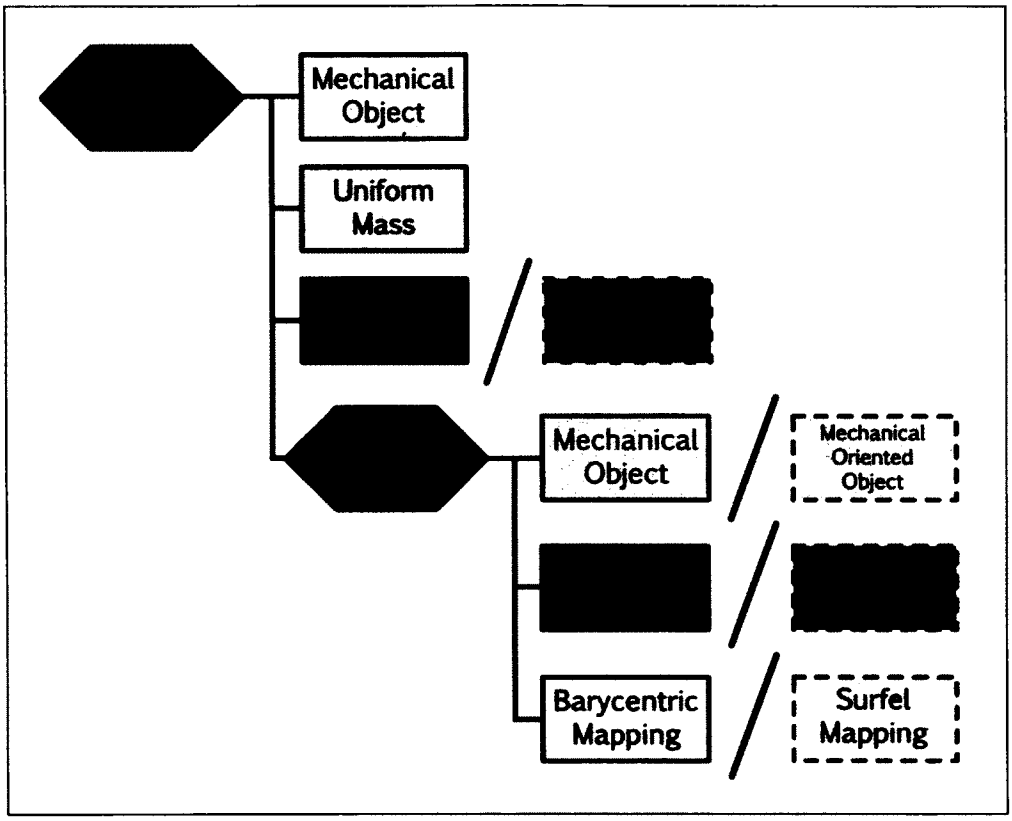


Fig. 59. Individual SOFA components that are used to model a deformable liver object. The core functionality has been extended by deriving new components (dashed rectangles) from the old ones.

The core functionality of the point-based deformable object is implemented as a *Force Field* component named *Meshless Force Field.* This component handles the point-based discretization of the continuum, Moving Least Square (MLS) approximation of the displacement gradient, and strain energy-based internal force computation. In addition to the *Force Field* component, the point-based visualization approach discussed earlier is integrated to the

framework by three new component implementations. The *Mechanical Oriented Object* component extends the *DoF* definition of the core framework by adding per-vertex normal information, the *Oriented Point Splat Model* component implements the surface splatting algorithm with alpha-blended oriented disks, and the *Surfel Mapping* component applies the updates of the point-based behavior model to the point-based visual model.

5.4 Cutting Operation for the Behavior Model

The cutting operation is implemented by additional auxiliary components that modify the internal states of the behavior and visual models. *Meshless Haptic Device* component interfaces with a haptic device to capture the location and orientation of the stylus. This component can be activated to record the cutting edge at specific intervals to create a cutting surface represented as a triangle strip. *Meshless Cut Event* component is responsible for transferring the cut surface information to the related components through SOFA's event handling mechanism, and finally, *Meshless Enrichment Grid Manager* component computes the enrichment values from the received cut surface information and passes these values to the *Meshless Force Field* component to let it update the weight values of the nodes (Fig. 60).

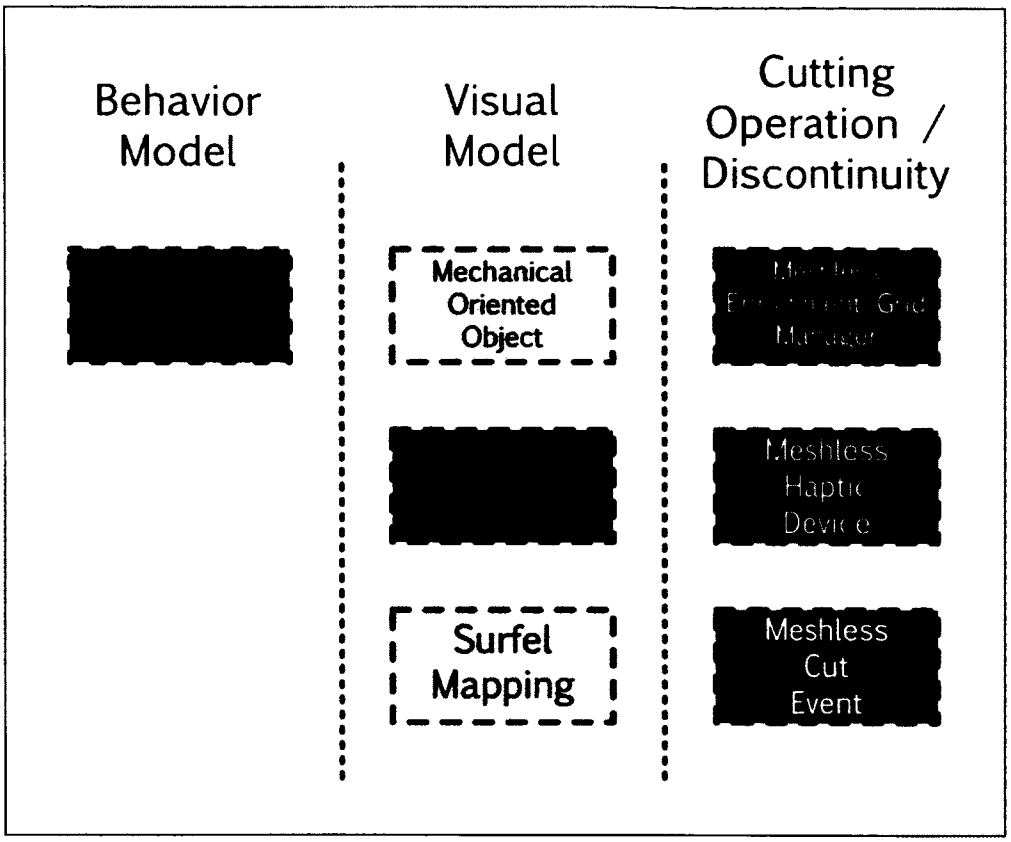


Fig. 60. The components that were implemented for the point-based approach are grouped according to their functionalities.

When there is a cut that intersects the deformable object, it affects the behavior and visual models of the object separately. For the behavior model, the cutting operation is handled in several steps:

1. The *Meshless Haptic Device* component records new cut points in the current configuration of the deformable object (in world coordinates).
2. The *Meshless Haptic Device* broadcasts this information to the related components in the scene graph.

3. The *Meshless Force Field* component converts the cut points from the current configuration to the reference configuration (from world coordinates to material coordinates) and passes these to the *Meshless Enrichment Grid Manager.*
4. The *Meshless Enrichment Grid Manager* component updates the current enrichment grid for the continuing cut surface (Fig. 61).

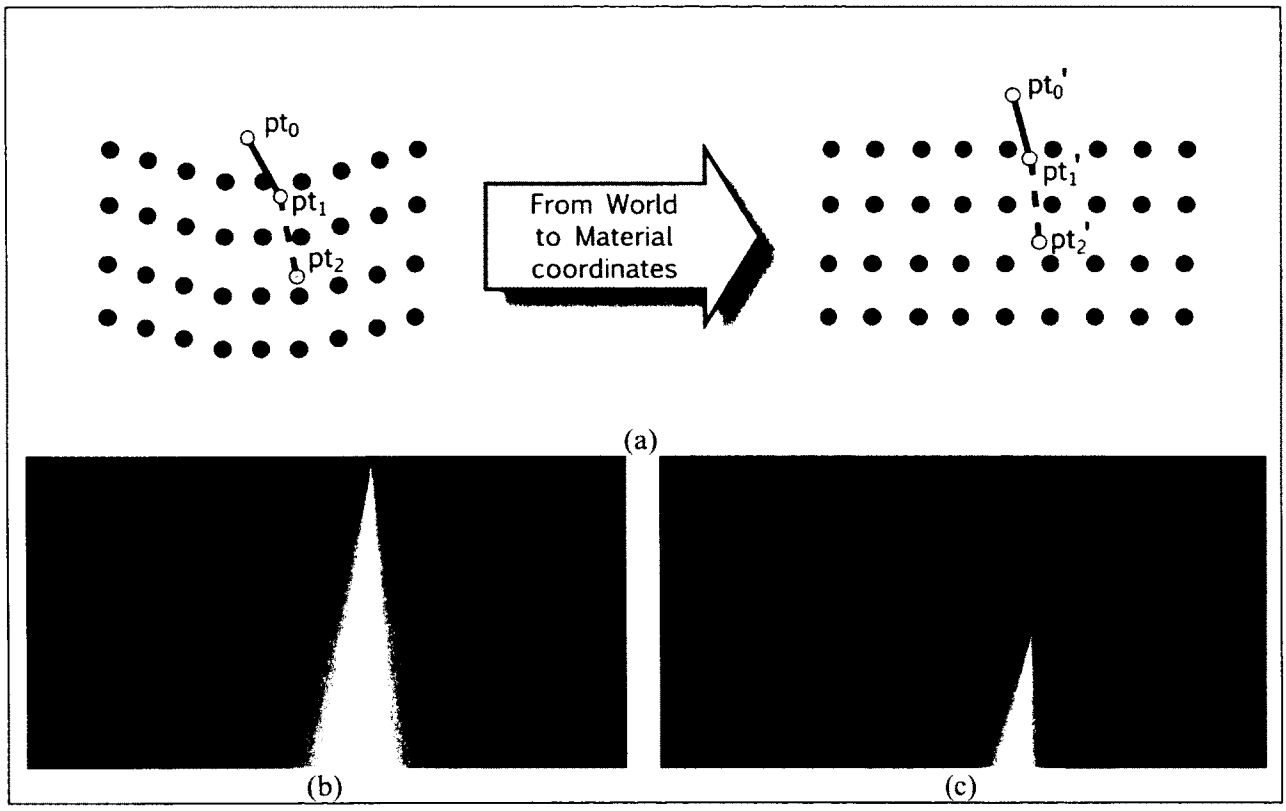


Fig. 61. Processing of the cut points in the enrichment grid. (a) When there is a cut, the points forming the cut surface are converted from the world coordinates to deformable body's material coordinates through an inverse mapping, (b-c) the cut points in the material coordinates are used to update the values in the enrichment grid.

As discussed in Chapter 2, in the Lagrangian formulation, a deformable object is represented in two configurations, the reference configuration and the current configuration. Deformable object point locations in these two configurations are defined in the material and

world coordinates respectively. There is not any direct mapping from the world coordinates to the material coordinates; however, an inverse mapping can be computed by approximating the weight of a given point in world coordinates. Recalling the Equation (22), the displacement of a point with material coordinate $\mathbf{X}_j$ can be approximated as

$$\tilde{u}_{x_j} = u_{x_i} + \left.\frac{\partial u_X}{\partial \mathbf{X}}\right|_i \cdot (\mathbf{X}_j - \mathbf{X}_i)\,. \tag{60}$$

For a point with the material coordinate $\mathbf{X}$ and that is in the neighborhood of i nodes, the displacement $\mathbf{u}$ can be computed based on the neighboring nodal displacements $\mathbf{u}_i$ as

$$\mathbf{p}' - \mathbf{X} = \mathbf{u} = \frac{1}{\sum_i w(r_i)} \sum_i w(r_i)(\mathbf{u}_i + \nabla \mathbf{u}_i(\mathbf{X} - \mathbf{X}_i))\,, \tag{61}$$

where $\mathbf{p}'$ is the world coordinate of the point, w is the weight function defined in Equation (15) and $r_i = \frac{\|\mathbf{X}-\mathbf{X}_i\|}{h_i}$, h_i being the support radius of the i-th node. It is desirable to obtain the material coordinate given the world coordinate, therefore Equation (31) is solved for $\mathbf{X}$, given $\mathbf{p}'$ and the following is obtained

$$\mathbf{X} = \left(\sum_i w(r_i)\,\mathcal{J}_i\right)^{-1} \left(\left(\mathbf{p}' \sum_i w(r_i)\right) - \left(\sum_i w(r_i)\mathbf{u}_i\right) + \left(\sum_i w(r_i)\nabla \mathbf{u}_i \mathbf{X}_i\right)\right), \tag{62}$$

where $\mathcal{J}_i$ is the Jacobian of the i-th node and defined as $\mathcal{J}_i = \mathbf{I} + \nabla \mathbf{u}_i$.

When the point for which we want to calculate the inverse mapping is not in the neighborhood of any node, or is in the neighborhood of a node with a very low weight value, the matrix summation $\sum_i w(r_i)\,\mathcal{J}_i$ will result in a singular matrix, which makes it impossible to invert. In such cases, a fall-back mechanism has been implemented that approximates the material coordinate of a point $\mathbf{p}'$ by

$$\mathbf{X} = \mathbf{p}' - \frac{\sum_i^k \mathbf{u}_i\,|\mathbf{p}_i - \mathbf{p}'|}{\sum_i^k |\mathbf{p}_i - \mathbf{p}'|}, \tag{63}$$

where $\mathbf{p}_i$ is one of the k-nearest neighbors of $\mathbf{p}'$ and equal to $\mathbf{X}_i + \mathbf{u}_i$.

5.5 Cutting Operation for the Visual Model

After obtaining the material coordinates of the cut surface, Meshless Force Field component forwards this information to the Meshless Enrichment Grid Manager for updating the behavior model as discussed earlier. For the visual model of the deformable object, the process involves the following steps:

1. The *Meshless Force Field* component converts the cut points from the current configuration to the reference configuration (from world coordinates to material coordinates) and passes these to the *Surfel Mapping* through SOFA event handling mechanism,
2. The *Surfel Mapping* samples regular points on the triangles that form the cut surface and for each of these points, checks whether the point is inside or outside of the surface of the deformable object,
3. For each of the points that are detected as inside of the boundary of the soft-tissue model, two new *DoF*s are added to the *Mechanical Oriented Object* component, forming the new surface of the cut deformable object.

When the *Surfel Mapping* receives the triangle strip vertex positions in material coordinates, it uniformly samples points on the triangles. The point sampling algorithm accepts as input the cutting surface and a density parameter that determines how many points should be sampled per unit area of the triangles (Fig. 62, Fig. 63).

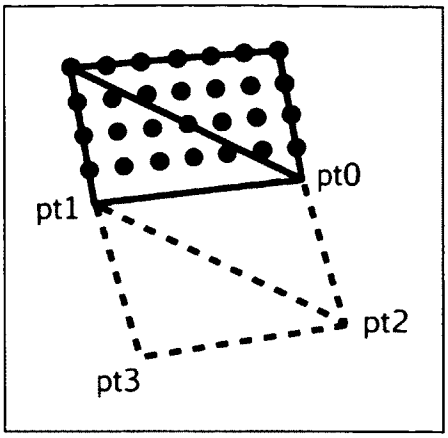


Fig. 62. New points are sampled uniformly on the cut surface for each pair of triangles. The density of the sampled points is defined by a density parameter.

```
1: procedure UNIFORMSAMPLE(cut, density)          ▷ cut is a triangle strip
2:   for each <p0,p1,p2> and <p2,p1,p3> ∈ cut
3:     tri1Samples ← {}
4:     tri2Samples ← {}
5:     totalArea ← Area(<p0,p1,p2>) + Area(<p2,p1,p3>)
6:     totalArea ← totalArea * density
7:     e1 ← Norm(p1 - p0)
8:     e2 ← Norm(p2 - p0)
9:     steps1 ← 1 / (⌈totalArea/e1⌉ − 1)
10:    steps2 ← 1 / (⌈totalArea/e2⌉ − 1)
11:    for u ← 0 to 1 step steps1
12:      for v ← 0 to 1 - u step steps2
13:        y ← 1 − u − v
14:        tri1Samples ← Push_back(tri1Samples, p2*u + p1*v + p0*y)
15:        tri2Samples ← Push_back(tri2Samples, p1*u + p2*v + p3*y)
16:    return <tri1Samples, tri2Samples>
17:  end procedure
```

Fig. 63. The algorithm for sampling uniform points on a cut surface represented as a triangle strip.

The points that are sampled in the reference configuration are input to an inside/outside test against the current points that represent the surface of the deformable object. The points that are classified as inside-points are used to generate new surface point elements (Fig. 64).

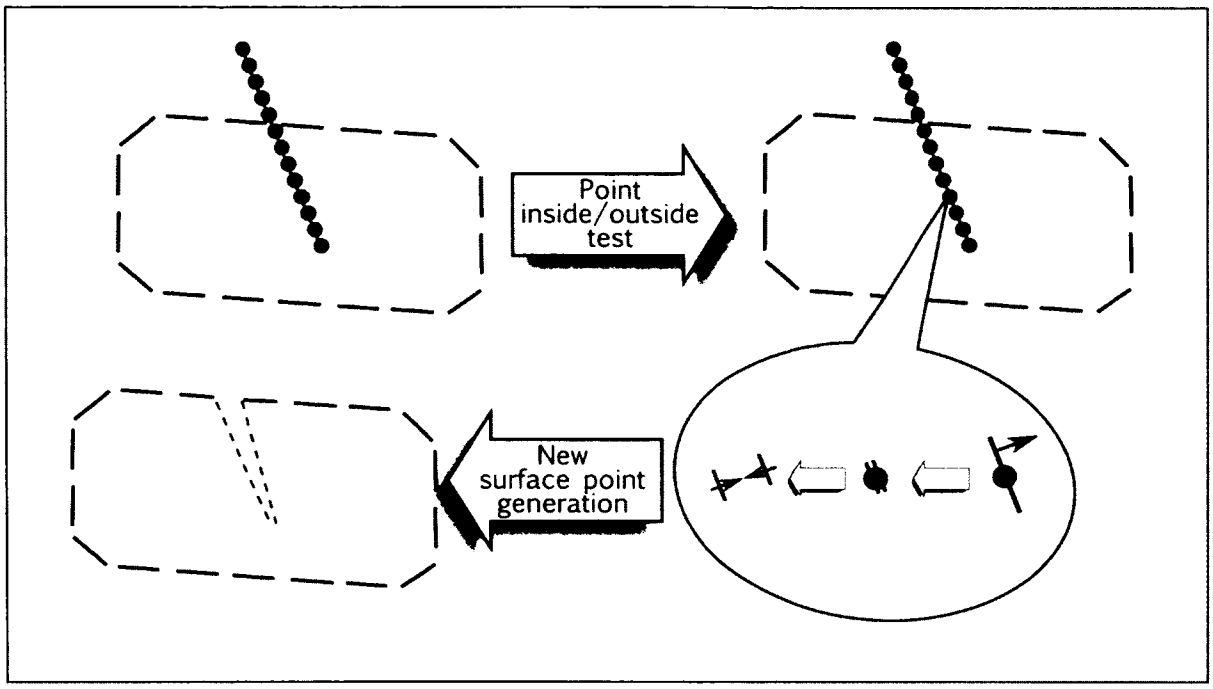


Fig. 64. Each point on the cut surface is input to an inside/outside test, and the ones that are designated as inside points are used to generate new surface points. For each inside point, two new surface points are generated, and then separated along the normal of the cut triangle that they belong to.

The inside/outside test for a sampled point $\mathbf{p}$ works by first finding the two surface points nearest to $\mathbf{p}$, which we label $\mathbf{p}_1$ and $\mathbf{p}_2$, and for which we define normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$. Next, the algorithm determines whether these oriented points $\mathbf{p}_1$ and $\mathbf{p}_2$ are convex to each other. If they are convex, the point $\mathbf{p}$ is categorized as inside the surface if both $\mathbf{p}_1$ and $\mathbf{p}_2$ are facing away from $\mathbf{p}$, based on a test on the normals $\mathbf{n}_1$ and $\mathbf{n}_2$. If the two-nearest points are concave to each other, the point is classified as an inside point when at least one of the points $\mathbf{p}_1$ or $\mathbf{p}_2$ does not face towards $\mathbf{p}$. The concavity and convexity definitions for two oriented points and sample inside/outside regions are illustrated in Fig. 65.

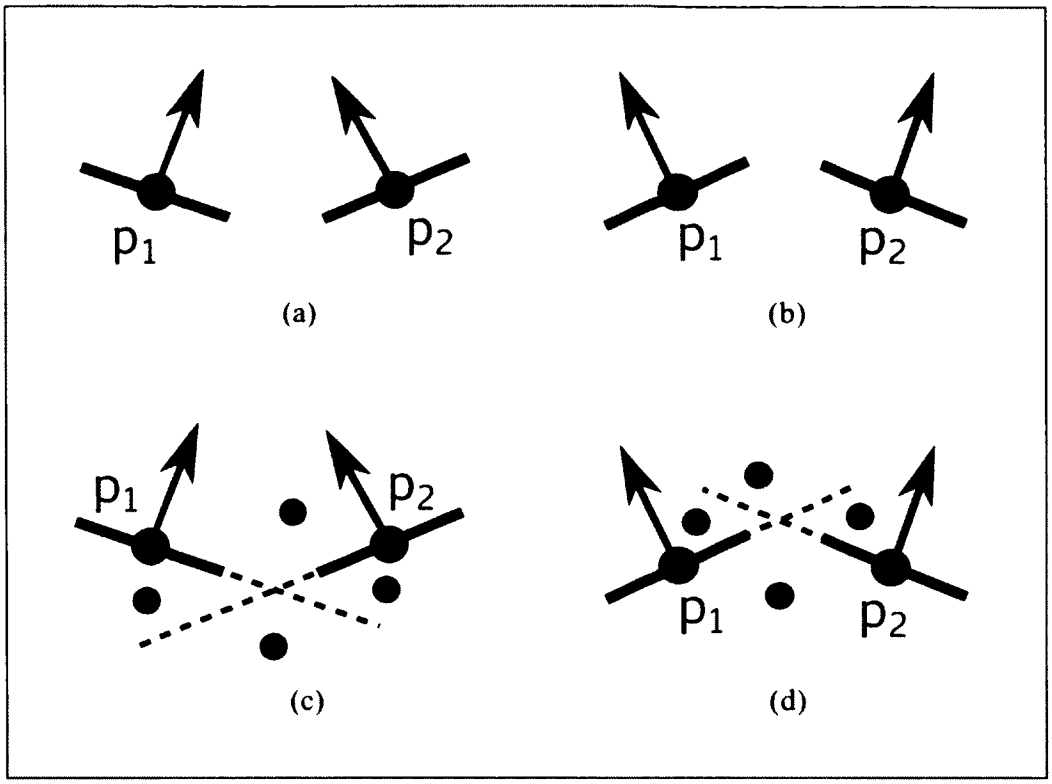


Fig. 65. Surface inside/outside tests with respect to two oriented surface points. (a) Two surface points spanning a concave surface, (b) spanning a convex surface, the regions classified as inside (blue) and outside (red) the local surface spanned by $\mathbf{p}_1$ and $\mathbf{p}_2$, which can be (c) concave or (d) convex.

In order to determine the convexity of the two points $\mathbf{p}_1$ and $\mathbf{p}_2$ with normals $\mathbf{n}_1$ and $\mathbf{n}_2$, we define two vectors $\mathbf{p}_{1\_2} = \mathbf{p}_1 - \mathbf{p}_2$ and $\mathbf{p}_{2\_1} = \mathbf{p}_2 - \mathbf{p}_1$. The points $\mathbf{p}_1$ and $\mathbf{p}_2$ are defined to be concave if the dot products $\mathbf{p}_{1\_2} \cdot \mathbf{n}_2$ and $\mathbf{p}_{2\_1} \cdot \mathbf{n}_1$ are both positive and they are defined as convex points if either of the dot products is negative.

5.6 Spatial Data Structures

The operations that are described throughout this dissertation such as setting up support radii of the meshless nodes, determining the meshless nodes that are affected by a cut surface, and testing whether a cut point sample is inside or outside of the deformable object all require

spatial queries such as like finding nearest neighbors or computing intersections between geometric primitive. Brute-force implementations of such operations typically have high computation costs and they are certainly not suitable for applications with interactivity requirements such as surgical simulation. When the performance is a critical factor for an application, spatial data structures become essential elements to accelerate time-consuming spatial queries.

There are various categories of spatial data structures. Regular grids, Octrees [75], Bounding Volume Hierarchies [76], Binary Space Partitioning (BSP) trees [77], and their generalization Kd-trees [78] all address different application requirements and problems. Among these, Kd-trees are particularly useful for organizing point data and they are also capable of performing specific queries efficiently such as nearest neighbor searches and range searches. These features of Kd-trees make them viable options for the purposes of this study. A subset of the FLANN library [79] called nanoflann [80] was used, which is optimized for 3D point cloud organization and querying.

The Kd-tree data structure is beneficial for point-based nearest neighbor queries and range searches; however, the cutting operation requires computing the intersection of a cut surface, which is a triangle strip, with multiple spherical regions that are essentially defined by meshless node and surface point locations as well as their domains of influence. This intersection query is critical for the cutting algorithm as it allows the algorithm to apply the enrichment only to the necessary nodes and surface points, i.e. modification of the weight function of a meshless node is only required if the cut surface intersects with its domain of influence. Considering the many triangles that a typical cut surface contain and the number meshless nodes as well as

surface points that this cut surface needs to be retested against, a brute-force approach is not a viable option.

Typically, the way to approach this problem is to partition the space that encloses the objects hierarchically and perform the intersection checks only between the objects that are located in the same space cell [81]. The approach in this dissertation to address this problem however, is a different, more practical approach. In order to find the nodes that are affected by a cut, by-products of the Enrichment Grid are being used as a spatial data structure. Recalling Equation (47) from Chapter 3 about handling discontinuities, a distance function was defined that formed the basis of the enrichment function definition

$$d_3(x,y,z) = \sqrt{d_2^{+2} + s^2}. \tag{64}$$

The d_3 function is a by-product of calculating the enrichment values for the enrichment grid. For each grid value, the d_3 function gives the minimum absolute distance of that grid point to the cut surface. This distance metric, therefore, provides a practical way of finding which meshless nodes/surface points are intersected by the cut surface as it translates the intersection problem into a d_3-look-up operation for the node location. If the distance of a node to the cut surface is smaller than its domain of influence size, then the cut surface affects that node (Fig. 66, Fig. 67).

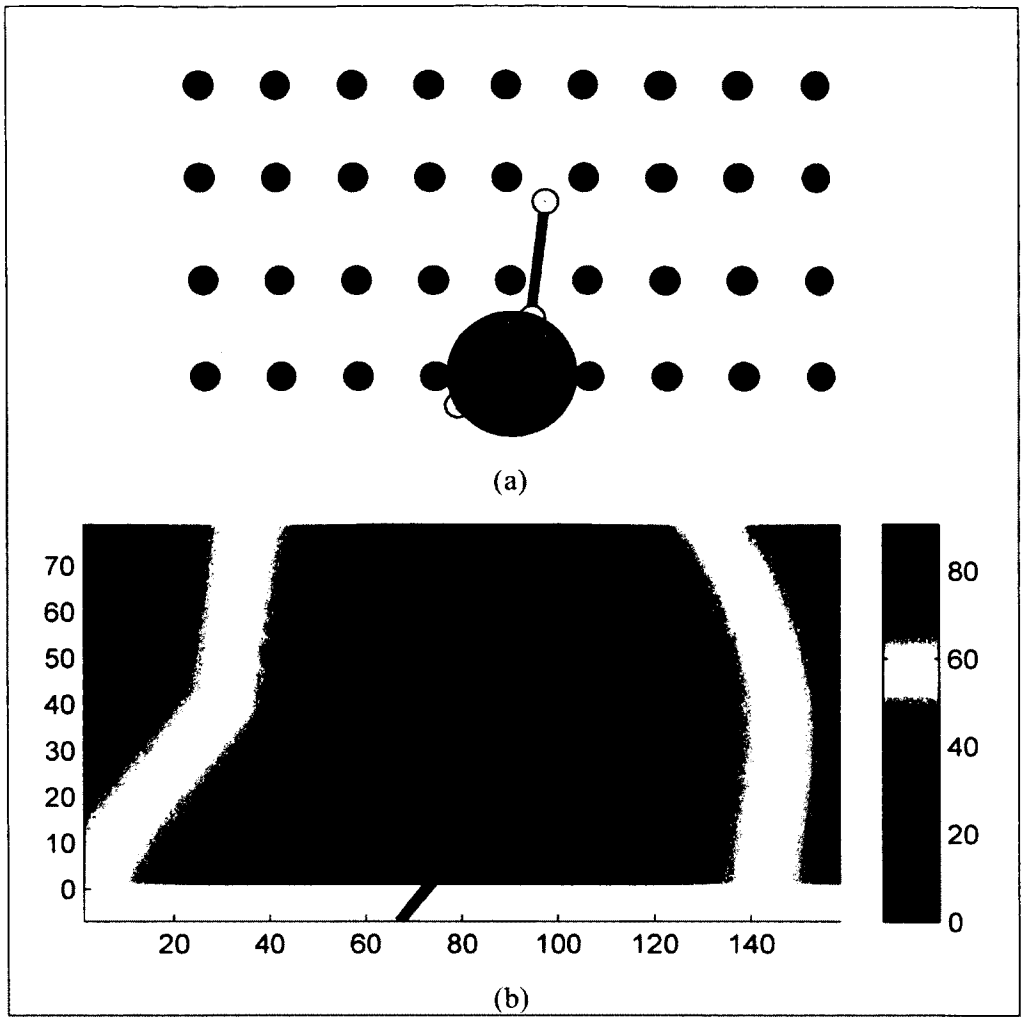


Fig. 66. A cut in a deformable body affects a meshless node as it intersects with its spherical domain of influence. (a) The calculated distance function (in 2D) can be used to find the list of nodes whose distances to the cut are smaller than their domain radii (b).

The enrichment grid related algorithms were implemented using the open-source Armadillo library [82]. Armadillo is an easy-to-use linear algebra library that provides matrix manipulation syntax similar to MATLAB [83]. In order to access the distance values of the nodes quickly at run time, the indices of the nodes are pre-calculated and stored in an index array. When one wants to query a distance grid, the stored indices are used to lookup the distance

values corresponding to the nodes. The nodes that have distance values smaller than their support radii are chosen as affected nodes by the cut, as shown in Fig. 68.

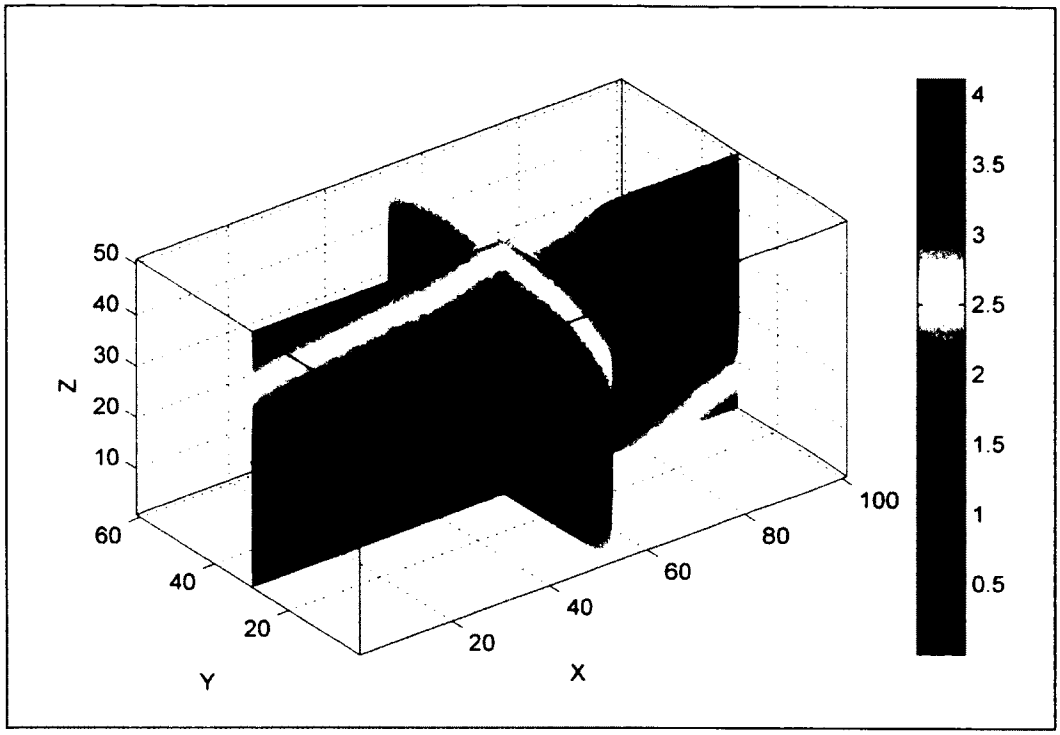


Fig. 67. The distance function d_3 is calculated for a cut surface in 3D.

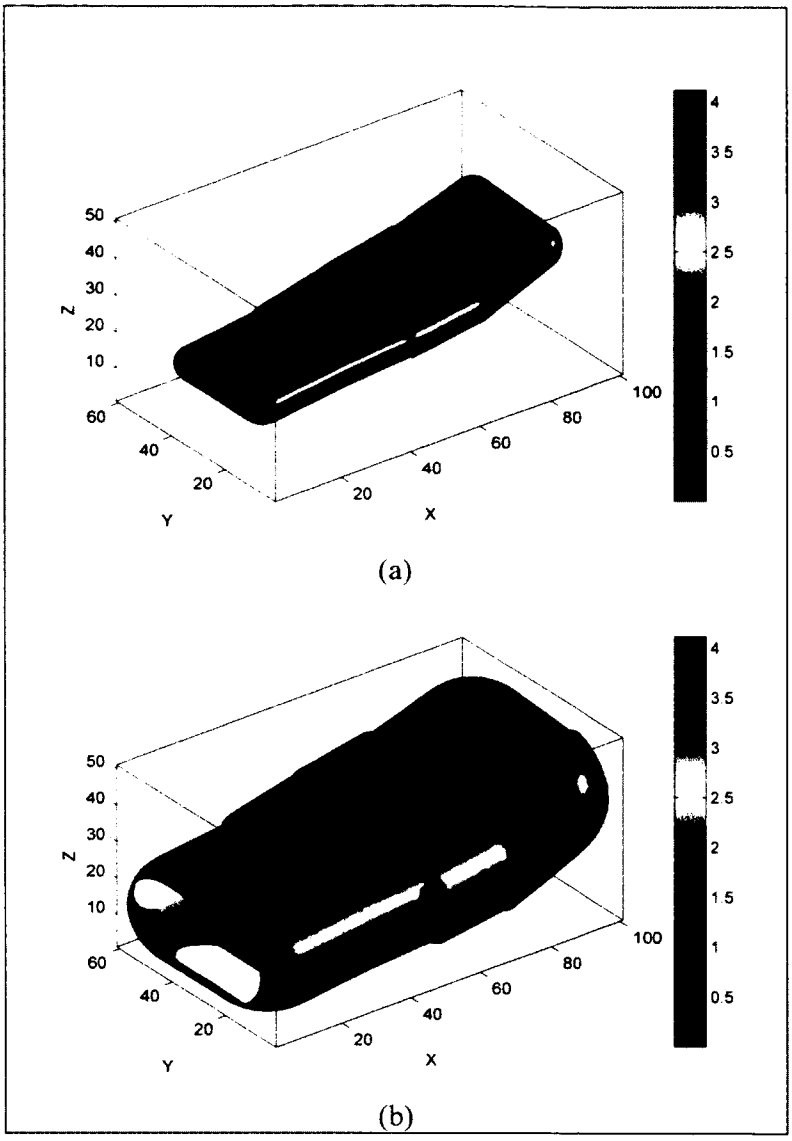


Fig. 68. The iso-surface visualization of the distance function at level-set (a) 0.5 and (b) 1.5.

5.7 Code Verification through Hertzian Contact Theory

The Hertzian theory of non-adhesive elastic contact [84] defines analytical solutions for the interaction of elastic half-spaces with simple shapes in terms of applied force and object indentation. For example, the amount of indentation of an elastic half-space under a spherical load is given by

$$f = \frac{4}{3} E^* \sqrt{r} d^{3/2}, \tag{65}$$

where f is the vertical force applied on the spherical load, r is the radius of the spherical load, d is the indentation amount, and E^* is the combined Young's modulus of the two materials and calculated using the Young's moduli (E_1, E_2) and Poisson's ratios (v_1, v_2) of the two materials as

$$\frac{1}{E^*} = \frac{1 - v_1^2}{E_1} + \frac{1 - v_2^2}{E_2}. \tag{66}$$

The Hertzian theory assumes 1) small strains within the elastic material, 2) much smaller area of contact compared to the areas of the objects in contact, and 3) continuous and frictionless contact surfaces. There have been numerous finite element analysis studies about the Hertzian theory that use both research and commercial finite element code [85-88].

In order to verify the usability of the Hertzian contact theory as a means of verification of soft-tissue deformation, experiments were first conducted using a well-established finite element platform: FEBio is an open-source software suite that is primarily targeted towards biomechanics and biophysics problems with a specific focus on nonlinear large deformation problems in biosolid mechanics [89]. FEBio provides several models and options to represent the non-adhesive Hertzian contact theory. In the experiments, the facet-to-facet sliding algorithm that is based on Laursen's contact formulation [90] was selected. In this algorithm, the contact constraints are enforced through Lagrange multipliers.

The FEBio experiment was setup by defining the fixed-position boundary conditions of the deformable block at its bottom and side faces, facet-to-facet sliding contact between the top face of the deformable block and the rigid spherical indenter, and the sphere's indentation amount. The simulation was run for 10 time steps of 0.1s each and the simulation runtime took

over 4 minutes. The node at the middle of the top of the deformable block was tracked for the vertical displacement and the vertical component of the contact force. The obtained load-displacement curve was compared to the theoretical solution, which were in very good agreement (Fig. 69), therefore verifying the usefulness of the Hertzian contact theory as a verification method.

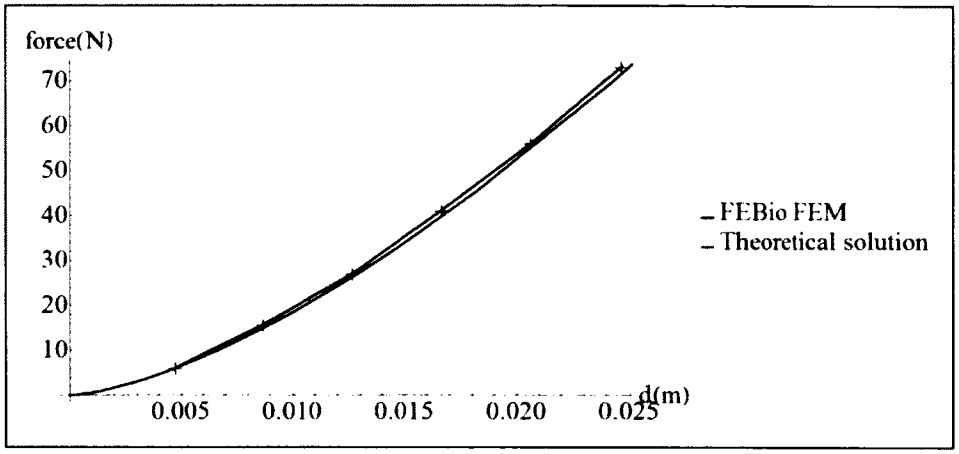


Fig. 69. Comparison of the FEBio FEM Code and the theoretical solution of the Hertzian non-adhesive frictionless contact theory.

The contact mechanics experiment was setup as a SOFA scene. In order to assess the ground-truth performance of the contact handling in SOFA, the rectangular deformable block was represented with the hexahedral finite element model in addition to the point-based deformable body approach. The validation of the hexahedral FEM implementation of SOFA was studied by Marchal et al. [91]. For a given sphere radius, simulations were performed for varying force values (f) applied to the spherical load as depicted in Fig. 70. With the applied force, the rigid sphere comes into contact with the elastic solid block and deforms it. The vertical velocity of the sphere is monitored and the indentation of the material (d) is measured when the sphere comes to rest. This f-d pair is compared to the theoretical solution.

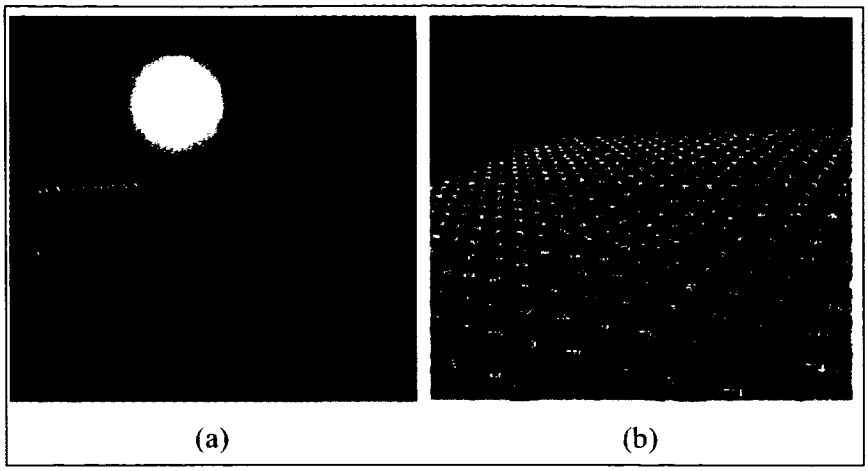


Fig. 70 Initial setup of the indentation experiment for the SOFA FEM model. (a) The close-up view of the indented deformable material (b). SOFA allows the user to track and monitor simulation values of indexed particles.

The meshless nodes are distributed uniformly inside a cubical volume with 2m long edge length. The indentation experiment is repeated for several distribution configurations, which play a critical role especially for the MLS approximation-based collocation methods. The convergence rate in the L2 (vector) error norm of the force-indentation pairs with respect to the theoretical values (Fig. 71) is investigated. The effect of different distribution schemes on the accuracy, stability, and performance of the meshless collocation methods has yet to be examined. In the implementation, the number of neighboring nodes is limited to 16 for each of the meshless nodes.

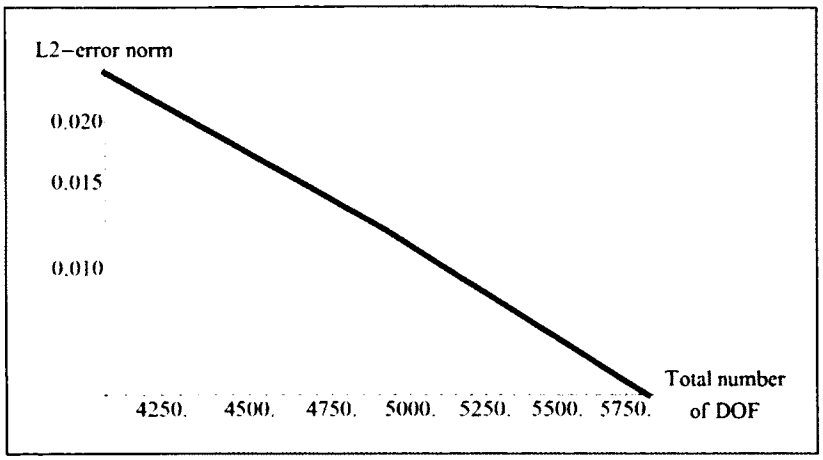


Fig. 71. Error in the L2 norm with respect to the theoretical solution as function of total number of the degree of freedom for the meshless method.

The SOFA FEM implementation and the meshless collocation method were compared with close accuracy (Fig. 72). For the meshless collocation method with nodal integration, an explicit time integration scheme was used with a time step of 0.001s without any stability problem. For the SOFA FEM implementation, implicit integration with a time step of 0.01s or greater were used. The calculations were performed within the SOFA application on a single Intel Core i5 CPU running at 2.67 GHz with 16 GB of RAM under Windows 7 operating system. The SOFA FEM implementation took 195ms of calculations per time step, whereas the meshless method consumed 20.11ms for calculations per time step. Therefore, the meshless collocation implementation in SOFA (along with other SOFA related operations such as collision detection) is roughly 25 times slower than the real-time operation, which is slightly better than the 30 times slower performance reported by the Meshless TLED algorithm [38]. The calculation speed of the meshless collocation algorithm is governed by the number of particles and the number of neighbors assigned to each particle.

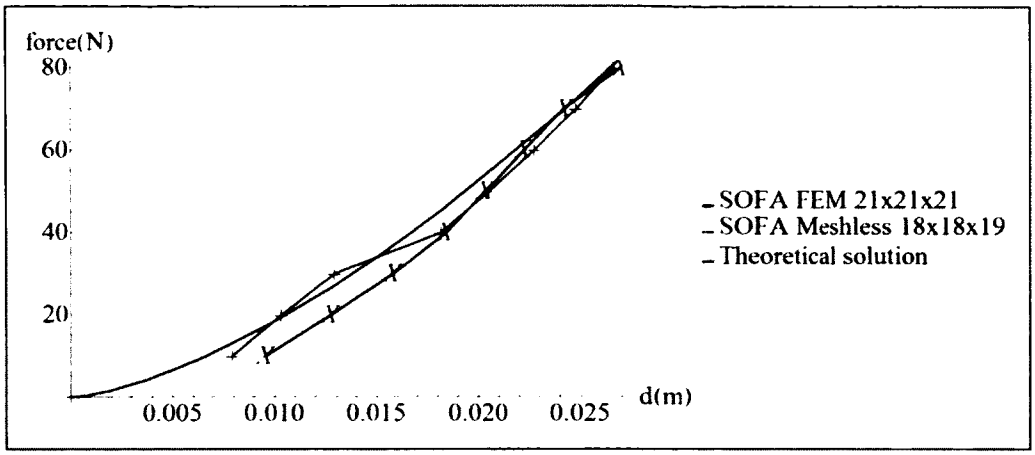


Fig. 72. Comparison of the SOFA FEM implementation and the point-based approach with close indentation accuracy and the theoretical solution.

A mesh convergence study for the meshless collocation method was also performed by investigating the convergence of the indentation amount to the theoretical value for a fixed amount of force (Fig. 73). After around 6000 particles, the indentation value converges to the theoretical indentation value, the equation of which is given in Equation (65).

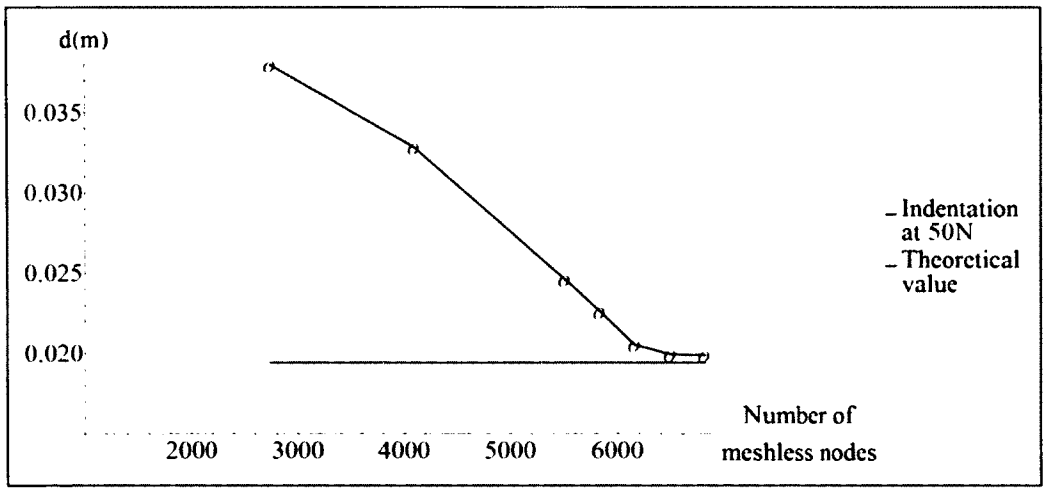


Fig. 73. Convergence of the indentation value with increasing number of meshless nodes.

5.8 Summary

In this chapter, the implementation details of the point-based approach for surgical simulation were discussed. The approach was integrated into a previous framework due to the fact that in a surgical simulation application there are several aspects of the problem that need to be considered and it is simply not feasible to address all the problems in a reasonable amount of time. Examples of several open-source frameworks were given and then the choice Simulation Open Framework Architecture (SOFA) was presented.

The SOFA framework has a modular and flexible architecture that enables developers to integrate novel algorithms with the rest of the framework. Details for several implemented components were presented. These components control different functionalities in the approach such as the deformable behavior, point-based visualization, and handling the cutting operation through the input of a haptic device. Several algorithms were developed for various tasks that were also described in detail in this chapter.

Finally, the formulation of the Hertzian non-adhesive contact theory was presented from which the verification technique was based. Various experiments were performed with different tools and techniques and the point-based deformable object modeling approach was compared with these techniques and the theoretical solution of the contact theory in terms of accuracy and performance.

CHAPTER 6

RESULTS

In this chapter, the results for the point-based deformable object modeling approach are presented by modeling two objects with the discussed techniques. The sample objects are a synthetic rectangular block and a liver model. The triangular mesh of the liver model, which is used to initialize the meshless node and surface point locations, is obtained from the SOFA repository [92]. The soft-tissue elastic properties that were reported by McKee et al. [93] are input to the point-based deformable model. The results are presented separately for a simple deformation of the objects under gravity and after cuts are introduced to the models.

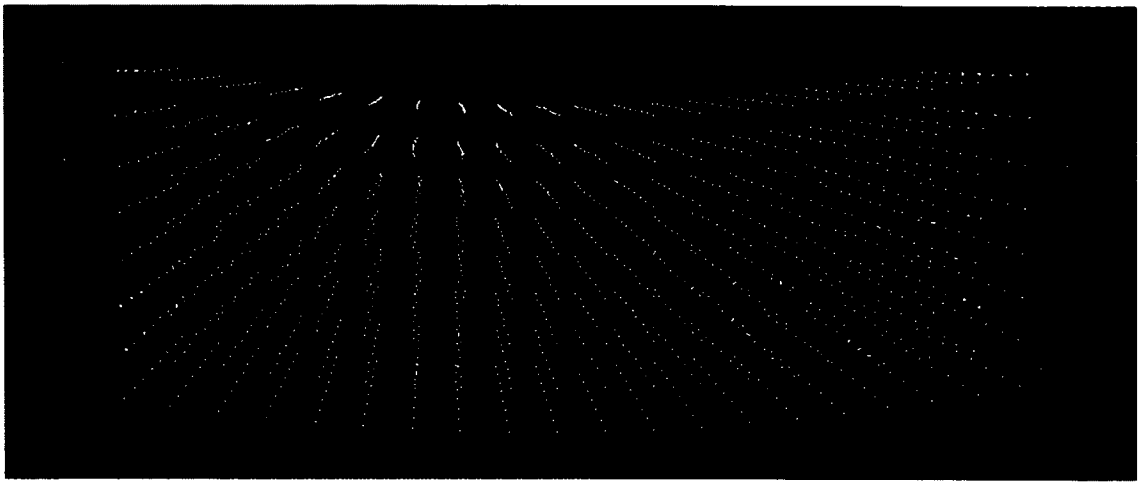


Fig. 74. The rectangular block, which is discretized by regularly distributed points, deforms under gravity.

The deformable block object is a simple geometric shape, which makes it possible for it to be discretized by uniformly distributed set of points in addition to the non-uniform tetrahedralization-based distribution that was discussed in Chapter 2. Fig. 74 shows the point-based regular discretization of the rectangular object with 1408 DoFs. The number of neighbors (the points that are connected by blue lines) for each DoF is limited to a maximum number of 16

neighbors and the simulation runs at about 80 simulation frames per second (FPS). It is also possible to discretize the object geometry hierarchically by first obtaining the tetrahedral mesh, and using the vertices of the mesh as point-based node locations (Fig. 75).

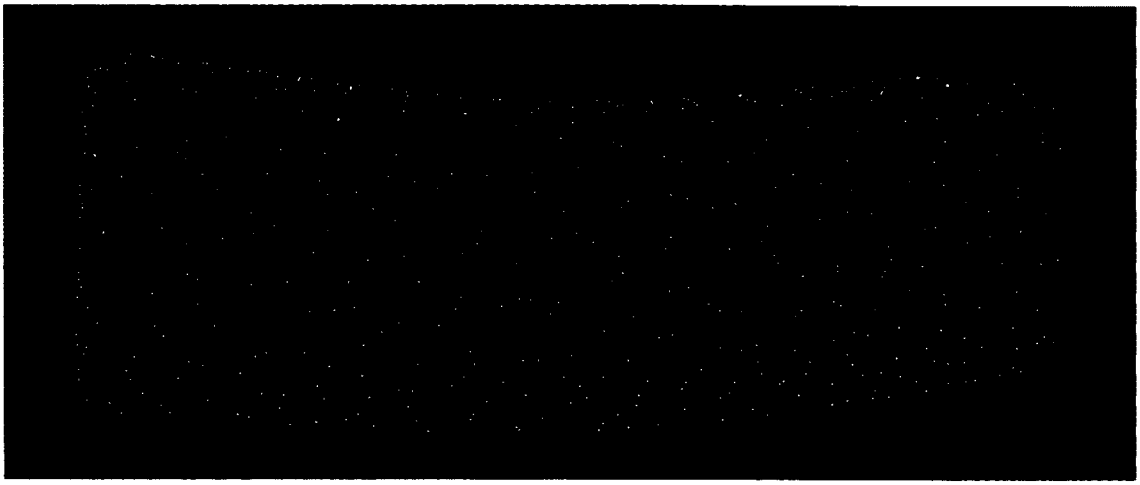


Fig. 75. The meshless node locations of the deformable block are obtained through hierarchical discretization. This simulation object has 436 DoFs, the number of neighbors is not limited, and simulated at around 125 FPS.

The meshless nodes shown above correspond to the behavior model of the simulation object. The visual model of the object, which is controlled by the behavior model, is also represented by point primitives.

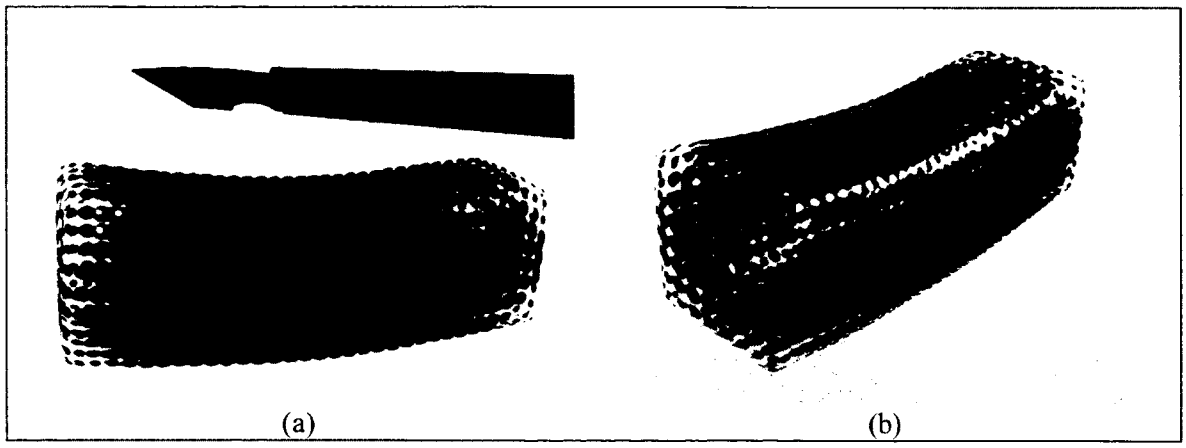


Fig. 76. The visual model of the simulation object is represented by point primitives.

Fig. 76 shows the visual representation of the deformable block. The oriented point splat primitives are scaled down to show the individual elements, and colored according to the curvature of the point – from low curvature, shown in blue to high curvature, depicted in red. Fig. 77 shows the continuous visualization of the block with oriented point splat primitives.

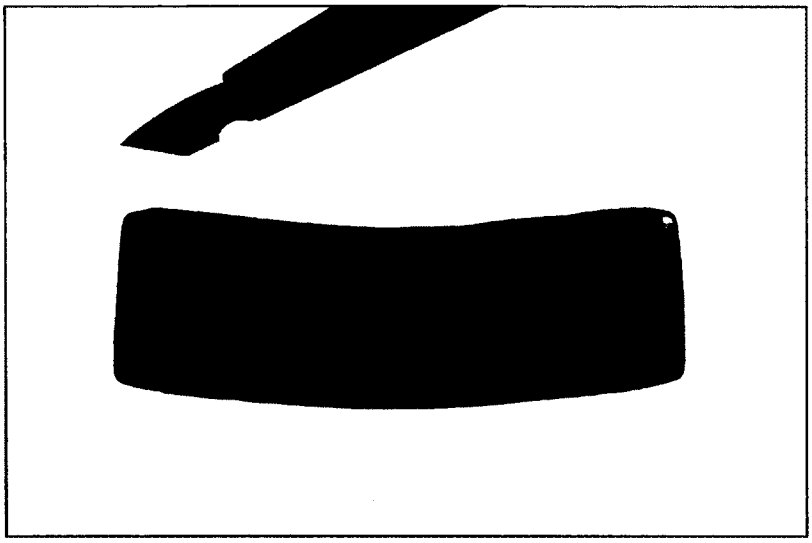


Fig. 77. The deformable block is visualized with oriented point splat primitives.

When a cutting operation is performed, an enrichment grid is generated for the cut, which is used to update the behavior and visual models of the deformable object. The changes applied to the visual model includes updating meshless node – surface point neighborhood relation, adding new surface points along the cut surface, and re-calculating the radii of the oriented point splats (Fig. 78, Fig. 79).

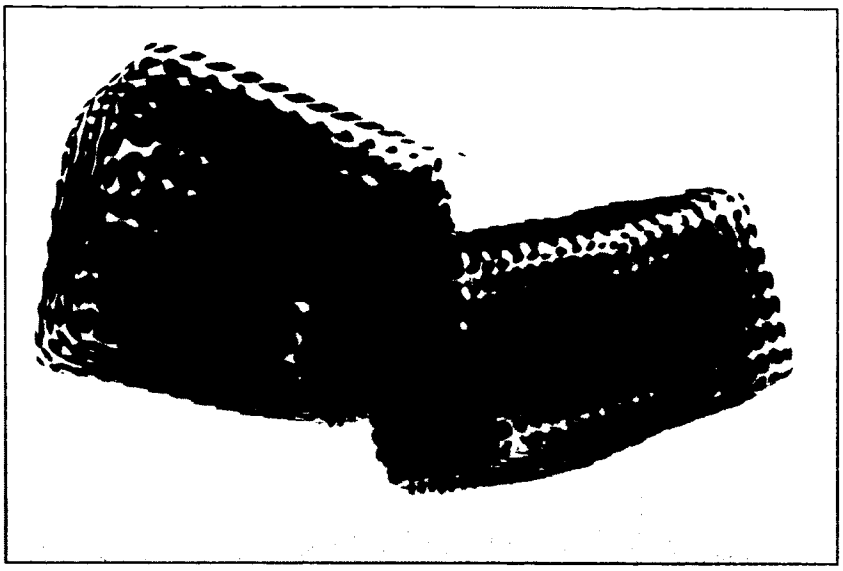


Fig. 78. The deformable block is cut, which results in the update of the behavior and visual models. The curvature of the affected surface points is re-calculated.

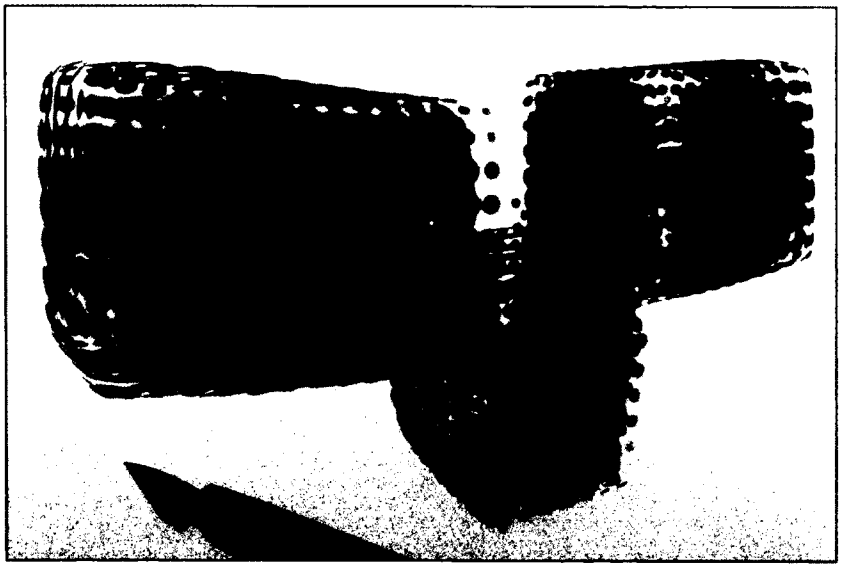


Fig. 79. The deformable block is cut a second time, completely separating a piece.

Fig. 80 shows the cut deformable block that is visualized with oriented point splat primitives. Although the radii of the affected splats are recalculated to conform to the new curvature values, some visual artifacts can still be seen because of the very sharp change in the splat orientation.

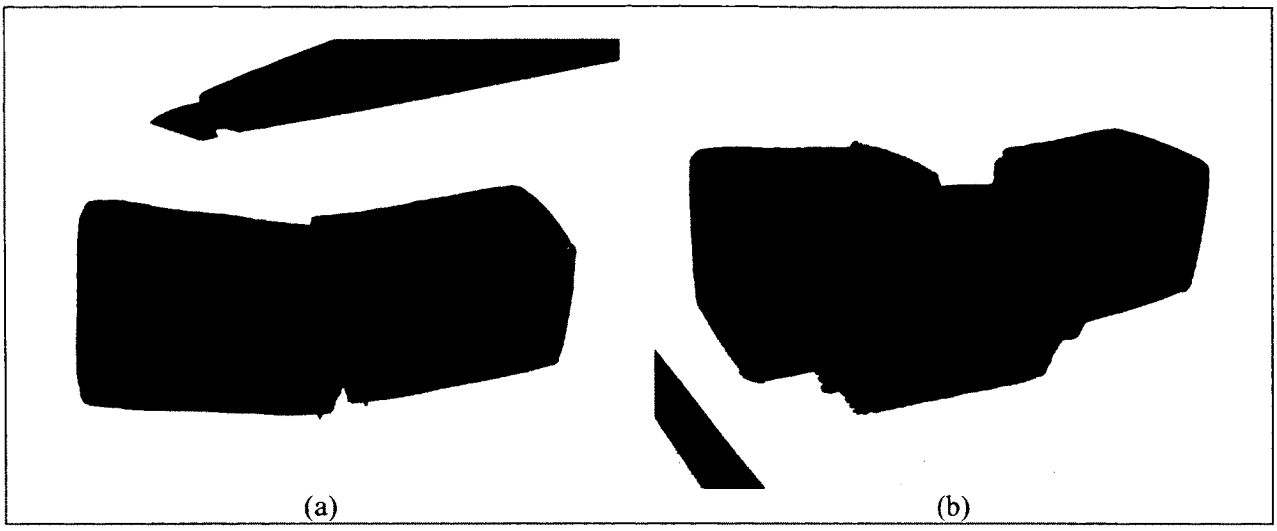


Fig. 80. The deformable block that underwent cut operations is visualized with oriented point splat primitives.

In addition to the deformable block, the liver model from the SOFA repository was also discretized to obtain the meshless node locations. The resulting point-based deformable model had 181 DoFs with an average number of 21 neighbors per each DoF, the simulation ran at around 120 FPS (Fig. 81, Fig. 82, Fig. 83, and Fig. 84).

Fig. 81. The meshless node locations of the liver object are obtained through hierarchical discretization.

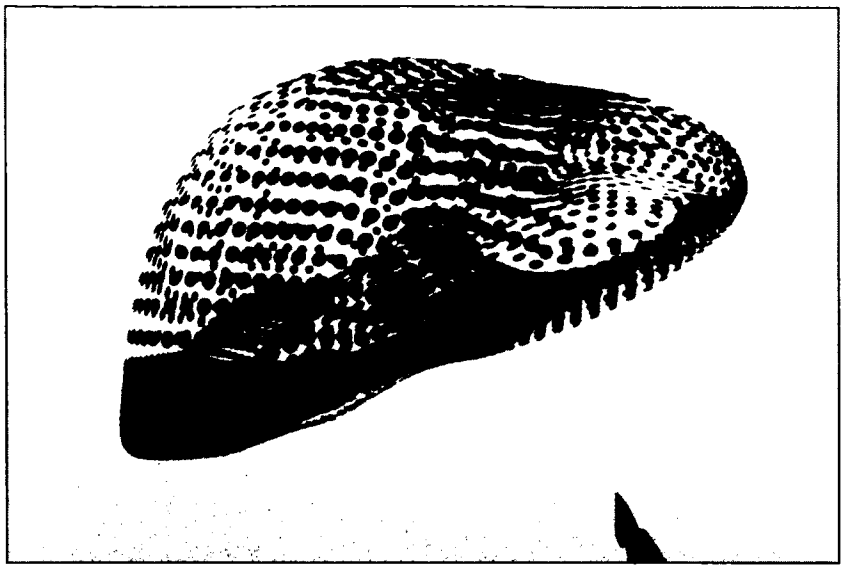


Fig. 82. Visual model of the liver object is represented by scaled-down oriented point splats.

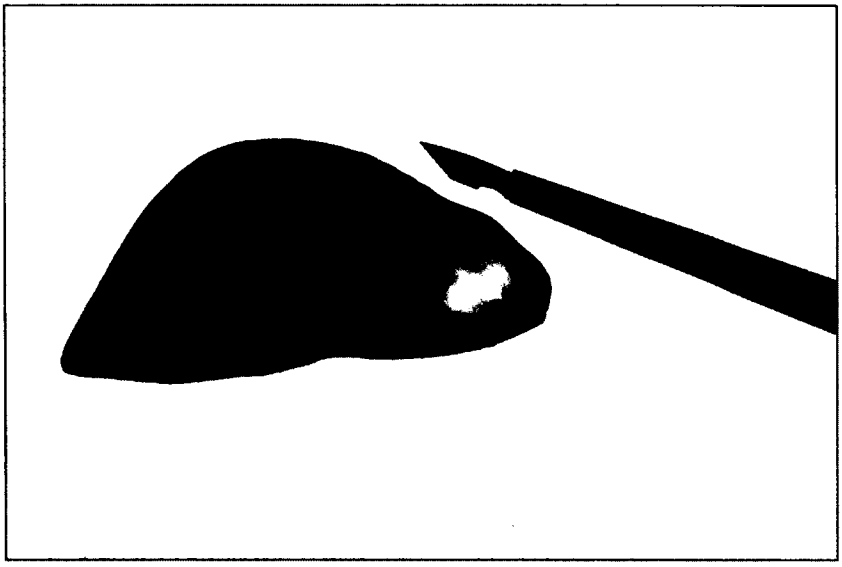


Fig. 83. The deformable liver model is visualized with oriented point splat primitives.

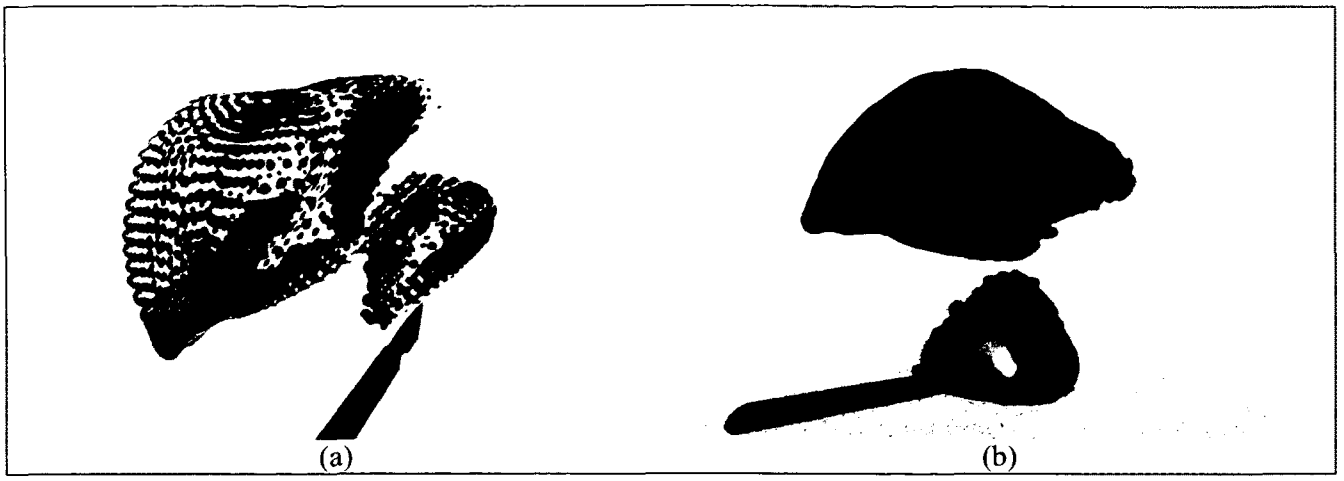


Fig. 84. An introduced cutting operation is visualized by (a) scaled-down point splats, and (b) regular-sized oriented point splat primitives.

CHAPTER 7

CONCLUSIONS

The feasibility of an entirely point-based approach for deformable body modeling in the context of surgical simulations was investigated in this dissertation. The problem was divided into three parts: assessing the use of point-based meshless methods for deformable object modeling, development of a visualization technique that does not rely on an explicitly defined mesh, and implementation of the cutting operation that takes advantage of these point-based approaches for the behavior and visual models. Each of these problems was examined separately by laying the theoretical foundation, which is followed by the presentation of the underlying implementation and algorithmic details.

The behavior model of the deformable object is formulated by first discretizing the domain of the simulation object with disconnected point primitives. The points in this discretization are positioned either by regular sampling for objects with regular geometric shapes or by tetrahedralization-based non-uniform sampling for objects with arbitrary shapes. After initializing the simulation related variables for each of the points such as density, volume, and inverse moment matrix, the internal elastic forces are computed at each time step by approximating the displacement gradient of the continuum with the Moving Least Squares (MLS) based approximation scheme. The behavior model implementation was verified by comparing the response of the model against the analytical solution of the Hertzian non-adhesive frictionless contact theory, which showed that the behavior model and the contact theory were in good agreement.

In the surgical simulation context, an important functionality that the soft tissue models need to have is the cutting operation. For the mesh-based models however, the cutting operation

is particularly problematic and usually becomes the bottleneck of the simulation in terms of performance. This dissertation presented a novel way of handling piecewise cut segments in 2D and 3D. The approach was achieved by extending the application of the mathematical enrichment function described earlier by Barbieri et al. [12]. The enrichment grid data structure that is proposed in this dissertation allowed the handling of consecutive cut segments in a correct way that prevented the occurrence of computational instabilities. This data structure has also proved itself to be a useful spatial query accelerator to find the meshless nodes as well as surface points that are affected by a cut.

Another aspect of the problem is the visualization of the simulation objects. For the scenarios where the object remains intact, i.e. the connectivity of the primitives is not disturbed by a discontinuity such as a cut; methods that use explicit meshes are the natural choice. When there is a cut though, these traditional methods suffer the same problems that the mesh-based deformable models do. Therefore, this dissertation adopted a purely point-based visualization technique that does not rely on explicit connectivity information, which enabled it to handle cuts in a more efficient way. Certain shortcomings of the point-based visualization technique were also described, while several solutions were proposed to address these problems.

The proposed soft-tissue model is based on the isotropic linear elastic material model. Although being a computationally efficient material definition, it may fail to reproduce the correct soft-tissue behavior when the linear property of the soft tissue falls out of a specific range of the strain-stress curve. The characteristic strain-stress curve of soft tissue is nonlinear, furthermore, they usually show anisotropic behavior due to the existence of fibers within the tissue [94, 95].

Another limitation of the presented work is the lack of a haptic force-feedback model. In a manner similar to this dissertation's visual model implementation, a local surface can be extracted from the neighborhood of the haptic tip and utilized in a haptic-constraint algorithm [96]. This approach only works for haptic rendering of the surface of the simulation object. Special care has to be taken during the haptic rendering of a cutting operation. This problem, investigated together with the effect of the fibrous structure of the soft tissue on haptic rendering, creates an interesting future research direction.

Point-based visualization gave promising results, when the deformable object's surface topology changed due to the introduced discontinuities. Although the presented curvature-based splat radius adaptation algorithm performed well and readjusted the radii of the splats that were affected by the cut, some visual artifacts can still be seen caused by the very sharp edges formed by the cut surfaces. Sharp edges need special processing in surface splatting-based visualizations such as sub-dividing the sharp regions into smaller splats [97] or clipping the splats that are along the sharp edges [98]. These techniques require multi-pass rendering algorithms, whose applicabilities need to be addressed within the SOFA framework.

Texture mapping is the process of mapping an image into a multidimensional space, and was pioneered by the work of Catmull [99]. Texture mapping is a cheap and easy way of increasing the visual realism of an object as it is supported natively by all modern graphics hardware. For an explicit mesh, each vertex of the surface is assigned a 2D coordinate called the texture coordinate, which maps a texture location to the vertex. After each vertex is assigned a texture coordinate, the connectivity information between the vertices is put into work to assign color values to the pixels in-between the vertices through interpolation. As the point-based visualization approach does not keep explicit connectivity information, obtaining the correct

color values in-between the point primitives becomes impossible with a single-pass algorithm. Although the previously discussed multi-pass surface splatting algorithm [63] implemented high-quality texture mapping, their approach did not allow visualizing point-based objects and mesh-based objects in the same scene, which is an important requirement for surgical simulation applications. Therefore, using objects having textured point-based visual models along with objects having other visual models is another possible research topic.

REFERENCES

[1] E. A. Ashley, "Medical education - beyond tomorrow?," *Medical Education,* vol. 34, pp. 455-459, 2000.

[2] B. Loftin, "Med school 1.0: Can computer simulation aid physician training," *Quest,* vol. 5, pp. 16-19, 2002.

[3] L. Grush. (2014, 04/23/2014). Surgical Theater: Flight simulation technology helps surgeons prep for surgery. Available: http://www.foxnews.com/health/2014/04/02/surgical-theater-flight-simulation-technology-helps-surgeons-prep-for-surgery/

[4] D. Ota, B. Loftin, T. Saito, R. Lea, and J. Keller, "Virtual reality in surgical education," *Computers in Biology and Medicine,* vol. 25, pp. 127-137, 1995.

[5] D. T. Chen, I. A. Kakadiaris, M. J. Miller, R. B. Loftin, and C. Patrick, "Modeling for plastic and reconstructive breast surgery," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2000*, 2000, pp. 1040-1050.

[6] J. Allard, S. Cotin, F. Faure, P. J. Bensoussan, F. Poyer, C. Duriez, *et al.*, "SOFA - An open source framework for medical simulation," in *Medicine Meets Virtual Reality*, 2007.

[7] 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.

[8] J. Wu, R. Westermann, and C. Dick, "Real-time haptic cutting of high-resolution soft tissues," *Studies in Health Technology and Informatics (Proc. Medicine Meets Virtual Reality 2014),* vol. 196, pp. 469-475, 2014.

[9] D. Steinemann, M. A. Otaduy, and M. Gross, "Splitting meshless deforming objects with explicit surface tracking," *Graphical Models,* vol. 71, pp. 209-220, 2009.

[10] N. Moës, J. Dolbow, and T. Belytschko, "A finite element method for crack growth without remeshing," *International Journal for Numerical Methods in Engineering,* vol. 46, pp. 131-150, 1999.

[11] P. Kaufmann, S. Martin, M. Botsch, E. Grinspun, and M. Gross, "Enrichment textures for detailed cutting of shells," *ACM Transactions on Graphics (TOG),* vol. 28, p. 50, 2009.

[12] E. Barbieri, N. Petrinic, M. Meo, and V. L. Tagarielli, "A new weight-function enrichment in meshless methods for multiple cracks in linear elasticity," *International Journal for Numerical Methods in Engineering,* pp. 177-195, 2011.

[13] T. W. Sederberg and S. R. Parry, "Free-form deformation of solid geometric models," *ACM Siggraph Computer Graphics,* vol. 20, pp. 151-160, 1986.

[14] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Transactions on Graphics (TOG),* vol. 24, pp. 471-478, 2005.

[15] S. F. Frisken-Gibson, "Using linked volumes to model object collisions, deformation, cutting, carving, and joining," *IEEE Transactions on Visualization and Computer Graphics,* vol. 5, pp. 333-348, 1999.

[16] K. Waters and D. Terzopoulos, "A physical model of facial tissue and muscle articulation," in *Visualization in Biomedical Computing*, 1990, pp. 77-82.

[17] A. Van Gelder, "Approximate simulation of elastic membranes by triangulated spring meshes," *Journal of Graphics Tools,* vol. 3, pp. 21-42, 1998.

[18] L. P. Nedel and D. Thalmann, "Real time muscle deformations using mass-spring systems," in *Computer Graphics International, 1998. Proceedings*, 1998, pp. 156-165.

[19] K.-J. Bathe, *Finite element procedures* vol. 2: Prentice hall Englewood Cliffs, 1996.

[20] R. Eymard, T. Gallouët, and R. Herbin, "Finite volume methods," *Handbook of numerical analysis,* vol. 7, pp. 713-1018, 2000.

[21] A. R. Mitchell and D. F. Griffiths, "The finite difference method in partial differential equations," *Chichester, Sussex, England and New York, Wiley-Interscience, 1980. 281 p,* 1980.

[22] M. Bro-Nielsen, "Finite element modeling in surgery simulation," *Proceedings of the IEEE,* vol. 86, pp. 490-503, 1998.

[23] K. Miller, G. Joldes, D. Lance, and A. Wittek, "Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation," *Communications in numerical methods in engineering,* vol. 23, pp. 121-134, 2007.

[24] S. Marchesseau, T. Heimann, S. Chatelin, R. Willinger, and H. Delingette, "Multiplicative jacobian energy decomposition method for fast porous visco-hyperelastic soft tissue model," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2010*, ed: Springer, 2010, pp. 235-242.

[25] Y. Chen, J. D. Lee, and A. Eskandarian, *Meshless Methods in Solid Mechanics*. Berlin: Springer Verlag, 2006.

[26] W. Ji, A. M. Waas, and Z. P. Bažant, "Errors caused by non-work-conjugate stress and strain measures and necessary corrections in finite element programs," *Journal of Applied Mechanics,* vol. 77, p. 044504, 2010.

[27] A. D. McNaught and A. Wilkinson, *IUPAC. Compendium of Chemical Terminology, 2nd ed. (the "Gold Book")* vol. 1669: Blackwell Scientific Publications, Oxford, 1997.

[28] K. Miller and K. Chinzei, "Constitutive modelling of brain tissue: experiment and theory," *Journal of Biomechanics,* vol. 30, pp. 1115-1121, 1997.

[29] I. S. Sokolnikoff and R. D. Specht, *Mathematical Theory of Elasticity* vol. 83. New York: McGraw-Hill, 1956.

[30] V. P. Nguyen, T. Rabczuk, S. Bordas, and M. Duflot, "Meshless methods: A review and computer implementation aspects," *Mathematics and Computers in Simulation,* vol. 79, pp. 763-813, 2008.

[31] H. Li and S. S. Mulay, *Meshless Methods and Their Numerical Properties*: CRC Press, 2013.

[32] L. B. Lucy, "A numerical approach to the testing of the fission hypothesis," *The Astronomical Journal,* vol. 82, pp. 1013-1024, 1977.

[33] N. Sukumar. (2002, 08/01/2012). *The Natural Element Method (NEM) in Solid Mechanics*. Available: http://dilbert.engr.ucdavis.edu/~suku/nem/

[34] B. Nayroles, G. Touzot, and P. Villon, "Generalizing the finite element method: Diffuse approximation and diffuse elements," *Computational mechanics,* vol. 10, pp. 307-318, 1992.

[35] T. Belytschko, Y. Y. Lu, and L. Gu, "Element-free Galerkin methods," *International Journal for Numerical Methods in Engineering,* vol. 37, pp. 229-256, 1994.

[36] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa, "Point based animation of elastic, plastic and melting objects," in *ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2004, pp. 141-151.

[37] X. Zhang, X. H. Liu, K. Z. Song, and M. W. Lu, "Least-squares collocation meshless method," *International Journal for Numerical Methods in Engineering,* vol. 51, pp. 1089-1100, 2001.

[38] A. Horton, A. Wittek, G. R. Joldes, and K. Miller, "A meshless total Lagrangian explicit dynamics algorithm for surgical simulation," *International Journal for Numerical Methods in Biomedical Engineering,* vol. 26, pp. 977-998, 2010.

[39] M. Pauly, R. Keiser, B. Adams, P. Dutr\, \#233, M. Gross, *et al.*, "Meshless animation of fracturing solids," *ACM Trans. Graph.,* vol. 24, pp. 957-964, 2005.

[40] H. Si, "TetGen: A quality tetrahedral mesh generator and three-dimensional delaunay triangulator," *Weierstrass Institute for Applied Analysis and Stochastic, Berlin, Germany,* 2006.

[41] J.-D. Boissonnat, O. Devillers, M. Teillaud, and M. Yvinec, "Triangulations in CGAL," in *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, 2000, pp. 11-18.

[42] D. Lübke. (2013, 03/17/2014). *QTetraMesher*. Available: http://qtm.dennis2society.de

[43] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM national conference*, 1968, pp. 517-524.

[44] I. M. Gelfand and S. V. Fomin, *Calculus of variations*: Courier Dover Publications, 2000.

[45] D. Organ, M. Fleming, T. Terry, and T. Belytschko, "Continuous meshless approximations for nonconvex bodies by diffraction and transparency," *Computational mechanics,* vol. 18, pp. 225-235, 1996.

[46] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics,* vol. 5, pp. 349-359, 1999.

[47] N. Amenta, S. Choi, and R. K. Kolluri, "The power crust," in *Proceedings of the sixth ACM symposium on Solid modeling and applications*, 2001, pp. 249-266.

[48] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006.

[49] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics (TOG),* vol. 32, p. 29, 2013.

[50] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel, "Multi-level partition of unity implicits," in *ACM SIGGRAPH 2005 Courses*, 2005, p. 173.

[51] S. Fleishman, D. Cohen-Or, and C. T. Silva, "Robust moving least-squares fitting with sharp features," in *ACM Transactions on Graphics (TOG)*, 2005, pp. 544-552.

[52] F. Calakli and G. Taubin, "SSD: Smooth signed distance surface reconstruction," in *Computer Graphics Forum*, 2011, pp. 1993-2002.

[53] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM Siggraph Computer Graphics,* vol. 21, pp. 163-169, 1987.

[54] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," *ACM Transactions on Graphics (TOG),* vol. 21, pp. 339-346, 2002.

[55] J. C. Hart, "Ray tracing implicit surfaces," *Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces,* pp. 1-16, 1993.

[56] J. M. Singh and P. Narayanan, "Real-time ray tracing of implicit surfaces on the GPU," *IEEE Transactions on Visualization and Computer Graphics,* vol. 16, pp. 261-272, 2010.

[57] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross, "Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces," in *Computer Graphics Forum*, 2005, pp. 303-312.

[58] M. Levoy and T. Whitted, "The use of points as a display primitive," University of North Carolina, Department of Computer Science TR 85-022, 1985.

[59] G. Schaufler and H. W. Jensen, "Ray tracing point sampled geometry," in *Rendering Techniques 2000*, ed: Springer, 2000, pp. 319-328.

[60] A. Adamson and M. Alexa, "Approximating and intersecting surfaces from points," in *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 2003, pp. 230-239.

[61] A. Adamson and M. Alexa, "Ray tracing point set surfaces," in *Shape Modeling International, 2003*, 2003, pp. 272-279.

[62] D. Schreiner, *Open GL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3*: Addison Wesley, 2013.

[63] S. Rusinkiewicz and M. Levoy, "QSplat: A multiresolution point rendering system for large meshes," in *Computer Graphics and Interactive Techniques*, 2000, pp. 343-352.

[64] H. Gouraud, "Continuous shading of curved surfaces," *IEEE Transactions on Computers,* vol. 100, pp. 623-629, 1971.

[65] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM,* vol. 18, pp. 311-317, 1975.

[66] M. Botsch, M. Spernat, and L. Kobbelt, "Phong splatting," in *Proceedings of the First Eurographics conference on Point-Based Graphics*, 2004, pp. 25-32.

[67] S. Gumhold, X. Wang, and R. MacLeod, "Feature extraction from point clouds," in *Proceedings of 10th international meshing roundtable*, 2001.

[68] M. Pauly, M. Gross, and L. P. Kobbelt, "Efficient simplification of point-sampled surfaces," in *Proceedings of the conference on Visualization'02*, 2002, pp. 163-170.

[69] J. Lander, "Skin them bones: Game programming for the web generation," *Game Developer Magazine,* vol. 5, pp. 10-18, 1998.

[70] K. Montgomery, C. Bruyns, J. Brown, S. Sorkin, F. Mazzella, G. Thonier, *et al.*, "Spring: A general framework for collaborative, real-time surgical simulation," *Studies in Health Technology and Informatics,* pp. 296-303, 2002.

[71] M. C. Cavusoglu, T. Goktekin, F. Tendick, and S. Sastry, "GiPSi: An open source/open architecture software development framework for surgical simulation," *Studies in Health Technology and Informatics,* pp. 46-48, 2004.

[72] M. Nakao, T. Kuroda, M. Komori, and H. Oyama, "Evaluation and User Study of Haptic Simulator for Learning Palpation in Cardiovascular Surgery," in *ICAT*, 2003.

[73] J. Bacon, N. Tardella, J. Pratt, and J. H. J. English, "The surgical simulation and training based language for medical simulation," *Medicine Meets Virtual Reality 14: Accelerating Change in Healthcare: Next Medical Toolkit,* vol. 119, p. 37, 2006.

[74] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik, S. Cotin, "SOFA: A multi-model framework for interactive physical simulation," in *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, ed: Springer, 2012, pp. 283-321.

[75] D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing,* vol. 19, pp. 129-147, 1982.

[76] T. L. Kay and J. T. Kajiya, "Ray tracing complex scenes," in *ACM SIGGRAPH Computer Graphics*, 1986, pp. 269-278.

[77] H. Fuchs, Z. M. Kedem, and B. F. Naylor, "On visible surface generation by a priori tree structures," in *ACM Siggraph Computer Graphics*, 1980, pp. 124-133.

[78] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM,* vol. 18, pp. 509-517, 1975.

[79] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *VISAPP (1)*, 2009, pp. 331-340.

[80] J. L. B. Claraco. (2013, 04/02/2014). *Nanoflann*. Available: https://code.google.com/p/nanoflann/

[81] C. Ericson, *Real-Time Collision Detection*. San Francisco, CA: CRC Press, 2004.

[82] C. Sanderson, "Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments," NICTA2011.

[83] The MathWorks, Inc. (2014, 04/04/2014). *MATLAB - The Language of Technical Computing*. Available: http://www.mathworks.com/products/matlab/

[84] H. Hertz, "Über die Berührung fester elastischer Körper," *Journal für die Reine und Angewandte Mathematik,* vol. 1882, pp. 156-171, 1882.

[85] D. Franke, A. Düster, and E. Rank, "The p-version of the FEM for computational contact mechanics," *Proceedings in Applied Mathematics and Mechanics,* vol. 8, pp. 10271-10272, 2008.

[86] N. Schwarzer, H. Djabella, F. Richter, and R. Arnell, "Comparison between analytical and FEM calculations for the contact problem of spherical indenters on layered materials," *Thin Solid Films,* vol. 270, pp. 279-282, 1995.

[87] D. Franke, A. Düster, V. Nübel, and E. Rank, "A comparison of the h-, p-, hp-, and rp-version of the FEM for the solution of the 2D Hertzian contact problem," *Computational Mechanics,* vol. 45, pp. 513-522, 2010.

[88] F. Pennec, H. Achkar, D. Peyrou, R. Plana, P. Pons, and F. Courtade, "Verification of contact modeling with COMSOL multiphysics software," in *Sixth EUROSIM Congress on Modelling and Simulation*, Ljubljana, Slovenia, 2007.

[89] S. A. Maas, B. J. Ellis, G. A. Ateshian, and J. A. Weiss, "FEBio: Finite elements for biomechanics," *Journal of Biomechanical Engineering,* vol. 134, pp. 110050-1--110050-10, 2012.

[90] T. Laursen and B. Maker, "An augmented Lagrangian quasi-Newton solver for constrained nonlinear finite element applications," *International Journal for Numerical Methods in Engineering,* vol. 38, pp. 3571-3590, 1995.

[91] M. Marchal, J. Allard, C. Duriez, and S. Cotin, "Towards a framework for assessing deformable models in medical simulation," in *Biomedical Simulation*, ed: Springer, 2008, pp. 176-184.

[92] F. Faure. (2014, 04/13/2014). *Getting Started - SOFAWiki.* Available: http://wiki.sofa-framework.org/wiki/Getting_Started

[93] C. T. McKee, J. A. Last, P. Russell, and C. J. Murphy, "Indentation versus tensile measurements of Young's modulus for soft biological tissues," *Tissue Engineering Part B: Reviews,* vol. 17, pp. 155-164, 2011.

[94] J. Humphrey, "Review Paper: Continuum biomechanics of soft biological tissues," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences,* vol. 459, pp. 3-46, 2003.

[95] Y. Fung, "Biomechanics: Mechanical properties of living tissues," ed: Springer-Verlag, New York, 1993, p. 568.

[96] A. Leeper, S. Chan, and K. Salisbury, "Point clouds can be represented as implicit surfaces for constraint-based haptic rendering," in *International Conference on Robotics and Automation (ICRA)*, Saint Paul, MN, USA, 2012, pp. 5000-5005.

[97] B. Adams and P. Dutre, "Interactive boolean operations on surfel-bounded solids," *ACM Transactions on Graphics (TOG),* vol. 22, pp. 651-656, 2003.

[98] M. Zwicker, J. Räsänen, M. Botsch, C. Dachsbacher, and M. Pauly, "Perspective accurate splatting," in *Proceedings of Graphics interface 2004*, 2004, pp. 247-254.

[99] E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces," PhD Thesis, University of Utah, 1974.

APPENDIX

The following is the generated Doxygen documentation of the components that are implemented in the SOFA framework.

sofa::component::forcefield::MeshlessForceField

Defines how meshless nodes behave according to the underlying constitutive equations.

Member Function Documentation

template<class DataTypes >
bool sofa::component::forcefield::MeshlessForceField< DataTypes >::getMaterialCoord
(const Coord & world,
Coord & mater
)

Get the material coordinate given world coordinate. Used when there is a cut.

Parameters
world the world coordinate to convert.
mater the material coordinate (OUT).

Returns
TRUE if converted successfully.

template<class DataTypes >
bool sofa::component::forcefield::MeshlessForceField< DataTypes >::getMaterialCoordApprx
(const Coord & world,
Coord & mater
)

Get the approximate material coordinate given world coordinate. Used when getMaterialCoord returns false.

Parameters
world the world coordinate to convert.
mater the approximate material coordinate (OUT).

Returns
TRUE if converted successfully.

template<class DataTypes >
void sofa::component::forcefield::MeshlessForceField< DataTypes >::getPhyxLinIndices
(arma::uvec & indices)

Get the linear indices of the meshless nodes to be used to query the meshless enrichment grid.

Parameters
indices The armadillo vector that contains the indices (OUT).

template<class DataTypes >
void sofa::component::forcefield::MeshlessForceField< DataTypes >::handleEvent (core::objectmodel::Event * event)
virtual

Processes the MeshlessCutEvent received from the MeshlessHapticDevice and first, converts the world coordinates to material coordinates. Then call the appropriate function of the EnrichmentGridManager.

template<class DataTypes >
void sofa::component::forcefield::MeshlessForceField< DataTypes >::init ( )

virtual

Initialization tasks:

Nodes should be already distributed in the mechanical state object
For each particle, evaluate the average distance to k-nearest neighbors Use this information to compute the support radius
Set particle neighborhood information (if the distance between nodes is smaller than the support)
Evaluate density scaling constant which is used to set the appropriate densities for the nodes
Evalute the mass, density, and volume of the particles
Evalute the inverse moment matrices of the particles
Do precomputations related to the enrichment grid.

Member Data Documentation

template<class DataTypes>
Data< Real > sofa::component::forcefield::MeshlessForceField< DataTypes >::dampingConstant

The damping constant for applying drag force. Scene graph name: dampingConstant

template<class DataTypes>
Data< Real > sofa::component::forcefield::MeshlessForceField< DataTypes >::densityScale

The scaling factor for all nodes, chosen s.t. phyxel densities are close to material density. Scene graph name: densityScale

template<class DataTypes>
Data< int > sofa::component::forcefield::MeshlessForceField< DataTypes >::displayEnrGridIndex

Display which enrichment grid? One enrichment grid is recorded for each cut. (DEBUG) Scene graph name: DisplayEnrichmentGridIndex

template<class DataTypes>
Data< bool > sofa::component::forcefield::MeshlessForceField< DataTypes >::displayEnrichmentGrid

Display the enrichment grid in material coordinates? (DEBUG) Scene graph name: DisplayEnrichmentGrid

template<class DataTypes>
Data< bool > sofa::component::forcefield::MeshlessForceField< DataTypes >::displayNeighbors

Display neighbors of a meshless node by drawing lines in-between. (DEBUG) Scene graph name: DisplayNeighbors

template<class DataTypes> Data< int > sofa::component::forcefield::MeshlessForceField< DataTypes >::displayNeighborsIdx Display the neighbors of which meshless node? (DEBUG) Scene graph name: DisplayNeighborsIndex
template<class DataTypes> Data< int > sofa::component::forcefield::MeshlessForceField< DataTypes >::kNearest The number of nearest neighbors to consider during initialization. Scene graph name: kNearest
template<class DataTypes> Data< Real > sofa::component::forcefield::MeshlessForceField< DataTypes >::materialDensity Material density in kg/m3. Scene graph name: materialDensity
template<class DataTypes> Data< int > sofa::component::forcefield::MeshlessForceField< DataTypes >::neighborLimit To limit number of neighbors for each node? (-1 no limit i.e. all nodes inside support radius are neighbors) Scene graph name: neighborLimit
template<class DataTypes> Data< Real > sofa::component::forcefield::MeshlessForceField< DataTypes >::poissonsRatio Poisson's Ratio elastic property. Scene graph name: poissonsRatio
template<class DataTypes> Data< Real > sofa::component::forcefield::MeshlessForceField< DataTypes >::supportRadiusScale Average distance to k-nearest neighbors is multiplied with this scaling constant to obtain support radius for the node. Scene graph name: supportScale
template<class DataTypes> Data< Real > sofa::component::forcefield::MeshlessForceField< DataTypes >::volumeConstant The volume conservation constant kV. Scene graph name: volumeConstant
template<class DataTypes> Data< Real > sofa::component::forcefield::MeshlessForceField< DataTypes >::youngsModulus Young's Modulus elastic property in Pa. Scene graph name: youngsModulus

PointBasedMethodsPlugin::MeshlessEnrichmentGridManager

The manager class that holds information regarding the calculated enrichments. For each new cut, this class generates new 3D grid and provide these grids to the related components.

Member Function Documentation

void PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::endCut (const sofa::defaulttype::Vector3 & pt0, const sofa::defaulttype::Vector3 & pt1) Called when the previously started cut is completed. Parameters

pt0 The first point of the cutting edge in MATERIAL coordinates. pt1 The second point of the cutting edge in MATERIAL coordinates.
void PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getCutInformation (Vector3Vector & cutPts, std::vector< double > & cutAreas, Vector3Vector & cutNormals) Return the cut information that were used to compute the last enrichment grid values. Parameters cutPts The vector of 3D points that form the cut surface (OUT). cutAreas The vector of area values of the triangles that form the cut surface (OUT). cutNormals The vector of 3D normals of the triangles that form the cut surface (OUT).
double PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getGridDX ( ) const inline Return the grid cell size in material x-dimension.
double PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getGridDY ( ) const inline Return the grid cell size in material y-dimension.
double PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getGridDZ ( ) const inline Return the grid cell size in material z-dimension.
const arma::vec * PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getLastDistanceGridVec ( ) Return the last distance grid computed to be used by the requesting components.
const arma::vec * PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getLastEnrichmentGridVec ( ) Return the last enrichment grid computed to be used by the requesting components.
int PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getResX ( ) const inline Return the number of grid cells in x-dimension.
int PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getResY ( ) const inline Return the number of grid cells in y-dimension.

int PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getResZ ( )
const
inline

Return the number of grid cells in z-dimension.

void PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::getTriCenters (
Vector3Vector & triCents)

Return the cut triangle centers that were used to compute the last enrichment grid values.

Parameters
triCents The vector of 3D coordinates of the triangle centers (OUT).

void PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::startCut (const sofa::defaulttype::Vector3 & pt0,
const sofa::defaulttype::Vector3 & pt1
)

Called when a new cut is started. Creates the necessary grid constructs.

Parameters
pt0 The first point of the cutting edge in MATERIAL coordinates.
pt1 The second point of the cutting edge in MATERIAL coordinates.

void PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::updateCut (const sofa::defaulttype::Vector3 & pt0,
const sofa::defaulttype::Vector3 & pt1
)

Called when there is a new cut edge of a previously started cut. Updates the contents of the associated grid structures.

Parameters
pt0 The first point of the cutting edge in MATERIAL coordinates.
pt1 The second point of the cutting edge in MATERIAL coordinates.

Member Data Documentation

Data<double> PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::inflateAmount

The size of the bbox of the enrichment grids are determined automatically from the enclosed object. This is the buffer amount to add to the bbox.

Data< int > PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::resX

Number of grid cells in x-dimension.

Data< int > PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::resY

Number of grid cells in y-dimension.

Data< int > PointBasedMethodsPlugin::MeshlessEnrichmentGridManager::resZ

Number of grid cells in z-dimension.

sofa::component::container::MechanicalOrientedObject

Derived from the MechanicalObject, this class keeps the DoFs for the OrientedPointSplatModel visual model.

Member Function Documentation

template<class DataTypes >
void sofa::component::container::MechanicalOrientedObject< DataTypes >::addOrientedPoints
(const sofa::helper::vector< Vec3d > & pos,
const sofa::helper::vector< Vec3d > & nor
)

Adds new DoFs to the SOFA vectors. Used when there is a cut.

Parameters
pos the standard vector that holds new positions.
nor the standard vector that holds new normals.

Member Data Documentation

template<class DataTypes>
Data< Deriv > sofa::component::container::MechanicalOrientedObject< DataTypes >::defaultNormal
protected

The default normal vector to assign when a new DoF is added and the vector is not specified.

template<class DataTypes>
Data< VecDeriv > sofa::component::container::MechanicalOrientedObject< DataTypes >::normals
protected

The SOFA vector that holds the additional normal information for the oriented splats.

template<class DataTypes>
Data< bool > sofa::component::container::MechanicalOrientedObject< DataTypes >::showNormals
protected

Show normals (DEBUG).

template<class DataTypes>
Data< float > sofa::component::container::MechanicalOrientedObject< DataTypes
>::showNormalsScale
protected

The scale of the visualized normal vectors (DEBUG).

template<class DataTypes>
Data< VecTexUV > sofa::component::container::MechanicalOrientedObject< DataTypes >::texCoords
protected

The SOFA vector that holds the texture coordinates for the oriented splats.

sofa::component::mapping::SurfelMapping

The deformable model controls the visual model. This class defines the relation between the two models. It maps the MechanicalObject of the deformable body to the OrientedMechanicalObject of the visual model.

Member Data Documentation

template<class TIn , class TOut > Data< Real > sofa::component::mapping::SurfelMapping< TIn, TOut >::cutOpening The amount that two surfels are separated when creating new cut surfaces.
template<class TIn , class TOut > Data< int > sofa::component::mapping::SurfelMapping< TIn, TOut >::cutPointDensity Number of points to sample on the cut surface per unit area.
template<class TIn , class TOut > Data< bool > sofa::component::mapping::SurfelMapping< TIn, TOut >::displayNeighbors Show the neighborhood information? (DEBUG)
template<class TIn , class TOut > Data< int > sofa::component::mapping::SurfelMapping< TIn, TOut >::displayNeighborsIdx The index of surfel to show the neighborhood information (DEBUG).
template<class TIn , class TOut > Data< int > sofa::component::mapping::SurfelMapping< TIn, TOut >::kNearestPhyxel Number of nearest phyxels to find when computing the relation between the two models.

sofa::component::visualmodel::OrientedPointSplatModel

The visual model that implements surface splatting algorithm.

Member Function Documentation

void sofa::component::visualmodel::OrientedPointSplatModel::updateSurfelRadii (int nbAddedSurfels, const arma::uvec & cutIndices) Called when there is a new cut, and the splat radii has to be updated because of the changed neighborhood.

PointBasedMethodsPlugin::MeshlessCuttingDevice

Extends the SofaHAPIHapticsDevice by adding functionality to handle cutting operation for meshless objects. Cutting starts by pushing the primary button on the haptic stylus.

Member Data Documentation

Data<int> PointBasedMethodsPlugin::MeshlessCuttingDevice::cutCaptureRate Capture a cutting edge and send it as an MeshlessCutEvent every this number of frame.
Data<Vec3d> PointBasedMethodsPlugin::MeshlessCuttingDevice::cutEdgePt0 The location of the first point of the cutting edge in this device's coordinate system.
Data<Vec3d> PointBasedMethodsPlugin::MeshlessCuttingDevice::cutEdgePt1 The location of the second point of the cutting edge in this device's coordinate system.

sofa::core::objectmodel::MeshlessCutEvent

This event notifies about haptic device interaction.

Member Enumeration Documentation

enum sofa::core::objectmodel::MeshlessCutEvent::EventType The type of the meshless cut event. START Starting a new cut. UPDATE Updating a new cut. END The cut is completed.

Member Function Documentation

sofa::defaulttype::Vector3 sofa::core::objectmodel::MeshlessCutEvent::getPt0 (void) const inline Return the first point of the cut edge in world coordinates.
sofa::defaulttype::Vector3 sofa::core::objectmodel::MeshlessCutEvent::getPt1 (void) const inline Return the second point of the cut edge in world coordinates.

VITA

Rifat Aras

Modeling, Simulation, and Visualization Engineering Department

Old Dominion University

Norfolk, VA 23529

Educational Background

Ph.D.: August 2014, Old Dominion University, Norfolk, VA, USA
Major: Modeling and Simulation
Dissertation: Meshless Mechanics and Point-Based Visualization Methods for Surgical Simulators

M.Sc.: January 2008, Bilkent University, Ankara, Turkey
Major: Computer Science
Thesis: 3D Hair Sketching for Real-Time Hair Modeling and Dynamic Simulations

Professional Background

Central Bank of Republic of Turkey - Database Administrator (2006-2009)

Worked as a database administrator of the IBM DB2 database management system running on multiple platforms like Linux, Windows, AIX, and z/OS.
Developed custom administration tools and a technology showcase web application that was presented at the annual International DB2 Users Group conference.

AYESAS - Software Engineer (2004-2005)

AYESAS (whose 30% ownership is of L3 communications), provides solutions in Aerospace and Defense business areas.
Worked as a part of the development team of the "SmartDeck" project, which involved of transforming mechanical and electronic avionics user interface elements into a central OpenGL-based interface.