Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2010)
J. Bender, K. Erleben, and M. Teschner (Editors)

GPU Accelerated Needle Insertion Simulation using Meshfree Methods

A. Shahingohar[1] S. deRibaupierre [2] and R. Eagleson[1]

[1]Department of Electrical and Computer Engineering, University of Western Ontario, London, Canada
[2]Department of Clinical Neurological Sciences, University of Western Ontario, London, Canada

Abstract

Needle insertion is a common practice used in many different medical procedures. Therefore, simulation of needle insertion is of great importance for multiple purposes such as training, planning and robotic assisted interventions. Modeling of soft tissue plays an important role in the needle insertion simulation, but the use of mesh based methods such as the Finite Element Method is frustrated by the need for remeshing in the neighbourhood of the needle tip. We have developed a novel method that uses meshfree methods for the tissue deformation model. In this method new tissue nodes are added on the needle shaft as the needle is inserted into the tissue. We have used a stack based approach to keep the state of the model; therefore, we have avoided over-sampling the model due to continuous needle insertion and extraction. Using this approach we have simulated the insertion of a straight rigid needle into soft tissue. In addition, we have utilized Nvidia's CUDA technology to accelerate the methods used in our framework. Our framework allows dynamic resampling and addition of new nodes while using CUDA. Our results show the usability and flexibility of the new method. By using the CUDA technology we were able to achieve up to 20 times speed for large meshes.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality I.3.1 [Computer Graphics]: Hardware Architecture—Graphics processors

Keywords: Needle Insertion Simulation, Deformable Object Modeling, CUDA, GPGPU

1. Introduction

Simulation of needle insertion is an important research area that has many applications in robotic and image guided operations such as brachytherapy cancer treatment, biopsies, neurosurgery and injections [APM07]. The success of these operations depends on the accuracy of reaching the target by the needle; however, in most cases the deflection of the needle, deformation of the soft tissue, imaging limitations and human error result in poor accuracy [APM07, RNM*97]. Modeling of soft tissue plays an important role in robotic needle insertion [DS02, DS03] and it is also essential for needle insertion planing and simulation [CAR*09]. Simulation of needle insertion is known to be challenging because of the disagreement in discretizations between needle and soft tissue. We will show here how our particle based framework can be an intuitive and efficient approach to simplify the discretization aspect on the tissue.

Modeling deformable objects such as soft tissue is still computationally expensive; however, most of the related calculations can be performed faster if they are executed in parallel. In [SE10] a framework was developed that utilized NVIDIA's CUDA technology [CUD10] to accelerate several classical methods in deformable object modeling by transferring their core computations to the GPU. Moreover, a new method called Local Shape Matching [SE10] was introduced based on the Shape Matching method [MHTG05].

1.1. Our Contributions

The main contributions of our work on needle insertion simulation are:

An algorithm that uses meshfree methods for modeling soft tissue.

DOI: 10.2312/PE/vriphys/vriphys10/135-144

A resampling approach based on the Shape Matching applicable to meshfree methods.
A stack-based handling of resampling that avoids oversampling during needle insertion/extraction.
Parallel implementation on the GPU while dynamic resampling is allowed.

In the next section we will review the literature on needle insertion simulation. In section 4 we outline the GPU-accelerated framework used in our needle insertion simulation. Then we introduce our new method developed for needle insertion simulation. Our results are provided in section 6. Finally, we will conclude and provide our directions for future research.

2. Related Work

Simulation of needle insertion in soft tissue is a very challenging application of deformable objet modeling, since special algorithms should be used to enforce the boundary conditions between needle and tissue. In this section we review different methods used to simulate needle and tissue interaction and the methods used to simulate the deformation.

2.1. Needle Tissue Interaction

Modeling the interaction forces between needle and tissue is very important in simulation of needle insertion. It affects tissue and needle deformation, trajectory of needle inside issue and the feedback force generated for simulation or control. It is known that before the puncture, the tissue applies a greater force on the needle because it is interacting with the membrane. After the puncturing, the tissue exerts a steady force due to friction and cutting which is less than maximum force at puncture. During extraction a negative force is applied to the needle due to friction only. In order to achieve similar behavior Simone et. al. [SO02] proposed a model in which the process is divided into two steps. Pre-puncture and post-puncture. In pre-puncture, the force increases steadily which is followed by a sharp drop in the amount of force due to puncturing of the surface of the tissue. During post-puncture, the amount of force depends on friction, cutting and collision with interior structures.
Crouch et. al. [CSWO05] further studied the dynamic effects such as relaxation on needle insertion. They recorded the position of the needle tip and the tissue needle forces during insertion and they repeated this experiment with different insertion velocities. They concluded that the tissue deformation and needle forces are time and velocity dependent. They also observed a gradual reduction in force after the needle halted. Recently Chentanez et. al. [CAR*09] used a stick-slip model of the friction between the tissue and the needle shaft. In static friction state the needle and the tissue move in lock step; in dynamic friction state they slide against each other. They derived and numerically solved the coupled equations of needle and tissue in both static and friction states.

2.2. Soft Tissue Modeling

Modeling of tissue deformation can become very complex since tissues are usually inhomogeneous, nonlinear, anisotropic, elastic, and exhibit the viscous behavior. Mass-spring and linear explicit finite element models are the most widely used methods for modeling tissue deformations.
Zhu et. al. [ZMRK07] used a localized mass-spring model to simulate needle tissue interaction. They assumed that needle only causes deformation in the region local to the insertion path. Therefore, once the needle is inside, a cylindrical deforming field centered at the predicted insertion path (i.e., the current direction of the needle) is created in the form of a tetrahedral mesh. In order to ensure that during the insertion a mesh node is always constrained at the tip and all penetrated nodes are constrained to only move along the needle shaft, they used a first-in-last-out stack, to store the shaft nodes.
DiMaio and Salcudean [DS02, DS03, DS05] used the Finite Element Method to model soft tissue deformation for their realtime haptic simulation system. Their system allowed the users to experience both visual and kinesthetic feedback while executing a virtual planar needle insertion. In [DS02,DS03] they described a method for estimating needle shaft forces that occur during insertions into tissue phantoms, based on empirical results. They applied estimated constant needle shaft force distribution based on penetration depth to nodes that are in contact with the needle. Therefore, they were able to simulate needle penetration with a constant speed in a predefined path in two dimensions. In [DS05] they developed a method to simulate insertion of a flexible needle in soft tissue using FEM. In their approach they applied the force boundary conditions only to the nodes in direct contact with the needle shaft, and deformation was calculated only for those working nodes. Displacement boundary conditions in this simulation constrained tissue nodes to the needle geometry and varied for rigid and flexible needles in their model. Displacement boundary conditions for a flexible needle and force boundary conditions were calculated from physical experiments. This was completed to include arbitrary 3d meshes in [GSD*05]. In [DGM*09], the interactions between soft tissues and needle are modeled using a set of soft constraints, eliminating the need for remeshing the soft tissue. They introduce corrective forces to constrain the tissue nodes to the needle. Additional constrains are added to the system to model other tissue properties such as puncture.

2.3. Remeshing During Needle Insertion

In the Finite Element Method, the boundary condition can only be applied at nodes. On the other hand, during insertion of the needle, it can generally penetrate inside an element. Therefore, a major issue in modeling needle insertion using Finite Element Method is the necessity to perform a remeshing on the mesh to make sure that the needle tip is always aligned with a node inside the tissue.

The simplest form of enforcing the boundary condition is achieved by *Snapping* a node to the needle trajectory. However, this method applies artificial deformations. A more sophisticated method is to increase the resolution of the tissue mesh around the needle by subdividing the elements to smaller elements. The subdivision could be done uniformly by dividing elements with a predetermined proportions; another approach is to subdivide the element such that the newly added node coincides with the needle trajectory. Even with progressive subdividing, it is not guaranteed to have the needle tip conform with the needle shaft. Therefore, after certain steps either a snapping is applied or the closest nodes are constrained to needle. Nienhuys et. al. [NvdS03] used a nonconforming scheme, that avoided snapping the nodes. After a few subdivision of tissue mesh around the needle instead of snapping the nodes, the friction were are adjusted based on geometry.
Instead of remeshing on the deformed mesh, remeshing can be done on the reference undeformed mesh (material space) such that the mesh is aligned with the needle. Therefore, the node can be constrained to the needle shaft with fewer artificial deformation. During this operation both the mesh and the stiffness matrix of the elements that are adjacent to the moving node should be updated [DS05].
Chentanez et. al. [CAR*09] used combination of operations in the material space to achieve high quality meshes after remeshing. These operations included node snapping, edge splitting, face splitting, and the tetrahedron splitting.

3. Local Shape Matching Method

We have extended the concept of clusters in the Shape Matching method, such that a cluster is defined for each point. An overview of this approach is shown in Figure 1. At the beginning of the simulation for each node i the center of mass $\mathbf{C}_i^0$ is calculated for that node and its neighbours; also the vector ν_i from the center of mass to each node is stored. At each iteration of the simulation, rotation should be extracted from the deformation. The rotation is approximated using the least square optimization explained in [MHTG05]. As shown in Figure 1(c), extracting the rotation is equivalent to rotating the coordinate system in reverse $\mathbf{x}^R = \mathbf{R}^{-1}\mathbf{x}$. For each node i in the new rotated coordinate system, the goal position is located at $\mathbf{g}_i^R = \nu_i + \mathbf{C}_i^R$ where $\mathbf{C}_i^R$ is the new center of mass of node i and its neighbours in the rotated coordinate system. Once the goal position are found in the new coordinate system, the goal positions are transformed to the original coordinate system $\mathbf{g}_i = \mathbf{R}\mathbf{g}_i^R$. Instead of using the integration schema proposed by Müller et.al., we introduce a restoring force from the current position to the calculated goal position:

$$\mathbf{f}_i^s = k_s(\mathbf{g}_i - \mathbf{x}_i) \tag{1}$$

Using this approach will allow us to use the standard integration methods used for other methods. It turns out that we don't have to transform all the nodes to the rotated coordinate system. The same results can be achieved if we only rotate ν_i. An overview of our algorithm is given in Algorithm 1.

Algorithm 1 Our Local Shape Matching algorithm

```
{Initialization}
for all nodes $i$ do
    $\mathbf{C}_i^0 = \frac{1}{N}\sum_{j\in\mathcal{N}(\mathbf{x}_i)} \mathbf{x}_j^0$
    $\nu_i = \mathbf{x}_i^0 - \mathbf{C}_i^0$
end for
{At each iteration}
Approximate the rotation $\rightarrow \mathbf{R}$.
for all nodes $i$ do
    $\mathbf{C}_i = \frac{1}{N}\sum_{j\in\mathcal{N}(\mathbf{x}_i)} \mathbf{x}_j$
    $\mathbf{g}_i = \mathbf{C}_i + \mathbf{R}\nu_i$
    $\mathbf{f}_i = k_s(\mathbf{g}_i - \mathbf{x}_i)$
end for
{Explicit integration}
for all nodes $i$ do
    $\dot{\mathbf{x}}_i \leftarrow \dot{\mathbf{x}}_i + \triangle t \mathbf{f}_i / m_i$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \triangle t \dot{\mathbf{x}}_i$
end for
```

4. Our CUDA Accelerated Framework for Deformable Object Modeling

In [SE10] a framework was developed for animating deformable objects using different particle based methods. The framework efficiently take advantage used parallel processing power of GPU using CUDA technology. In addition to the Local Shape Matching Method(LSHM) method mentioned in the previous section, we have implemented weighted Mass-Spring system (MSM), Debunne's meshfree Finite Element method (DEB) [DDBC99] and Point Based Animation (PBA) method described in [MKN*04].
In this framework, for each node a set of arrays are generated to store variables associated with the simulation. These variable are stored for all nodes and include position, velocity, elastic force, list of neighbours and weight of each neighbour. Other than these variable there are some variables that are specific to each method and should be declared separately. These arrays generally keep the information regarding to the shape of each object. In this paper the arrays specific to each method are known as *Shape Information* arrays. All arrays are created and initialized at the beginning of simulation on the host CPU and copied into GPU device. However, during simulation only positions and velocities are sent back and forth between host and device. Positions and velocities are transferred to host for collision detection, the corrected values are sent back to GPU for next iteration.
Table 1 shows the proprieties of each variable that is created on the GPU. Velocity, position and force are of factor 4 instead of 3. The reason is that in our CUDA kernel we use float4 since CUDA is not optimized for access

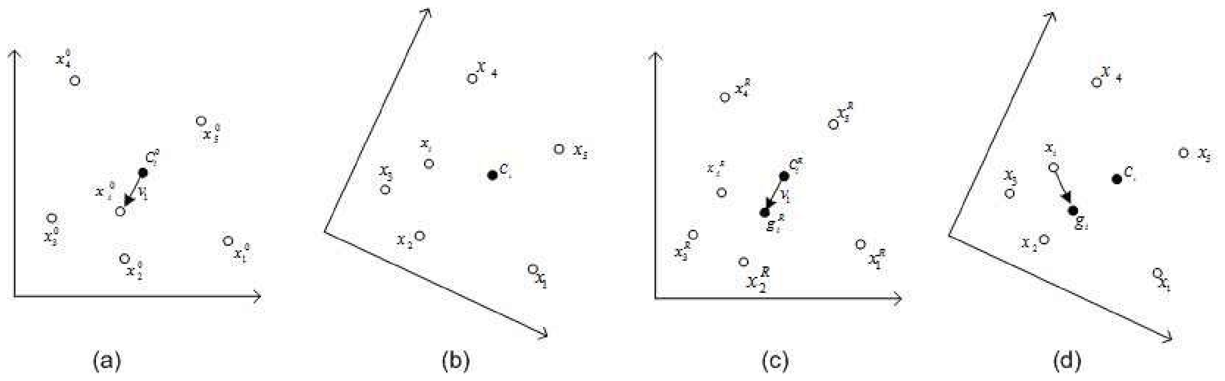


Figure 1: *Local Shape Matching. (a) Node $\mathbf{x}_i$ and its neighbours at the beginning of the simulation. $\mathbf{C}_i^0$ is the center of mass node i and its neighbours, $\nu_i = \mathbf{x}_i^0 - \mathbf{C}_i^0$ (b) Node $\mathbf{x}_i$ and its neighbours after deformation. (c) The rotation is extracted by rotating the coordinate system. Then the goal position is calculated at the rotated coordinate system; it is located at the same vector position from the center of the mass $\mathbf{g}_i^R = \nu_i + \mathbf{C}_i^R$. (d) Goal positions are rotated back to the original coordinate system and a force is applied in the direction of $\mathbf{x}_i - \mathbf{g}_i^R$.*

Table 1: *Data arrays and their sizes*

Array Name	Type	Size	GPU Memory	Transfer
Position	float4	$4 \times n\_Nodes$	Global	Each Iteration
Velocity	float4	$4 \times n\_Nodes$	Global	At Initialization
Force	float4	$4 \times n\_Nodes$	Global	Each Iteration
Neighbours	int	$n\_Neighbours \times n\_Nodes$	Texture	At Initialization
Nei_Weights	float	$n\_Neighbours \times n\_Nodes$	Texture	At Initialization
Method Specific				
node distances (MSM)	float	$n\_Neighbours \times n\_Nodes$	Texture	At Initialization
ν_i (LSHM)	float4	$4 \times n\_Nodes$	Texture	At Initialization
Original Positions (DEB,PBA)	float4	$4 \times n\_Nodes$	Texture	At Initialization
$\mathbf{M}^{-1}$ (PBA)	float	$9 \times n\_Nodes$	Texture	At Initialization

of type float3. The Neighbours array keeps the index of *n_Neighbours* closest nodes to each node and effect of each neighbour is weighted according a weight function. We have used two kernel function in our simulation. The first kernel calculates the forces(or accelerations) and the second kernel is used for explicit integration.

5. Description of Meshfree Needle Insertion Method

We have developed an algorithm to use meshfree methods for needle insertion simulation. Our main contribution is that instead of remeshing or snapping the nodes, we add nodes to the system. Adding a node is done in such a way that it does not impose any artificial strain in the tissue. We have used the GPU based framework introduced in the previous section to accelerate the simulation. In this section we explain the additions that enables us to simulate needle insertion. While we have used the concept of *Shape Vector* explained in section 3 for the purpose of resampling, the developed algorithm can be applied to any meshfree approach as explained in this section.

5.1. Adding a Node

In FEM the tissue is remeshed at the needle tip as the needle is inserted. Instead of remeshing, we add nodes to the system as the needle is inserted. A node is added and placed at the needle tip if the distance between the needle tip and the last added node is greater than a threshold. When a new node is added to the mesh, a corresponding node should be added to the reference mesh (material space). It is not always straightforward to find a mapping between the material space and the world space. If a tetrahedral mesh is used barycentric coordinates can be used to find this mapping as it is done in [WRK*10].
In our meshfree framework we have used the *shape vector* ν introduced in Local Shape Matching (section 3) to approximate this mapping. This operation involves a few a steps:

1. **Update the list of neighbours and weights:**
   Once the new node is added to the mesh in the world space, we calculate the distance of the new node to all other nodes and pick the $n = num\_neighbours$ closest nodes as its neighbours. If i is the new node, its neighbours are shown by $\mathcal{N}(\mathbf{x}_i)$ and rth neighbour of i is shown

by $\mathcal{N}_r(\mathbf{x}_i)$ and its corresponding weights is shown by $\mathcal{W}_r(\mathbf{x}_i)$. The neighbourhood relationship should be mutual to preserve the third law of Newton (action equals reaction); therefore, i should be a neighbour of all nodes in $\mathcal{N}(\mathbf{x}_j)$ as well, where $j \in \mathcal{N}(\mathbf{x}_i)$. Suppose we want to add i to the list of neighbours of node j. In the list of neighbours of j we look for a node with zero weight.[†] Suppose there is such a node and that node is the rth neighbour of j (which means $\mathcal{W}_r(\mathbf{x}_j) = 0$). We place i at the empty position of r: $\mathcal{N}_r(\mathbf{x}_j) \leftarrow i$ and $\mathcal{W}_r(\mathbf{x}_j) \leftarrow w_{ij}$.
If all weights associated with node j are nonzero we add i to the last place in the list of neighbours of j: $\mathcal{N}_n(\mathbf{x}_j) \leftarrow i$ and $\mathcal{W}_n(\mathbf{x}_j) \leftarrow w_{ij}$. However, since we are dropping a neighbour from j we have to break the corresponding relationship. If before updating the last neighbour of j it was pointing to node k, we search for j in $\mathcal{N}(\mathbf{x}_k)$. If j is the sth neighbour of k ($j = \mathcal{N}_s(\mathbf{x}_k)$) we assign its corresponding weight to zero: $\mathcal{N}_s(\mathbf{x}_k) \leftarrow 0$.

2. **Calculate the *shape vector* ν for the newly added node:** The *shape vector* ν_i is the vector from the center of the mass of the neighbourhood of node i to the node i itself. In the world space we find the center of mass of the neighbourhood: $\mathbf{C}_i = (1/n)\sum_{j \in \mathcal{N}(\mathbf{x}_i)} \mathbf{x}_j$ then we find the *shape vector*. The object might be rotated then vector from the center of the mass to the node i is in fact the rotated *shape vector*: $\nu_i^R \leftarrow \mathbf{x}_i - \mathbf{C}_i$. After approximation of the rotation we obtain the *shape vector*: $\nu_i \leftarrow \mathbf{R}^{-1}\nu_i^R$.
3. **Find the position of the new node in the reference mesh:** After calculating the *shape vector* we approximate the position of the new added node in the reference mesh by: $\mathbf{x}_i^0 \leftarrow \mathbf{C}_i^0 + \nu_i$ where $\mathbf{C}_i^0 = (1/n)\sum_{j \in \mathcal{N}(\mathbf{x}_i^0)} \mathbf{x}_j^0$. This approximation produces no artificial deformation in LSHM but it might cause a small non-existent deformation in other methods.
4. **Calculate the shape information for the newly added node and update the shape info for its neighbours:** Now that the we have placed the new node in the material space we have to find the shape information specific to each method. During updating list of neighbours and weights, some neighbourhood relationships might be broken. Therefore, it is necessary to update the shape info for all other nodes that might be affected. In general the new node, its neighbours and neighbours of neighbours could be affected.

5.2. Deleting a node

When the needle is inserted new nodes are added and they are attached to the needle like beads. When the needle is pulled out the new nodes that are added should be deleted as the needle is withdrawn. Thus we create stacks of arrays of neighbours list, neighbour weights and shape info. Figure 2 shows a representation of these stacks. Before a new node is added these arrays are pushed into their corresponding stacks and after a node is deleted they are popped back. This ensures that we don't impose any modification or complexity to the model.

5.3. Needle Boundary Conditions

We use the stick-slip friction model [Kar85] by considering two friction states. Each node can be in the stick or slip mode independently from other nodes. In the stick state the node moves with the needle and the static friction governs between the needle and that node. If the magnitude of the part of the elastic tissue force on node i which is projected on the needle shaft $\|\mathbf{f}_i^p\|$ is greater than the static friction threshold we change the state of that node to slip in which dynamic friction applies.
A node which is bound to the needle should always lie on the needle. To apply the needle boundary condition in the dynamic friction state, we first move the node to the closest point on the needle, then we apply the dynamic friction force. The closest point on the needle is found by projecting the position of the node on the needle. Crouch et. al. [CSWO05] demonstrated that friction forces are velocity dependent. In addition a tissue that is under deformation will generate a greater friction force on the needle and vice versa. In order to model the layer forces we use the *Coulomb friction* model which states that the friction is proportional to the normal force. To include all these states we declare the dynamic friction force on node i, f_i^{df} as follows:

$$\mathbf{f}_i^{df} = k_{vf}\mathbf{v}_i^p + k_{df}\, sign(\mathbf{v}_{needle}^p - \mathbf{v}_i^p)\|\mathbf{f}_i^N\|\hat{\mathbf{bt}} \quad (2)$$

where k_{vf} is the velocity friction coefficient and k_{df} is the dynamic friction coefficient. $\mathbf{v}_i^p$ is the velocity of the node i projected on the needle, $\mathbf{v}_{needle}^p$ is the velocity of the needle at the position of the node projected on the needle shaft. $\|\mathbf{f}_i^N\|$ is the magnitude of the part of the elastic tissue force on node i which is normal to the needle shaft and $\hat{\mathbf{bt}}$ is the normal vector in the direction of needle base to needle tip.
The projection vector is found by a dot product. The projection of the elastic force exerted on node i is equal to:

$$\mathbf{f}_i^p = (\mathbf{f}_i \cdot \hat{\mathbf{bt}})\hat{\mathbf{bt}} \quad (3)$$

and the remaining part will be perpendicular to the needle:

$$\mathbf{f}_i^N = \mathbf{f}_i - \mathbf{f}_i^p \quad (4)$$

5.4. The Complete Model

In the previous section we explained the elements used in our method. In this section we put all the elements together and explain the overall process in detail. We have to mention that

[†] We might have zero weights since we first form the list of neighbours by finding the closest neighbours of each node. Then we perform a one to one comparison to make sure that the neighbourhood relationship is mutual. This ensures that nodes apply equal forces to each other (action equals reaction). If we find a non-mutual neighbourhood relationship, we assign zero to weight of that neighbour

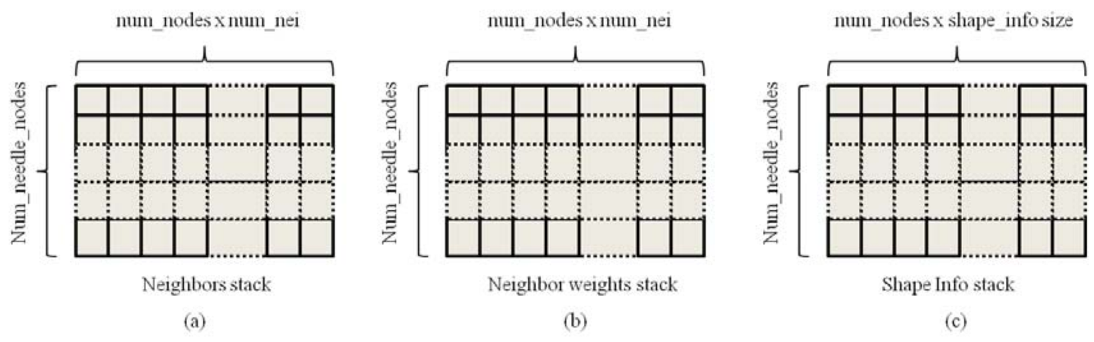


Figure 2: *Stacks are used for node deletion*

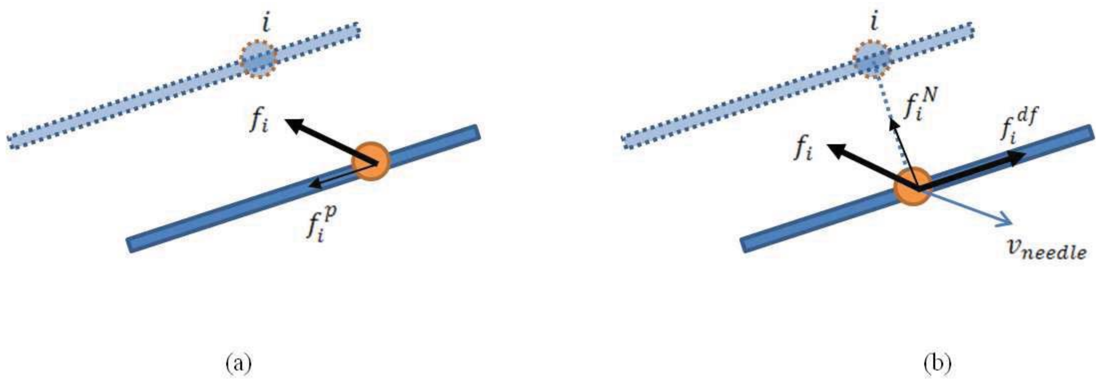


Figure 3: *Needle Tissue Friction Model. (a) Static friction: the node i moves along with the needle. If $\| f_i^p \|$ is greater than the static friction direction we change the state of that node to dynamic (b) Dynamic friction: The node i is projected on the needle and the dynamic friction force f_i^{df} is applied. The dynamic friction force depends on the velocity of the needle and elastic tissue force on the direction of the needle normal vector.*

although we have used meshfree methods to simulate the tissue, we use a triangular mesh to create the enclosing surface of the model. Using this method we can use the fixed functionality of the graphics hardware for rendering. It also helps us to develop an effective algorithm for puncture simulation. We have assumed that the needle and the tissue can be in three different states: *Outside*, *Touching* and *Inside*.

5.4.1. Needle in Outside State

When the needle is *Outside* it has no interaction with the tissue. To detect a collision between the needle tip and the mesh, we project the needle tip to all planes and if it is close enough to a triangle plane, we project the needle tip to see if it lies inside that triangle [Eri04]. If a collision is detected between the needle tip and the tissue surface a node called *Surface node* is added to the tissue nodes and placed at the needle tip. We also delete the triangle that collided with needle and replace it with 3 triangles created by nodes of the deleted triangle and the *Surface node*. From now on the needle tip is in Touching state.

Needle in Touching State

Once the needle is in the *touching* state with the needle, we attach the *Surface node* to the needle like the static friction state. As we explained in the previous section, we have replaced a triangle with 3 new ones. Edges of the new triangles form three angles with the needle itself shown by α_1, α_2 and α_3 in Figure 4. We take the average of these 3 angles if the average is greater than a certain threshold (eg. 95°) we change the state to *Outside*, delete the surface node and replace the 3 newly added triangles with the original one.
If the average of the angles is less than a threshold (eg. 80°) it means that puncture has occurred. Then we change the state to *Inside* and immediately add another node called *Tip node*. We also change the friction state of the *Surface node* to dynamic friction state.

5.4.2. Needle in Inside State

When the needle is inside the tissue boundary conditions mentioned in section 5.3 are applied. At each iteration we update all nodes including the *Surface node* but the *Tip node*

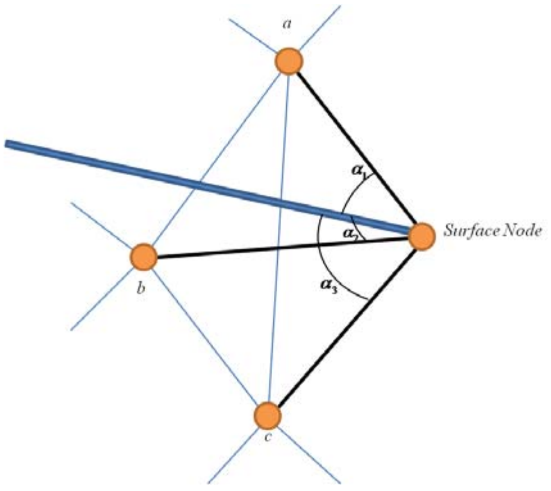


Figure 4: *Needle Puncture and Extraction Condition Checking.*

is a special case. We always measure the distance between the *Tip node* and the node before it (which is always the last added node), if the needle is pulled out this distance becomes negative at some point(the positive direction is in from needle base to its tip) in which case we delete the last added node (section 5.2). If the needle is inserted and the distance between needle tip and the last added node becomes greater than a threshold we add another node at needle tip node (section 5.1). Figure 6 shows different steps of inserting a needle into the tissue in our simulator.

The neighbours list and weights for the *Tip node* is generated as soon as the tissue surface is punctured but it can penetrate deep into the tissue. Therefore, we have to update the neighbours list and neighbours weight for it regularly. One option is to update that neighbours for the *Tip node* as soon as adding or deleting a node. In practice we do this update more often. We update the neighbours list and their weights, if the distance of the needle tip to the last added node changes more than a threshold. The threshold that we considered was $1/10$th of the node adding threshold. Therefore, during insertion the tip node is updated 10 times before a new node is added.

5.4.3. Data Structure

In order to be able to simulate the needle insertion with the framework that was introduced in section 4, we needed to do some modifications to the data structure. Suppose there are N_mesh nodes in the original tissue mesh. Instead of creating the arrays for N_mesh nodes, we create the arrays for $N\_mesh + N\_needle$ nodes. Where N_needle is the maximum number of nodes that can be added as needle nodes. This value should be sufficiently large to accommodate for the state which the needle is fully inserted into the tissue. However, a very large value will result in waste of resources. At each iteration, the simulation is only done for $N\_mesh + N\_needle\_active$ where N_needle_active is the number of nodes added as the needle nodes including the *Tip node* and the *Surface node*. Since boundary conditions are enforced on the CPU side, we have to transfer the calculated elastic force from the device to host and transfer the dynamic friction force from host to device. However, these arrays are transferred back and forth for the needle nodes only.

6. Results

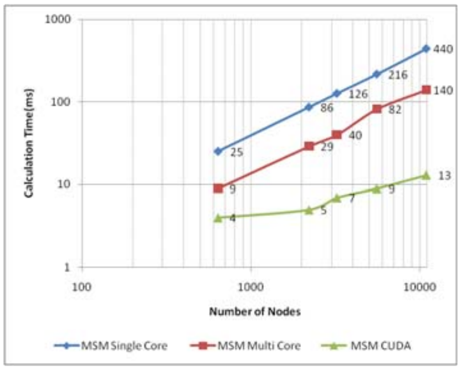
In our framework, Wavefront *.obj* format is used to import triangular meshes. In order to compare the deformation using different methods, we have inserted a needle into a cube mesh with 639 nodes. In Figure 7 screen shots for needle insertion is given for four different methods used in our framework. For all 4 simulations we first inserted the needle by moving it 15cm to the right in the x axis direction; then we rotated the needle along the z axis by 25°.

In general the weighted Mass Spring system is able handle a limited range of deformations and needle movements and results in a visually good response. However for very sudden large movements it results in stability. The stability is very good under LSHM and almost no kind of deformations causes numerical breakdown; however, its lack of volume preservation is evident on those circumstances. The Debunne method gives the most naturally looking result however the range of deformations that it can endure without getting unstable is far less than LSHM. Although PBA is the most computationally expensive method in our framework it does not give the best results. A similar result is obtained in [GBB09] for large elastic deformation. When there is large elastic deformations, using Moving Least Square for approximation results in ill-conditioned deformation gradient and thereby destabilizes the simulation.

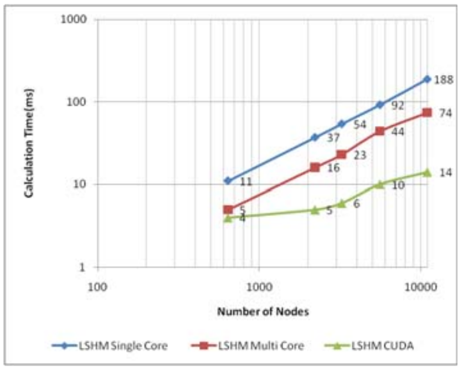
6.1. Performance

In order to compare the performance of different methods together and examine effect of CUDA we ran the simulation on cube meshes with different resolutions. We ran the simulation on a PC with Intel(R) Core 2 Quad 2.4Ghz CPU with 4GB of RAM and graphics processing unit of Nvidia GTX 8800 with 128 CUDA cores and 768MB of memory. We measured the calculation time for single core CPU, 4 core CPU and CUDA.

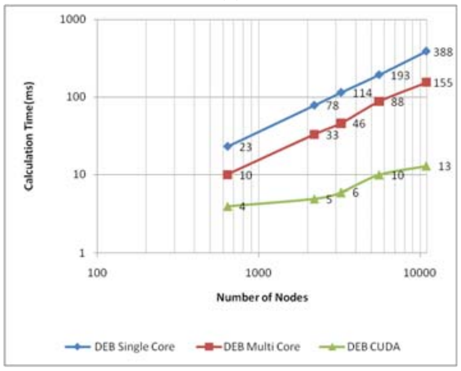
In Figure 5 the calculation time is compared for different implementations. We have used logarithmic scale to better observe the differences. The simulation is repeated for all four methods for different mesh sizes (639, 2207, 3232, 5567 and 10932 nodes) and the average calculation time is given in milliseconds when running the algorithm on a single core of CPU, on 4 cores of CPU and on the GPU.



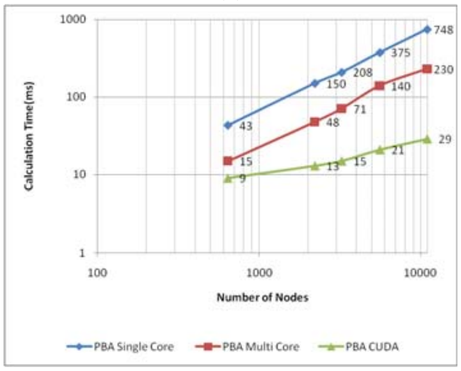
(a)



(b)



(c)



(d)

Figure 5: *Comparison of calculation time for different methods when 16 neighbours were considered. (a) Mass-Spring Method. (b) Local Shame Matching. (c) Discretized Finite Element method (Debunne). (d) Point Based Animation.*

Using all 4 cores of the CPU we were able to accelerate the simulation almost three times. The nodes added during insertion didn't have a significant impact on the calculation time since the needle nodes are far less than the total number of nodes. Although, additional data is sent back and forth between host and the GPU, however that does not cause any noticeable increase since we only transfer the required information for the needle nodes instead of all nodes.

7. Conclusion

A new algorithm was developed using meshfree (non-grid) methods for simulation of needle insertion into soft tissue. It was shown that meshfree methods have better flexibility for needle insertion simulation. By adding nodes, we eliminated the need for remeshing the tissue around the needle tip. We also enforced the boundary conditions by defining static and dynamic friction forces. A dynamic friction model was developed that took both the velocity of the needle and deformation of the tissue into consideration and therefore resulted in a more realistic simulation. Our results overview the benefits and drawbacks of each method. Although LSHM did not provide the most accurate result it was stable for a wide variety of the needle movements. Therefore it is a good choice for a training software in which the user has the freedom to move the needle around. When accuracy is required, the Debunne method is a better choice – but needle movement should be restrained.
We have implemented the needle insertion algorithm in our CUDA based framework and therefore we were able to accelerate the simulation to 20 times when using large meshes. In the future we intend to extend our algorithm to be able to simulate flexible needles. This will be achieved by modeling the curved needle as a set of small rigid needles. Simulation of a flexible needle enables us to simulate a wider range of procedures such as insertion of a ventricular shunt commonly used to treat hydrocephalus. On the other hand, by using the GPU we where able to achieve tremendous acceleration, we are encouraged to implement more advanced meshfree methods. In addition, we are interested in validation of the achieved simulation results.

References

[APM07] ABOLHASSANI N., PATEL R., MOALLEM M.: Needle insertion into soft tissue: A survey. In *Medical Engineering &*

Physics (Aire-la-Ville, Switzerland, Switzerland, 2007), vol. 29, Eurographics Association, pp. 413–431. 1

[CAR*09] CHENTANEZ N., ALTEROVITZ R., RITCHIE D., CHO L., HAUSER K. K., GOLDBERG K., SHEWCHUK J. R., O'BRIEN J. F.: Interactive simulation of surgical needle insertion and steering. In *Proceedings of ACM SIGGRAPH 2009* (Aug 2009). 1, 2, 3

[CSWO05] CROUCH J. R., SCHNEIDER C. M., WAINER J., OKAMURA A. M.: A velocity-dependent model for needle insertion in soft tissue. *Med Image Comput Comput Assist Interv 8*, Pt 2 (2005), 624–32. 2, 5

[CUD10] Nvidia cuda (compute unified device architecture) programming guide 3.0. 1

[DDBC99] DEBUNNE G., DESBRUN M., BARR A. H., CANI M.-P.: Interactive multiresolution animation of deformable models. In *Eurographics Workshop on Computer Animation and Simulation'99, September, 1999* (Milan, Italie, Sept. 1999), Magnenat-Thalmann N., Thalmann D., (Eds.), Computer Science, Springer, pp. 133–144. 3

[DGM*09] DURIEZ C., GUÉBERT C., MARCHAL M., COTIN S., GRISONI L.: Interactive simulation of flexible needle insertions based on constraint models. In *MICCAI '09: Proceedings of the 12th International Conference on Medical Image Computing and Computer-Assisted Intervention* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 291–299. 2

[DS02] DIMAIO S. P., SALCUDEAN S. E.: Needle insertion modelling for the interactive simulation of percutaneous procedures. In *MICCAI (2)* (2002), pp. 253–260. 1, 2

[DS03] DIMAIO S. P., SALCUDEAN S. E.: Needle steering and model-based trajectory planning. In *MICCAI (1)* (2003), pp. 33–40. 1, 2

[DS05] DIMAIO S., SALCUDEAN S.: Interactive simulation of needle insertion models. *Biomedical Engineering, IEEE Transactions on 52*, 7 (july 2005), 1167 –1179. 2, 3

[Eri04] ERICSON C.: *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. 6

[GBB09] GERSZEWSKI D., BHATTACHARYA H., BARGTEIL A. W.: A point-based method for animating elastoplastic solids. In *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), ACM, pp. 133–138. 7

[GSD*05] GOKSEL O., SALCUDEAN S. E., DIMAIO S. P., ROHLING R., MORRIS W. J.: 3d needle-tissue interaction simulation for prostate brachytherapy. In *MICCAI* (2005), pp. 827–834. 2

[Kar85] KARNOPP D.: Computer simulation of stick-slip friction in mechanical dynamic systems. *Journal of Dynamic Systems, Measurement, and Control 107*, 1 (1985), 100–103. 5

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3 (2005), 471–478. 1, 3

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2004), Eurographics Association, pp. 141–151. 3

[NvdS03] NIENHUYS H.-W., VAN DER STAPPEN A. F.: Interactive needle insertions in 3d nonlinear material. In *Medical Image Computing and Computer Assisted Intervention 2003 (MICCAI2003)* (2003), LNCS, Springer Verlag. Submitted for consideration. 3

[RNM*97] ROBERSON P. L., NARAYANA V., MCSHAN D. L., WINFIELD R. J., MCLAUGHLIN P. W.: Source placement error for permanent implant of the prostate. *Medical Physics 24*, 2 (1997), 251–257. 1

[SE10] SHAHINGOHAR A., EAGLESON R.: A framework for gpu accelerated deformable object modeling. In *Proceedings of 2010 International Workshop on GPUs and Scientific Applications* (Sept 2010), University of Vienna. 1, 3

[SO02] SIMONE C., OKAMURA A. M.: Modeling of needle insertion forces for robot-assisted percutaneous therapy. In *ICRA* (2002), pp. 2085–2091. 2

[WRK*10] WICKE M., RITCHIE D., KLINGNER B. M., BURKE S., SHEWCHUK J. R., O'BRIEN J. F.: Dynamic local remeshing for elastoplastic simulation. In *Proceedings of ACM SIGGRAPH 2010* (July 2010), pp. 1–11. 4

[ZMRK07] ZHU Y., MAGEE D. R., RATNALINGAM R., KESSEL D.: A training system for ultrasound-guided needle insertion procedures. In *MICCAI (1)* (2007), pp. 566–574. 2

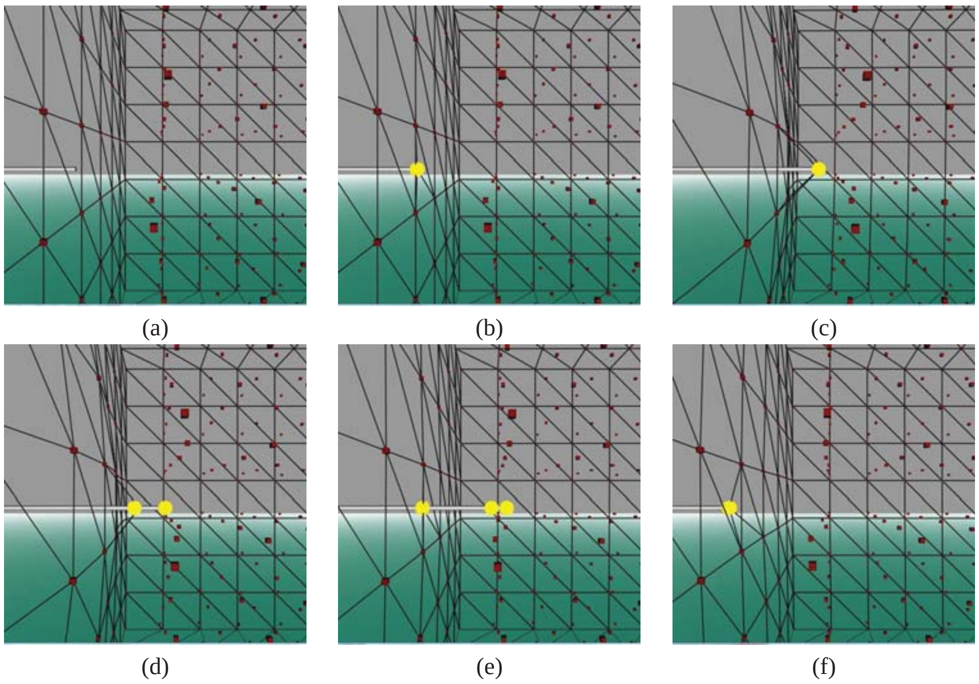


Figure 6: *Needle insertion steps. The cubes are the original tissue nodes and the spheres are added nodes. (a) Needle is* Outside. *(b) Needle has collided with tissue and is now in* Touching *state. The* Surface node *is added. (c) Needle is pushed into the tissue but puncture is not happened yet. (d) Needle has punctured the tissue surface and is* Inside *the tissue now. The* Tip node *is added. (e) Another tissue node is added and is constrained to the needle. (f) Needle is being pulled out and is in* Touching *state again.*

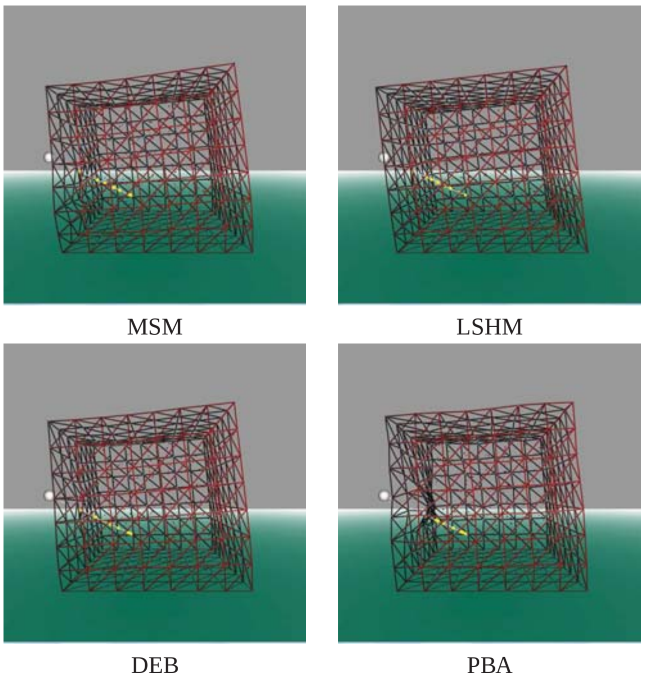


Figure 7: *Comparison of needle insertion simulation with different methods.*