Coupling between Meshless FEM Modeling and Rendering on GPU for Real-time Physically-based Volumetric Deformation

Muhammad Mobeen Movania
Nanyang Technological University, Singapore
mova0002@e.ntu.edu.sg

Feng Lin
Nanyang Technological University, Singapore
asflin@ntu.edu.sg

Kemao Qian
Nanyang Technological University, Singapore
mkmqian@ntu.edu.sg

Wei Ming Chiew
Nanyang Technological University, Singapore
chie0017@e.ntu.edu.sg

Hock Soon Seah
Nanyang Technological University, Singapore
ashsseah@ntu.edu.sg

ABSTRACT

For real-time rendering of physically-based volumetric deformation, a meshless finite element method (FEM) is proposed and implemented on the new-generation Graphics Processing Unit (GPU). A tightly coupled deformation and rendering pipeline is defined for seamless modeling and rendering: First, the meshless FEM model exploits the vertex shader stage and the transform feedback mechanism of the modern GPU; and secondly, the hardware-based projected tetrahedra (HAPT) algorithm is used for the volume rendering on the GPU. A remarkable feature of the new algorithm is that CPU readback is avoided in the entire deformation modeling and rendering pipeline. Convincing experimental results are presented.

Keywords

Volumetric deformation, physically based deformation, finite element method, meshless model, GPU transform feedback, volume rendering

1. INTRODUCTION

Interactive visualization of physically-based deformation has been long pursued as it plays a significant role in portraying complex interactions between deformable graphical objects. In many applications, such subtle movements are necessary, for example, surgical simulation systems in which a surgeon's training experience is directly based on the feedback he/she gets from the training system.

Prior to the advent of the Graphics Processing Unit (GPU), such interactions were only restricted to sophisticated hardware and costly workstations. Thanks to the massive processing capability of modern GPUs, such interactions can now be carried out on a consumer desktop or even a mobile device. However, even with such high processing capability, it is still difficult to simultaneously deform and visualize a volumetric dataset in realtime. Several promising volumetric deformation techniques have been proposed, but they have mostly favored a specific stage of the programmable graphics pipeline. Therefore, these approaches could not utilize the full potential of the hardware efficiently.

With new hardware releases, new and improved features have been introduced into the modern GPU. One such feature is transform feedback in which the GPU feedbacks the result from the geometry shader stage back to the vertex shader stage. While this method was usually used for dynamic tessellation and level-of-detail (LOD) rendering, we have proposed to use this mode for an efficient deformation pipeline. Since this deformation uses the vertex shader stage, we may streamline the fragment shader stage for volume rendering, forming a coupled graphics pipeline.

In Section 2, we report a comprehensive survey on deformation algorithms and GPU acceleration technologies. Then, we describe our new meshless FEM approach and the formulation of the physical model in Section 3. In Section 4, we present the techniques for coupling between the novel deformation pipeline and the GPU-based volume rendering. Experimental results and comparisons of the performance are given in Section 5. And finally, Section 6 concludes this paper.

2. PREVIOUS WORK

Up to now, physically-based deformation can be broadly classified into mesh-based and meshless methods. Mesh-based methods include finite element method (FEM), boundary element method (BEM), and mass spring system. Meshless methods include smoothed point hydrodynamics (SPH), shape matching and Lagrangian methods. We refer the reader for meshless methods to [ST99], [HF00] [BBO03], [NRBD08] and for physically-based deformation approaches in computer graphics to [NMK06].

One of the first mass spring methods for large deformation on the GPU for surgical simulators is attributed to Mosegaard et al. [MHSS04], in which Verlet integration is implemented in the fragment shader. Using the same technique, Georgii et al. [GEW05] implemented a mass spring system for soft bodies. The approach by Mosegaard et al. [MHSS04] requires transfer of positions in each iteration. Georgii et al. [GEW05] thus focused on how to minimize this transfer by exploiting the ATI Superbuffers extension. They described two approaches for implementation: an edge centric approach (ECA) and a point centric approach (PCA). A CUDA-based mass spring model has been proposed recently [ADLETG10].

All of the mass spring models and methods discussed earlier used explicit integration schemes which are only conditionally stable. For unconditional stability, implicit integration could be used as demonstrated for the GPU-based deformation by Tejada et al. [TE05].

The mass spring models are fast but inaccurate. FEM methods have been proposed for more accurate simulation and animation. The model assumes linear elasticity so the deformation model is limited to small displacements. In addition, the small strain assumption produces incorrect results unless the corotational formulation is used [MG04] which isolates the per-element rotation matrix when computing the strain.

With the increasing computational power, non-linear FEM has been explored, in which both material and geometric non-linearities are taken into consideration [ML03], and [ZWP05]. The fast numerical methods for solving FEM systems for deformable bodies are based on the multi-grid scheme. These approaches have been extended in animation [SB09] and medical applications for both the tetrahedral [GW05] [GW06] and hexahedral FEM [DGW10].

In addition to the above approaches, explicit non-linear methods have been proposed using the Lagrangian explicit dynamics [MJLW07] which are especially suitable for real-time simulations. A single stiffness matrix could be reused for the entire mesh. Especially with the introduction of the CUDA architecture, the Lagrangian explicit formulation has been applied for both the tetrahedral FEM [TCO08] as well as the hexahedral FEM [CTA08].

The problem with explicit integration is that it is only conditionally stable, that is, for convergence, the time step value has to be very small. In addition, such integration schemes may not be suitable during complex interactions as in surgery simulations and during topological changes (for example, cutting of tissues). Allard et al. [ACF11] circumvent these cons by proposing an implicitly integrated GPU-based non-linear corotational model for laparoscopic surgery simulator. They use the pre-conditioned Conjugate Gradient (CG) solver for solving the FEM. They solve the stiffness matrices directly on the mesh vertices rather than building the full stiffness assembly. Ill-conditioned elements may be generated in the case of cutting or tearing which may produce numerical instabilities.

3. THE MESHLESS FEM APPROACH

Although there have been significant achievements in deformable models, a few difficulties in the mesh-based models still exist in real-time volumetric deformation, as highlighted in the followings:

- Approximating a volumetric dataset requires a large number of finite tetrahedral elements. Numerical solution of such a large system would require a large stiffness matrix assembly. This makes the model unsuitable for real-time volumetric deformation. In addition, the corotated formulation is needed which further increases the computational burden.
- The solution of the tetrahedral FEM requires an iterative implicit solver for example Newton Raphson (Newton) or Conjugate Gradient (CG) method. These methods converge slowly. Moreover, the implicit integration solvers reduce the overall energy of the system causing the physical simulation to dampen excessively.
- Even though multi-grid schemes are fast, they have to update the deformation parameters across different grid hierarchy levels. This requires considerable computation. Moreover, the number

of grid levels required is subjective to the dataset at hand and there is no rule to follow for accurate results.

On the other hand, we have noticed that the meshless FEM approach has not been applied for volumetric deformation in the literature. Our preliminary study shows that the meshless formulation possesses a few advantages:

- It supports deformations without the need for stiffness warping (the corotated formulation).
- The solution of meshless FEM is based on a semi-implicit integration scheme which not only is stable but also converges faster as compared to the implicit integration required by the tetrahedral FEM solver. In addition, it does not introduce artificial damping.
- It does not require an iterative solver such as conjugate gradient (CG) method which is required for conventional FEM.

Therefore, in this study, we are interested in exploiting the meshless FEM approach for volumetric deformation, coupled with simultaneous GPU-based real-time visualization.

Formulation of the Physical Model

We base our deformation modeling and rendering on the continuum elasticity theory. Key parameters in the physical model are stress, strain and displacement. Strain (ε) is defined as the relative elongation of the element. Assuming an element undergoing a displacement (ΔL) having length (l), the strain may be given as:

$$\varepsilon = \frac{\Delta L}{l}$$

For a three-dimensional problem, the strain (ε) is represented as a symmetric 3×3 tensor. There are two popular choices for the strain tensor in computer graphics, the linear Cauchy strain tensor given as

$$\varepsilon_{\text{Cauchy}} = \frac{1}{2}(\nabla U + [\nabla U]^T) \tag{1}$$

and the non-linear Green strain tensor given as

$$\varepsilon_{\text{Green}} = \frac{1}{2}(\nabla U + [\nabla U]^T + [\nabla U]^T \nabla U) \tag{2}$$

In Eq. (1) and (2), the ∇U is the gradient of the displacement field U. Similar to the strain, in the three-dimensional problem, the stress tensor (σ) is also given as a 3×3 tensor. Assuming that the material under consideration is isotropic and it undergoes small deformations (geometric linearity), the stress and strain may be linearly related (material linearity) using Hooke's law, given as

$$\sigma = D\varepsilon \tag{3}$$

Since stress and strain are symmetric matrices, there are six independent elements in each of them. This reduces the isotropic elasticity matrix (D) to a 6×6 matrix as follows:

$$D = B\begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}$$

$$B = \frac{E}{(1+\nu)(1-2\nu)}$$

where, E is the Young's modulus of the material which controls the material's resistance to stretching and ν is the Poisson's ratio which controls how much a material contracts in the direction transverse to the stretching.

In a finite element simulation, we try to estimate the amount of displacement due to the application of force. There are three forces to consider (see Fig. 1):

- Stress force (σ) which is an internal force,
- Nodal force (q) which is an external force applied to each finite element node, and
- Loading force (t) which is an external force applied to the boundary or surface of the finite element.

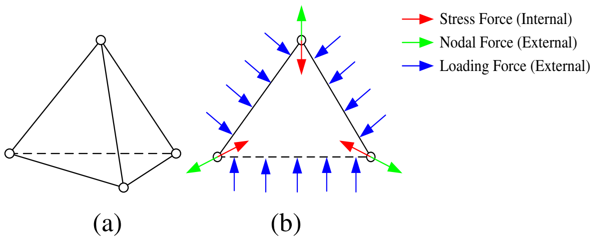


Figure 1. Different forces acting on a finite tetrahedral element (a), with (b) its cross sectional view highlighting the different internal and external forces acting on the finite element

For the finite element to be in static equilibrium, the amount of work done by the external forces must be equal to that of the internal forces, given as

$$\int_{V^e} W_\sigma dV = W_q + \int_{A^e} W_t dA \tag{4}$$

where, W_σ is the internal work done per unit volume by stress σ, W_q is the external work done by the nodal force q on the element's node and W_t is the external work done by the loading force t on the element per unit area. V^e is the volume and A^e is the area of the finite element e. W_σ is given as

$$W_\sigma = (\delta\varepsilon)^T \sigma$$

where, $\delta\varepsilon$ is the strain produced by the stress σ. Similarly, W_q is given as

$$W_q = (\delta u^e)^T q^e$$

where, δu^e is the displacement of the finite element e produced by the force q^e. W_t is given as

$$W_t = (\delta u)^T t$$

Substituting these in Eq. (4), we get

$$\int_{V^e} (\delta\varepsilon)^T \sigma dV = (\delta u^e)^T q^e + \int_{A^e} (\delta u)^T t dA \tag{5}$$

Since δu^e provides the displacement of node e at vertices only, to get the displacement at any point within the finite element, we can interpolate it with the shape function **N.** After applying a differential operator **S** to the shape functions, we get the change in strain ($\delta\varepsilon$). The matrix product (**SN**) can be replaced by B

$$\delta\varepsilon = B\delta u^e$$

Substituting ($\delta\varepsilon$) in Eq. (5), we get

$$\int_{V^e} (B\delta u^e)^T \sigma dV = (\delta u^e)^T q^e + \int_{A^e} (N\delta u^e)^T t dA \tag{6}$$

Simplifying Eq. (6), taking the constant terms out of the equation and solving integral (see Appendix) gives

$$B^T D B u^e V^e = q^e + \int_{A^e} (N^T t) dA \tag{7}$$

The left side is replaced by the element stiffness matrix (K^e=$B^T DBV^e$) and the right by the element surface force (f^e), which gives us the stiffness matrix assembly equation:

$$K^e u^e = q^e + f^e \tag{8}$$

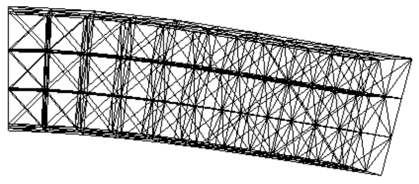
The Meshless FEM

The conventional FEM methods discretize the whole body into a set of finite elements. Calculation of the element stiffness matrix in Eq. (8) requires the volume of the body which is represented as the sum of the finite elements' volume. For instance, the widely used corotated linear FEM [MG04] has to construct the global stiffness matrix for each deformation frame, and the matrix is then solved using an iterative solver such as the conjugate gradients (CG). This makes the implementation inefficient for a large volume.

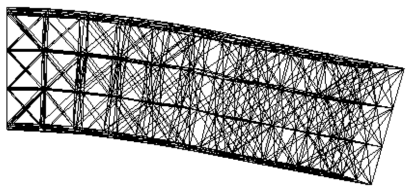
In our meshless FEM, the whole body is sampled at a finite number of points. The typical simulation quantities such as the position (x), velocity (v) and density (ρ) are all stored with the points, and the displacement field is estimated from the volume of the point and its mass distribution. The gradient of the displacement field is then estimated to obtain the Jacobian. Finally, the Jacobian is used to calculate the stresses and strains. These, in turn, allow us to obtain the internal forces. Since the meshless FEM uses the moving least square approximation, it does not require the stiffness matrix assembly, enabling a much better execution performance.

To ascertain that our proposed meshless FEM is able to produce the same deformation as that in the conventional FEM such as the corotated linear FEM, we conducted a computational experiment on a horizontal beam as shown in Fig. 2. The two results show a horizontal beam having Young's modulus of 500,000 psi and the Poisson ratio of 0.33. The dimensions of the two beams are the same. The beam in Fig. 2 (a) contains 450 tetrahedra for the corotated linear FEM whereas its equivalent one in Fig. 2 (b) contains 176 points for the meshless FEM.

While the two computations yield the same deformation under the given load, our meshless FEM has a significantly improved execution performance: 40 msecs per frame by the corotated linear FEM, compared to 1.25 msecs per frame with the meshless FEM. These timings include both the deformation as well as rendering time.



(a)



(b)

Figure 2. Comparison of deformation of a horizontal beam using (a) linear FEM and (b) meshless FEM

In a dynamic simulation, we are to solve the following system:

$$m\ddot{x} = -c\dot{x} + \sum (f_{int} + f_{ext}) \tag{9}$$

The first term on the right is the velocity damping term with c being the damping coefficient. For an infinitesimal element, the mass is approximated using density (ρ). This changes Eq. (9) to

$$\rho\ddot{x} = -c\dot{x} + \sum (f_{int} + f_{ext}) \tag{10}$$

The external forces (f_{ext}) are due to gravity, wind, collision and others. Since our system assumes geometric and material linearity, Eq. (10) becomes a linear PDE that may be solved by discretizing the domain of the input dataset using finite differences over finite elements. This system may be solved using either explicit or implicit integration schemes.

Smoothing Kernel

In the conventional FEM, the volume of the body is estimated from the volume of its constituent finite elements. Calculation of the element stiffness matrix requires the volume of the body which is usually represented as the sum of the finite elements volume. In the case of the meshless FEM, it is approximated from the point's neighborhood. For each point, its mass is distributed into its neighborhood by using a smoothing kernel (w)

$$w(r,h)=\begin{cases}\frac{313}{64\pi h^9}(h^2-r^2)^3 & \text{if } (r<h)\\ 0 & \text{else}\end{cases} \tag{11}$$

where, r is the distance between the current particle and its neighbor, and h is the kernel support radius. The density is approximated by summing the product of the current point's mass with the kernel.

We analyzed the effect of varying the smoothing kernel [MCG03]. These kernels include the normal smoothing kernel (as given in Eq. (11)) the spiky kernel given as

$$w(r,h)=\begin{cases}\frac{15}{\pi h^6}(h-\|r\|)^3 & 0\le\|r\|\le h\\ 0 & \|r\|>h\end{cases} \tag{12}$$

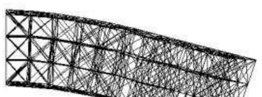
and the blobby kernel given as

$$w(r,h)=\begin{cases}\frac{15}{2\pi h^3}\left(-\frac{r^3}{2h^3}+\frac{r^2}{h^2}+\frac{h}{2r}-1\right) & 0\le\|r\|\le h\\ 0 & \|r\|>h\end{cases} \tag{13}$$

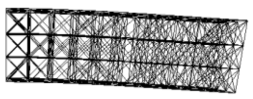
The deformation results on a horizontal beam containing 176 points are shown in Fig. 3. Note that for all the beams shown in Fig. 3, the Young's modulus of 500,000 psi and the Poisson ratio of 0.33 are used. As can be seen, changing the smoothing kernel alters the stiffness of the soft body. This is because each kernel has a distinct support radius which influences the neighboring points. Moreover, each of these kernels has a different falloff (or, different derivative) which gives a different deformation result even though the rest of the simulation parameters are the same.

Propagation of Deformation

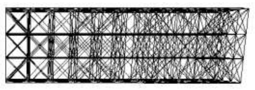
For propagating the stress, strain and body forces in the meshless FEM, we compute the gradient of the displacement field (U) by a moving least square interpolation between the displacement values at the current point (u_i) and its neighbor (u_j) as given by



(a)



(b)



(c)

Figure 3. Effects of different smoothing kernels on the deformation: (a) the normal smoothing kernel (Eq. 11), (b) the spiky kernel (Eq. 12) , and (c) the blobby kernel (Eq. 13)

$$e=\sum_i (u_j-u_i)^2 w_{ij} \tag{14}$$

where, w_{ij} is the kernel function given in Eq. (11).

The displacement values (u_j) are given using the spatial derivatives approximated at point (i) as

$$u_j = u_i + \nabla u.(x_j - x_i)$$

We want to minimize the error (e) in Eq. (14) so we differentiate e with respect to X, Y and Z and set the derivatives equal to zero. This gives us three equations for three unknowns

$$\nabla u|_x = A^{-1}\left(\sum_i (u_j-u_i)(x_j-x_i)w_{ij}\right)$$

where, $A=\Sigma_i(x_j-x_i)(x_j-x_i)^T w_{ij}$ is the moment matrix that can be pre-calculated since it is independent of the current position and displacement. Once ∇u is obtained, the strain (ε) is obtained using Eq. (2). Using this strain, the stress (σ) may be obtained using Eq. (3). The internal forces (f_{int}) in Eq. (10) are calculated as the divergence of the strain energy which is a function of the particle's volume

$$U_i = v_i \frac{1}{2}(\varepsilon_i.\sigma_i)$$

where, v_i is the volume of the particle. The force acting on neighboring particle (j) due to particle (i) is given as

$$f_j = -\nabla U_i = -v_i \frac{1}{2}\nabla\varepsilon_i.\sigma_i$$

To sum up, the internal forces acting on the particles i and j may be given as

$$\begin{aligned} f_i &= -2v_i J\sigma_i d_i \\ f_j &= -2v_j J\sigma_j d_j \end{aligned} \tag{15}$$

where, $d_i=M^{-1}(\Sigma_j(x_j-x_i)w_{ij})$ and $d_j=M^{-1}(x_j-x_i)w_{ij}$, J is the Jacobian, v is the volume of the point and σ is the stress at the given point.

Note that in the case of point masses, the volume may be calculated from the mass density in the point's neighborhood. The mass (m_i) of the point (i) is calculated using

$$m_i = s r_i^3 \rho$$

where, r_i is the average distance between the mass point and its neighbors, ρ is the material density and s is a scaling constant which is calculated as

$$\beta_j = \frac{1}{\sum_j r_j^3 w_j}$$

$$s = \frac{\sum_i \beta_i}{n}$$

where, n is the total number of points. Once the mass is obtained, the per-point density is then obtained by summing the product of the mass of its neighbor (m_j) with the kernel evaluated at the neighbor j (w_j), as follow:

$$\rho_i = \sum_j m_j w_j$$

This density can then be used to obtain the volume of the point which is given as

$$vol_i = \frac{m_i}{\rho_i}$$

During the force evaluation, the internal forces are scattered in the neighborhood of the point using Eq. (15). Since scattering cannot be implemented in a GPU shader program, therefore, for parallel processing, we convert the scatter operation into a gather operation by reformulation [Buc05]. First, the sum of force matrices (F_e and F_v) is obtained

$$sumF_i = (F_e + F_v)$$

Instead of calculating the force on the point i as given in Eq. (15), we multiply the matrix ($sumF_i$) with (d_i)

$$F_i = F_i + sumF_i d_i$$

where d_i is obtained as in Eq. (15). The internal force due to the neighbor points is then given as

$$F_i = F_i - sumF_j d_j$$

where, F_i is the net internal force, j loops for each neighbor of the current point i and d_j is obtained as in Eq. (15). This allows us to run the program in parallel on all points simultaneously.

4. COUPLING BETWEEN VOLUMETRIC DEFORMATION AND RENDERING

Prior rendering algorithms have resorted to the GPGPU-based techniques for evaluating the position and/or velocity integration on the fragment shader [GEW05], [GW06], [GW08], [TE05], [VR08]. This involves rendering a screen sized quad with the appropriate textures setup and then the fragment shader is invoked to solve the integration for each fragment. The output from the fragment shader is written to another texture. On the contrary, we adopt a different approach (see Fig. 4).

We implement the meshless deformation by using the transform feedback mechanism of the modern GPU. The original unstructured mesh vertices are used directly in our implementation. Our deformable pipeline is implemented in the vertex shader stage and it outputs to the buffer object registered to the transform feedback. We use a pair of buffer objects for both the positions and velocities to avoid the simultaneous read/write race condition [ML12a]. So when we are reading from a pair of position and velocity buffers, we write to another pair. In each iteration, the pair is swapped.

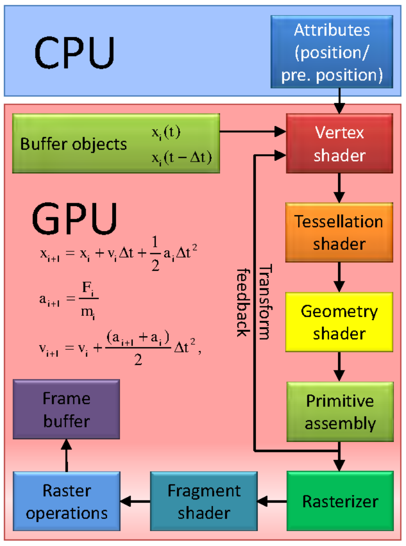


Figure 4. Proposed deformation pipeline using transform feedback

Attribute Setup for Transform Feedback

All the per-point attributes such as the current position (x_i), previous position (x^0_i), and velocity (v_i) are passed to the transform feedback vertex shader as per-vertex attributes.

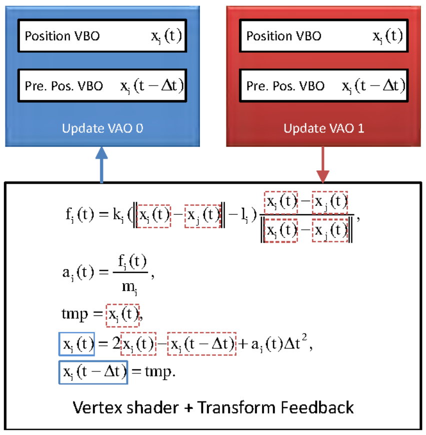
The inverse mass matrices of individual nodes are stored in a texture ($texM_{inv}$) and the rest distances are stored in an attribute (r_{dist}); The point neighbor distance (r_i), support radius (h_i), the point mass (m_i) and volume (vol_i) are pre-computed and stored into a set of textures: the neighborCountsTexture and the neighborListTexture.

A pair of position and velocity buffer objects is bound as a transfer feedback buffer. This enables the vertex shader to output the results to a buffer object directly without CPU readback.

Dataflow

Referring to Fig. 5, for each rendering cycle, we swap between the two buffers to alternate the read/write pathways. Before the transform feedback can proceed, we need to bind the update array objects. Once the update array object is bound, we bind the appropriate buffer objects (for reading the current positions and velocities) to the transform feedback.

The draw point call is issued to allow the writing of vertices to the buffer object. The transform feedback is then disabled. Following the transform feedback, the rasterizer is enabled and then the points are drawn. This time, the render array objects are bound. This renders the deformed points on screen.



Position VBO $x_i(t)$
Pre. Pos. VBO $x_i(t-\Delta t)$
Update VAO 0
Position VBO $x_i(t)$
Pre. Pos. VBO $x_i(t-\Delta t)$
Update VAO 1
$f_i(t) = k_i(\|x_i(t) - x_j(t)\| - l_i)\frac{x_i(t) - x_j(t)}{\|x_i(t) - x_j(t)\|}$,
$a_i(t) = \frac{f_i(t)}{m_i}$,
$tmp = x_i(t)$,
$x_i(t) = 2x_i(t) - x_i(t-\Delta t) + a_i(t)\Delta t^2$,
$x_i(t-\Delta t) = tmp$.
Vertex shader + Transform Feedback

Figure 5. The vertex array object and vertex buffer object setup for transform feedback: the blue/solid rectangles show the attributes written to and the red/dotted rectangles show the attributes being read simultaneously from another vertex array object

Evaluation of Forces

The forces are evaluated per-vertex. Rather than storing the isotropic elasticity matrix (D) as a 6×6 matrix, we store it into a single vec3 attribute containing the three non-zero entries. The inverse mass matrix is pre-calculated at initialization on the CPU and then transferred to the GPU as a per-vertex attribute.

At initialization, the nearest K neighbors of the point are found using a neighborhood search. For the examples shown in this paper, the K is set as 10. This value was arrived at after some experiments. Large size of K generally makes the object stiffer. For fast and efficient search, we use a Kd-tree extracted from the given point set. The found points become the neighbors of the current point. Then, the mass, volume, weights and moment matrices are calculated for each point.

We first calculate the external forces such as the gravity force and the velocity damping force. We then calculate the Jacobians, the stresses and the internal forces using the neighbor node attributes.

Numerical Integration

Following the calculation of the forces, we perform the leap frog integration. The leap frog integration works by evaluating the velocities at 1/2 time step offset from the position. Mathematically, the leap frog integration is given as

$$\begin{aligned} x_{i+1} &= x_i + v_i\Delta t + a_i\frac{\Delta t^2}{2} \\ v_{i+1} &= v_i + \frac{a_i + a_{i+1}}{2}\Delta t \end{aligned} \tag{16}$$

The advantage that we obtain from this integration scheme is that it conserves the overall energy of the system. The expressions given in Eq. (16) may be converted directly into shader statements. The current position (x_i) and velocity (v_i) are passed in as per-vertex attributes whereas the next position (x_{i+1}) and velocity (v_{i+1}) are written using the transform feedback mechanism.

Cell Projection of Transformed Volume

In view of the various aspects of modeling and rendering requirements for fast rendering of transformed points, we use the hardware assisted projected tetrahedra (HAPT) algorithm [MMF10]. The deformation pipeline outputs a pair of buffer objects. These are used directly as positions for cell projection. Thus, we do not need to transfer the deformation results to CPU.

The nonrigid transformation is incorporated in the HAPT pipeline by streaming our deformed points directly. The HAPT algorithm stores positions in texture objects. In our implementation, we reuse the buffer objects output from our deformation pipeline directly. This involves no CPU readback, and thus, the data can be visualized directly.

User Interaction and Collision Detection with Deformed Volume

In a simulation system, it is often necessary to interact with the volume in realtime. In our proposed pipeline, we can interact with the volume in two ways: by modifying the vertex positions directly and by modifying the tetrahedra. In either case, we map the GPU memory to obtain the pointer to the GPU memory, index to the appropriate value and modify the value directly [ML12b].

Collision detection and response can be handled directly in the integration vertex shader by applying constraints to the calculated positions. For example, if we have a mass point x_i, a sphere with a radius r and center C, the collision constraint may be given as

$$x_{i+1} = \begin{cases} C + \dfrac{(x_i - C).r}{|x_i - C|} & \text{if } |x_i - C| < r \\ x_i & \text{else} \end{cases}$$

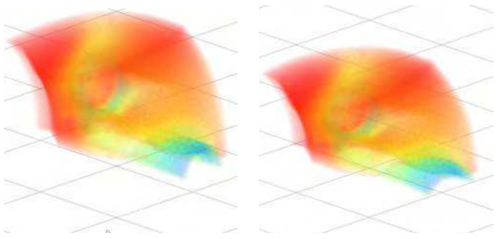
Likewise, other constraints may be integrated directly in the proposed pipeline using the vertex or geometry shader.

5. EXPERIMENTAL RESULTS AND PERFORMANCE ASSESSMENT

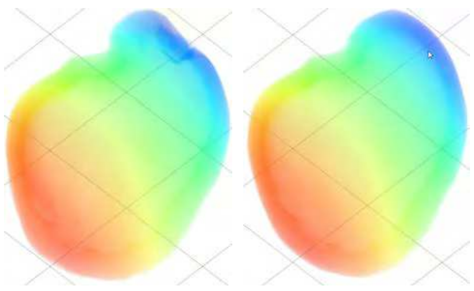
The coupled deformation and rendering pipeline has been implemented on a Dell Precision T7500 desktop with an Intel Xeon E5507 @ 2.27 MHz CPU. The machine is equipped with an NVIDIA Quadro FX 5800 graphics card. The viewport size for the renderings is 1024×1024 pixels.

The output results with deformation and rendering are shown in Fig. 6. We applied the meshless FEM to two volumetric datasets, the spx dataset (containing 2896 points and 12936 tetrahedra) and the liver dataset (1204 points and 3912 tetrahedra).

For all our experiments, the normal smoothing kernel Eq. (11) is used. We allowed the spx dataset to fall under gravity while the liver dataset was manipulated by the user. The time step value (dt) used for this experiment is 1/60. Thanks to the convenience of our proposed deformation pipeline, we can integrate our deformation pipeline directly into the HAPT algorithm.



(a)



(b)

Figure 6. Two frames of deformation of (a) the spx dataset falling due to gravity on the floor and (b) the liver dataset manipulated by the user

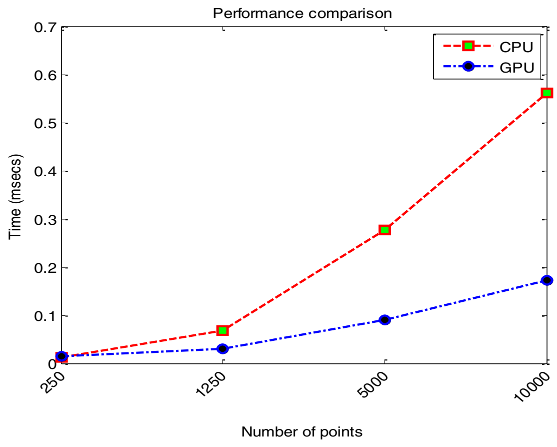


Figure 7. Performance of meshless FEM on GPU and CPU

In the second experiment, to assess the performance of the proposed method, we compare our GPU-based meshless FEM with an already optimized CPU implementation that utilized all available cores of our CPU platform. For this experiment, the bar model containing varying number of points (from 250 to 10000) was used with the time step value (dt) of 1/60. These timings only include the time for deformation using a single iteration and they do not include the time for rendering. These results are given in Fig. 7. The results clearly show that the performance of the meshless FEM scales up well with the large datasets and we gain an acceleration of up to 3.3 times compared to an optimized CPU implementation. From this graph, it is clear that the runtime for CPU would be exponentially increased for larger datasets and the performance gap between the CPU and the GPU would be widened further.
It is the first time a meshless FEM model is applied to unstructured volumetric datasets. Our method uses the leap-frog integration which is semi-implicit whereas the existing mesh based FEM approaches use implicit integration schemes. The amount of time required for convergence in the case of implicit integration schemes is much more as compared to semi-implicit integration.

Moreover, the simulation system built using such formulation requires the stiffness matrix assembly which is then solved using an iterative solver such as Newton Raphson method or Conjugate Gradients (CG) method. Such stiffness assemblies are not required in meshless FEM. Therefore, ours converges much faster.

Nevertheless, for completeness, in the third experiment, we compared the performance of meshless FEM with an implicit tetrahedral FEM [ACF11]. These results are presented in Table 1.

Dataset	Tetrahedra	Frame rate (frames per second)	
		Implicit FEM	Meshless FEM
liver	3912	118.50-78.70	249.50-331.01
spx	12936	76.70-81.90	124.80-128.23
raptor	19409	40.30-43.80	71.22-71.71

Table 1. Comparison of meshless FEM against implicit tetrahedral FEM solver [ACF11]

As expected, the performance of meshless FEM is better as compared with the implicit tetrahedral FEM. Our meshless FEM is based on a semi-implicit integration scheme which does not require an iterative solver as is required for the implicit tetrahedral FEM.

6. DISCUSSION AND CONCLUSION

We have applied meshless FEM to nonrigid volumetric deformation. Using the proposed approach, interactive visualization of deformation on large volumetric dataset is made possible. We are confident of the results obtained from our experiments and would like to expand the model to address specific applications such as biomedical modeling [MCZLQS09], [MLQS09]; simulation [LSL96], [LSL97], [LSWM07]; fast ubiquitous visualization [YLS00], [ML12c]; and confocal imaging [TOTMLQS11].

We reiterate our main contributions. Firstly, we have applied meshless FEM for volumetric deformation. Secondly, we have integrated the meshless FEM into a novel deformation pipeline exploiting the transform feedback mechanism. And finally, we have integrated our novel deformation pipeline into the HAPT algorithm. There are some considerations on the meshless FEM. The deformation result is directly dependent on the number of mesh points used. More points generally give better approximation and vice versa. For better performance, we can think of two strategies. Firstly, we can use the image load-store extension in OpenGL 4.2 which allows shader programs to have access to arbitrary GPU memory. Since the hardware used in this study did not support OpenGL 4.2, this approach could not be verified. Secondly, we may use a CUDA kernel to do scattered writes, alongside a GLSL shader. This will possibly be a future research direction.

7. ACKNOWLEDGMENTS

This work is partially supported by a research grant (M408020000) from Nanyang Technological University and another (M4080634.B40) from Institute for Media Innovation, NTU. We would also like to thank the anonymous reviewers for their helpful suggestions and feedback.

8. REFERENCES

[ACF11] Allard J., Courtecuisse H., Faure F.: Implicit fem and fluid coupling on GPU for interactive multiphysics simulation. ACM SIGGRAPH 2011.

[ADLETG10] Andres D. L. C., Eliuk S., Trefftz G. H.: Simulating soft tissues using a GPU approach of the mass-spring model. Virtual Reality Conference (VR'2010), pp. 261–262, 2010.

[BBO03] Babuška I., Banerjee I., Osborn J. E., Survey of meshless and generalized finite element methods: A unified approach, Acta Numerica, 12, pp:1-125, 2003.

[Buc05] Buck I., Taking the plunge into GPU computing. Chapter 32 in GPU Gems 2, Matt Pharr and Randima Fernando (editors), 2005.

[CTA08] Comas O., Taylor Z., Allard J., Ourselin S., Cotin S., Passenger J., Efficient nonlinear fem for soft tissue modelling and its GPU implementation within the open source framework SOFA. International Symposium on Computational Models for Biomedical Simulation'08, pp. 28–39, 2008.

[DGW10] Dick C., Georgii J., Westermann R., A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. In Simulation Modelling Practice and Theory'10, 19, No. 2, pp. 801–816, 2010.

[GEW05] Georgii J., Echtler F., Westermann R.: Interactive simulation of deformable bodies on GPUs. In Simulation and Visualization'05, 2005.

[GW05] Georgii J., Westermann R.: A multi-grid framework for real-time simulation of deformable volumes. Workshop on Virtual Reality Interactions and Physical Simulations'05, 2005.

[GW06] Georgii J., Westermann R., A generic and scalable pipeline for GPU tetrahedral grid rendering. IEEE Transaction on Visualization and Computer Graphics'06, 2006.

[HF00] Huerta A., Fernandez M. S., Enrichment and coupling of the finite element and meshless methods, International Journal for Numerical Methods in Engineering, 48, No. 11, pp:1615–1636, August 2000.

[LSL97] Lin F., Seah H. S., Lee Y. T., Structure Modeling and Context-Free-Grammar: Exploring a new approach for surface construction. Computers and Graphics (1997), 21, No. 6, pp. 777–785, 1997.

[LSL96] Lin F., Seah H. S., Lee Y. T., Deformable volumetric model and isosurface: Exploring a new approach for surface construction. Computers and Graphics (1996), 20, No. 1, pp. 33–40, 1996.

[LSWM07] Lin F., Seah H. S., Wu Z., Ma D., Voxelisation and fabrication of freeform models. Virtual and Physical Prototyping'07, 2 No. 2, pp. 65–73, 2007.

[MCG03] Mueller M., Charypar D., Gross M.: Particle based fluid simulation for interactive applications. In Proceedings of the ACM SIGGRAPH Symposium on Computer Animation (SCA'03), pp. 154–159, 2003.

[MCZLQS09] Movania M. M., Cheong L. S., Zhao F., Lin F., Qian K., Seah H. S., "GPU-based Surface Oriented Interslice Directional Interpolation for Volume Visualization," The 2nd International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL'09), Bratislava, Slovak Republic, November 24-27, 2009.

[MG04] Mueller M., Gross M., Interactive virtual materials. Proceedings of Graphics Interface (GI'04), pp. 239–246, 2004.

[MHSS04] Mosegaard J., Herborg P., Sangild Sorensen T.: A GPU accelerated spring mass system for surgical simulation. Health Technology and Informatics'04, pp. 342–348, 2004.

[MJLW07] Miller K., Joldes G., Lance D., Wittek A., Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. Communications in Numerical Methods in Engineering'07, 23, No. 1, pp. 801–816, 2007.

[ML03] Mendoza C., Laugier C., Simulating soft tissue cutting using finite element models. IEEE International Conference on Robotics and Automation'03, pp. 1109–1114, 2003.

[ML12a] Movania M. M., Lin F., A novel GPU-based deformation pipeline. ISRN Computer Graphics, vol. 2012 (2012), p. 8.

[ML12b] Movania M. M., Lin F., Real-time physically-based deformation using transform feedback. Chapter 17 in The OpenGL Insights, Christophe, Riccio and Patrick, Cozzi (Ed.), AK Peters/CRC Press, 2012, pp. 233-248.

[ML12c] Movania M. M., Lin F., High-Performance Volume Rendering on the Ubiquitous WebGL Platform, 14th IEEE International Conference on High Performance Computing and Communications (HPCC'12), Liverpool, UK, 25-27 June, 2012.

[MLQS09] Movania M. M., Lin F., Qian K., Seah H. S., "Automated Local Adaptive Thresholding for Real-time Feature Detection and Rendering of 3D Endomicroscopic Images on GPU," The 2009 International Conference on Computer Graphics and Virtual Reality (CGVR'09), Las Vegas, US, July 13-16, 2009.

[MMF10] Maximo A., Marroquim R., Farias R., Hardware assisted projected tetrahedra. Computer Graphics Forum, 29, No. 3, pp: 903–912, 2010.

[NMK06] Nealen A., Mueller M., Keiser R., Boxermann E., Carlson M., Physically based deformable models in computer graphics. In STAR Report Eurographics 2006 vol. 25, pp. 809–836, 2006.

[NRBD08] Nguyena V. P., Rabczukb T., Bordasc S., Duflotd M., Meshless methods: A review and computer implementation aspects, Mathematics and Computers in Simulation, 79, No. 3, pp:763–813, December 2008.

[SB09] Sampath R., Biros G., A parallel geometric multigrid method for finite elements on octree meshes. In review, available online accessed in 2012 http://www.cc.gatech.edu/grads/r/rahulss/, 2009.

[ST99] Shapiro V., Tsukanov I., Meshfree Simulation of Deforming Domains," Computer Aided Design, 31, No. 7, pp: 459–471, 1999.

[TCO08] Taylor Z., Cheng M., Ourselin S., High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. IEEE Trans. Medical Imaging, vol. 27, pp. 650–663, 2008.

[TE05] Tejada E., Ertl T., Large steps in GPU-based deformable bodies simulations. In Simulation Modeling Practice and Theory, 13 No. 8, Elsevier, pp. 703–715, 2005.

[TOTMLQS11] Thong P. S. P, Olivo M., Tandjung S. S., Movania M. M., Lin F., Qian K., Seah H. S., Soo K. C., "Review of Confocal Fluorescence Endomicroscopy for Cancer Detection," IEEE Photonics Society (IPS) Journal of Selected Topics in Quantum Electronics, Vol. PP, Issue 99, 2011. DOI: 10.1109/JSTQE.2011.2177447.

[VR08] Vassilev T., Rousev R., Algorithm and data structures for implementing a mass-spring deformable model on GPU. Research and Laboratory University Ruse 2008 (2008), pp. 102–109, 2008.

[YLS00] Yang Y. T., Lin F. and Seah H. S., "Fast Volume Rendering," Chapter 10 in Volume Graphics, Springer, January 2000.

[ZWP05] Zhong H., Wachowiak M., Peters T.: A real time finite element based tissue simulation method incorporating nonlinear elastic behavior. Computer Methods Biomechan. Biomed. Eng., 6, No. 5, pp. 177–189, 2005.

9. APPENDIX

$$\int_{V^e} (B\delta u^e)^T \sigma dV = (\delta u^e)^T q^e + \int_{A^e} (N\delta u^e)^T t dA$$

$$(\delta u^e)^T \int_{V^e} B^T \sigma dV = (\delta u^e)^T (q^e + \int_{A^e} N^T t dA)$$

$$\int_{V^e} B^T \sigma dV = q^e + \int_{A^e} N^T t dA$$

Substitution using Eq. (3)

$$\int_{V^e} B^T D \varepsilon dV = q^e + \int_{A^e} N^T t dA$$

$$\int_{V^e} B^T D B u^e dV = q^e + \int_{A^e} N^T t dA$$

$$B^T D B u^e \int_{V^e} dV = q^e + \int_{A^e} N^T t dA$$

$$B^T D B u^e V^e = q^e + \int_{A^e} N^T t dA$$