IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 26 January 2023, accepted 10 February 2023, date of publication 14 February 2023, date of current version 23 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3245130

RESEARCH ARTICLE

Real-Time Surface-Based Volume Constraints on Mass-Spring Model in Unity3D

HONGLY VA[1], MIN-HYUNG CHOI[2], AND MIN HONG[3]
[1]Department of Software Convergence, Soonchunhyang University, Asan-si 31538, South Korea
[2]Department of Computer Science and Engineering, University of Colorado Denver, Denver, CO 80217, USA
[3]Department of Computer Software Engineering, Soonchunhyang University, Asan-si 31538, South Korea

Corresponding author: Min Hong (mhong@sch.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2022R1I1A3069371, in part by the Brain Korea 21 Fostering Outstanding Universities for Research (BK21 FOUR) under Grant 5199990914048, and in part by the Soonchunhyang University Research Fund.

ABSTRACT This paper describes a parallel method to simulate real-time 3D deformable objects using volume preservation constraints on a mass-spring model (MSM) to achieve plausible results in real-time performance. Instead of considering a volumetric mesh which is mostly used to simulate deformable objects, we purely take the surface of the 3D object into account to reduce the time complexity and obtain a high-quality deformable. In the conventional MSM, we can simply control the shape of the deformable object through the stiffness and damping coefficients which is beneficial for our volume constraint. We use the divergence theorem and implicit constraint enforcement scheme to maintain the volume of the object and deform it freely. The surface-based volume constraint is applied to correct the force of the spring network. The proposed algorithm was designed on compute shader in Unity3D and runs outside the normal rendering pipeline in the graphics processing unit (GPU) in order to utilize the massive parallel process to accelerate the performance of the simulation. The performance of the simulation can be accelerated by using the parallel processing method on the GPU with an average speedup factor of 4.27 using the conventional mass-spring method, and an average speedup factor of 2.54 for the volume preservation constraint method. We present several scenes which demonstrate the volume-preserving deformations using the conventional mass-spring method and volume-preservation constraint method and the volume loss of deformable objects for all 3D models is compared. The volume preservation method obtains a volume loss significantly lower than the conventional MSM for all experimental tests.

INDEX TERMS Dynamic simulation, GPU parallel computing, implicit constraint enforcement, mass-spring system, Unity3D, volume preservation constrain.

I. INTRODUCTION

Due to the growing demand for computer animation and graphical modeling, real-time physically-based simulation in computer graphics has become a significant issue. Physically-based simulation became a quickly growing area that is used by many interesting fields including entertainment, education, medical science, architecture, astronomy, biology, and more [1]. Although physically-based simulation requires computing power to calculate and model the time-varying behavior of a dynamical system in real-time. Therefore, it is challenging to simulate and model a high-resolution model in real-time that is measured by frames per second (fps) [2]. For example, in the film industry, 24 fps are operated, and a game is considerable at 30 fps. However, immersive simulation using virtual reality (VR) technology required higher frames per second for keeping the immersive experience and avoiding simulation sickness [3], [4].

Many researchers have developed many approaches over the years to realistically model and simulate plausible soft bodies to be used in interactive applications. Among the previous approaches of physical simulation, there are many

The associate editor coordinating the review of this manuscript and approving it for publication was Gang Mei.

approaches such as the Finite Element Method (FEM), the Finite Differences Method (FDM) and the Finite Volume Method (FVM), and the Boundary Element Method (BEM) trying to replicate the real-world processes and mechanics into the graphics simulation as accurate as possible [5], [6], [7], [8]. On the other hand, performance and controllability are the main reasons to be considered the method for physical simulation with plausible realism. In contrast, many of the established method using FEM is commonly applied to deformable object simulations due to their accuracy and robustness. However, the computational cost is also expensive which required optimization algorithms. Therefore, most deformable object simulations can easily be implemented using the mass-spring model (MSM), in which a 3D mesh is discretized into the system of spring where each spring is made by a pair of nodes or vertex in the geometry mesh [9]. In the conventional MSM, each spring is attracted to the other by spring force to resist the internal and external forces in the system. Therefore, the behavior of the simulated object depends on the network of the spring system. Unlike the most stable and accurate method, FEM easily controls the physical behavior by Young's modulus and Poison ratio, while the parameter in MSM is extremely difficult to control since the numerical errors and instability are frequently occurring in the explicit scheme [10]. The volumetric model such as the tetrahedral model is used instead of a triangular surface since the interior structures well maintain the shape and interact as real-world objects [11]. However, the insufficient volumetric effects are caused by the missing volume preservation on the MSM. As the result, the volume of deformable objects suffers from a heavy loss that can be distracting artifacts when performing large and complex deformation. Due to its simplicity, robustness, and speed, Position-based Dynamic (PBD) has been introduced and applied in various applications, especially in the entertainment industry [12]. With any type of geometry constraint solving in explicit time integration, the PBD approach is used for soft bodies simulation including cloth simulation, and deformable object simulation. Distance and bending constraints are commonly used to preserve the overall volume of the deformable object with the algebraic constraint functions. Nonetheless, many researchers also introduced other approaches for deformable object simulations to obtain plausible and stable simulations in real-time. The meshless approach such as shape matching can simulate visually plausible and plastic deformation which considers only the surface structure of the deformable object [13]. Similarly, Pressure constraint based on the ideal gas law is applied to simulate the suction cups to reproduce the physics of suction cups of varying shapes [14].

Most physics simulations required high performance to simulate multi-resolution of the deformable object in real-time which can be beneficial for an immersive experience such as VR, augmented reality (AR), and mixed reality (MR) [15], [16], [17]. Especially in VR practical applications, virtual simulation recreates an actual environment and allows users to interact with the object. Therefore, virtual simulation in VR impacts many trainings and education [18]. VR technologies are well suited to medical education including treatment, rehabilitation, and especially virtual surgery [19], [20], [21], [22], [23], [24]. The study demonstrates that expensive on-site device can be replaced by plausible VR images and achieves promising results. Furthermore, virtual surgery VR-based is the most challenging process since it required practical skills from the experts and a more realistic virtual environment to make the process more plausible and interactive.

Due to the limitation of hardware architecture, most approaches implement the serial algorithm on the central processing unit (CPU). Most games, graphic applications, and some image processing applications strongly rely on serial processing and mostly suffer from the time-consuming problem that needs to downscaling the dimension of the object. Therefore, parallel computation is used instead of serial computation to speed up the algorithm since the instructions are executed simultaneously is faster than sequential processing [25]. In terms of parallelization, there are two types of parallelism in processing execution [26]. Data parallelism refers to the data distribution that is concurrently executed across multiple processing cores for the same task regarding the multiprocessor system. Additionally, synchronous computation is performed for the general structured data such as arrays and matrices also known as single instruction multiple data (SIMD). Therefore, the parallel job on arrays can be done by a single execution thread that executes the same code for the different threads in order to control the operation. Differently, task parallelism refers to the task distribution concurrently perform by the specific thread in different computing processors. Furthermore, the threads can be executed on the same or different code on the set of input data, and each thread required communication after one is finished. It's well suited for the system that load balancing depends on the hardware and scheduling algorithm [27].

Therefore, in this paper, we implemented the method for deformable object simulation that purely uses surface triangle mesh to represent the 3D object. Since the MSM model is not well preserver and potentially suffers from a huge loss of volume over time on the simulation, we employ volume constraint to correct and preserve the volume of the deformable object. The proposed design and implementation of the simulation are performed on Unity3D, a well-known game engine for VR/AR content development, 2D/3D games, and physics simulation. In order to speed up the algorithm, we utilize compute shader, general-purpose computing on the graphics processing unit (GPGPU) as a GPU program to perform the simulation using parallel computing. The specific contributions of this paper are:

- The volume preservation constraint for a mass-spring system to achieve a realistic and efficient deformable object behavior without volumetric meshing involved.
- The performance acceleration using parallel computing in the GPU method is used to simulate the deformable

object in real-time for large and complex 3D mesh models.

The rest of the paper is organized as follows. Section II provides a summary of the previous works and research on deformable object simulations with various methods on the different scenarios. Section III indicates the overview of the general dynamic concepts and equations used in this paper. Section IV presents the proposed method for designing and implementing a force-based volume preservation constraint method on MSM in the Unity3D engine that purely uses a surface triangle model. In the CPU implementation, the method is simply focused on a single-thread program in which all computations are done in the CPU, and update the GPU memory for rendering. Therefore, in the parallel implementation, the algorithms are entirely computed on the GPU kernel program in compute shader script which is a benefit for the rendering purpose. Section V provides the performance comparison of the proposed method with the conventional method in various resolutions of the 3D model. The volume error is used to measure the efficiency of each method for the deformable object simulation for the large and complex 3D model. Section V concludes the paper, provides the limitation of this paper, and also potential future research.

II. RELATED WORK

Deformable body simulation is a fundamental research topic in surgical simulation. A survey of the current state-of-the-art deformable model for modeling and simulation of soft tissue deformation is given in [28]. Many mechanics-based techniques were proposed sequentially which were applied to mesh-based and meshless approaches. FEM is a typical mesh-based method for the simulation of soft bodies which divide the objects into a number of elementary building component and applies to govern differential equations to them [29]. Due to the high computational cost, BEM simplified the FEM complexity by numerically solving the boundary integral equation on a surface mesh [30]. A series of recent studies have indicated that the PBD method is widely used in various real-time interactive graphics applications and physics simulations. Muller et al. [12] employed PBD techniques which describe the use of general constraint and the solver for the approach that considers only position information. In the PBD method, the velocity layer is omitted in order to avoid the problem of explicit time integration schemes in a force-based system. A recent study by Lee et al. [31] proposed an internal shape-preserving constraint (ISPC) algorithm to be integrated into PBD to maintain the 3D deformable object while reducing the number of interior structures in the object. The ISPC algorithm has simply relied on the algebraic constraint function including distance, strain, and bending constraint which potentially generates some artifact and leads to the volume loss problem. Abu Rumman et al. [32] have driven the further usage of the PBD method to simulate the soft body characters by enforcing the tetrahedral volume constraint on the volumetric mesh and reconstructing the surface information later for rendering. The volume constraint that only considers surface meshed for the PBD approach was introduced by Diziol et al. [33]. This method does not require interior nodes and interior links to simulate the object to reduce the computational burden since the volume of a 3D object can be calculated using only triangles information. To obtain a high-quality simulation, they also apply the shape matching algorithms with arbitrary triangle meshes. The PBD approach is a fast, unconditionally stable, controllable, and effective method however, it is not an accurate method compared to the other method such as FEM and MSM. Another disadvantage of PBD is a time step and iteration count independent manner which was discussed and proposed a new method name extended PBD (XPBD). A survey of position dynamics for real-time applications is given in [34].

Due to the simplicity and speed, a method of mass-spring was proposed for a real-time physic simulation [9], [10], [35], [36]. However, the MSM only well perform for simple deformable objects such as linked springs simulation and cloth simulations. Complex and large deformable objects such as soft biological typically suffer the problem of heavy loss of volume since the conventional MSM cannot completely preserve the entire volume of the object effectively. Therefore the constraint projection on MSM is used to avoid the overstretching and over-compressing problem of the spring [37]. However, this remains a problem for general deformable object simulation due to insufficient interior structures.

Research by Vassilev et al. [38] is well documented, it also acknowledged the approach to simulated real-time cloth simulation that utilizes the shape-matching technique on the mass-spring model to allow the stable simulation with large time steps conditions. Zang et al. [39], enforced the tetrahedral volume constraint on the MSM model, which required a volumetric model to simulate the deformable object. Although results appear consistent, the proposed approaches still insufficiently consider the general geometry such as surface triangle mesh. The early research by Hong et al. [40] was conducted for the fast volume preservation constraint for MSM that purely consider closed triangle surface mesh to represent the human muscles and some general deformable bodies.

In terms of performance, the physics simulation required higher framerates which also means the latency of each frame in simulation has to be as low as possible. Therefore, many studies have partitioned their method to be able to fit the concept of parallelism to gain the benefit of performance acceleration. The previous research by Va et al. [41] proposed the GPU-based mass-spring method for cloth simulation using the OpenGL platform. Several researchers have also proposed the parallel method for GPU computing with a remarkable improvement compared to CPU-based approaches [42], [43], [44]. A large number of existing studies utilized GPGPU as the GPU program to perform the simulation, while a method of GPGPU using shading language is still insufficient.

III. OVERVIEW

In this section, we will formulate the general time integration approach for the general deformable body. We consider the coordinate system in this paper as a Cartesian coordinate where a point is in three-dimensional space. For performance advantage, explicit Euler is used as a time integration method.

A. ALGORITHM OVERVIEW

A deformable object composed of a set of N nodes or vertices and a vertex $i \in [1, \ldots, N]$ has its physical property following the concept of particle physics [45]. Then each node consists of:

- A position: $x_i \in \mathbb{R}^3$
- A velocity: $v_i \in \mathbb{R}^3$
- An acceleration: $a_i \in \mathbb{R}^3$
- A Mass: $m_i \in \mathbb{R}$

In general, the deformable object is more difficult compared to the normal rigid body since the deformable object generates a different shape when the external force is given or it contacts a collidable object. Fig. 1 demonstrates the different shapes of a simple deformable object when using various types of coefficients to control.

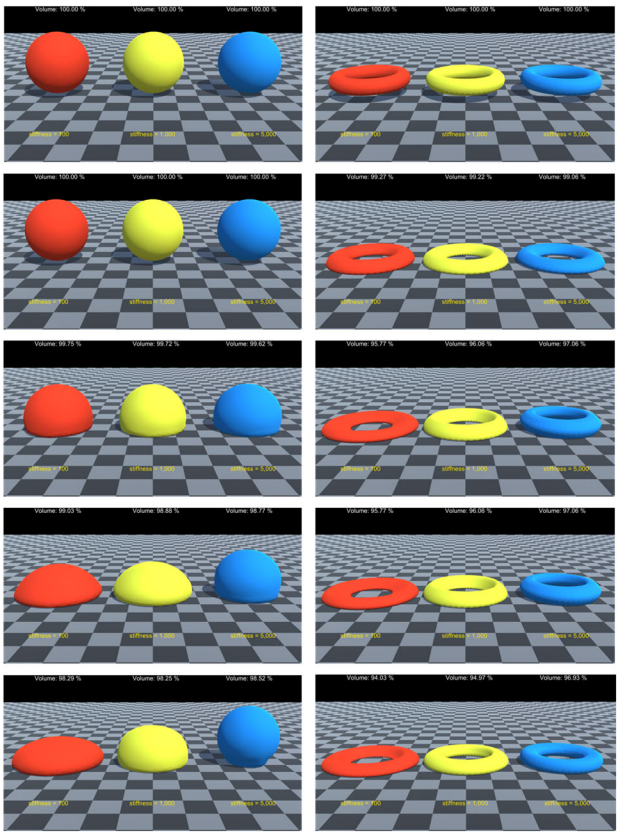

FIGURE 1. Snapshots of the simulation using different numbers of stiffness coefficients to control the deformable bodies using volume preservation constraint on the surface-based mass-spring system method.

1) TIME INTEGRATION

Following Newton's second law of motion, the motion equation of a particle with respect to the time t can be written as follows:

$$\ddot{x}_i(t) = \dot{v}_i(t) = a_i(t) = \frac{1}{m_i} f_i(t) \tag{1}$$

where f_i is the total sum of all forces given to the particle i. Since we only consider soft body simulations, additional attributes such as inertia tensor, orientation, and angular velocity are not mentioned. The goal of the dynamic simulation is to find the positions at the next time step Δt. Based on this information, the time integration for a particle-based on explicit Euler is then performed by (2) and (3).

$$v_i\,(t + \Delta t) = v_i\,(t) + \Delta t \frac{1}{m_i} f_i(t) \tag{2}$$

$$x_i\,(t + \Delta t) = x_i\,(t) + \Delta t v_i\,(t + \Delta t) \tag{3}$$

2) MASS-SPRING SYSTEM

In the general MSM for deformable object simulation, the spring system required a complex structure to simulate various effects like stretching, shearing, and bending. Therefore, the different springs are used to generate different forces to resist internal and external forces. However, in the MSM for surface mesh, the spring is easily manipulated by the edge of each triangle. Even though, some researchers try to generalize the compensated edge inside each triangle to obtain plausible behavior of simulation [46].

The MSM consists of a set of M springs, where each spring is made by two pairs of nodes linked together with the initial length of L_0 as shown in Fig. 2.

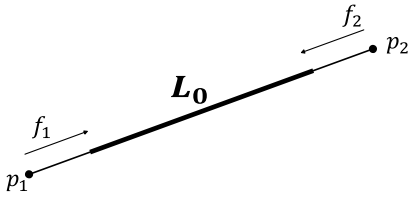

FIGURE 2. A spring property in the mass-spring system.

The spring force acting on both nodes f_1 and f_2 can be computed by Hooke's law since the force is the magnitude proportional to the increase or decrease of the length of the spring to equilibrium. Besides the elastic force given to the spring force, the damping force can be introduced into the system directly to resist the motion by proportioning the velocity of the mass point. Therefore, the spring force can be calculated as follows:

$$f_1 = -f_2 = \Big\{ [k_s\,(|x_2 - x_1| - L_0) + k_d \left(\frac{(v_2 - v_1) \cdot (x_2 - x_1)}{|x_2 - x_1|} \right)] \frac{x_2 - x_1}{|x_2 - x_1|} \Big\} \tag{4}$$

where the proportional constant k_s is the spring constant to control the elastic stiffness of the spring. Similarly, the proportional constant k_d is the spring constant to control the damping of the spring. However, it easily suffers the unstable condition when improper time-step is used for time

integration. In another word, the mass-spring system requires a small integration time-step to use a large stiffness coefficient efficiently and steadily. Many techniques were applied to solve the problems by introducing the constraint on the spring when it stretches and compresses over a certain defined threshold. However, most of the methods tried to move the position directly which ignored the physical consequences of the dynamic behavior of motion, and it can cause many further problems whenever the collision phase involves. A constraint enforcement method that corrects the force magnitude is considered instead of its discussion in [41].

3) VOLUME PRESERVATION CONSTRAINT

The major reason to use volume preservation is to accurately model the motion of a deformable body including soft tissue, muscle, and even large deformation object. The most concerning reason for a deformable body is the incompressibility of the object that uses compressible springs to deform freely while maintaining the overall volume of the object. Another purpose of volume preservation is using only the surface mesh model since it is useful in the computer graphics area in terms of visual quality and rendering performance. Note that the deformable object's surface has to be closed to provide a complete boundary of the object. The convexity and topology of the object are also unrestricted in the divergence theorem.

Fortunately, the volume of the surface mesh can be calculated by the divergence theorem since the triple integral over a deformable object's volume and a surface integral over a deformable object's surface is high computational costs [47]. The relationship between the integration of objects shows as follows:

$$\iiint_V \nabla \cdot f(x)\, dx = \iint_{\partial V} f(x)^T \hat{n}(x)\, dx \tag{5}$$

where V is the volume of the surface triangle and ∂V is the boundary of the volume. $f(x)$ is the vector field of the object and $\hat{n}$ is a unit normal vector of the surface triangle. By using the identity function $f(x) = \mathrm{x}$, the volume integral of the body can be written as follows:

$$\iiint_V \nabla \cdot x dx = \iint_{\partial V} x^T \hat{n} dx = 3V \tag{6}$$

Given a set of triangles in the surface model T_e, where e is the index of the triangle. Therefore, the total volume of the object can be simply calculated by summing over the triangles on the surface mesh.

$$V = \frac{1}{9}\sum_{e=1}^{nt} A_e \left(a_e + b_e + c_e\right)^T \hat{n}_e \tag{7}$$

where nt is the number of surface triangles and $\mathrm{A_e}$ is the area of the e-th triangle. The vertices position a_e, b_e, and c_e are the form of the vertices on the t-th triangle. Then, $\hat{n}_\mathrm{e}$ is the unit normal vector of the e-th triangle. In the entire simulation time, the object's volume must remain steady and unchanged. Therefore, we can turn this into the constraint dynamic system equation.

The formula of constraint Lagrange multiplier s results in a mixed system of ordinary differential equations (ODE) and algebraic equations. Since the particles have three translational degrees of freedom, we can write this system using generalized coordinates in the Cartesian coordinate system as follows.

$$q = [x_1, y_1, z_1, x_2, y_2, z_2 \ldots x_n, y_n, z_n]^T \tag{8}$$

In the multi-constraint dynamic system, the general formulation of the constraint equation composed of m component is formed to the algebraic constraint vector and represents as $\Phi(\mathrm{q}(\mathrm{t}), \mathrm{t}) = \left[\Phi^1(\mathrm{q}, \mathrm{t}), \Phi^2(\mathrm{q}, \mathrm{t}) \ldots \Phi^{\mathrm{m}}(\mathrm{q}, \mathrm{t})\right]^T$, where $\Phi^{\mathrm{i}}(\mathrm{q}, \mathrm{t})$ is the individual scalar algebraic constraint. However, the volume preservation constraint for MSM is only one to preserve the object's volume, then the algebraic constraint vector can be written as follows:

$$\Phi(q(t), t) = \frac{1}{9}\sum_{e=1}^{nt} A_e \left(a_e + b_e + c_e\right)^T \hat{n}_e - V(0) = 0 \tag{9}$$

where $V(0)$ is the object's volume in non-deformed states or the original volume of the object. Fortunately, the partial differentiation of the constraint function can be easily computed and obtain a Jacobian matrix size of $m \times 3n$. If the triangle surface is defined with three nodes, where $(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)$ are the position of three nodes in a triangle, then the formulas for the Jacobian $\partial \mathrm{V}/\partial \mathrm{q}$ are calculated as follows:

$$\frac{\partial V}{\partial x_1} = \frac{1}{2}(y_3 z_2 - y_2 z_3) \quad \frac{\partial V}{\partial y_1} = \frac{1}{2}(x_2 z_3 - x_3 z_2)$$
$$\frac{\partial V}{\partial z_1} = \frac{1}{2}(x_3 y_2 - x_2 y_3) \tag{10a}$$
$$\frac{\partial V}{\partial x_2} = \frac{1}{2}(y_1 z_3 - y_3 z_1) \quad \frac{\partial V}{\partial y_2} = \frac{1}{2}(x_3 z_1 - x_1 z_3)$$
$$\frac{\partial V}{\partial z_1} = \frac{1}{2}(x_3 y_2 - x_2 y_3) \tag{10b}$$
$$\frac{\partial V}{\partial x_3} = \frac{1}{2}(y_2 z_1 - y_1 z_2) \quad \frac{\partial V}{\partial y_3} = \frac{1}{2}(x_1 z_2 - x_2 z_1)$$
$$\frac{\partial V}{\partial z_3} = \frac{1}{2}(x_2 y_1 - x_1 y_2) \tag{10c}$$

In an implicit first-order constraint enforcement scheme, the relation between Lagrange multiplier λ and constraint force is added into the motion equation to predict the new appropriate position as follows:

$$\dot{q}(t + \Delta t) = \dot{q}(t) - \Delta t M^{-1} \Phi_q^T \lambda + \Delta t M^{-1} f^A(q, t) \tag{11}$$
$$q(t + \Delta t) = q(t) + \Delta t \dot{q}(t + \Delta t) \tag{12}$$

where M is a $3n \times 3n$ diagonal mass matrix, f^A is the accumulation of internal force and external forces affecting the node. Φ_q is the Jacobian matrix size of $m \times 3n$, which is made by partial differentiation of q. The constraint function of the next time is treated implicitly and can be written as $\Phi(q(t + \Delta t), t + \Delta t) = 0$. We approximate the solution by

using a truncated first-order Taylor series as follows:

$$\Phi\left(q\left(t\right),t\right)+\Phi_q\left(q\left(t\right),t\right)\left(q\left(t+\Delta t\right)-q\left(t\right)\right)+\Phi_t\left(q\left(t\right),t\right)\Delta t=0 \quad (13)$$

We substitute $\dot{q}\left(\mathrm{t}+\Delta\mathrm{t}\right)$ from (11) into (12) and we can eliminate the implicit generalized coordinates by substituting the result into (13) as follows:

$$\Phi_q\left(q,t\right)M^{-1}\Phi_q^T\lambda=\frac{1}{\Delta t^2}\Phi\left(q,t\right)+\frac{1}{\Delta t}\Phi_t\left(q,t\right)+\Phi_q\left(q,t\right)\left(\frac{1}{\Delta t}\dot{q}\left(t\right)+M^{-1}f^A\left(q,t\right)\right) \quad (14)$$

Since only volume preservation constraint is used, Equation (14) seizes from being a linear system problem and is easy to be solved.

The local deformation technique is used to improve the behavior of the volume preservation constraint because it distributes the preserving condition or the affected region on the surface model. As shown in (14), the volume constraint preserves all of the triangles and nodes on the surface which is not suitable for some materials with complex properties. To provide a flexible technique to get a smooth transition between global and local deformation, the weight vector is used to localize the volume correction. The weight vector W_i where $i \in [1,\ldots,N]$ is set to every surface node. Therefore, $\mathrm{W}_i \in [0,1]$allows the user to control the constraint terms by multiplication in the momentum equations to scale the constraint forces. Hong et al. [40] also proposed the weight distribution function to localize the affected area and non-affected area on the surface smooth as the following equation:

$$W_i=\frac{1}{2}\left[\cos\left(\frac{\|x_i-c\|}{r}\right)+1\right] \quad (15)$$

where c is the position of the loaded external force. r is the radius for the affected zone or deformation zone.

B. GENERAL-PURPOSE COMPUTING ON GRAPHICS PROCESSING UNITS (GPGPU)

In the past, the CPU was designed as a processor that executes the instructions from a computer program. And the CPU contains all the circuitry that needs to perform basic arithmetic, logic, controlling, and input/output (I/O) operations given by the instructions from the computer program. Although the design of the CPU's architecture has changed over time, its core functionality has remained largely unchanged including the control unit (CU), arithmetic logic unit (ALU), and cache, as shown in Fig. 3. Architecturally, the CPU is composed of just a few cores with lots of cache memory that can handle a few software threads at a time [48].

Over the past decade, a graphic processing unit (GPU) was designed as a specialized electronic circuit to handle the image rendering and animation on the computer display. Due to the designing architecture of many cores processing

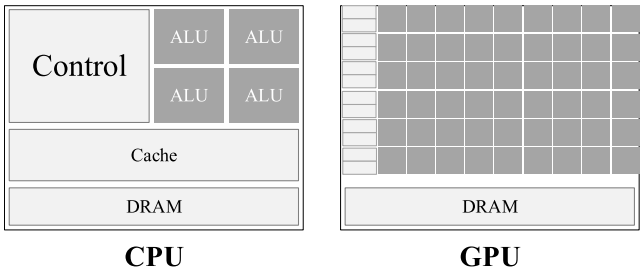

FIGURE 3. Comparison of CPU vs GPU architecture.

and high throughput, its potentials are beyond the rendering purpose as shown in Fig. 3. GPU offers a way to continue accelerating various applications such as graphic modeling and simulation, general computing, and even artificial intelligence (AI) industry by parallel the task within the thousand of available processors [49].

The Unity3D game engine was developed by Unity Technologies. It is most well-known for game development, specifically for VR/AR content and physical simulation. Primary scripting in the C# programming language is available in Unity3D to control the general computation of object movement in the physics simulation [50]. However, it executes all operations as a serial process and can be accelerated by using a multi-thread computing system. Even, though this approach works well still has a limitation when dealing with GPU rendering since the data conversation between CPU and GPU is a bottleneck problem. In order to solve this problem, the GPGPU is required.

GPGPU pipeline is specially designed for parallel processing for accelerating general computation with as many threads as available. A software type or library, such as CUDA and OpenCL, is necessary for GPGPU to implement and design the programmable script on GPU. However, the lightweight GPGPU such as compute shader is more compatible with applications that do not need additional configuration [51].

In general, the shaders are a kind of program that is responsible for the rendering of graphic data. However, the compute shader is a shader program that executes on the GPU, and it runs outside of the normal rendering pipeline. In Unity3D, the compute shader extends the DirectCompute technology of Microsoft Direct3D 11 for cross-platform applications development [52]. Additionally, the compute shader acts as a kernel program that is defined by the specifier number of workgroup blocks, and each workgroup block is made up of a number of threads. There is a limit on the number of threads allowed in a block because they are all expected to share the same processor core. Computing blocks are organized into a one-dimensional, two-dimensional, or three-dimensional grid of thread blocks as illustrated in Fig. 4. The size of the data being processed usually determines how many total thread blocks should be included in the grid. Therefore, all threads are executed in a non-sequential order, which gives an advantage to the performance of the extremely large data.

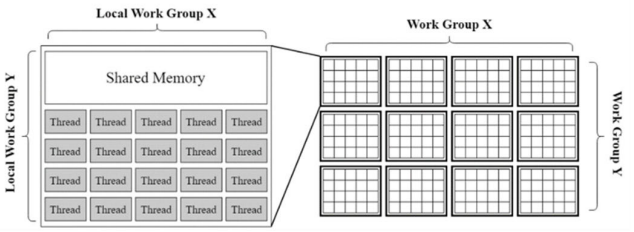

FIGURE 4. The compute space dimension of the kernel in Unity.

IV. IMPLEMENTATION DETAILS

In this section, we present volume preservation constraint techniques for a deformable object using a mass-spring system that purely uses a closed triangle surface in GPU using parallel processing.

Since the force-based method is more accurate than the position-based dynamic, we utilize a mass-spring system even if it causes visual implausibility when using large integration time steps [34]. Unlike, Lee et al. [31], our approach doesn't require any additional preserving constraints to the system for maintaining the shape of the object. Compare our method to Diziol et al. [33], we used only surface volume constraint to correct the forces to avoid the implausible effect and volume loss of the object while they also apply shape matching method to simulate the elasticity of the objects. Our approach is also different from the previous work where the authors use the volume constraint to preserve the element of the deformable object such as a tetrahedron [39], [53].

Typically, the implementation of a conventional MSM approach is simply performed in the following steps in Algorithm 1.

Algorithm 1 Pseudocode for the Conventional MSM Approaches

1: **Initialization**: generate the elastic springs for the surface model.
2: **Loop**
3: Accumulation of spring forces at mass nodes
4: Time integration of mass nodes.
5: Update the next position of the nodes.
6: **End loop**

As previously mentioned, the behavior of the object is strongly dependent on the configuration of the spring network. Since the simulation of a deformable object respects the force-driven method, the calculation process is done in the 1st step of the algorithm. The forces information for each node is then applied to the explicit Euler integration method in order to update the velocity and position of all nodes.

The volume preservation constraint with respect to the force-driven method is then applied to preserve the object's volume in every step of the simulation. Note that, the volume preservation constraint solving is applied after the force accumulation of all nodes to correct the object's volume through forces information. The process of volume preservation constraint using the surface triangle model is given following Algorithm 2. Note that the triangle information is required to solve the volume preservation constraint.

TABLE 1. List of GPU's buffer for GPU-based volume preservation constraint on MSM.

Buffer	Size	Description
Position buffers	N	Contain the list of node's position
Velocity buffer	N	Contain the list of node's velocity
Force buffer	N	Contain the list of node's force
Spring buffer	M	Contain the list of springs on the surface
Triangle buffer	E	Contain the list of surface's triangle

Algorithm 2 Pseudocode for the Volume Preservation Constraint Method

1: **Begin**
2: **for all** triangles i **do**:
3: Compute Jacobian vector
4: Compute error of object's volume
5: **end loop**
6: **for all** nodes j **do**:
7: Compute temporary velocity
8: Compute the right-hand side value of (14)
9: Compute the left-hand side value of (14)
10: Solve Lagrange multiplier λ
11: **end loop**
12: **for all** nodes j **do**:
13: Correct the accumulated force follows (11)
14: **end loop**
15: **End**

In the GPU-based implementation, there are five different GPU buffers to store the list of data needed to compute in the compute shader as shown in Table 1.

The data structure for Position buffers, Velocity buffers, and Force buffers are the same since each element contains a 3D point in space. Since a single spring is made by two linked nodes, the data structure for the Spring buffer has simply contained the indices of the linked node in the Position buffer, and it also contains the initialized distance. In addition, each cell of the Triangle buffer contains the indices of the three vertices or nodes for performing the calculation and visualization purposes.

In the Unity3D engine, the compute shader can be created as a normal script and triggered the kernel program to execute in the C# script. In a compute shader, the users can create multiple kernels in a compute shader. It also can share multiple buffers in a certain number; however, only eight buffers in a maximum number can be assigned to be used in a single kernel program.

Therefore, we build only one compute shader for the entire simulation, and we only utilize two kernels program (*ComputeSpringForce()*, and *UpdateNodes()*)to perform the conventional mass-spring method following Algorithm 1. Additionally, we used four different kernels program for the volume preservation constraint solving as shown in Table 2.

TABLE 2. List of kernel programs for volume preservation constraint on MSM approach.

Kernel	Grid size	Number of thread
ComputeSpringForce()	$\lceil \frac{Springs}{1024} \rceil$	(1024,1,1)
ComputeVolume()	$\lceil \frac{Triangles}{1024} \rceil$	(1024,1,1)
ComputeJacobian()	$\lceil \frac{Nodes}{1024} \rceil$	(1024,1,1)
BuildSystem()	$\lceil \frac{Nodes}{1024} \rceil$	(1024,1,1)
AccumulateForce()	$\lceil \frac{Nodes}{1024} \rceil$	(1024,1,1)
UpdateNodes()	$\lceil \frac{Nodes}{1024} \rceil$	(1024,1,1)

In the *ComputeSpringForce()* kernel, we calculate the elastic spring force following Hooke's law as shown in (4). In the previous discussion in [53] and [54] the node-centric and spring-centric are the most consistent method for adding the new force to the mass nodes. Each mass node adjacency is given information in the node-centric approach. In the spring-centric approach, each spring calculates its spring force only once and propagates it to the mass points it linked. In terms of parallelism, the node-centric approach outperforms the spring- centric approach when the spring is sparsely connected to the corresponding node.

Therefore, we choose the spring-centric approach to calculate the spring force in parallel even if it suffers from the problem of data racing in the writing operation. However, we use the atomic operation concept by locking the block memory which has to be written in the current thread. Note that Unity3D does not provide the atomic add operation for floating-point data type to the compute shader, but the atomic compare and exchange only supports unsigned integers. Therefore, we create a temporary buffer with the unsigned integer as the data type to solve this problem. The pseudocode to compute the spring force and add the force to the linked node is given thin the Algorithm 3.

Algorithm 3 Pseudocode Spring-Centric on Compute Shader

1: Begin
2: j = global thread index
3: initialize force = (0,0,0)
4: i_1, i_2 = indices of nodes in *Spring buffers[j]*
5: force = calculate spring force *Spring buffers[j]*
6: *Force buffer*$[i_1]$ = InterlockAdd(force)
7: *Force buffer*$[i_2]$ = InterlockAdd(-force)
8: End

We also required an additional buffer to solve the volume preservation constraint in compute shader as shown in Table 3. Note that Volume buffer and Lambda buffer is just normal float-point value and they have to be read-write type. And the Jacobian vector is made by the partial derivate with respect to the generalized coordinate so its size is the same as

TABLE 3. List of GPU's buffer to solve the volume preservation constraint.

Buffer	Size	Description
Volume buffer	*1*	Contain the current object's volume
Jacobian buffer	*N*	Contain the list of Jacobian vectors made by partial derivatives on the generalized coordinate.
Lambda buffer	*1*	Contain the Lagrange multiplier to correct the object's volume by force

the number of nodes on the surface model. Additionally, the partial derivative on the node's position obtains three values for each axis (x, y, and z), so each element of the Jacobian vector contains another three values.

In the *ComputeVolume()* kernel, it is the most crucial part of the simulation since the object's volume must be done to compute the error between the current volume with the volume in a non-deformed state. We follow the divergence theorem equation to compute the object's volume as shown in Algorithm 4. Since the object's surface triangles are summing together, the *Interlock()* function is called to perform the atomic add operation on the GPU.

Algorithm 4 Pseudocode of Divergence Theorem

1: Begin
2: i = global thread index
3: i_1, i_2, i_3 = indices of nodes in *Triangle buffers[i]*
4: p_1, p_2, p_3 = *Position buffer*$[i_1, i_2, i_3]$
5: area = $\frac{(|(p_2-p_1)\times(p_3-p_1)|)}{2}$
6: tmp = area $\times$ $(p_1 + p_2 + p_3)$
7: $\text{normal}_{normalized} = (|(p_2 - p_1) \times (p_3 - p_1)|)$
8: volume = $\frac{(\text{tmp}\cdot\text{normal}_{normalized})}{9}$
9: *Volume buffer*[0] = InterlockAdd(volume)
9: End

In line (3) of Algorithm 2, we can simply calculate the Jacobian vector by looping through the triangle of the surface model. However, in the GPU implementation, we cannot apply the same mechanism to obtain the Jacobian vector since the data racing problem occur when the thread invokes the same buffer block to write the new Jacobian element of the buffer. Therefore, we use the node-centric approach since we can easily assign GPU thread per node and loop over corresponding triangles to construct the Jacobian element by partial derivative as shown in Algorithm 5.

In the *BuildSystem()* kernel, the GPU thread invokes the Jacobian buffer, volume buffer, Velocity buffer, and Force buffer to compute the left-hand and right-hand side of the system in (14) to compute the Lagrange multiplier. Therefore, we also need a temporary buffer to store the left-hand and right-hand as the scalar value. Next, those two temporary buffers are used in the *AccumulateForce()* kernel to compute the new force for correcting the object's volume. Note that, we don't require an additional buffer to store the Lagrange multiplier, since it is easily found by the division of the

TABLE 4. Experimental environment.

Component	Specification
OS	Windows 10 Pro 10.0.19044 Build 19044
CPU	Intel® Core™ i7-7700
RAM	32GB
GPU	NVIDIA GeForce GTX 1070 8 GB V-RAM
Shader	HLSL Shader model 5.0 (Windows Graphics API)
IDE	Unity engine version 2020.3.8f1 Personal, Microsoft Visual Studio Community 2019 Version 16.11.2

right-hand side and left-hand side of the linear system in (14). Each element of the Force buffer is then easily computed since each GPU thread has access to each element of the Jacobian vector correspondingly.

Algorithm 5 Pseudocode of Jacobian Vector Calculation

```
Begin
  i = global thread index
  for all triangle t affecting node i do:
    i1, i2, i3 = indices of nodes in Triangle buffers[t]
    p1, p2, p3 = Position buffer[i1, i2, i3]
    if i = i1
      Compute Jacobian respect to (10a)
    End if
    Else if i = i2
      Compute Jacobian respect to (10b)
    End if
    Else if i = i3
      Compute Jacobian respect to (10c)
    End if
  End loop
End
```

After the correcting forces at the next time-step are found, the *UpdateNodes()* kernel is executed to calculate the new velocity and position of each node. Each GPU thread is responsible for the calculation process for each node based on the explicit Euler method as shown in (2) and (3).

V. RESULT

A. EXPERIMENTAL ENVIRONMENT

Table 4 shows the specifications with which our experiment was conducted on the desktop. Since the simulation was developed on the Unity3D engine, we only target the window PCs as the main platform for this simulation. However, it is easy to integrate into other platforms such as mobile and even AR/VR platforms.

B. PERFORMANCE RESULT

We used various 3D surface models to conduct a comparative simulation with different mesh resolutions. Fig. 5 shows the 3D surface model used in this paper.

3D Sphere and Torus models were made by Blender [55], a 3D modeling software to generate various kinds of 3D surface objects. The rest of the 3D models were obtained by the Stanford repository which is made by 3D scanning and available to download [56]. Note that the 3D model is stored in polygon file format (PLY).

We use the TetGen library to generate a closed surface triangle mesh to perform the simulation since general 3D object generally has overlapping vertices and triangles which is a problem applying the volume preservation constraint [57]. Therefore, we only use the surface information for simulating and rendering the 3D object. Another reason to use a 3D model made by the TetGen library is that the edge information is easily extracted to compute the spring force.

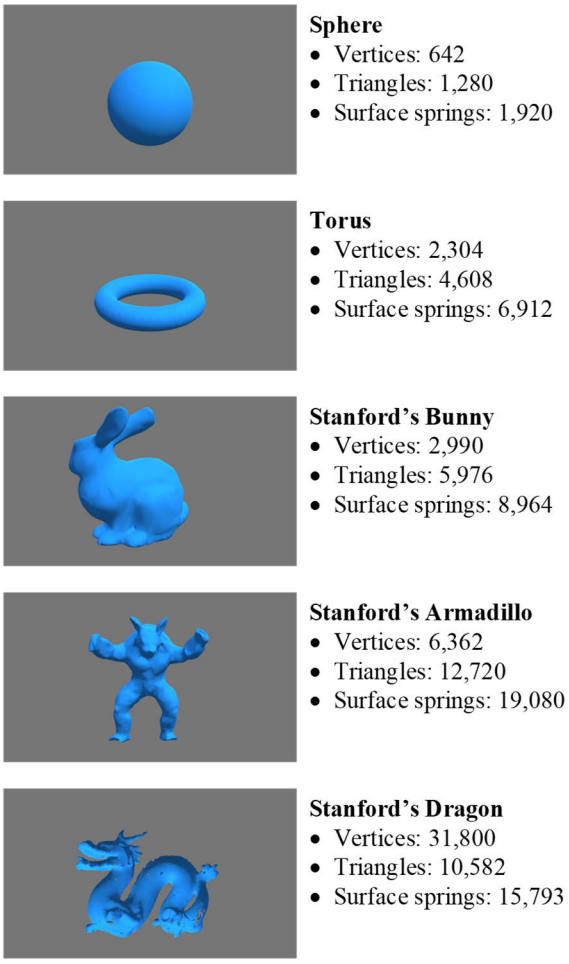

FIGURE 5. The 3D model is used in this simulation for performance comparison.

To compare the performance of the simulation, we calculate the average fps for 400 frames of the simulation single object in a scene. The VSync Count is set to "Don't Sync" in order to get a fair performance comparison for the simple and complex objects since the default settings of Unity3D set the VSync Count to "Every V-Blank" to target the framerate of the animation to match with the monitor's framerate. As mentioned in Table 2, the grid size for the local workgroup of thread is set to 1,024 for each GPU kernel since it is the maximum size for the one-dimension of workgroup size in HLSL's compute shader. Fig. 6 shows the performance

comparison between CPU-based implementation and GPU-based implementation methods. Where MSM refers to the mass-spring model on a surface triangle model, VPC refers to the volume preservation constraint on the MSM. The the computational complexity of our approach is represented by the GPU VPC bar graph and measured in fps. Although our simulation method can handle large time steps, we run all simulations with a fixed time step size of 0.0001 to ensure the stability of the simulation.

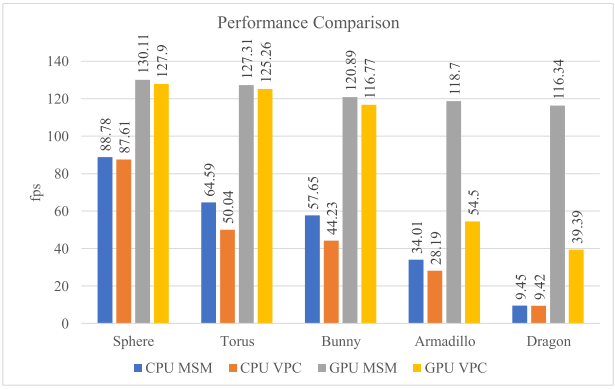

FIGURE 6. Performance comparison of deformable object simulation using CPU-based and GPU-based approaches.

As the expectation, the performance of the simulation can be accelerated by using the parallel processing method on the GPU with an average speedup factor of 4.27 using the MSM method, and an average speedup factor of 2.54 for volume preservation constraint. In the CPU-based MSM, it can only simulate Stanford's armadillo model in real-time. Due to the simple calculation of spring force using spring centric method in the GPU kernel, the GPU-based MSM achieves real-time performance for all cases. The MSM method for deformable object simulation is a fast method but problematic. The surface-based volume preservation constraint is used to correct the object's volume through the internal force of the surface's nodes. Only one volume constraint is enforced on the MSM, and it can simulate Stanford's bunny in real-time using a CPU-based approach with an average framerate of 44.23 fps. When the mesh's resolution is increased, it only achieves 28.19fps and 9.42fps for the Stanford armadillo and dragon models, respectively. However, the GPU-based method improves the performance of volume constraint solving to be able to perform the simulation of the Stanford armadillo and dragon in real-time with an average framerate of 54.5fps and 39.39fps, respectively.

Compare the performance result to previous works where most GPU-based implementations, used CUDA, therefore our performance can be performed with the mesh with a maximum resolution of 31K nodes and about 20K triangles. While the state of art methods such as PBD can be used to simulate higher resolution mesh up to 32K nodes for real-time simulation [33]. However, Lee et al. [31], implement the PBD method on a unity3D engine using compute shader using only distance, bending, and strain constraint. The real-time performance ($\geq$ 30 fps) can handle about 6K vertices and 19K constraints.

Additionally, the GPU-based volume preservation constraint is not as fast as conventional MSM in the GPU scheme because of the usage of locking processing when doing the concurrent sum to calculate the object's volume every frame of the simulation. The usage of concurrent sum in parallel to compute an object's volume using atomic operation is a bottleneck since the atomic slows down the calling thread and locks the accessed buffer element from another thread.

Even though the performance of volume preservation constraint on the MSM approach is slower than conventional MSM, it can preserve the object's volume more correctly. As shown in Fig. 1, we use different stiffness proportional constants to control the stiffness of the deformable object that purely taking surface mesh into account, the 3D sphere model with the stiffness coefficient of 100, 1000, and 5000 obtains maximum volume loss of 3.67%, 2.69%, and 1.56%, respectively. Similarly, the 3D torus model with the stiffness coefficient of 100, 1000, and 5000 obtains maximum volume loss of 11.71%, 9.51%, and 7.12%, respectively.

Extensive results carried out show that the volume preservation constraint improves the result of deformable object simulation by using a suitable stiffness coefficient. Therefore, we conduct another comparison of deformable object simulation where the stiffness and damping coefficient was set to 5000 and 200, respectively. Fig. 7 has shown the maximum percentage of volume loss for the object using MSM with only surface spring, and volume preservation constraint on (VPC) MSM.

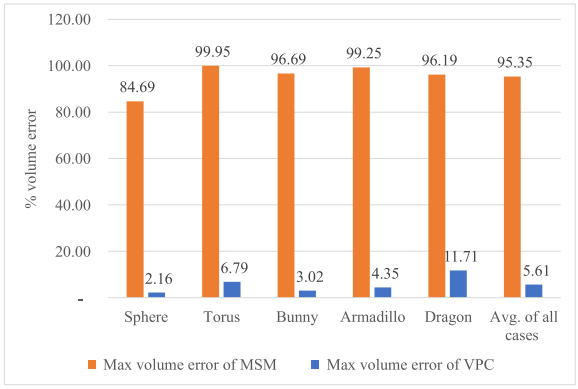

FIGURE 7. The maximum loss of free-fall simulation.

We captured the error of volume percentage for all experimental models in a free fall with a simple flat surface as a collider. The result reveals significant differences in volume error. For all experimental models using only the MSM approach, the maximum loss on average is 95.35% which also means that almost the entire object's volume cannot be well-preserved and does not remain the same as the initialized state. We also calculated the average volume error for 400 frames of the simulation using the experimental 3D models with conventional MSM and the proposed approach as shown in Fig. 8.

Overall, the proposed approach of volume preservation constraint delivers sophisticated results with significant improvement for the loss of the deformable object's volume. The volume preservation constraint approaches are successfully applied to improve the stability of the mass-spring system for the large and complex model of the deformable object. We also prove that the usage of parallel processing in GPU is way faster than the naïve approach that used serial processing in GPU. Fig. 9 shows the snapshots of the simulation we conducted.

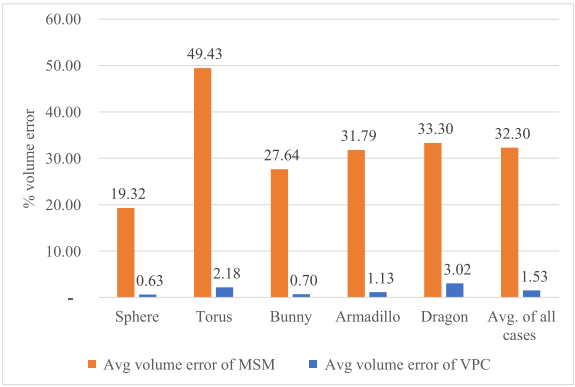

FIGURE 8. The average loss of volume preservation for the three methods.

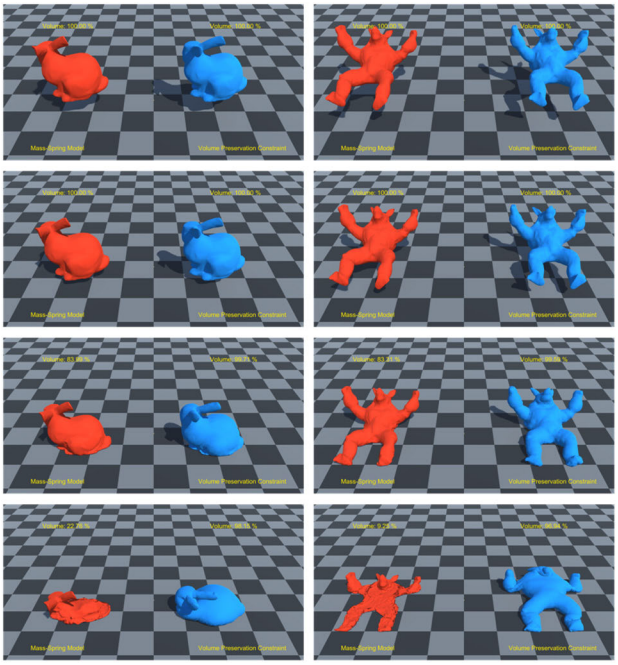

FIGURE 9. Snapshots of the simulation using the MSM (red color), and volume preservation on the MSM (blue color).

To confirm the efficiency of our proposed method, the experimental test of pressing two flat objects with the deformable object was conducted. The transparent object is moving to press the deformable objects to the ground. Fig. 10 shows the motion comparison of the Stanford bunny model and Stanford armadillo model using the conventional mass-spring method and our proposed method. In this experiment, the spring's stiffness and spring damping are set to 1000 and 200, respectively.

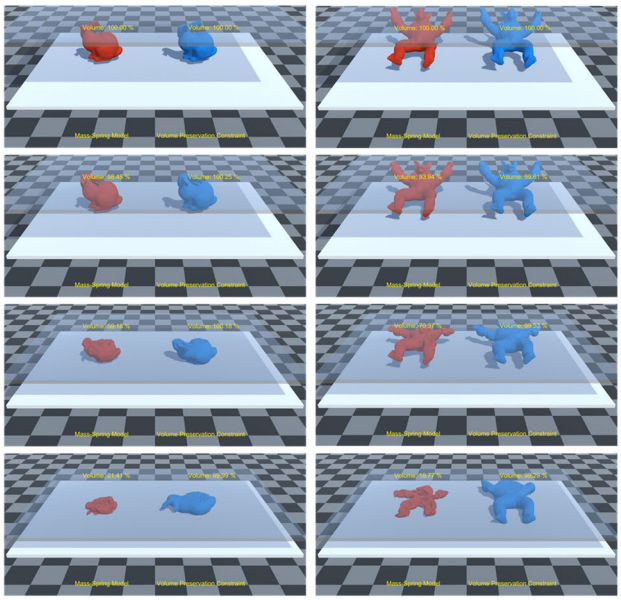

FIGURE 10. Comparison of objects when heavy pressure is applied to deformable objects using the MSM (red), and our proposed method (blue).

As a result of the experimental test, we recorded the volume of the deformable object for 600 frames, the maximum volume loss of Stanford bunny and armadillo using MSM are 79.84% and 80.35%, respectively. However, the maximum volume loss of Stanford bunny and armadillo using our proposed approach is only 0.33% and 0.97%, respectively. It was clear that the volume preservation constraint approach well maintained the object volume and only consider the surface mesh as a volume constraint. The performance of the heavy pressure applied to the bunny models and armadillo model are 37.31fps and 18.59fps, respectively.

Another experimental test of volume preservation was also conducted. We used a cube (386 vertices, 768 triangles, and 1,777 springs) and performed the twisting behavior on its top side while the bottom side is set to be fixed. In this experimental test, we mimic the internal spring to improve the behavior of the volume constraint.

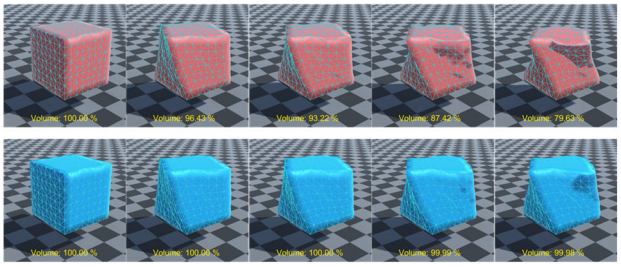

FIGURE 11. Snapshot of the cube twisting simulation using the MSM (top row) and volume preservation constraint on the MSM (bottom row).

As a cube twisting test, the spring's stiffness is set to 1000 and the damping force of the spring is eliminated.

Therefore, the cube twisting using the conventional MSM obtains the maximum of volume loss 20.37% of the total volume. Alternatively, when using our approach for cube twisting, the maximum volume loss is only 0.02%. The performance of cube twisting using MSM, and volume preservation constraints are 125.17fps and 123.69fps, respectively. Note that we calculate the torque force and applied to the affected node on the top side of the cube.

VI. CONCLUSION

We proposed a method to design and implement the simulation of the deformable object that purely considers only surface mesh based on a conventional mass-spring system and applied the volume preservation constraint to improve the volume preservation in the Unity3D environment. The simulation is specifically implemented on the GPU-based approach in order to accelerate the performance compared to the simple CPU-based approach.

Comparing the GPU-based to CPU-based MSM, we obtain an average speedup of factor 4.27. Likewise, the average speedup of factor 2.53 is achieved for GPU-based acceleration using volume preservation constraint on the MSM. Even though the volume preservation constraint method is not as fast as the conventional mass-spring method, the volume preservation result is much more accurate in terms of volume loss during the simulation. In the free-fall experiment, the maximum volume loss of the deformable body is an average of 95.35 for the conventional MSM which is much higher than the volume preservation constraint that only achieves the maximum volume loss of 5.61% on average. Furthermore, the pressing experimental test and cube twisting test also achieve a similar result which is the volume preservation constraint is well maintained in the deformable object's volume.

However, the use of atomic operation is the main bottleneck of the entire operation in a simulation, even though we use it to perform concurrent sum in the GPU to compute the object's volume. Many optimization algorithms such as reduction and graph coloring have been introduced which required further study in order to apply in our GPU-based algorithm. Another limitation is that the plausible elastic and plastic deformation can be performed by our volume preservation constraint method under the circumstance in which shape matching is applied to keep the detail of the mesh during the simulation as mentioned in [33]. Presently, we only explore the built-in features in the Unity engine in order to implement our simulation. Our method can be implemented on other engines as well but additional libraries might be required for GPU computing and rendering purposes.

In future work, we will explore the possibility of implementing our proposed method on another engine as well. Additionally, we will apply collision detection to our GPU-based simulation since the current study only focuses on CPU-based which required read-write data from the GPU which is time-consuming. We will focus on the complex 3D biological model in order to perform the virtual cutting in real-time.

REFERENCES

[1] A. Wirtz, M. Meißner, P. Wiederkehr, D. Biermann, and J. Myrzik, "Evaluation of cutting processes using geometric physically-based process simulations in view of the electric power consumption of machine tools," *Proc. CIRP*, vol. 79, pp. 602–607, Jan. 2019.
[2] O. Robert, P. Iztok, and B. Borut, "Real-time manufacturing optimization with a simulation model and virtual reality," *Proc. Manuf.*, vol. 38, pp. 1103–1110, Jan. 2019.
[3] Y. Ryu and E. Ryu, "Overview of motion-to-photon latency reduction for mitigating VR sickness," *KSII Trans. Internet Inf. Syst.*, vol. 15, no. 7, pp. 2531–2546, 2021.
[4] S. Saeed and U. Park, "A study on presence quality and cybersickness in 2D, smartphone, and VR," *KSII Trans. Internet Inf. Syst.*, vol. 16, no. 7, pp. 2305–2327, 2022.
[5] W. K. Liu, S. Li, and H. S. Park, "Eighty years of the finite element method: Birth, evolution, and future," *Arch. Comput. Methods Eng.*, vol. 29, no. 6, pp. 4431–4453, Oct. 2022.
[6] B. E. Griffith and X. Luo, "Hybrid finite difference/finite element immersed boundary method," *Int. J. for Numer. Methods Biomed. Eng.*, vol. 33, no. 12, Dec. 2017.
[7] L. Van Hoecke, D. Boeye, A. Gonzalez-Quiroga, G. S. Patience, and P. Perreault, "Experimental methods in chemical engineering: Computational fluid dynamics/finite volume method—CFD/FVM," *Can. J. Chem. Eng.*, vol. 101, no. 2, pp. 545–561, Feb. 2023.
[8] L. Najarzadeh, B. Movahedian, and M. Azhari, "Numerical solution of scalar wave equation by the modified radial integration boundary element method," *Eng. Anal. with Boundary Elements*, vol. 105, pp. 267–278, Aug. 2019.
[9] X. Zhang, J. Duan, W. Sun, T. Xu, and S. K. Jha, "A three-stage cutting simulation system based on mass-spring model," *Comput. Model. Eng. Sci.*, vol. 127, no. 1, pp. 117–133, 2021.
[10] T. Liu, A. W. Bargteil, J. F. O'Brien, and L. Kavan, "Fast simulation of mass-spring systems," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1–7, Nov. 2013.
[11] W. Zhang, Y. Ma, J. Zheng, and W. J. Allen, "Tetrahedral mesh deformation with positional constraints," *Comput. Aided Geometric Design*, vol. 81, Aug. 2020, Art. no. 101909.
[12] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *J. Vis. Commun. Image Represent.*, vol. 18, pp. 109–118, Apr. 2007.
[13] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 471–478, Jul. 2005.
[14] A. Bernardin, E. Coevoet, P. Kry, S. Andrews, C. Duriez, and M. Marchal, "Constraint-based simulation of passive suction cups," *ACM Trans. Graph.*, vol. 42, no. 1, pp. 1–14, Sep. 2022.
[15] M. Sung and H. Choi, "Real-time augmented reality physics simulator for education," *Appl. Sci.*, vol. 9, no. 19, p. 4019, Sep. 2019.
[16] K. Kounlaxay, Y. Shim, S. Kang, H. Kwak, and S. K. Kim, "Learning media on mathematical education based on augmented reality," *KSII Trans. Internet Inf. Syst.*, vol. 15, no. 3, pp. 1015–1029, 2021.
[17] G. ZhaoZhe, L. XiaoDong, S. XiaoYing, L. Geng, and C. Bin, "Research of 3D image processing of VR technology in medicine based on DNN," *KSII Trans. Internet Inf. Syst.*, vol. 16, no. 5, pp. 1584–1596, 2022.
[18] Z. Feng, V. A. González, R. Amor, R. Lovreglio, and G. Cabrera-Guerrero, "Immersive virtual reality serious games for evacuation training and research: A systematic literature review," *Comput. Educ.*, vol. 127, pp. 252–266, Dec. 2018.
[19] T. Baniasadi, S. M. Ayyoubzadeh, and N. Mohammadzadeh, "Challenges and practical considerations in applying virtual reality in medical education and treatment," *Oman Med. J.*, vol. 35, no. 3, pp. e125–e125, May 2020.
[20] M.-S. Bracq, E. Michinov, and P. Jannin, "Virtual reality simulation in nontechnical skills training for healthcare professionals: A systematic review," *Simul. Healthcare: J. Soc. Simul. Healthcare*, vol. 14, no. 3, pp. 188–194, 2019.
[21] S. Lee, M. Hong, S. Kim, and S. J. Choi, "Effect analysis of virtual-reality vestibular rehabilitation based on eye-tracking," *KSII Trans. Internet Inf. Syst.*, vol. 14, no. 2, pp. 826–840, 2020.
[22] S. G. Han, Y. D. Kim, T. Y. Kong, and J. Cho, "Virtual reality-based neurological examination teaching tool(VRNET) versus standardized patient in teaching neurological examinations for the medical students: A randomized, single-blind study," *BMC Med. Educ.*, vol. 21, no. 1, p. 493, Dec. 2021.

[23] S. Lee, M. Hong, H. Va, and J.-Y. Park, "Implementation of a subjective visual vertical and horizontal testing system using virtual reality," *Comput., Mater. Continua*, vol. 67, no. 3, pp. 3669–3679, 2021.

[24] J. Ma, H.-J. Kim, J.-S. Kim, E.-S. Lee, and M. Hong, "Virtual reality-based random dot kinematogram," *Comput., Mater. Continua*, vol. 68, no. 3, pp. 4205–4213, 2021.

[25] H. Wang, H. Peng, Y. Chang, and D. Liang, "A survey of GPU-based acceleration techniques in MRI reconstructions," *Quant. Imag. Med. Surg.*, vol. 8, no. 2, pp. 196–208, Mar. 2018.

[26] F. M. Madsen and A. Filinski, "Streaming nested data parallelism on multicores," in *Proc. 5th Int. Workshop Funct. High-Performance Comput.*, Nara, Japan, Sep. 2016, pp. 44–51.

[27] X. Wang, Y. Qiu, S. R. Slattery, Y. Fang, M. Li, S.-C. Zhu, Y. Zhu, M. Tang, D. Manocha, and C. Jiang, "A massively parallel and scalable multi-GPU material point method," *ACM Trans. Graph.*, vol. 39, no. 4, p. 30, Aug. 2020.

[28] J. Zhang, Y. Zhong, and C. Gu, "Deformable models for surgical simulation: A survey," *IEEE Rev. Biomed. Eng.*, vol. 11, pp. 143–164, 2018.

[29] M. Freutel, H. Schmidt, L. Dürselen, A. Ignatius, and F. Galbusera, "Finite element modeling of soft tissues: Material models, tissue interaction and challenges," *Clin. Biomechanics*, vol. 29, no. 4, pp. 363–372, Apr. 2014.

[30] B. Zhu and L. Gu, "A hybrid deformable model for real-time surgical simulation," *Computerized Med. Imag. Graph.*, vol. 36, no. 5, pp. 356–365, Jul. 2012.

[31] D.-K. Lee, T.-W. Kim, Y.-J. Choi, and M. Hong, "Volumetric object modeling using internal shape preserving constraint in unity 3D," *Intell. Autom. Soft Comput.*, vol. 32, no. 3, pp. 1541–1556, 2022.

[32] N. Abu Rumman and M. Fratarcangeli, "Position-based skinning for soft articulated characters," *Comput. Graph. Forum*, vol. 34, no. 6, pp. 240–250, Sep. 2015.

[33] R. Diziol, J. Bender, and D. Bayer, "Robust real-time deformation of incompressible surface meshes," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, Vancouver, BC, Canada, Aug. 2011, pp. 237–246.

[34] J. Bender, M. Müller, and M. Macklin, "A survey on position based dynamics," in *Proc. Eur. Assoc. Comput. Graphics, Tuts.*, Lyon, France, 2017, pp. 1–31.

[35] Y. Duan, W. Huang, H. Chang, W. Chen, J. Zhou, S. K. Teo, Y. Su, C. K. Chui, and S. Chang, "Volume preserved mass–spring model with novel constraints for soft tissue deformation," *IEEE J. Biomed. Health Informat.*, vol. 20, no. 1, pp. 268–280, Jan. 2016.

[36] P. M. Pieczywek and A. Zdunek, "Compression simulations of plant tissue in 3D using mass–spring system approach and discrete element method," *Soft Matter*, vol. 13, no. 40, pp. 7313–7331, 2017.

[37] R. Durikovic and E. Siebenstich, "A mass spring model for string simulation with stress-strain handling," in *Proc. 6th Int. Symp. Comput. Netw. Workshops (CANDARW)*, Takayama, Japan, Nov. 2018, pp. 8–14.

[38] T. I. Vassilev, "Mass-spring cloth simulation with shape matching," in *Proc. 18th Int. Conf. Comput. Syst. Technol.*, Ruse, Bulgaria, Jun. 2017, pp. 257–264.

[39] X. Zhang, H. Wu, W. Sun, and C. Yuan, "An optimized mass-spring model with shape restoration ability based on volume conservation," *KSII Trans. Internet Inf. Syst.*, vol. 14, no. 4, pp. 1738–1756, 2020.

[40] M. Hong, S. Jung, M.-H. Choi, and S. W. J. Welch, "Fast volume preservation for a mass-spring system," *IEEE Comput. Graph. Appl.*, vol. 26, no. 5, pp. 83–91, Sep. 2006.

[41] H. Va, M.-H. Choi, and M. Hong, "Parallel cloth simulation using OpenGL shading language," *Comput. Syst. Sci. Eng.*, vol. 41, no. 2, pp. 427–443, 2022.

[42] Z. Zhang, "Soft-body simulation with CUDA based on mass-spring model and verlet integration scheme," in *Proc. ASME Int. Mech. Eng. Congr. Expo.*, vol. 7A, Nov. 2020, pp. 1–9.

[43] M. Hong, J.-H. Jeon, H.-S. Yum, and S.-H. Lee, "Plausible mass-spring system using parallel computing on mobile devices," *Human-Centric Comput. Inf. Sci.*, vol. 6, no. 1, pp. 1–11, Dec. 2016.

[44] M. Kim, N.-J. Sung, S.-J. Kim, Y.-J. Choi, and M. Hong, "Parallel cloth simulation with effective collision detection for interactive AR application," *Multimedia Tools Appl.*, vol. 78, no. 4, pp. 4851–4868, Feb. 2018.

[45] M. M. Bhatti, M. Marin, A. Zeeshan, and S. I. Abdelsalam, "Editorial: Recent trends in computational fluid dynamics," *Frontiers Phys.*, vol. 8, p. 4, Oct. 2020.

[46] J.-H. Zhu, Y. Li, W.-H. Zhang, and J. Hou, "Shape preserving design with structural topology optimization," *Structural Multidisciplinary Optim.*, vol. 53, no. 4, pp. 893–906, Apr. 2016.

[47] T. Lu, B. Chen, Z. Zou, and D. Wang, "Hybrid method of FEM and divergence theorem to analyze ion flow field including dielectric film's accumulation charges," *IEEE Trans. Magn.*, vol. 57, no. 6, pp. 1–4, Jun. 2021.

[48] M. Forsell, S. Nikula, J. Roivainen, V. Leppänen, and J. L. Träff, "Performance and programmability comparison of the thick control flow architecture and current multicore processors," *J. Supercomput.*, vol. 78, no. 3, pp. 3152–3183, Feb. 2021.

[49] C. Deng, X. Fang, X. Wang, and K. Law, "Software orchestrated and hardware accelerated artificial intelligence: Toward low latency edge computing," *IEEE Wireless Commun.*, vol. 29, no. 4, pp. 110–117, Aug. 2022.

[50] *Unity Real-Time Development Platform*. Accessed: Oct. 12, 2022. [Online]. Available: https://unity.com/

[51] H. Va, M.-H. Choi, and M. Hong, "Real-time cloth simulation using compute shader in Unity3D for AR/VR contents," *Appl. Sci.*, vol. 11, no. 17, p. 8255, Sep. 2021.

[52] *Compute Shaders-Unity*. Accessed: Oct. 12, 2022. [Online]. Available: https://docs.unity3d.com/Manual/class-ComputeShader.html

[53] H. Va, M.-H. Choi, and M. Hong, "Real-time volume preserving constraints for volumetric model on GPU," *Comput., Mater. Continua*, vol. 73, no. 1, pp. 831–848, 2022.

[54] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 1–12, Jul. 2014.

[55] *Blender*. Accessed: Oct. 12, 2022. [Online]. Available: https://www.blender.org/

[56] M. Levoy, J. Gerth, B. Curless, and K. Pull. *The Stanford 3D Scanning Repository*. Accessed: Oct. 12, 2022. [Online]. Available: http://www-graphics.stanford.edu/data/3Dscanrep/

[57] H. Si, "TetGen, a Delaunay-based quality tetrahedral mesh generator," *ACM Trans. Math. Softw.*, vol. 41, no. 2, pp. 1–36, Jan. 2015.

HONGLY VA received the B.E. degree in information technology engineering from the Royal University of Phnom Penh, in 2019. He is currently pursuing the Ph.D. degree with the Department of Software Convergence, Soonchunhyang University, Asan-si, Republic of Korea. His research interests include computer graphics, virtual reality, parallel computing, physically-based modeling, and simulation.

MIN-HYUNG CHOI received the M.S. and Ph.D. degrees from the University of Iowa, in 1996 and 1999, respectively. Currently, he is a Professor in computer science and the Director of the Computer Graphics and VR Laboratory, University of Colorado Denver. His research interests include computer graphics, physically-based modeling and simulation, scientific visualization, and human–computer interaction in VR.

MIN HONG received the B.S. degree in computer science from Soonchunhyang University, Asan-si, Republic of Korea, in 1995, and the M.S. degree in computer science and the Ph.D. degree in bioinformatics from the University of Colorado, in 2001 and 2005, respectively. He is currently a Professor with the Department of Computer Software Engineering, Soonchunhyang University. His research interests include computer graphics, mobile computing, physically-based modeling, and simulation.