Volume xx (*200y*), *Number z, pp. 1–19*

Boundary-Particle-Driven Gaussian Splatting for One-Pass Particle Dynamics-to-Rendering with Deformation and Topology Changes

HyeonMin Jeong[1], Jong-Hyun Kim[2]

[1]Dept. of Computer Engineering, Inha University, 100 Inha-ro, Michuhol-gu, Incheon 22212, South Korea
[2]College of Software and Convergence (Dept. of Artificial Intelligence, Design Technology), Graduate School of Electrical and Computer Engineering, Inha University, 100 Inha-ro, Michuhol-gu, Incheon 22212, South Korea

Abstract

Real-time simulation of deformable objects together with high-quality rendering remains an important challenge in computer graphics. This problem becomes particularly difficult in scenarios involving topology changes, such as cutting or tearing, where physical simulation and visual representation must be tightly coupled in a stable manner. Existing Gaussian-based rendering approaches primarily focus on static or quasi-static scenes optimized from captured images, making them unsuitable for handling dynamically changing connectivity or topology changes in real time. Moreover, many deformable simulation frameworks assume a fixed mesh connectivity, which limits their ability to directly handle operations such as cutting or fracture. In this paper, we represent deforming surfaces using boundary particles that serve as physically meaningful samples, and directly map them to Gaussian primitives. Based on this idea, we propose a one-pass *framework that unifies simulation and rendering within a single pipeline. The proposed method directly couples Position-Based Dynamics (PBD) based deformable simulation with Gaussian splatting rendering, allowing stable constraint projection and continuous rendering even when topology changes such as cutting and tearing occur. Furthermore, by using the boundary-particle representation, our approach can render dynamically evolving surfaces without requiring explicit surface reconstruction or re-optimization. This enables the generation of temporally coherent visual results even in real-time interactive environments. Experimental results demonstrate that the proposed method achieves both stable simulation and consistent rendering under large deformations and topology-changing events.*

(see http://www.acm.org/about/class/class/2012)

CCS Concepts

•***Computing methodologies*** → ***Physical simulation; Animation; Computer graphics;*** *Point-based graphics; Image-based rendering; Physically based modeling; Simulation of solid mechanics;*

1. Introduction

Real-time physics-based simulation has long been a central research topic in computer graphics. In particular, soft deformable objects such as cloth, balloons, and thin shells exhibit complex geometric deformation and dynamic behavior in response to external forces, collisions, and user interactions. Therefore, simulation methods that are both numerically stable and capable of real-time interaction are highly desirable. One representative approach that satisfies these requirements is Position-Based Dynamics (PBD), which provides high stability and computational efficiency through a constraint-projection-based formulation.

Despite the significant progress in deformable simulation techniques, the rendering stage that visually represents simulation results is still often constructed as a separate pipeline. Most real-time systems either render the mesh directly generated by the simulation stage or perform visualization through additional surface reconstruction and post-processing steps. While such a separated pipeline works well for simple deformation scenarios, it becomes problematic when topology changes occur, such as cutting or tearing. In these cases, remeshing, connectivity reconstruction, and visual discontinuities often arise, which increase computational cost and introduce inconsistencies between simulation and rendering.

These limitations are closely related to the structural constraints of representation methods. Mesh-based representations rely on explicit connectivity structures, which makes dynamic updates expensive and implementation complex when connectivity changes over time. In contrast, point-based or implicit representations have lower dependency on explicit connectivity and are therefore more suitable for dynamic scenes. However, existing point-based representations have rarely been designed to integrate tightly with physics simulation.

Recently, 3D Gaussian Splatting (3DGS) has been proposed as

an efficient representation that models scenes as a set of Gaussian primitives and directly renders them through screen-space splatting, achieving both high visual quality and real-time rendering performance. However, existing 3DGS approaches are primarily designed for static or quasi-static scenes reconstructed from multi-view images, and have not been developed as a representation that can directly drive real-time physics simulations with dynamically changing topology.

To bridge this gap, we propose a new framework that integrates physics simulation and rendering not as separate stages but at the *representation level*. The key idea is to maintain the deforming surface as a set of physically meaningful samples using boundary particles, and to directly map these samples to Gaussian primitives without requiring reconstruction or learning processes. This design enables a unified *one-pass* pipeline in which particle states updated during simulation immediately become rendering primitives.

Boundary particles maintain stable sampling density and geometric information along surface regions, and particularly preserve surface continuity and silhouette structure even after cutting or tearing events. As a result, simulation and rendering can continue seamlessly within the same representation structure even when topology changes occur.

Furthermore, this work explicitly supports dynamic topology change through a solver extension that includes line-based cutting operations. Although mesh connectivity is updated during cutting, both the particle-based dynamics and the Gaussian-based rendering representation maintain the same data structure, thereby ensuring temporal continuity and visual consistency.

At the rendering stage, we adopt isotropic Gaussian primitives and position-based appearance computation, which alleviates flickering and shading discontinuities that may arise from anisotropic Gaussian estimation or normal-based shading. This design choice is particularly important for maintaining stable visual results in real-time interactive environments.

Overall, the proposed approach combines the numerical stability of physics-based simulation with the efficiency of Gaussian-based point representations, resulting in a unified framework that preserves real-time performance and visual continuity even under large deformation and topology-changing scenarios.

Contributions. The key contribution of this work lies in *establishing an equivalence between deformable simulation states and rendering primitives*, thereby eliminating the *representation gap between simulation and rendering*, and presenting a real-time unified pipeline that explicitly supports *topology change*. The main contributions are summarized as follows:

- Boundary particle-based surface sampling representation. We define boundary particles as *physically meaningful samples of evolving surfaces*, and integrate them into a PBD-based deformable solver. This representation preserves boundary geometry and surface sampling density *even after cutting or tearing*, directly addressing the limitations of existing mesh/particle hybrid pipelines that rely on fixed-topology assumptions.
- Optimization-free (one-pass) dynamics-to-rendering equivalence. We present a *one-pass particle dynamics-to-rendering pipeline* that maps simulation particle states directly to Gaussian primitives *without reconstruction, remeshing, or learning-based optimization*. This eliminates latency, discontinuities, and implementation complexity associated with traditional surface reconstruction pipelines.
- Solver–renderer co-design supporting topology change. We design a line-based cutting solver extension that explicitly supports dynamic connectivity updates, while ensuring that both *PBD constraint projection stability* and *temporal continuity of Gaussian-based rendering* are maintained within the same pipeline. In other words, topology change is treated not as an exceptional case but as a *native operation of the pipeline*.
- Validation of stability and efficiency in real-time settings. Through a variety of scenarios involving rotation, large deformation, volume contraction/expansion, and topology changes, we demonstrate that the proposed framework maintains temporally stable visual quality *without noise or flickering*, while operating within a performance range suitable for real-time interactive environments.

In summary, this work elevates deformable simulation into a "renderable dynamic representation," fundamentally replacing the conventional paradigm of visualizing simulation results through post-processing. Consequently, the proposed framework provides a practical unified solution that can be directly applied to applications such as games, animation, and immersive content, where *real-time interaction and dynamic deformation or destruction* are simultaneously required.

2. Problem Statement

3D Gaussian Splatting (3DGS) has recently gained significant attention in the fields of static scene representation and view synthesis, demonstrating excellent visual quality and real-time rendering performance. However, most existing 3DGS approaches assume that the positions, orientations, covariances, and colors of Gaussian primitives are *pre-optimized* from a set of input images. Such image-based optimization pipelines inherently assume *static or quasi-static scenes*, which introduces structural limitations when attempting to directly apply them to real-time physics-based simulations involving dynamic deformation and structural changes.

In particular, deformable objects handled in real-time physics engines often undergo not only continuous deformation but also *mesh topology changes*, including cutting, tearing, and fracture. In such scenarios, the geometric structure and surface connectivity may change drastically between frames, which fundamentally conflicts with the assumption of temporal consistency required by image-based 3DGS optimization. Consequently, existing image-driven 3DGS representations struggle to directly reflect real-time simulation results or to stably represent states after topology changes.

Furthermore, Gaussian splats estimated from images tend to be strongly dependent on the observation viewpoint, and directional information as well as covariance estimation often contain noise. This can lead to visual instability and temporal incoherence in real-time interactive environments where viewpoint changes and structural deformations occur frequently. Conversely, approaches that directly convert meshes into Gaussian primitives also face difficulties in fully exploiting the advantages of 3DGS, due to issues

such as mesh connectivity, normal discontinuities, and reconstruction challenges under topology-changing conditions.

These observations reveal a more fundamental problem. While 3DGS was originally designed for *image-based scene reconstruction*, real-time physics engines rely on *particle- or mesh-based dynamic geometry representations*. As a result, the two systems make different assumptions regarding input data structures, temporal update mechanisms, and stability requirements, making their direct integration non-trivial. This discrepancy becomes even more pronounced in deformable simulations involving topology changes.

Motivated by this observation, we argue that a *new intermediate representation* is required to directly connect physics engines and 3D Gaussian splatting. We observe that *Boundary Particles* computed on mesh surfaces can serve as an effective intermediate representation. Boundary particles maintain stable sampling density and geometric information along boundary regions during deformation and topology change, providing physically meaningful references for determining the positions and spatial distributions of Gaussian splats.

Based on this insight, the core research question addressed in this work can be formulated as follows: *Can we stably and consistently transform the results of real-time physics simulations, including topology changes, into Gaussian-primitive-based rendering representations, without relying on image-based inputs or expensive optimization processes?*

To answer this question, we propose a new framework that integrates physics simulation and 3D Gaussian splatting into a unified *one-pass* pipeline through the use of Boundary Particles.

3. Related Work

3.1. Real-Time Deformable Simulation and Constraint-Based Dynamics

Real-time deformable simulation is a core enabling technology for applications where stability and controllability are critical, such as games and interactive authoring tools. Traditionally, mass–spring systems and FEM-based approaches have been widely used, while implicit integration and constraint-based methods have been developed to ensure stability under large time steps [BW98, TPBF87]. Position-Based Dynamics (PBD) is a representative approach that provides strong stability and controllability by projecting positions toward constraint satisfaction rather than directly integrating forces [MHHR06]. Subsequently, XPBD was introduced to alleviate stiffness dependence on the number of solver iterations and the time step size [MMCK16], and Projective Dynamics was proposed to combine constraint projection with an optimization perspective for fast implicit integration [BML*14]. In addition, unified frameworks have been presented to represent various materials such as solids, fluids, and cloth within a single constraint-based particle formulation [MMCK14]. From both practical and research perspectives, tutorials and surveys have also continuously accumulated to summarize the design principles and applications of the PBD family [BMM15].

3.2. Topology Change: Cutting, Tearing, and Fracture

Topology-changing events such as cutting, tearing, and fracture are known to be significantly more challenging than simple continuous deformation, because they involve both changes in connectivity and redefinition of surface boundaries, thereby affecting both simulation and rendering. Classical work introduced brittle fracture simulation that models stress-based crack generation and propagation [OH99], and meshless approaches have also been explored to handle fracture without relying on explicit meshes [PKA*05]. For thin sheets, adaptive fracture techniques have been proposed to achieve high-quality tearing and cracking behavior [PNdJO14]. Furthermore, studies on rigid fracture simulation based on surface meshes have emphasized the difficulty of post-fracture surface reconstruction and the complexity of connectivity management [ZSTB15]. Taken together, these studies show that topology change simultaneously challenges both *representation* and *pipeline design*, especially in real-time settings, where remeshing, reconstruction cost, and temporal artifacts become major bottlenecks.

3.3. Collision, Contact, and Particle-Based Simulation Practices

In real-time deformable simulation, collision, contact, and friction have a significant impact on both visual quality and numerical stability. For example, robust collision and friction handling techniques have been proposed for cloth animation [BFA02], and constraint-based particle frameworks have been reported to be advantageous for handling such contact phenomena in an integrated manner [MMCK14]. In addition, Position-Based Fluids, which models incompressible fluids within the PBD framework, demonstrates the practical effectiveness of particle-based simulation in real-time environments [MM13].

3.4. Point-Based Rendering and Splatting

Point-based rendering has long been studied as a flexible representation that can express and render geometry without requiring explicit mesh connectivity, making it relatively well suited to topology-changing scenarios. Surfels [PZvBG00] and surface splatting [ZPvBG01] established the foundations for effectively filtering and projecting point samples in screen space, while QSplat is another representative technique for rendering large-scale meshes using point-based level-of-detail representations [RL00]. Point Set Surfaces based on moving least squares (MLS) made an important contribution to the geometric justification of point-based representations by defining smooth surfaces from point sets [ABCO*01]. In addition, Pointshop3D presented an editing and processing pipeline for point-based surfaces, demonstrating that point-based representations can be used not only for rendering but also for modeling [ZPKG02]. These developments were later consolidated into a systematic treatment of point-based graphics [GP07].

3.5. Screen-Space Particle Rendering

Screen-space approaches that render particle-based fluids as visually coherent surfaces have become a representative strategy for achieving real-time performance through splatting and filtering. For

Method Category	Representative Work	Primary Input / Assumption	Topology Change	Temporal Stability	Real-Time	Key Limitations
Mesh-based deformable pipelines	PBD/XPBD backbones [MHHR06, MMCK16]	Explicit mesh connectivity; mesh shading	△	△	✓	Topology change requires connectivity updates and often remeshing; sim–render decoupling can cause artifacts near cuts.
Particle screen-space rendering	Screen-space fluids [vdLGS09, TY18]	Particles + view-dependent surface reconstruction	△	△	✓	Effective for dense fluids, but thin shells/tear boundaries are hard to preserve; view-dependent smoothing may blur sharp features.
Point-based graphics / splatting	Surfels / surface splatting [PZvBG00, ZPvBG01]	Pre-sampled oriented points; local splats	△	△	✓	Without simulation-aware resampling, maintaining coherent sampling density under large deformation/tearing is nontrivial.
Neural radiance fields	NeRF and fast variants [MST*20, MESK22]	Multi-view images; per-scene optimization/training	△	✓	×	Training/optimization cost and static-scene assumptions hinder tight integration with real-time physics and topology change.
3D Gaussian Splatting (image-driven)	3DGS [KKLD23] and dynamic variants [WYF*24, LWJ*24]	Multi-view images; optimized Gaussians	△	△	△	Primarily image-driven; orientation/covariance estimation can be noisy, and direct mesh-to-3DGS mapping lacks simulation-consistent sampling under topology change.
Ours	**This work**	**Physics engine state represented using mesh-derived boundary particles**	✓	✓	✓	Training-free sim-to-splat mapping; boundary particles provide coherent splats after cutting/tearing. Remaining limits include simplified appearance (e.g., isotropic Gaussians / position-based shading) and potential oversampling cost near highly detailed boundaries.

Table 1: Comparison of related pipelines for real-time deformable simulation and splatting-based rendering. Symbols ✓/△/× indicate strong/moderate/weak suitability for each criterion.

instance, screen-space fluid rendering based on curvature flow was proposed [vdLGS09], and subsequent work improved filtering to better preserve boundaries and suppress noise [TY18]. This line of research demonstrates that the pipeline of "particles → screen projection → filtering" is a powerful tool in real-time rendering, and it shares conceptual common ground with the one-pass integration direction pursued in this paper.

3.6. Neural Rendering, Radiance Fields, and Gaussian Splatting

NeRF significantly improved novel view synthesis by optimizing a continuous radiance field from multi-view images [MST*20], but its scene-specific training cost and rendering overhead have been widely recognized as major limitations. To address these issues, explicit and grid-based alternatives such as Plenoxels were proposed [FKYT*22], and instant-ngp further accelerated both training and rendering through hash-based encoding [MESK22]. In point-based neural rendering, Neural Point-Based Graphics demonstrated the potential of point-cloud-based view synthesis [ASK*20], and Point-NeRF introduced a point-based radiance field for efficient rendering [XXP*22].

3D Gaussian Splatting (3DGS) is a representative method that models a scene using 3D Gaussian primitives and achieves high-quality real-time rendering through visibility-aware splatting [KKLD23]. However, the dominant paradigm in this family still assumes "multi-view image input + optimization," and even extensions to dynamic scenes often require learning or estimation pipelines [WYF*24, LWJ*24]. Against this background, directly connecting the state of a real-time physics engine, including topology change, to a Gaussian splat representation without image-based optimization remains a largely unresolved challenge.

3.7. Positioning of Our Work

In summary, (1) PBD and other constraint-based simulation methods provide real-time stability and controllability [MHHR06, MMCK16], (2) topology change introduces reconstruction cost and instability at both the representation and pipeline levels [PNdJO14, OH99], (3) point-based rendering and splatting offer flexibility through connectivity-free representations [ZPvBG01, PZvBG00], and (4) 3DGS achieves high-quality real-time rendering but still primarily assumes image-based input and optimization [KKLD23]. The proposed method is distinguished by using boundary particles computed from meshes as a physically meaningful sampling basis for Gaussian splats, thereby directly integrating real-time physics simulation — especially topology-changing simulation — with 3DGS-style rendering at the representation level.

Table 1 summarizes and compares the main characteristics of representative approaches related to real-time deformable simulation and splatting-based rendering. The comparison criteria are centered on factors that are particularly important in real-time physics

engine settings, including the ability to handle topology change, temporal stability under dynamic deformation, suitability for real-time processing, and the degree of integration between simulation and rendering.

Mesh-based deformable pipelines such as PBD and XPBD provide excellent real-time performance and controllability, but they rely on explicit mesh connectivity at the rendering stage. As a result, when topology-changing events such as cutting or tearing occur, connectivity updates or remeshing become necessary, which may lead to visual discontinuities or temporal instability around newly created boundaries. Screen-space particle rendering techniques are effective for visualizing dense particle sets in real time, but due to their view-dependent surface reconstruction characteristics, they often struggle to stably preserve sharp structures such as thin shells or tearing boundaries.

Point-based graphics approaches such as surfels and surface splatting provide flexible representations that are less dependent on connectivity, but when large deformation or topology change is involved, it remains difficult to maintain temporal consistency in sampling density and distribution. Meanwhile, NeRF-style neural rendering methods can achieve high visual quality, but because they rely on scene-specific training and optimization, they are not well suited to serve as direct output representations for real-time physics simulations in which geometric structure changes rapidly. Image-based 3D Gaussian Splatting likewise provides efficient real-time rendering, but the orientations and covariances of Gaussians estimated from multi-view images can become unstable under fast deformation and topology-changing events.

In contrast, the method proposed in this paper directly transforms the simulation state of the physics engine into a Gaussian splat representation using boundary particles computed from the mesh, thereby integrating simulation and rendering within a one-pass pipeline. Boundary particles maintain sampling density and geometric information in boundary regions consistently with the simulation even after deformation and cutting, allowing Gaussian splats to produce stable and temporally coherent visual results even after topology changes. As shown in Table 1, this representation-level integration constitutes the central distinction of our work from existing approaches.

4. Preliminaries

In this section, we summarize the fundamental concepts and background knowledge required to understand the unified framework proposed later. Specifically, we briefly introduce Position-Based Dynamics (PBD), which is widely used for real-time deformable simulation, the concept of Boundary Particles for sampling and representing boundary regions, and 3D Gaussian Splatting, a point-based rendering representation.

4.1. Position-Based Dynamics for Deformable Objects

Position-Based Dynamics (PBD) is a real-time simulation framework that iteratively corrects particle positions so that they satisfy given constraints, rather than directly integrating physical forces [MHHR06]. An object is represented as a set of particles, where each particle has a mass and a position, and the relationships among particles are modeled through various physical constraints.

In a typical PBD simulation loop, predicted positions are first computed under external forces, and then each constraint is iteratively projected to correct particle positions so that the physical constraints are satisfied. This process is numerically stable and has the advantage of maintaining real-time performance even under relatively large time steps [MMCK14, MMCK16].

The distance constraint between two particles p_i and p_j is defined as follows:

$$C_{\text{dist}}(p_i, p_j) = \|p_i - p_j\| - d_0 = 0, \tag{1}$$

where d_0 denotes the initial (rest) distance. During the constraint projection step, the positions of the particles are corrected along the gradient direction of the constraint to satisfy this condition.

For soft objects such as balloons, volume preservation is important in addition to distance constraints. To this end, a global constraint can be added so that the current volume V formed by the entire particle set preserves the initial volume V_0 [MHHR06, MMCK14].

$$C_{\text{vol}} = V - V_0 = 0. \tag{2}$$

Such a volume constraint suppresses excessive expansion or contraction of the entire object, even under local deformation caused by external forces or collisions.

By handling these constraints within the same iterative projection process, PBD can stably and controllably simulate a wide range of deformable objects, and is therefore widely adopted in real-time interactive environments.

4.2. Boundary Particles

Boundary Particles are an auxiliary particle representation introduced to more explicitly represent the boundary regions of an object in particle-based simulation. Since a standard particle distribution treats interior and boundary regions in the same manner, the geometric characteristics and sampling density of boundaries can become unstable under large deformation or topology-changing events.

Attempts to explicitly sample or reinforce boundary regions in particle-based representations can also be found in earlier work on meshless fracture simulation and oriented-particle-based surface modeling [PKA*05, ST92, ABCO*01]. These approaches aim to improve geometric consistency after deformation or separation by maintaining the position, orientation, and density of boundary samples more stably.

Boundary particles are typically generated based on edge or face information of a mesh, and serve to reinforce the sampling density near the boundary. For example, by generating additional particles at regular intervals h along an edge formed by two particles p_i and p_j, the boundary shape can be tracked more robustly even after boundary stretching or separation.

Rather than directly participating in the physical computation of the simulation, Boundary Particles are used as an intermediate representation that provides auxiliary geometric information, such as

boundary positions, normal directions, and sampling density. This property is particularly important in topology-changing scenarios such as cutting or tearing, where new boundaries are created dynamically.

To maintain stable sampling density in boundary regions, the generation interval of boundary particles is defined in proportion to the local geometric scale of the mesh. Let l_{edge} denote the edge length. Then, the target interval h for boundary particles is defined as

$$h = \eta\, l_{\text{edge}}, \tag{3}$$

where η is a user-defined constant that controls the density of boundary samples. With this definition, a consistent boundary representation can be maintained across meshes with different resolutions.

4.3. 3D Gaussian Splatting

3D Gaussian Splatting (3DGS) is a point-based rendering technique that represents a scene as a set of 3D Gaussian primitives and renders them by directly projecting (splatting) them in screen space [KKLD23]. Each Gaussian is defined as a continuous geometric representation that includes position, size, shape, and color information.

A general Gaussian primitive has a mean position μ and a covariance matrix Σ, and its spatial contribution is expressed as follows:

$$G(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^{\top}\Sigma^{-1}(\mathbf{x}-\mu)\right). \tag{4}$$

After being projected into screen space, such Gaussians are accumulated at the pixel level to form the final rendered color.

Recently proposed 3DGS approaches have achieved high visual quality by pre-optimizing the positions, orientations, and covariances of Gaussians from multi-view images [KKLD23, WYF*24, LWJ*24]. These methods have the advantage of enabling continuous surface representation without relying on explicit mesh connectivity.

However, existing 3DGS methods primarily target static or quasi-static scenes and assume image-based input and optimization processes. Therefore, additional considerations are required to use them as a direct output representation for real-time physics simulations in which geometric structure and topology change from frame to frame. Motivated by this observation, this paper proposes a more direct connection between physics simulation and Gaussian-based rendering.

5. Proposed Method

The core idea of our method is to interpret boundary particles as a physically meaningful sampling representation of a surface that evolves during deformation, and to directly and stably transform them into Gaussian primitives without requiring additional reconstruction or optimization processes.

In this section, we present the proposed method, which directly converts the results of mesh-based deformation and cutting simulation together with *boundary particles* into a Gaussian splat representation, thereby integrating simulation and rendering within a *one-pass* pipeline. The key idea is as follows: (1) particle states are updated on the CPU using a PBD-based simulation, and (2) the updated particles (and boundary particles) are immediately treated as Gaussian primitives on the GPU, where the Geometry Shader (GS) constructs their screen-space footprints, and the Pixel Shader (PS) performs elliptical Gaussian masking as well as lighting- and environment-map-based shading.

5.1. Overview

Our system is organized as a one-pass pipeline that performs *simulation state update* and *immediate rendering* sequentially at every frame. At frame t, the input consists of the simulation state from the previous frame (particle positions, velocities, and Gaussian attributes), the time step Δt, external forces (e.g., gravity), a set of constraints (distance constraints and volume constraints), as well as the camera transformation matrices $(\mathbf{M}, \mathbf{V}, \mathbf{P})$ and lighting/environment maps. The outputs are (1) the updated particle state and the corresponding Gaussian rendering parameters (position, scale, rotation, color, and opacity), and (2) the final rendered target color image. In other words, instead of re-estimating Gaussians through image-based optimization, our method directly converts the geometric information produced by physics simulation into Gaussian representations for rendering.

The pipeline is divided into two stages. In Stage A, PBD updates are performed on the CPU. Specifically, predicted positions are first computed by incorporating external forces, after which distance and volume constraints are projected for a fixed number of iterations to stabilize the deformation, and finally velocities and normals are updated. In addition, to account for topology changes such as tearing, boundary particles are updated on a per-frame basis to preserve sampling density near the surface and boundary regions. In Stage B, the updated particles (and boundary particles) are interpreted as Gaussian primitives on the GPU for rendering. The Geometry Shader expands each Gaussian into a screen-space footprint, and the Pixel Shader performs elliptical Gaussian masking and lighting/environment-map-based shading. Because the simulation results are immediately connected to rendering without any additional reconstruction step, the proposed method directly integrates a real-time physics engine with a 3DGS-style representation.

5.2. Deformable Simulation via PBD Constraints

At this stage, we update the physical state of the deformable object on a per-frame basis using the PBD (Position-Based Dynamics) framework. Each particle i is represented by its position $\mathbf{x}_i \in \mathbb{R}^3$, velocity $\mathbf{v}_i \in \mathbb{R}^3$, and mass m_i. A single frame update proceeds in the following order: external force integration, iterative constraint projection, velocity recomputation, normal update, and boundary particle update.

We first compute the predicted velocity and position using the external force $\mathbf{f}_i$ (e.g., gravity $m_i\mathbf{g}$):

$$\mathbf{v}_i^* = \mathbf{v}_i^t + \Delta t\,\frac{\mathbf{f}_i}{m_i}, \qquad \mathbf{x}_i^* = \mathbf{x}_i^t + \Delta t\,\mathbf{v}_i^*. \tag{5}$$

These predicted positions are then used as the input to the subsequent constraint projection step. For real-time stability, we iteratively project the constraints for a fixed number of iterations I ($I = 5$ in our implementation).

The distance constraint is defined to preserve the rest length d_{ij} between adjacent particle pairs (i, j):

$$C_{\text{dist}}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\| - d_{ij} = 0 \tag{6}$$

In addition, to stably maintain balloon-like behavior, a volume constraint is applied so that the total volume preserves the initial volume:

$$C_{\text{vol}} = V - V_0 = 0 \tag{7}$$

At each iteration, the two constraints are sequentially applied as follows:

$$\begin{aligned} \mathbf{x} &\leftarrow \text{ProjectDistanceConstraints}(\mathbf{x}), \\ \mathbf{x} &\leftarrow \text{SolveOverpressureConstraints}(\mathbf{x}). \end{aligned} \tag{8}$$

The volume V is computed from the triangle set $\mathcal{T}$ of the deformed mesh using the divergence theorem:

$$V(\{\mathbf{x}\}) = \frac{1}{6} \sum_{(i,j,k)\in\mathcal{T}} \mathbf{x}_i \cdot \big((\mathbf{x}_j - \mathbf{x}_i) \times (\mathbf{x}_k - \mathbf{x}_i)\big). \tag{9}$$

Here, each triangle contributes the signed volume of the tetrahedron formed with the origin, and the total volume can be stably computed as long as the mesh maintains a consistent orientation. The computed volume is used to evaluate the volume constraint and is reflected in the positional correction during constraint projection.

After all constraints have been applied, the velocity is recomputed using the corrected positions:

$$\mathbf{v}_i^{t+1} = \frac{\mathbf{x}_i^{t+1} - \mathbf{x}_i^t}{\Delta t}. \tag{10}$$

Figure 1 shows an example of balloon deformation simulated using PBD. By repeatedly applying distance and volume constraints, the shape is maintained relatively stably even under large deformation, and balloon-like overpressure behavior is naturally reproduced. This result serves as an example of the basic simulation stage, which later provides input to the rendering and representation method proposed in this work.

To maintain rendering consistency, normals are then recomputed from the deformed shape. These normals are directly used for subsequent lighting and environment-map evaluation.

Finally, boundary particles are updated. Boundary particles preserve the sampling density of the surface and boundary regions, helping prevent boundary shapes from collapsing even after large deformation or cutting. Based on the particle state updated by the simulation, the positions and connectivity information of the boundary particles are updated, thereby allowing the rendering stage to stably maintain visual continuity near the surface.

Figure 2 shows examples of the proposed boundary particle generation and distribution. Each particle is visualized as a disk with a uniform radius, and particles are sampled so as to maintain a minimum inter-particle distance, thereby ensuring uniform density and stable representation in boundary regions. Such a distribution contributes to preserving boundary shapes robustly even under large deformation or topology-changing events.

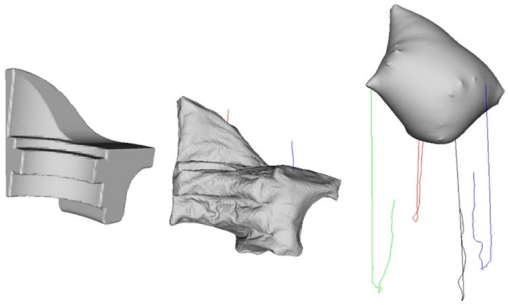


Figure 1: Balloon deformation simulated using position-based dynamics (PBD). The figure shows representative deformation results under constraint-based simulation, including large shape changes and stable volume preservation.

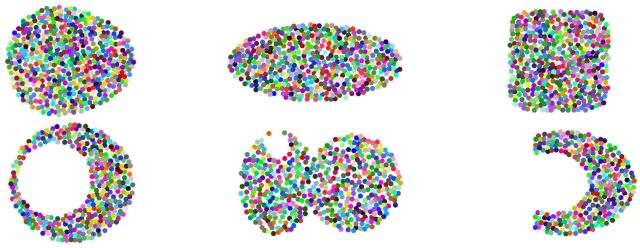


Figure 2: Examples of boundary particle distributions used in our system. Each colored disk represents a boundary particle rendered with a uniform radius. The particles are sampled so that overlap is avoided, providing a stable and uniform boundary representation. Different shapes demonstrate that the sampling strategy remains robust under large geometric variations and topology changes.

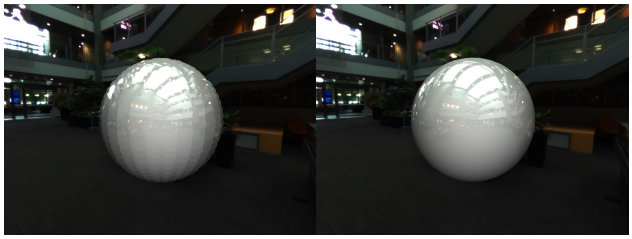
5.3. Gaussian Parameterization from Particles

In this subsection, we describe the particle-to-Gaussian mapping method, which directly interprets particle states obtained from physics simulation as renderable Gaussian primitives. Conventional 3D Gaussian Splatting approaches estimate Gaussian positions, covariances, and colors through an optimization process from multi-view images. In contrast, our method directly constructs Gaussian primitives by exploiting geometric information already available in the simulation. By eliminating the optimization stage, this design enables direct integration with a real-time pipeline.

In the proposed system, each particle (and boundary particle) is mapped one-to-one to a Gaussian primitive, and the Gaussian center is directly determined from the particle position. Furthermore, for stable real-time rendering, we maintain initialized isotropic Gaussians and update only the center and the geometric information required for shading at each frame. This design reduces the numerical instability and computational overhead that may arise when covariances are repeatedly estimated or shape parameters are optimized.

Figure 3 compares the effect of directly using boundary particles as rendering primitives for Gaussian splatting. When rendering is

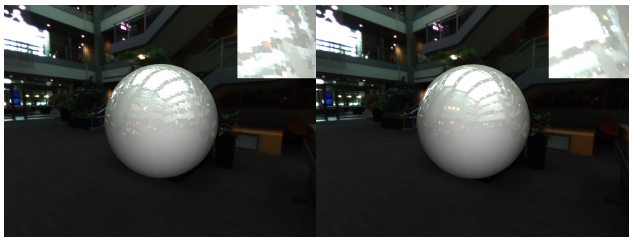
performed using only the initial particle set, sampling near the surface and boundary regions is sparse, resulting in non-uniform Gaussian coverage. Consequently, artifacts such as unstable silhouettes and rough surface appearance can occur. In contrast, when mesh-derived boundary particles are added, the sampling density near the boundary is reinforced, surface coverage is increased, silhouettes and highlights are preserved more stably, and visual noise is reduced. This demonstrates that boundary particles serve effectively as an intermediate representation that directly preserves the geometric boundary of the physics simulation while complementing the surface sampling required for point-based rendering.



(a) (b)

Figure 3: Boundary particles as rendering primitives. (a) Rendering using only the initial particle set yields sparse Gaussian coverage near the surface, causing an unstable silhouette and visible artifacts. (b) Adding mesh-derived boundary particles densifies the surface sampling, improves coverage, and produces a smoother and more stable appearance under identical settings.

Figure 4 compares the visual differences between two environment-map sampling strategies. In the conventional normal-based sampling scheme, the lookup direction tends to remain nearly constant within a single Gaussian, and therefore the entire footprint often appears with a nearly uniform tone. In contrast, with the world-position-based sampling used in our method, the lookup direction varies per pixel, allowing the environmental reflection to change continuously even within a single Gaussian footprint. This produces a more natural and visually rich result. Such a difference is particularly effective for reducing the flat appearance of large splats or regions with low curvature.



(a) (b)

Figure 4: Comparison of environment map sampling strategies. (a) Normal-based sampling results in nearly uniform shading across each Gaussian footprint. (b) Our world-position-based sampling varies the lookup direction per pixel, allowing environment reflections to change smoothly within a single splat, which improves visual richness and reduces flat appearance.

5.3.1. Particle-to-Gaussian Mapping

The Gaussian primitive G_i corresponding to particle i is defined as follows:

$$G_i = \{\mu_i, \mathbf{R}_i, \mathbf{s}_i, \alpha_i, \mathbf{c}_i\} \tag{11}$$

Here, μ_i denotes the Gaussian mean (center), $\mathbf{R}_i$ is the local coordinate rotation, $\mathbf{s}_i$ is the scale corresponding to the standard deviation, α_i is the opacity, and $\mathbf{c}_i$ represents the color or lighting coefficients.

In our method, the Gaussian center is set to be identical to the particle position:

$$\mu_i^{t+1} = \mathbf{x}_i^{t+1}. \tag{12}$$

This definition simplifies the mapping between the simulation state and the rendering representation, and enables a direct correspondence without requiring any additional interpolation or reconstruction process.

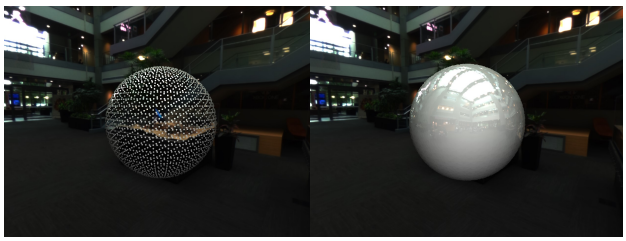
5.3.2. Isotropic Initialization and Update Strategy

The initial Gaussian scale is set in an isotropic form:

$$\mathbf{s}_i = (s_0, s_0, s_0), \qquad \mathbf{R}_i = \mathbf{I}. \tag{13}$$

Such isotropic initialization has the advantage of stably representing the spatial distribution of simulation particles, while avoiding numerical instability that may arise during covariance estimation or update. Moreover, the footprint orientation and size are determined in screen space during the Geometry Shader stage based on camera transformation, and therefore it is unnecessary to maintain anisotropic Gaussians in world space.

During the frame update process, only the Gaussian center is updated according to the particle position, while the base scale and rotation are preserved. This strategy helps improve rendering stability by maintaining temporally consistent splat sizes.



(a) (b)

Figure 5: Isotropic Gaussian initialization and update strategy. (a) Initial particle distribution at the beginning of the simulation. (b) The same particles rendered as isotropic Gaussian primitives. Changing the Gaussian scale affects the screen-space footprint and perceived volume without altering particle positions.

Figure 5 visually illustrates the proposed isotropic Gaussian initialization strategy. In the initial state, simulation particles are distributed along the surface and define its boundary and overall shape. When the same particles are then interpreted as Gaussian primitives, each particle is rendered as an isotropic Gaussian with a uniform scale.

The Gaussian scale parameter directly affects the footprint size in screen space and the visually perceived surface continuity. However, because the particle positions themselves remain unchanged, geometric consistency between the physics simulation result and the rendering representation is preserved.

This initialization strategy enables stable surface representation in real-time environments without requiring complex covariance estimation or optimization procedures.

5.3.3. Boundary Particles as Rendering Primitives

Boundary particles are also converted into Gaussian primitives according to the same rule. They serve to reinforce the sampling density near the surface and boundary regions, and in particular, they alleviate the problem that the Gaussian distribution near the surface may become excessively sparse when topology changes occur.

This plays an important role in preserving the continuity and stability of Gaussian-splat-based surface representation even after cutting or tearing. Furthermore, a uniform Gaussian distribution including boundary particles promotes the generation of uniformly distributed screen-space footprints, thereby reducing visual noise and empty regions.

5.4. Visibility, Sorting, and Blending

Gaussian splat rendering can be regarded as a *semi-transparent accumulation* problem in which each primitive occupies a finite footprint (elliptical support region) in screen space and contributes color and opacity. Accordingly, multiple splats may overlap on the same pixel, and in this case the accumulation order and blending scheme directly affect the final image quality, especially depth cues and boundary sharpness. Since the goal of our one-pass pipeline is to immediately visualize the results of physics simulation as splats, we adopt a real-time-friendly accumulation scheme instead of relying on complex global sorting or multi-pass composition.

Each splat basically has an opacity α_i and contributes an output color $\mathbf{C}_i$. In our implementation, standard alpha blending (Porter–Duff over) is used as the default option, while additive accumulation can also be optionally enabled depending on the scene characteristics (e.g., emissive or particle-like effects). For alpha blending, far-to-near sorting based on the view-space depth z_i can be applied as an optional strategy to reduce color mixing artifacts that may arise from semi-transparent overlap. However, the key point of our method is not the sorting itself, but rather the fact that *simulation particles and boundary particles are maintained sufficiently densely* so that structural continuity can be preserved even without explicit sorting. The sorting and blending policy is treated as an implementation parameter, allowing quality–performance trade-offs to be adjusted as needed.

5.5. Geometry Shader: View-Plane Footprint Reconstruction

The goal of the Geometry Shader (GS) is to expand the Gaussian primitive corresponding to each particle into a rasterizable primitive in screen space. In our implementation, one Gaussian is approximated by one quad (two triangles), and Gaussian masking is later applied in the Pixel Shader (PS) so that only the actual footprint (the elliptical Gaussian region) remains. That is, the GS quad acts as a *carrier geometry* for footprint evaluation.

The key observation is that, when a 3D Gaussian is projected into camera space, its influence region appears as an ellipse in screen space. To approximate this efficiently, we (1) compute the projections of the local axes in view space, (2) select the two axes that contribute most strongly to the view plane (XY), (3) construct an orthogonal right/up basis using 2D Gram–Schmidt, and (4) generate four corners around the center and emit them as a triangle strip. This process is performed in constant time per splat and does not require additional mesh generation or surface reconstruction.

5.5.1. Transform Composition

We first construct the local rotation $\mathbf{R}_{\text{local}}$ from the quaternion, and combine it with the local scale $\mathbf{S}_{\text{local}} = \text{diag}(s_x, s_y, s_z)$:

$$\mathbf{M}_{\text{final}} = \mathbf{S}_{\text{local}}\mathbf{R}_{\text{local}}\mathbf{R}_{\text{model}}\mathbf{R}_{\text{view}}. \tag{14}$$

Here, $\mathbf{R}_{\text{model}}$ denotes the object-space model rotation, and $\mathbf{R}_{\text{view}}$ denotes the view rotation. The row (or column) vectors of $\mathbf{M}_{\text{final}}$ can be interpreted as the axis vectors $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2$ mapped into view space, and these axes determine the orientation and scale of the screen-space footprint.

The center point in view space is obtained as

$$\mathbf{c} = \mathbf{VM}\,[\mathbf{x}_{\text{model}}, 1]^{\mathsf{T}} \tag{15}$$

where $\mathbf{V}$ is the view transform and $\mathbf{M}$ is the model transform.

5.5.2. Dominant Axis Selection on the View Plane

To determine which of the three axes contributes most strongly to the screen-space footprint, we define the view-plane contribution of each axis as follows:

$$c_k = \|\mathbf{a}_{k,xy}\|^2 = \mathbf{a}_{k,xy} \cdot \mathbf{a}_{k,xy}, \qquad k \in \{0,1,2\}. \tag{16}$$

The axis with the smallest c_k is discarded because its projection onto the view plane is weak. The remaining two axes, denoted by $\mathbf{u}$ and $\mathbf{v}$, are then used to construct the footprint. This selection process also helps alleviate cases in which the footprint would otherwise become overly thin or degenerate.

5.5.3. 2D Gram–Schmidt and Quad Construction

Since the selected axes $\mathbf{u}$ and $\mathbf{v}$ are generally not orthogonal, we construct a stable right/up basis using 2D Gram–Schmidt:

$$\mathbf{u}_2 = \mathbf{u}_{xy}, \qquad \mathbf{r}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\| + \varepsilon}, \tag{17}$$

$$\mathbf{v}_2 = \mathbf{v}_{xy}, \qquad \mathbf{v}_\perp = \mathbf{v}_2 - (\mathbf{v}_2 \cdot \mathbf{r}_2)\mathbf{r}_2, \qquad \mathbf{u}_2^{\text{up}} = \frac{\mathbf{v}_\perp}{\|\mathbf{v}_\perp\| + \varepsilon}. \tag{18}$$

In degenerate cases where $\|\mathbf{v}_\perp\| \approx 0$, a vector perpendicular to $\mathbf{r}_2$ is used as a fallback. In addition, to preserve the projected lengths, we store

$$\ell_r = \|\mathbf{u}_2\|, \qquad \ell_u = \|\mathbf{v}_\perp\| \tag{19}$$

and construct the final axis vectors as

$$\mathbf{axis}_{\text{right}} = \ell_r\,(\mathbf{r}_2, 0), \qquad \mathbf{axis}_{\text{up}} = \ell_u\,(\mathbf{u}_2^{\text{up}}, 0) \tag{20}$$

From these, the quad corners around the center **c** are computed and emitted as a triangle strip (4 vertices), while world-space position, normal, color, and opacity are passed to the PS.

5.5.4. Opacity Mapping

The input opacity o_i is mapped to $[0,1]$ using a sigmoid for numerical stability:

$$\alpha_i = \sigma(o_i) = \frac{1}{1+\exp(-o_i)}. \tag{21}$$

This mapping mitigates excessive fluctuations in the blending result, even when the scale of user-defined or simulation-driven parameters changes.

5.6. Pixel Shader: Gaussian Masking, Lighting, and Environment Mapping

For each fragment of the quad generated by the GS, the Pixel Shader (PS) determines whether the fragment lies inside the Gaussian footprint and performs shading only for valid regions. In other words, if the GS provides the *bounding rectangle* of the footprint, the PS retains only the Gaussian elliptical region inside it. As a result, each splat is rendered as a compact circular or elliptical footprint on the screen.

5.6.1. Elliptical Gaussian Footprint Masking

Let the normalized pixel coordinate be $\mathbf{u} = (u,v) \in [-1,1]^2$. We normalize it using the sharpness parameter $\sigma = (\sigma_x, \sigma_y)$ and compute

$$\mathbf{u}' = \left(\frac{u}{\sigma_x}, \frac{v}{\sigma_y}\right), \qquad r^2 = \mathbf{u}' \cdot \mathbf{u}' \tag{22}$$

Then, pixels outside the ellipse are discarded via `clip`:

$$\texttt{clip}(1-r^2) \quad \Rightarrow \quad r^2 > 1 \text{ 인 픽셀 discard.} \tag{23}$$

This masking limits the effective support region of the Gaussian footprint, thereby reducing unnecessary overdraw and alleviating excessive alpha spreading, especially near boundaries.

5.6.2. Environment Mapping and Final Color

For valid pixels, the surrounding illumination is approximated using an environment map. In our implementation, environment-map sampling is performed based on world position rather than surface normal, and the diffuse and specular environment terms are combined as

$$\begin{aligned} \mathbf{E}_d &\leftarrow \text{EnvDiffuse}(\mathbf{p}) \odot \mathbf{k}_d, \\ \mathbf{E}_s &\leftarrow \text{EnvSpecular}(\text{reflect}(-\mathbf{t}, \mathbf{p})) \odot \mathbf{k}_s \, g(\text{shininess}) \end{aligned} \tag{24}$$

Here, $\mathbf{p} = \text{posWorld}$, $\mathbf{t}$ is the view direction, $\odot$ denotes element-wise multiplication, and $g(\cdot)$ is a function that modulates the specular response according to the shininess parameter. The final color is obtained by adding the environment terms and the direct lighting term:

$$\mathbf{C} = \mathbf{E}_d + \mathbf{E}_s + \mathbf{L}_{\text{dir/pt/spot}} \tag{25}$$

Finally, α_i is used in the blending stage for splat accumulation, forming a continuous surface or particle representation composed of many Gaussians.

5.7. Solver Extension: Dynamic Topology Update via Line-Based Cutting

Most deformable simulation frameworks assume a fixed mesh connectivity structure. Under this assumption, topology-changing events such as tearing or cutting require additional remeshing steps or simulation reinitialization, which can undermine numerical stability and temporal continuity in real-time environments.

In this work, we propose a dynamic topology extension that handles connectivity updates directly at the solver level, so that PBD-based constraint projection and Gaussian splatting rendering can continue to operate under a unified data representation even when topology changes occur. The main goal is to preserve the following two invariants:

1. Numerical stability of the constraint projection process
2. Structural consistency of the particle-to-Gaussian mapping

To this end, the cutting operation is defined with respect to a line segment ℓ in 2D screen space, and local connectivity is reconstructed by computing the intersection relationship between that segment and mesh edges.

5.7.1. Edge–Line Intersection Analysis

We first determine whether a mesh edge e_{ij} intersects the cutting line ℓ, and in the case of a proper intersection, we compute the intersection parameter t:

$$\mathbf{x}_{\text{cut}} = (1-t)\mathbf{x}_i + t\mathbf{x}_j, \quad t \in (0,1). \tag{26}$$

This intersection point is inserted as a new vertex, and the triangles containing that edge are locally subdivided. This step is a purely structural operation for preserving geometric consistency, while the dynamic state variables (position and velocity) remain unchanged.

5.7.2. Local Connectivity Reconstruction

An intersected triangle $t \in \mathcal{T}$ is replaced by a new set of triangles:

$$\mathcal{T} \leftarrow (\mathcal{T} \setminus \{t\}) \cup \{t_1, t_2, \dots\}. \tag{27}$$

This process is performed locally and does not require global remeshing. Therefore, its computational complexity is proportional only to the limited region near the cutting line.

5.7.3. Vertex Duplication and Constraint Separation

To allow the two separated regions to deform independently after cutting, shared vertices are duplicated when necessary:

$$\mathbf{x}_i \rightarrow \{\mathbf{x}_i^{(A)}, \mathbf{x}_i^{(B)}\}. \tag{28}$$

Each region is then updated to reference a different constraint set, so that the PBD constraint system is split into two independent connected components. Importantly, this process does not modify the original constraint definitions themselves, but only updates index references. Therefore, the constraint projection algorithm can be preserved without any modification.

Figure 6: Real-time results on a deformable monkey model. The proposed Gaussian splatting framework remains stable under continuous rotation, volume contraction and expansion, and topology-changing events. The method produces temporally coherent rendering without visible noise or flickering, demonstrating stable integration with PBD-based dynamics.

5.7.4. Boundary Particle Resampling

After the connectivity update, boundary particles are resampled to reflect the newly created boundaries. Since boundary particles act as an intermediate representation that preserves surface sampling density for Gaussian splatting, this step is a key element for ensuring visual continuity after topology change. In particular, because the particle-to-Gaussian mapping follows the same rule before and after cutting, the rendering pipeline can immediately resume without any reinitialization.

5.7.5. Computational Behavior

For the topology-changing sphere example shown in Figure 7, we quantitatively analyzed the average computational cost before and after the cutting event. When topology change occurs, the number of particles increases from 18,122 to 19,626, corresponding to an increase of 1,504 particles (+8.30%).

Metric (ms)	Before	After	Δ	Δ%
CPU Frame	27.581	29.835	+2.254	+8.17%
GPU Scene (incl. Gaussian Splatting)	3.672	3.683	+0.012	+0.32%

Table 2: Average computational cost before and after topology change (sphere model in Figure 7).

In terms of frame rate, the performance decreases from 36.257 fps to 33.518 fps, which corresponds to a change of approximately -7.55%. However, this reduction is mainly caused by the increased CPU-side constraint computation, whereas the GPU cost of Gaussian splatting remains almost unchanged.

A particularly notable observation is that even though the number of particles increases by 8.3%, the GPU scene cost rises by only 0.012 ms (+0.32%). This indicates that the proposed particle-to-Gaussian mapping and the Geometry/Pixel Shader-based splatting structure operate in an almost linear and stable manner with respect to particle count.

To analyze the source of the CPU-side overhead in more detail, we further measured the individual update stages.

Update Stage	Before	After	Δ	Δ%
Apply Forces	0.114	0.173	+0.059	+51.46%
Constraint Projection	4.791	6.015	+1.225	+25.57%
Integrate + Normal Update	1.414	1.591	+0.177	+12.51%
Update Particles (incl. Boundary)	4.997	5.383	+0.386	+7.73%

Table 3: Breakdown of CPU update stages (average per frame).

As shown in Table 3, most of the additional CPU cost comes from the Constraint Projection stage (+1.225 ms). This is because the connectivity structure is locally updated after topology change, and the distance and volume constraints must be repeatedly projected for the added boundary particles.

In contrast, the increase in the Update Particles stage remains within a range similar to the particle-count increase (+7.73% versus 8.3%), which indicates that the additional boundary particles are processed in an approximately linear manner. The Integrate and Normal Update stage also exhibits a relatively moderate increase.

Overall, despite the connectivity update and the increase in boundary particles after cutting, the additional computational cost

is largely confined to the CPU-side constraint projection and particle update stages, while the increase in GPU cost for Gaussian splatting rendering remains limited. This suggests that the proposed solver–renderer integration structure retains computational characteristics suitable for real-time interactive environments even under topology-changing conditions.

This also indicates that the proposed method is dominated more by simulation overhead than by rendering overhead, quantitatively showing that topology change does not make the GPU pipeline the performance bottleneck.

5.7.6. Stability Considerations

Unlike conventional deformable simulation methods that assume fixed topology, the proposed solver extension maintains an integrated representation for both dynamics and rendering even when connectivity changes dynamically. In other words, topology change is processed continuously within the same simulation–rendering pipeline, rather than being handled as a separate post-processing step.

This distinguishes our method from conventional approaches, in that it is not merely a mesh cutting algorithm, but rather an *integrated particle dynamics-to-rendering framework* that explicitly supports dynamic connectivity updates.

6. Experimental Results

In this section, we experimentally validate the dynamic stability, rendering continuity, and robustness of the proposed Boundary-Particle–Driven Gaussian Splatting framework under topology-changing conditions. In particular, we focus on analyzing (1) temporal consistency under continuous deformation and rotation, (2) stability under nonlinear behaviors such as volume contraction and expansion, and (3) whether the integration between simulation and rendering is maintained under dynamically changing connectivity caused by cutting and tearing. In addition, we measure both the CPU cost of constraint projection and the GPU cost of Gaussian-splatting-based rendering, thereby quantitatively evaluating the applicability of the method to real-time interactive environments.

Figure 6 shows the real-time experimental results obtained using the monkey model. The first and second images in the top row represent the case where the model is continuously rotating, demonstrating that the proposed Gaussian splatting rendering framework maintains temporally stable results regardless of camera motion or object transformation. In particular, even though a Gaussian-primitive-based representation is used, a consistent surface appearance is preserved without flickering or visible noise.

Moreover, even during dynamic deformation in which the model contracts and then expands again in a balloon-like manner, surface continuity and smooth shading remain stable. This is because the proposed method is directly integrated with PBD-based dynamics, so that the particle states are immediately reflected in the Gaussian representation.

The results in the bottom row include a scene in which topology change occurs, and show that even after cutting and connectivity changes, the rendering pipeline continues without interruption and produces continuous results in real time. In particular, the boundary-particle-based representation maintains sufficient sampling density even in newly created boundary regions, thereby minimizing visual discontinuities and artifacts.

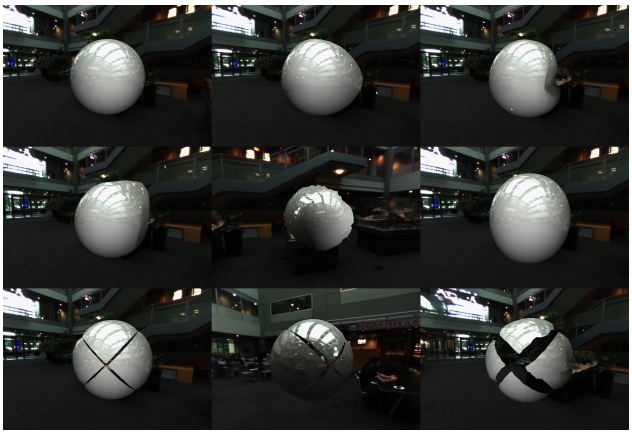


Figure 7: Real-time results on a deformable sphere model.

These results demonstrate that the proposed method constitutes a one-pass simulation-to-rendering framework efficiently integrated with physics-based dynamics, and suggest that it can be applied to a wide range of applications requiring real-time interaction, including games, animation, and immersive interactive content.

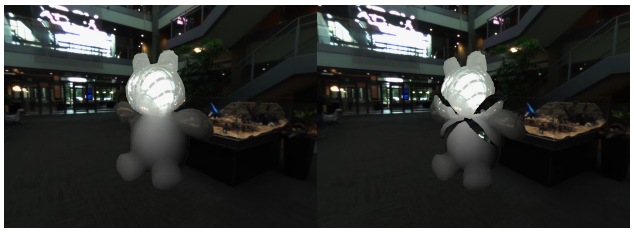
Figure 7 shows real-time results on a spherical model under user interaction. When the user drags a specific region with the mouse, deformation is immediately propagated around the selected area, and the deformed result is simultaneously visualized in real time through Gaussian-splatting-based rendering. This is possible because the simulation stage and the rendering stage share the same particle-based representation.

In particular, because the spherical model has an initially uniform curvature, it is possible to clearly observe how a localized input propagates across the entire surface distribution. The proposed method shows that the deformation generated around the input region expands continuously, while stably preserving both surface sampling density and the outer silhouette. Since the Gaussian primitives are updated immediately according to the deformation, continuous visual feedback is provided without requiring any additional surface reconstruction.

The bottom-row results also include the case in which tearing occurs in the same model. Even after the connectivity structure is dynamically changed, the newly formed boundary regions are immediately reflected in the rendering, and natural results are produced without sampling-density loss or visual gaps around the cutting path. This is because the boundary-particle-based representation supports a uniform surface representation even when dynamic boundaries are created.

This experiment demonstrates that the proposed framework is not limited to the visualization of static scenes, but can simultaneously achieve immediate deformation response and consistent rendering even in environments with continuous user interaction.

Figure 8 shows the results of topology-change experiments con-



(a) Teddy Bear

(b) Stanford Bunny

Figure 8: Topology-changing results on complex deformable models. The proposed framework handles cutting and tearing events while maintaining stable particle dynamics and continuous Gaussian-based rendering across diverse geometric structures.

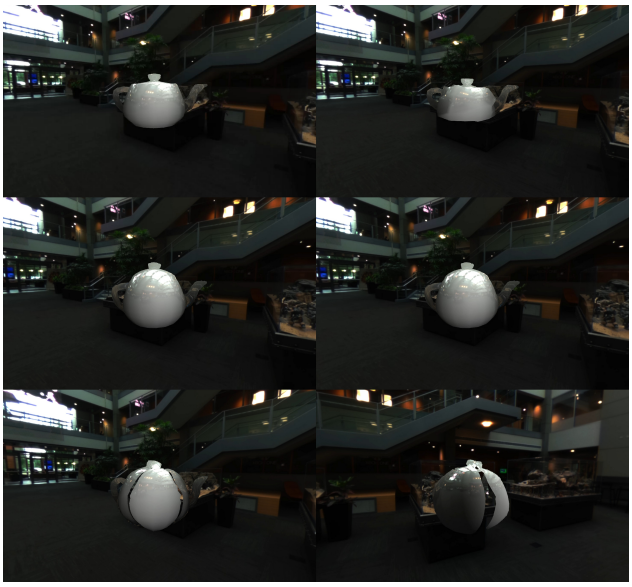


Figure 9: Real-time results on a deformable teapot model. The sequence demonstrates volume contraction and expansion, user-driven interaction, and topology-changing events. Boundary particles are dynamically updated according to surface area variation, enabling stable and continuous Gaussian splatting rendering without visual artifacts or temporal discontinuities.

ducted on the Teddy Bear and Stanford Bunny models. The two models have different geometric structures and curvature distributions, and include more complex surface details than spherical or otherwise simple models.

Even when cutting and tearing occur, the proposed framework stably maintains particle-based dynamics while dynamically updating the mesh connectivity structure. In particular, even in regions with large curvature variation, such as ears, limbs, and protruding parts, the Gaussian primitives are immediately updated, so that continuous surface representation is preserved without visual discontinuities or gaps.

In the Teddy Bear model, the cutting path crosses multiple curved surface regions and locally creates new boundaries. However, through boundary particle resampling, uniform sampling density is maintained even in the newly formed boundary regions. In the Stanford Bunny model as well, even with irregular geometry and fine structural details, the separated regions deform independently after cutting, and the rendering stage reflects these changes immediately without requiring additional reconstruction.

These results show that the proposed method is not limited to a specific shape, but can consistently handle topology change for a wide variety of three-dimensional models with different levels of complexity. In other words, the proposed framework is applicable not only to simple test models, but also to complex assets used in practical applications.

Figure 9 shows the results of applying the proposed framework to a teapot model. This experiment includes not only simple shape deformation, but also volume contraction and expansion, user interaction, and topology-changing events including tearing.

The top and middle rows show a balloon-like behavior in which the object volume first decreases and then increases again. During this process, as the surface area changes dynamically, boundary particles are resampled and redistributed, and the spatial distribution of the Gaussian primitives is also updated automatically. As a result, surface coverage is preserved, and neither silhouette collapse nor non-uniform Gaussian footprints are observed during contraction and expansion.

The bottom row presents scenes including user-driven interaction and surface tearing events. Even after cutting, the connectivity structure is dynamically updated, and boundary particles are stably regenerated along the newly formed boundaries. In particular, because the dynamics stage and the Gaussian splatting rendering stage share the same particle representation, continuous results are maintained without temporal mismatch between simulation and rendering or abrupt visual popping artifacts.

This experiment shows that the proposed unified pipeline operates stably in real time even under complex scenarios that simultaneously include (1) large deformation and surface-area variation, (2) user interaction, and (3) topology change. This suggests that the boundary-particle-based representation serves not merely as a static surface enhancement, but as a mechanism for dynamic surface reconstruction.

We further analyzed the frame-wise average computational cost for the topology-change experiment on the teapot model shown in Figure 9. After cutting, the number of particles increased from 10,350 to 10,894, corresponding to an increase of approximately 5.26%. Unlike the sphere model, this scene contains a non-uniform curvature structure, including the handle and the spout, so that connectivity updates and boundary reconstruction are performed under more complex conditions.

The average frame rate before and after topology change was

Metric (ms)	Before	After	Δ	Δ%
CPU Frame	16.374	18.386	+2.012	+12.29%
GPU Scene (incl. GS)	2.137	2.316	+0.180	+8.41%
CPU Update Breakdown				
Apply	0.076	0.113	+0.037	+49.53%
Constraints	2.892	4.245	+1.353	+46.79%
Integrate + Normal	0.809	0.986	+0.177	+21.82%
UpdateParticles	2.866	3.152	+0.286	+9.97%

Table 4: Integrated performance analysis before and after topology change on the teapot model.

measured as 61.073 FPS and 54.389 FPS, respectively, corresponding to a decrease of approximately 10.95%. Nevertheless, the system still maintains real-time performance above 50 FPS.

The increase in CPU frame time mainly comes from the constraint projection stage (+46.79%). This is because after cutting, the constraint relationships between separated regions must be reconstructed, requiring additional constraint computation. The Apply stage and the Integrate+Normal stage also increase, but the increase in the UpdateParticles stage is relatively limited (+9.97%). This is because boundary particle resampling is performed only for local surface changes.

An interesting observation is that, despite the increase in particle count, the GPU scene time increases by only 0.180 ms. The Gaussian splatting stage is not directly coupled to dynamic connectivity updates, and its time complexity increases relatively mildly with respect to particle count. This suggests that the rendering cost is governed primarily by the screen-space splat workload rather than by the structural complexity of topology change itself.

Compared with the sphere-model experiment, the teapot experiment includes more complex geometry and a more non-uniform curvature distribution, yet exhibits a similar level of performance stability. This indicates that the proposed boundary-particle-based extension is not limited to simple symmetric structures, but can also be generalized to more typical mesh configurations.

Figure 10 shows topology-changing scenes involving cutting and tearing for various mesh shapes under a modified environment map. The experiments use (a) Fandisk, (b) Sphere, (c) Stanford Bunny, and (d) Teddy Bear models, and each row presents the deformation and fracture process over time as cutting progresses.

As can be seen from the results, the proposed boundary-particle-based Gaussian splatting framework can stably represent a wide range of tearing behaviors, from initial rupture starting as thin tearing to complex fracture structures that expand in multiple directions. In particular, even under large deformation, the method can reconstruct continuous shape changes and topology changes with visually consistent results using only the particle-based representation, without requiring any intermediate surface reconstruction process.

Furthermore, even under changed lighting conditions induced by the modified environment map, the Gaussian-splatting-based rendering stably reflects the dynamically evolving surface structure, allowing complex geometric changes and tearing patterns to be expressed in a visually natural manner. These results demonstrate that the proposed method operates effectively not only for simple cutting, but also in complex physical scenarios where large deformation and topology change occur simultaneously.

7. Discussion

7.1. Comparison with Original 3D Gaussian Splatting

Figure 11 compares the results of the original 3D Gaussian Splatting (3DGS) with those of the proposed Boundary-Particle–Driven Gaussian Splatting. The left image shows a conventional optimization-based 3DGS result, while the right image shows the result of our method under the same scene conditions.

Even though the original 3DGS uses a large number of Gaussian primitives, color instability near boundaries and structural blur are still observed. This is because the scene is represented from the perspective of radiance-field optimization, which primarily focuses on minimizing pixel-wise reconstruction error rather than explicitly preserving dynamic deformation or geometric structure.

In contrast, the proposed method maintains the surface as a physically meaningful set of samples through boundary particles, and directly maps them to Gaussian primitives, thereby preserving the geometric structure and surface silhouette more explicitly. In particular, despite using far fewer Gaussians than the original 3DGS, our method still provides a sharper and more coherent surface representation, demonstrating an advantage in terms of representational efficiency.

These results show that the proposed method does not rely on learning-based radiance-field optimization, but instead directly utilizes particle-based dynamic states as a rendering representation, thereby achieving structural stability and temporal consistency. This suggests that our method can complement the limitations of conventional 3DGS approaches, especially in environments involving dynamic deformation and topology change.

7.2. Practical Considerations: Environment Map Sampling

Figure 12 compares the visual differences arising from different environment-map sampling strategies. The left image shows a case where the diffuse term is computed based on surface normals while only the specular term is computed in a position-based manner, whereas the right image shows the result obtained when both the diffuse and specular terms are computed in a position-based manner.

Normal-based diffuse shading depends on surface orientation information, and therefore tends to preserve depth cues and volumetric shape perception more stably. However, because similar colors are grouped at the level of individual Gaussian primitives, “chunky” particle-wise artifacts may become visible, especially when the number of particles is small or the Gaussian size is large.

In contrast, position-based sampling can sample different environmental lighting values for each pixel even within the same Gaussian footprint, so color variation appears more naturally interpolated. However, in this case, the shading information based

(a) Fandisk

(b) Sphere

(c) Stanford Bunny

(d) Teddy Bear

Figure 10: Examples of large deformation and topology-changing tearing under a modified environment map. Each row demonstrates progressive cutting events on different models. The proposed boundary-particle–driven Gaussian splatting framework robustly captures a wide range of tearing behaviors, from thin crack initiation to complex fracture patterns, while maintaining stable visual reconstruction during severe deformation. The results demonstrate that the method can handle both simple and highly complex topology changes without requiring intermediate surface reconstruction.

on surface orientation becomes weaker, which may slightly reduce shape cues.

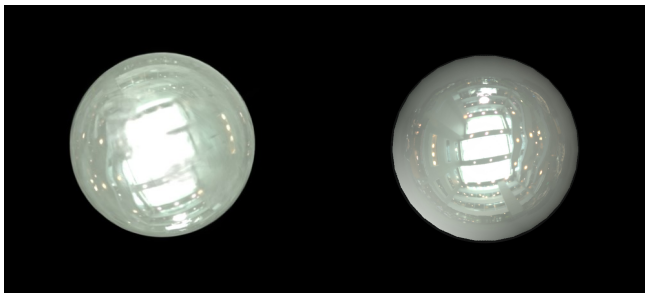
This comparison is not a core contribution of the proposed unified framework itself, but rather illustrates a practical implementation trade-off that arises when choosing an environment-lighting strategy for Gaussian-based representations. Experimentally, we found that a hybrid use of the two approaches is advantageous for maintaining a balance between shape preservation and color stability.

7.3. Opacity Sensitivity and Rendering Artifacts

Figure 13 shows the visual changes caused by different alpha (opacity) settings for Gaussian primitives. In our implementation, the per-particle alpha value is controlled through a sigmoid function, which is an important factor in determining surface transparency and coverage during Gaussian splatting.

The experimental results show that when the alpha value is sufficiently high, Gaussian footprints overlap naturally, so that a uniform surface appearance is maintained. However, when the alpha value decreases below a certain threshold (empirically around 88% or lower), the kernel shape of individual Gaussians begins to become visible on the screen, and the boundaries between particles become perceptible. This may lead to artifacts in which the surface appears not as a smooth continuum, but as a collection of Gaussian particles.

This phenomenon is not a core algorithmic contribution of the



(a) (b)

Figure 11: Comparison with the original 3D Gaussian Splatting (3DGS). (a) shows a baseline 3DGS reconstruction trained using a standard optimization pipeline, while (b) shows our boundary-particle–driven Gaussian splatting result. Despite using significantly fewer Gaussian primitives, our method produces a more stable and coherent surface representation. The baseline exhibits color instability and blurred structural details, whereas the proposed approach maintains sharper geometry and consistent appearance.

(a) (b)

Figure 12: Comparison of environment map sampling strategies. (a) Diffuse shading computed from surface normals while specular reflection is position-based. (b): Both diffuse and specular components computed in a position-based manner. While the position-based approach provides smoother color variation within each Gaussian footprint, the normal-based diffuse term preserves stronger shape cues. This comparison illustrates practical implementation trade-offs rather than a core algorithmic contribution.

proposed unified framework, but rather stems from an intrinsic characteristic of Gaussian-based representations. That is, if sufficient overlap and appropriate opacity settings are not maintained, the kernel structure inherent to point-based representations may become exposed.

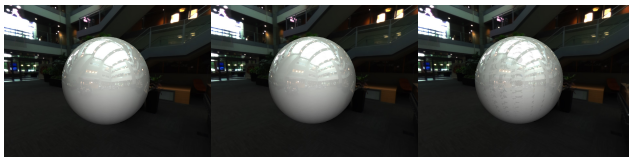
Therefore, in practical applications, it is important to jointly consider the balance among particle density, Gaussian size, and alpha value. This experiment can be viewed as a qualitative example showing how the opacity parameter affects visual stability in Gaussian-splatting-based rendering.

7.4. Effect of Boundary Sampling and Shading Strategy

Figure 14 compares the influence of boundary particle generation strategies and environment-mapping-based shading on Gaussian-splatting-based surface representation.

7.4.1. Vertex-Only Boundary Sampling

In (a), boundary particles are generated only at mesh vertex positions, resulting in a surface representation composed of 528 splats.



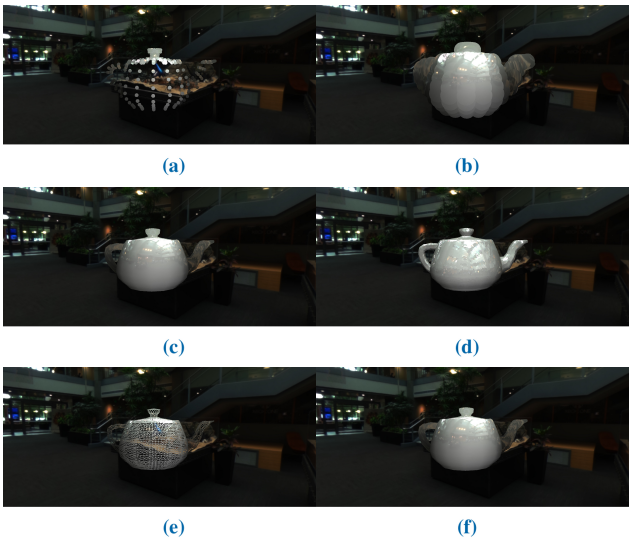
(a) 99.91% **(b)** 99% **(c)** 88%

Figure 13: Effect of particle alpha (opacity) on Gaussian splatting appearance. As the sigmoid-based alpha value decreases, individual Gaussian footprints become increasingly visible, leading to noticeable particle boundaries and non-uniform surface appearance. This illustrates a practical sensitivity of opacity control in Gaussian-based rendering rather than a core contribution of our framework.

Even though the screen-space footprints are expanded through the Geometry Shader (GS), the number of particles is insufficient, leading to non-uniform surface coverage, and the silhouette and curved surface cannot be properly reconstructed. To compensate for this, the particle size may be increased, which can make the result appear visually continuous as in (b). However, this also causes distortion of the original geometric shape. In other words, simply enlarging the splat size is not a fundamental solution from the viewpoint of shape preservation.

7.4.2. Normal-Based Environment Mapping

In (c), normal-based environment mapping is applied. In this case, because the lookup direction remains nearly constant within the same Gaussian footprint, the overall surface color tends to appear uniform or accompanied by noticeable fine-grained noise. In particular, in regions with large curvature variation, highlights may appear broken or visually rough. This is caused by the structural



(a) (b)

(c) (d)

(e) (f)

Figure 14: Comparison of boundary particle sampling and environment mapping strategies. (a)–(b) Vertex-only boundary particles (528 splats). (c) Normal-based environment sampling. (d)–(f) Proposed boundary sampling and world-position-based shading.

limitation of normal-based sampling, where shading is effectively fixed at the splat level.

7.4.3. Proposed Sampling and Shading Strategy

Figures (d)–(f) show the results of the proposed method. Our approach uniformly resamples mesh-based boundary particles around the surface to ensure sufficient surface coverage, and performs environment mapping based on world position. As a result, the lookup direction varies per pixel even within the same Gaussian footprint, allowing environmental reflections to appear continuously. Figure (d) demonstrates the effect of the improved shading strategy, while (e), which uses 10,350 splats, shows that when the particle density is sufficient, Geometry-Shader-based splatting can reproduce a smooth and stable surface. Moreover, even when the particle size is increased, the representation remains natural without geometric distortion.

Overall, simple vertex-based sampling or normal-based shading has difficulty simultaneously satisfying shape preservation and visual continuity in Gaussian-splatting environments. In contrast, the proposed boundary-particle resampling and world-position-based environment mapping strategy provide stable surface quality while preserving the advantages of particle-based representations.

7.5. Limitations

Although this work proposes a framework that integrates deformable simulation and Gaussian-based rendering at the representation level, several limitations remain.

7.5.1. Isotropic Gaussian Assumption

In the current implementation, we use isotropic Gaussians for stability and computational efficiency. This is advantageous for ensuring temporal consistency and numerical stability under dynamic deformation and topology change. However, in regions with strong directional anisotropy or fine curvature variation, the geometric fidelity may be more limited than that of anisotropic Gaussian representations. Future work should therefore explore anisotropic parameterizations that remain stable even in dynamic scenarios.

7.5.2. Dependence on Particle Sampling Density

The rendering quality of the proposed method is directly affected by the sampling density of particles and boundary particles. If the particle spacing is not sufficiently small or the Gaussian radius becomes excessively large, particle-like artifacts may appear, with individual Gaussian footprints becoming visually noticeable. This is one of the practical issues in our implementation, and calls for appropriate radius control and alpha-mapping strategies.

7.5.3. Scalability under Extensive Topology Changes

The proposed line-based cutting scheme operates stably in real-time settings. However, if highly complex multiple cuts or repeated topology changes occur on high-resolution meshes, the cost of connectivity reconstruction and boundary resampling may increase. Although the current implementation was shown to operate stably within the range of real-time interactive applications, additional acceleration strategies may be required for large-scale destruction simulations or scenarios involving fine fracture patterns.

7.5.4. Simplified Illumination Model

The position-dependent appearance computation based on environment maps provides temporal stability and implementation simplicity, but it does not explicitly account for complex global illumination effects or inter-reflections. Therefore, compared with physically rigorous ray-tracing-based rendering, it remains limited in terms of lighting accuracy.

7.5.5. Absence of Learning-Based Parameter Optimization

This work adopts a deterministic Gaussian generation strategy that excludes optimization or learning processes. This provides the advantage of real-time performance and predictability, but follows a different design philosophy from data-driven methods that can recover super-resolved details or material characteristics. A hybrid integration with learning-based Gaussian representations remains an interesting direction for future research.

Nevertheless, this work demonstrates that physics-based deformable simulation including dynamic topology change can be converted into an immediate renderable representation without reconstruction or optimization, thereby illustrating the practical potential of a real-time integrated pipeline.

8. Conclusion

In this work, we proposed a new real-time framework that directly integrates physics-based deformable simulation and Gaussian-based rendering at the representation level. Built upon PBD-based particle dynamics, the proposed method introduces boundary particles to maintain the evolving surface as a physically meaningful set of samples, and directly maps them to Gaussian primitives without requiring additional reconstruction or optimization. This design enables a one-pass simulation-to-rendering pipeline.

In particular, the proposed framework does not assume fixed topology. Instead, through a solver extension that supports line-based cutting, it incorporates topology change as a native operation of the pipeline. As a result, we showed that the stability of PBD constraint projection and the temporal continuity of Gaussian-based rendering can be preserved simultaneously even after cutting and tearing events. Experimental results demonstrated that the proposed method produces stable visual results without noticeable noise or flickering across a variety of scenarios, including rotation, large deformation, volume variation, and topology change.

The central significance of this work lies in going beyond the conventional decoupled paradigm in which simulation results are visualized only through post-processing. By establishing an equivalence between particle states themselves and rendering primitives, our framework removes the structural mismatch between simulation and rendering. Such an integrated design is directly applicable to real-time interactive applications, including games, animation, and virtual or augmented reality content.

As future work, it would be valuable to explore extensions toward anisotropic Gaussian representations, improved scalability for large-scale destruction simulation, and hybrid integration with learning-based Gaussian representations. Through these directions, we expect the framework to evolve into a more generalized real-time representation for physics-based dynamic scenes.

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *Proceedings of IEEE Visualization* (2001). 3, 5

[ASK*20] ALIEV K.-A., SEVASTOPOLSKY A., KOLOS M., ULYANOV D., LEMPITSKY V.: Neural point-based graphics. In *Proceedings of ECCV* (2020). 4

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics 21*, 3 (2002). 3

[BML*14] BOUAZIZ S., MARTIN S., LIU T., KAVAN L., PAULY M.: Projective dynamics: Fusing constraint projections for fast simulation. *ACM Transactions on Graphics 33*, 4 (2014). 3

[BMM15] BENDER J., MACKLIN M., MÜLLER M.: Position-based simulation methods in computer graphics. Eurographics Tutorial Notes, 2015. 3

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Proceedings of SIGGRAPH* (1998). 3

[FKYT*22] FRIDOVICH-KEIL S., YU A., TANCIK M., CHEN Q., RECHT B., KANAZAWA A.: Plenoxels: Radiance fields without neural networks. In *Proceedings of CVPR* (2022). 4

[GP07] GROSS M., PFISTER H.: *Point-Based Graphics*. Morgan Kaufmann / Elsevier, 2007. eds. 3

[KKLD23] KERBL B., KOPANAS G., LEIMKÜHLER T., DRETTAKIS G.: 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics 42*, 4 (2023). 4, 6

[LWJ*24] LEE J., WON C.-Y., JUNG H., BAE I., JEON H.-G.: Fully explicit dynamic gaussian splatting. In *Proceedings of NeurIPS* (2024). Also available as arXiv:2410.15629. 4, 6

[MESK22] MÜLLER T., EVANS A., SCHIED C., KELLER A.: Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics 41*, 4 (2022). 4

[MHHR06] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. In *Proceedings of VRIPHYS (Eurographics)* (2006). 3, 4, 5

[MM13] MACKLIN M., MÜLLER M.: Position based fluids. *ACM Transactions on Graphics 32*, 4 (2013). 3

[MMCK14] MACKLIN M., MÜLLER M., CHENTANEZ N., KIM T.-Y.: Unified particle physics for real-time applications. *ACM Transactions on Graphics 33*, 4 (2014). 3, 5

[MMCK16] MACKLIN M., MÜLLER M., CHENTANEZ N., KIM T.-Y.: Xpbd: Position-based simulation of compliant constrained dynamics. In *Proceedings of MIG* (2016). 3, 4, 5

[MST*20] MILDENHALL B., SRINIVASAN P. P., TANCIK M., BARRON J. T., RAMAMOORTHI R., NG R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of ECCV* (2020). 4

[OH99] O'BRIEN J. F., HODGINS J. K.: Graphical modeling and animation of brittle fracture. In *Proceedings of SIGGRAPH* (1999). 3, 4

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRÉ P., GROSS M., GUIBAS L. J.: Meshless animation of fracturing solids. *ACM Transactions on Graphics 24*, 3 (2005). 3, 5

[PNdJO14] PFAFF T., NARAIN R., DE JOYA J. M., O'BRIEN J. F.: Adaptive tearing and cracking of thin sheets. *ACM Transactions on Graphics 33*, 4 (2014). 3, 4

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proceedings of SIGGRAPH* (2000). 3, 4

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of SIGGRAPH* (2000). 3

[ST92] SZELISKI R., TONNESEN D.: Surface modeling with oriented particle systems. In *Proceedings of SIGGRAPH* (1992). 5

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. *Computer Graphics (Proceedings of SIGGRAPH) 21*, 4 (1987). 3

[TY18] TRUONG N., YUKSEL C.: A narrow-range filter for screen-space fluid rendering. *Proc. ACM Comput. Graph. Interact. Tech. 1*, 1 (2018). 4

[vdLGS09] VAN DER LAAN W. J., GREEN S., SAINZ M.: Screen space fluid rendering with curvature flow. In *Proceedings of I3D* (2009). 4

[WYF*24] WU G., YI T., FANG J., XIE L., ZHANG X., WEI W., LIU W., TIAN Q., WANG X.: 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of CVPR* (2024). 4, 6

[XXP*22] XU Q., XU Z., PHILIP J., BI S., SHU Z., SUNKAVALLI K., NEUMANN U.: Point-nerf: Point-based neural radiance fields. In *Proceedings of CVPR* (2022). 4

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3d: An interactive system for point-based surface editing. *ACM Transactions on Graphics 21*, 3 (2002). 3

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings of SIGGRAPH* (2001). 3, 4

[ZSTB15] ZHU Y., SIFAKIS E., TERAN J., BRIDSON R.: Simulating rigid body fracture with surface meshes. *ACM Transactions on Graphics 34*, 4 (2015). 3