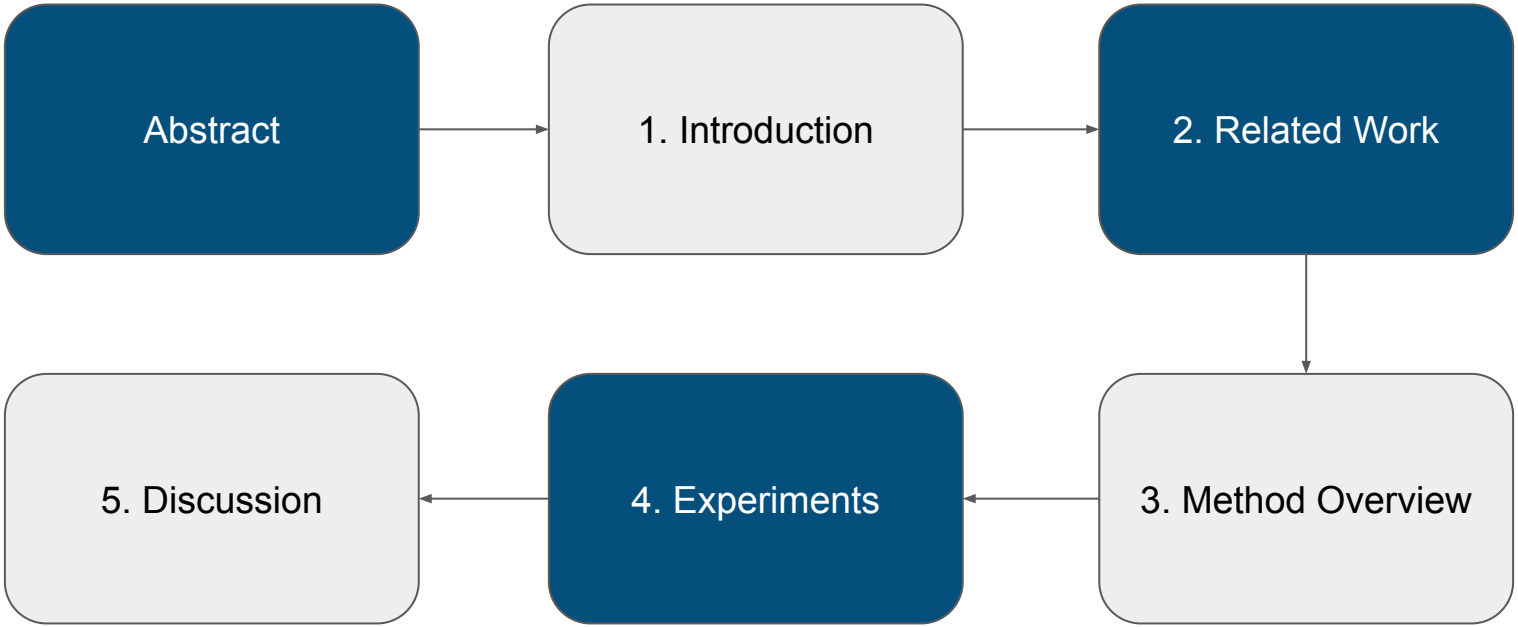
PhysGaussian: Physics-Integrated 3D Gaussians for Generative Dynamics

Mohara Nascimento - Reviewer
Leonardo Mendonça - Archaeologist
Marcelo de Sousa - Hacker
Marcelo de Sousa - PhD Student

PhysGaussian

Reviewer - Mohara Nascimento

Paper Structure



Motivation and key idea

Integrates a physics-based dynamic model within a Gaussian Splatting representation, unifying simulation with scene reconstruction.

3D Gaussian Splatting

+

Gaussian Evolution

+

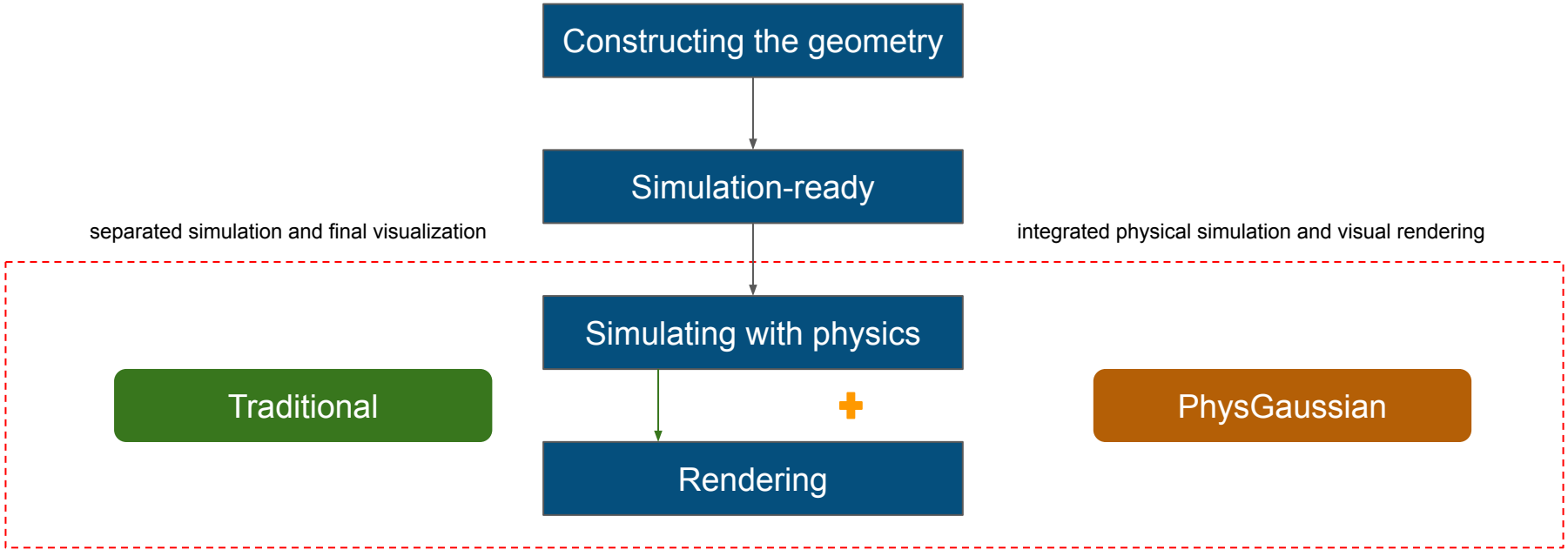
Continuum Mechanics + t

Kinematics Dynamics

Motivation and key idea

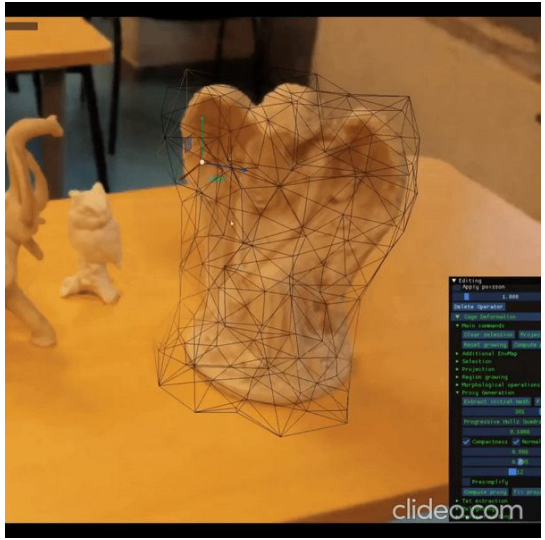
"what you see is what you simulate" (WS²)

Physics-based visual content generation pipeline:



Motivation and key idea

Traditional



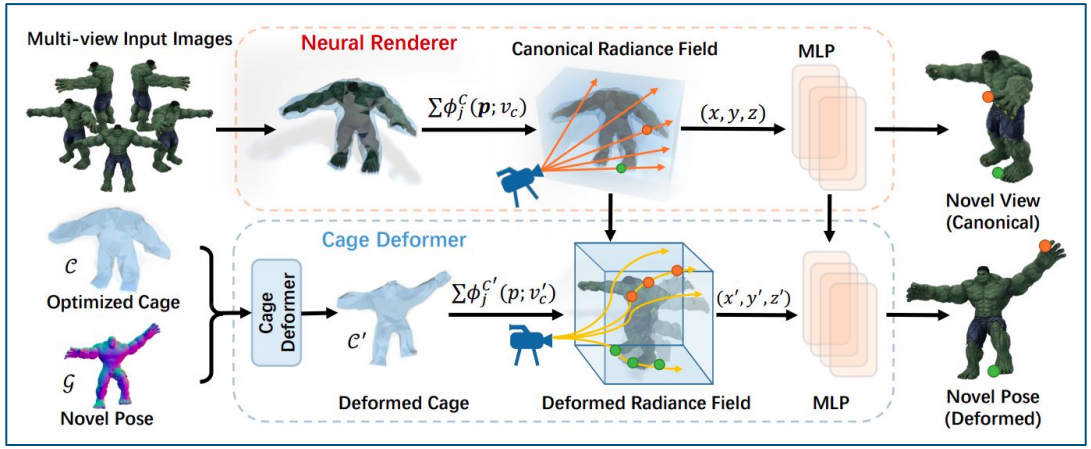
- mesh or geometry embedding;
- manual movement editing.

PhysGaussian

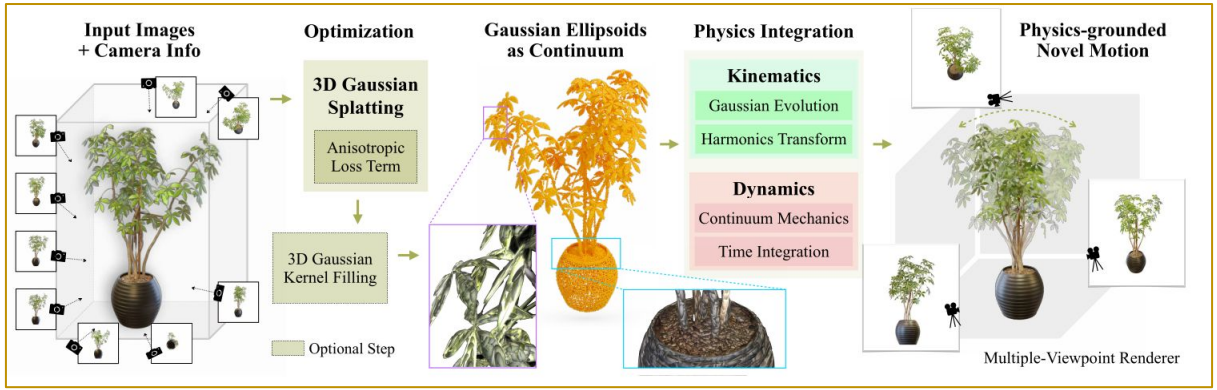
- geometry as 3D Gaussians;
- integrated physical behavior with a Material Point Method (MPM).

Motivation and key idea

CageNeRF



PhysGaussian



Cagenerf: Cage-based neural radiance field for generalized 3d deformation and animation. NeuIPS, 2022.

Physics Method Overview

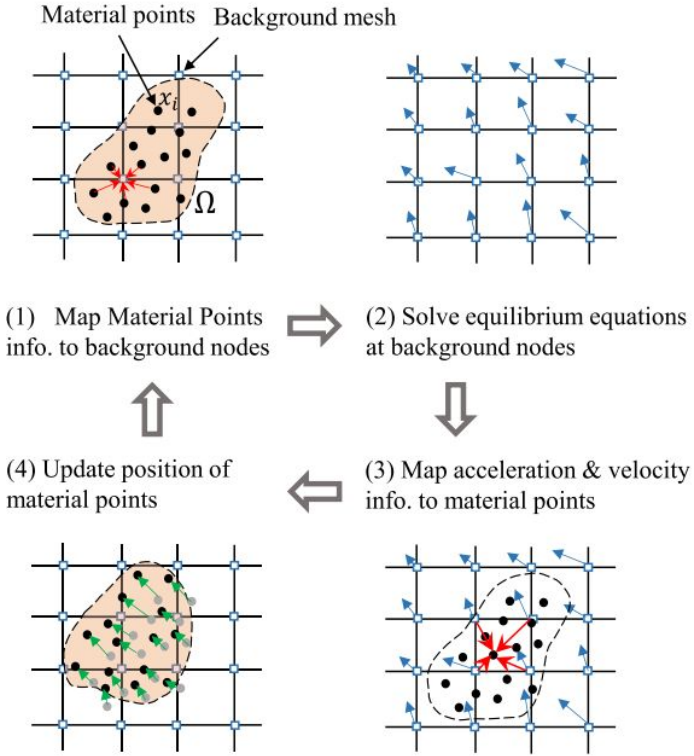
Material Point Method

Summary

A hybrid Lagrangian-Eulerian approach

That represents the continuum as a collection of material points with a background mesh.

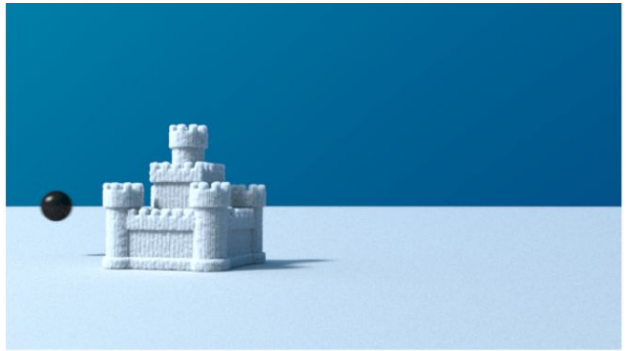
- the material points (Lagrangian particles) carry mass, volume, velocity and movements information;
- the background mesh (Eulerian grid) is used to solve the momentum equations;



Material point method for large-deformation modeling of coseismic landslide and liquefaction-induced dam failure. Soil Dynamics and Earthquake Engineering. (2021)

Physics Method Overview

Material Point Method

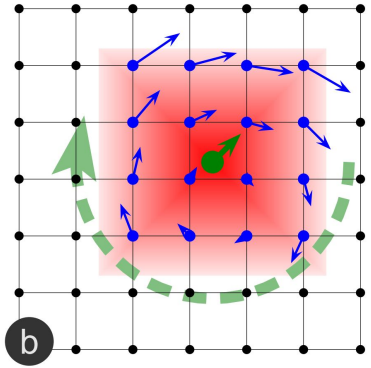


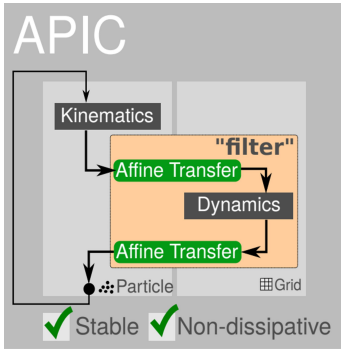
1. Rasterize particle data to the grid
2. Compute particle volume and densities
3. Compute grid forces
4. Update velocities on grid
5. Grid-based body collisions
6. Solve linear system to velocity
7. Update deformation gradient
8. Update particle velocities
9. Particle-based body collisions
10. Update particle positions

A material point method for snow simulation. TOG, 2013.

Physics Method Overview

Material Point Method





m_p	mass	particle
v_i^n	velocity	grid node
w_{ip}^n	weigths	grid +particle
x_i	position	grid node
C_p^n	affine momentum	particle
τ_p^n	Kirchhoff stress	particle

1. Transfer Particles to Grid

$$m_i^n = \sum_p w_{ip}^n m_p$$

$$m_i^n v_i^n = \sum_p w_{ip}^n m_p (v_p^n + C_p^n (x_i - x_p^n))$$

2. Grid Update

$$v_i^{n+1} = v_i^n - \frac{\Delta t}{m_i} \sum_p \tau_p^n \nabla w_{ip}^n V_p^0 + \Delta t g$$

3. Transfer Grid to Particles

$$v_p^{n+1} = \sum_i v_i^{n+1} w_{ip}^n$$

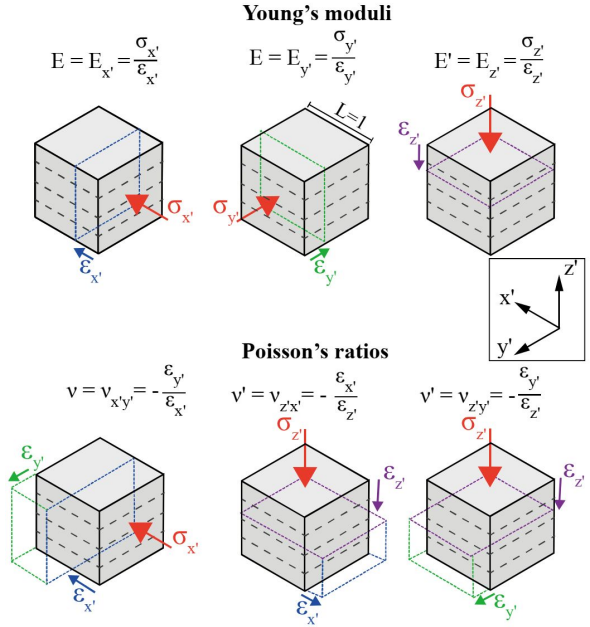
$$x_p^{n+1} = x_p^n + \Delta t v_p^{n+1}$$

C. Jiang et al. The affine particle-in-cell method. TOG, 2015.

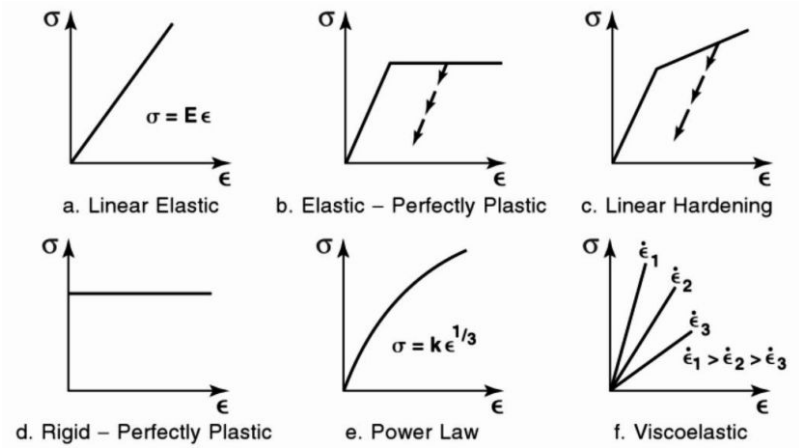
Physics Method Overview

Material Point Method

Material characteristic properties



Constitute behavior

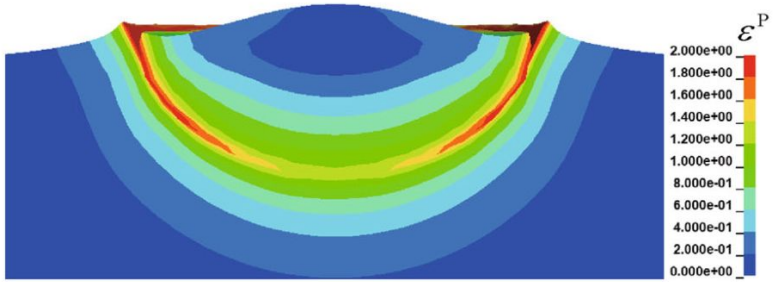


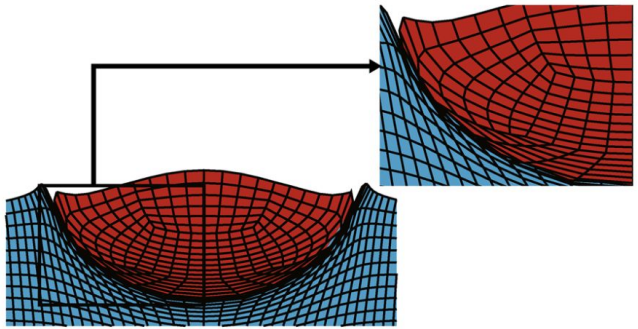
A methodology to determine the elastic properties of anisotropic rocks from a single uniaxial compression test. Jour. of Rock Mech. and Geo. Eng., 2019.

Physics Method Overview

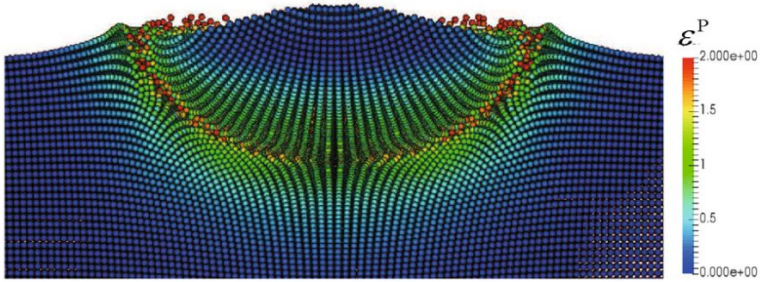
Material Point Method

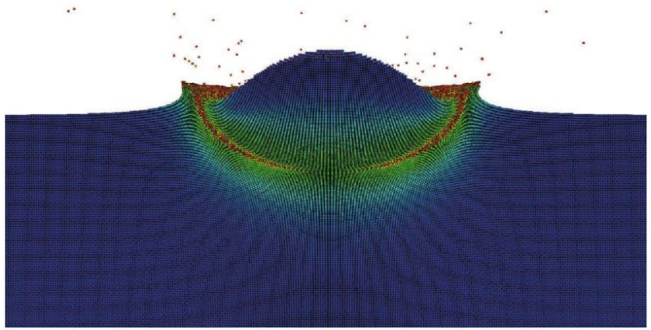
FEM





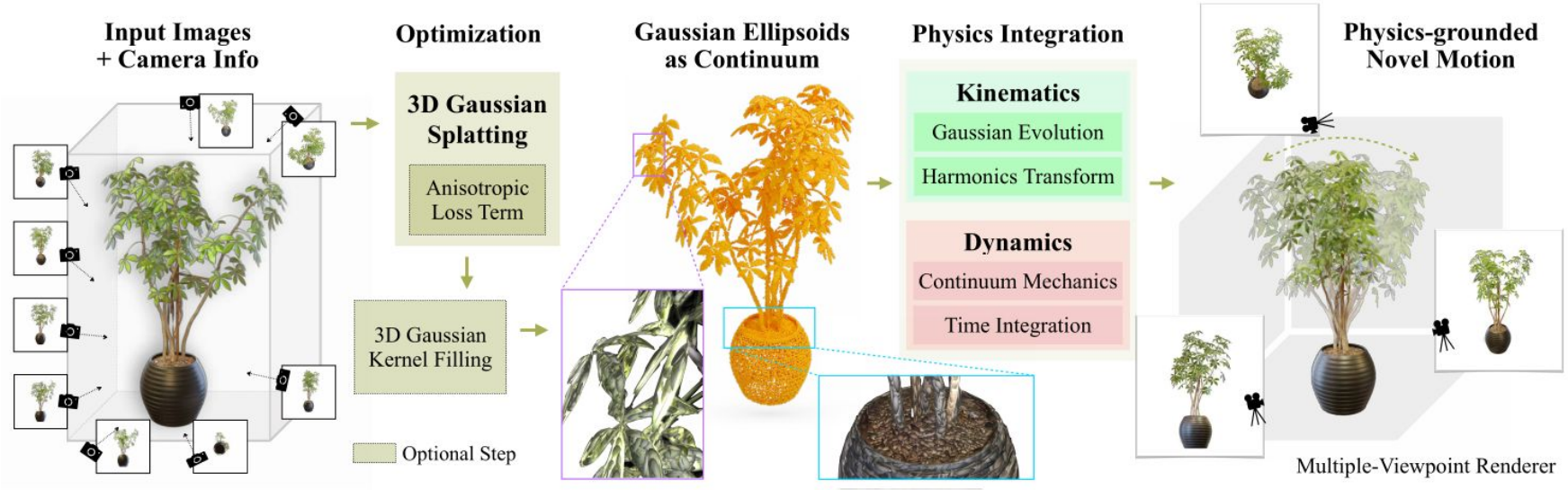
MPM





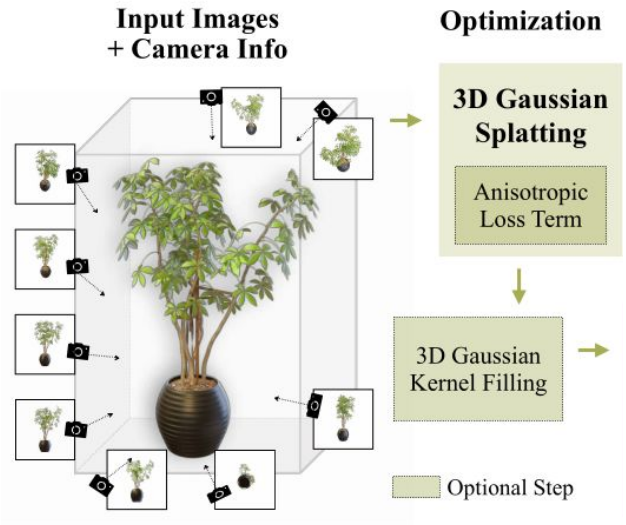
Investigating the cold spraying process with the material point method. Int J Mech Mater Des, 2019.

PhysGaussian Overview



PhysGaussian Overview

3D Gaussian Splatting

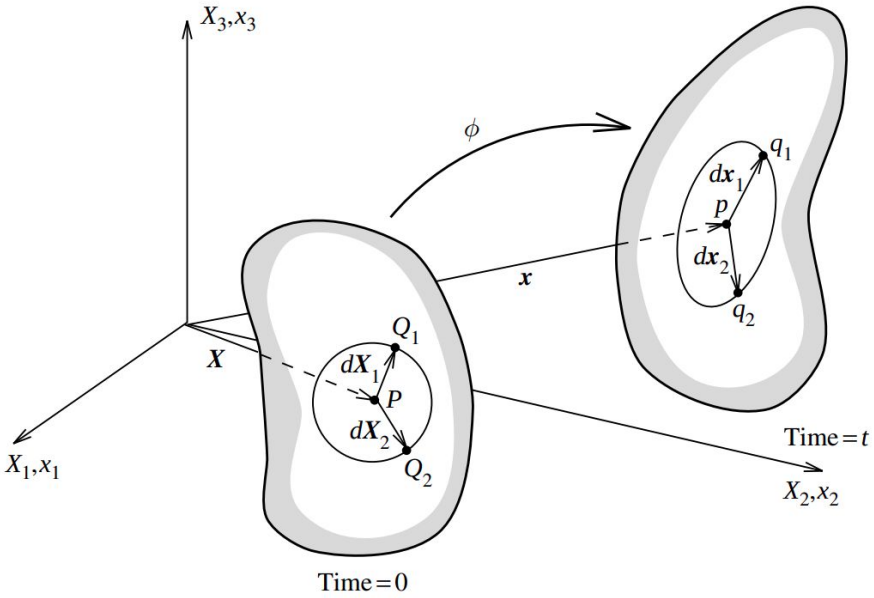


- PhysGaussian uses 3DGS to create the geometry and adapt this;
- Initially, reconstruct a GS representation of a static scene (w/ or wo/ anisotropic loss);
- 3D Gaussians are the scene discretization (for physic simulation);

PhysGaussian Overview

Material Point Method

Motion in the neighborhood of a particle P:



Time-dependent continuous deformation map:

$$x_p = \phi(X_P, t) \quad x_{q_1} = \phi(X_{Q_1}, t) \quad x_{q_2} = \phi(X_{Q_2}, t)$$

$$x = \phi(X, t)$$

Deformation gradient tensor *F*:

encodes local transformations: stretch, rotation, and shear

$$F(X, t) = \nabla_X \phi(X, t)$$

Total deformation gradient decomposed into elastic and plastic:

$$F = F^E F^P$$

J. Bonet and R. D. Wood. Nonlinear continuum mechanics for finite element analysis. Cambridge university press, 1997.

Physics-Integrated 3D Gaussians

3D Static Gaussians ⟶ Dynamic Gaussians

The Gaussian kernels discretize the continuum so, if the continuum deforms, the gaussians deform as well. The centers of the gaussians and the covariance matrices are time-dependent.

- gaussian kernel defined at Xp in the material space

$$G_p(X) = e^{-\frac{1}{2}(X-X_p)^T A_p^{-1}(X-X_p)}$$

- gaussian in the deformed kernel under the deformation map

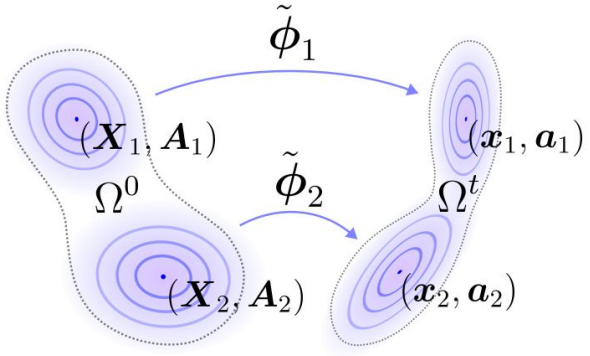
$$G_p(x,t) = e^{-\frac{1}{2}(\phi^{-1}(x,t)-X_p)^T A_p^{-1}(\phi^{-1}(x,t)-X_p)}$$

- first-order approximation is assumed

$$\tilde{\phi}_p(X,t) = x_p + F_p(X - X_p)$$

- Gaussian deformed kernel:

$$G_p(x,t) = e^{-\frac{1}{2}(x-x_p)^T (F_p A_p F_p^T)^{-1}(x-x_p)}$$



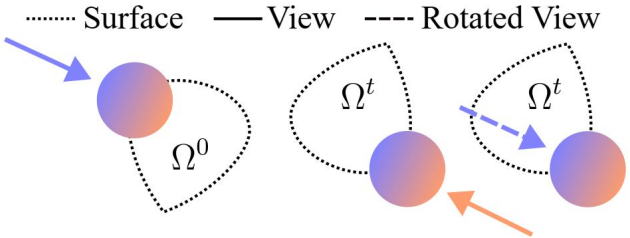
$$x_p(t) = \phi\,(X_p, t)$$

$$a_p(t) = F_p(t) A_p F_p(t)^T$$

PhysGaussian Overview

Evolving Orientations of Spherical Harmonics

When an ellipsoid is rotated over time, is needed rotate the orientations of its spherical harmonics (represented in the material space), to acompain the object rotation (in world-space).



$$f^t(d) = f^0(R^T d)$$

Incremental Evolution of Gaussians

incremental update of the Gaussian kernel shapes without use the deformation gradient

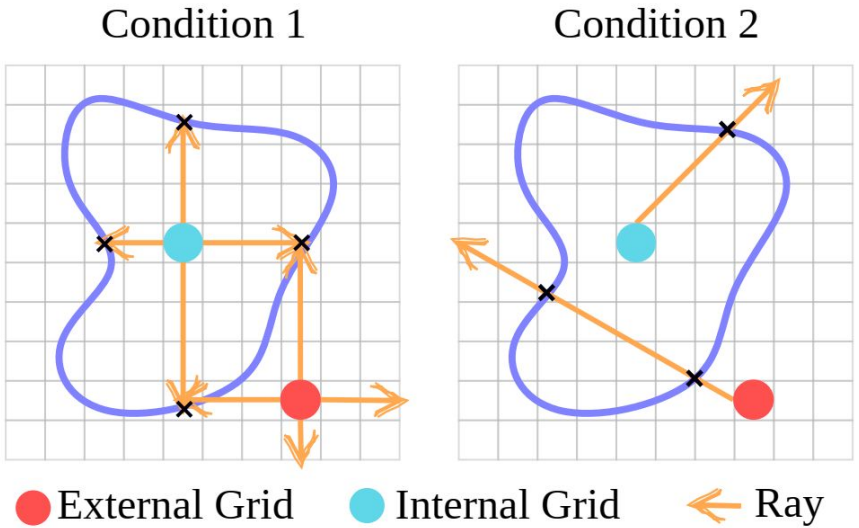
$$a_p^{n+1} = a_i^n + \Delta v_p a_p^n + a_p^n \nabla v_p^T$$

PhysGaussian Overview

Internal Filling

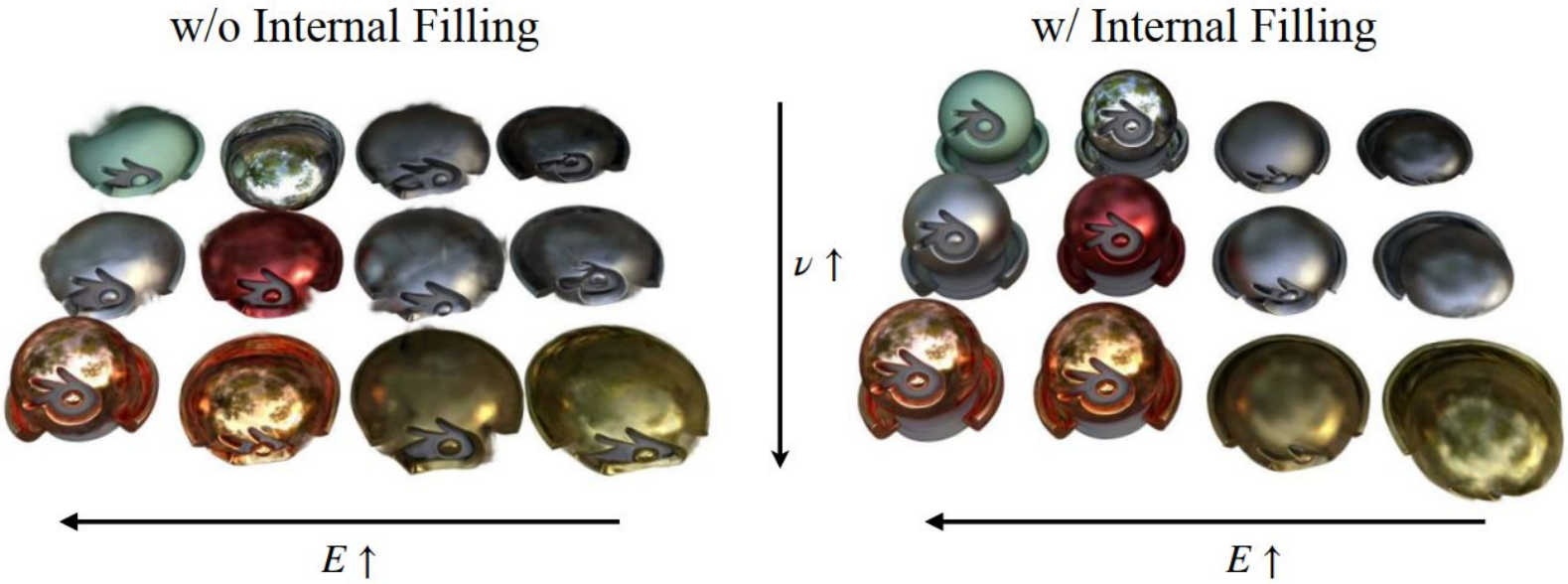
The internal structure is occluded by the object's surface, as the reconstructed Gaussians tend to distribute near the surface, resulting in inaccurate behaviors of volumetric objects. To fill particles into the void internal region, the 3D opacity field is borrowed from 3D Gaussians:

$$d(x) = \sum_p \sigma_p \exp(-\frac{1}{2}(x - x_p)^T A_p^{-1}(x - x_p))$$



PhysGaussian Overview

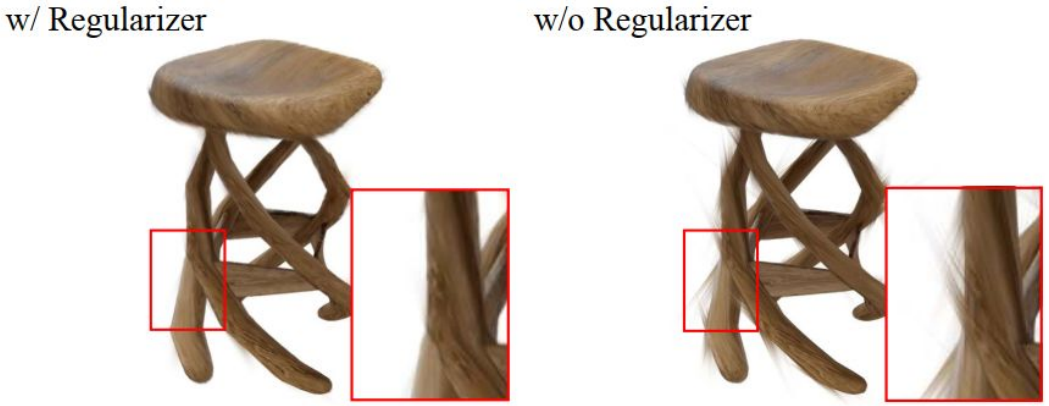
Internal Filling



PhysGaussian Overview

Anisotropy Regularizer

$$\mathcal{L}_{aniso} = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} \max\{\max(S_p)/\min(S_p), r\} - r$$



Experiments

Model settings

Scene	Figure	Constitutive Model
Vasedeck	Fig. 1	Fixed corotated
Ficus	Fig. 2	Fixed corotated
Fox	Fig. 3	Fixed corotated
Plane	Fig. 3	von Mises
Toast	Fig. 3	Fixed corotated
Ruins	Fig. 3	Drucker-Prager
Jam	Fig. 3	Herschel-Bulkley
Sofa Suite	Fig. 3	Fixed corotated
Materials	Fig. 6	Fixed corotated
Microphone	Fig. 7	Neo-Hookean
Bread	Fig. 9	Fixed corotated
Cake	Fig. 9	Herschel-Bulkley
Can	Fig. 9	von Mises
Wolf	Fig. 9	Drucker-Prager

Model Parameters

Notation	Meaning	Relation to E, ν
E	Young's modulus	/
ν	Poisson's ratio	/
μ	Shear modulus	$\mu = \frac{E}{2(1+\nu)}$
λ	Lamé modulus	$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}$
κ	Bulk modulus	$\kappa = \frac{E}{3(1-2\nu)}$

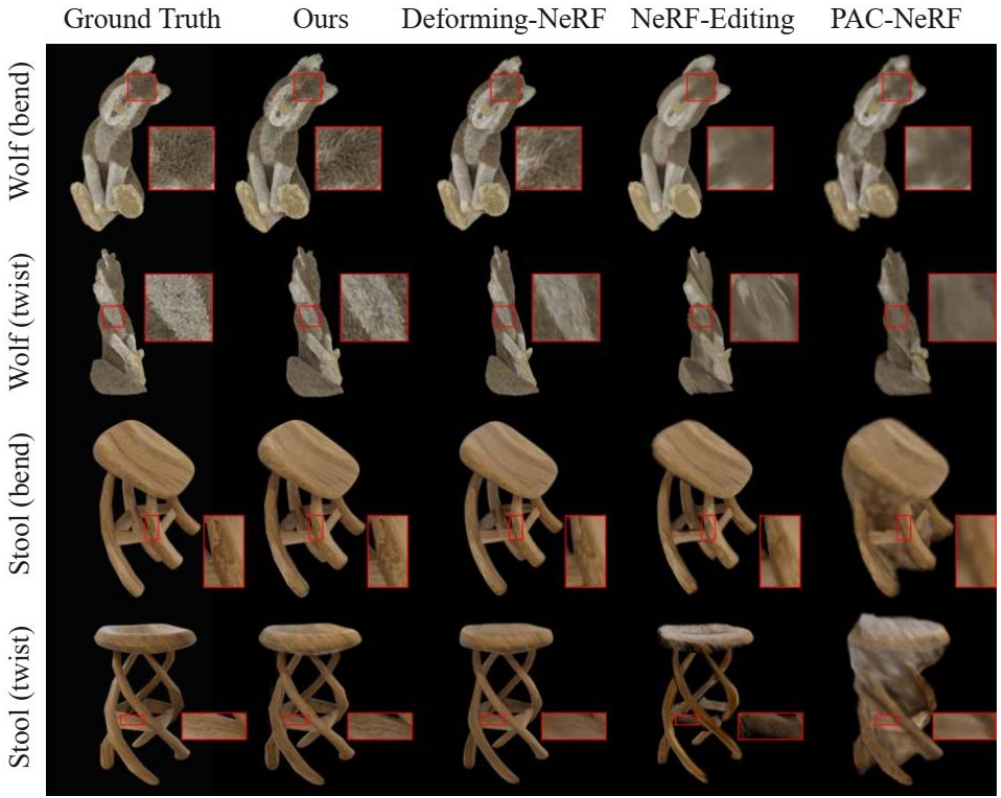
Experiments

Elastic Entity

Plastic Metal

Collision

Experiments



Positive Points

- Physical Method choice: the choice of MPM possibilities the use of the 3DGS framework integrated to the physics behavior;
- Concerns about choosing the adequate constitutive model according to the material;
- Despite to propose a new and innovative approach, the paper uses two consolidated methods in their fields (MPM and 3DGS);
- The method captures volumetric behaviors, as seen below:

Negative Points

- Problems with physical modeling:

The airplane propeller seams more elastic than reality and its movement exert an unrealistic influence on the fuselage

Do not represent the wall as a unified structure (with contact forces)

Blur in fractured region

Evaluation and justification

- The paper achieves what proposes to do;
- Innovative and efficient method;
- Usable to different scenarios and materials;
- Paper text is concise and well explained;
- Adjustments in material properties, contact conditions and in constitutive models can provide a more realistic simulation.

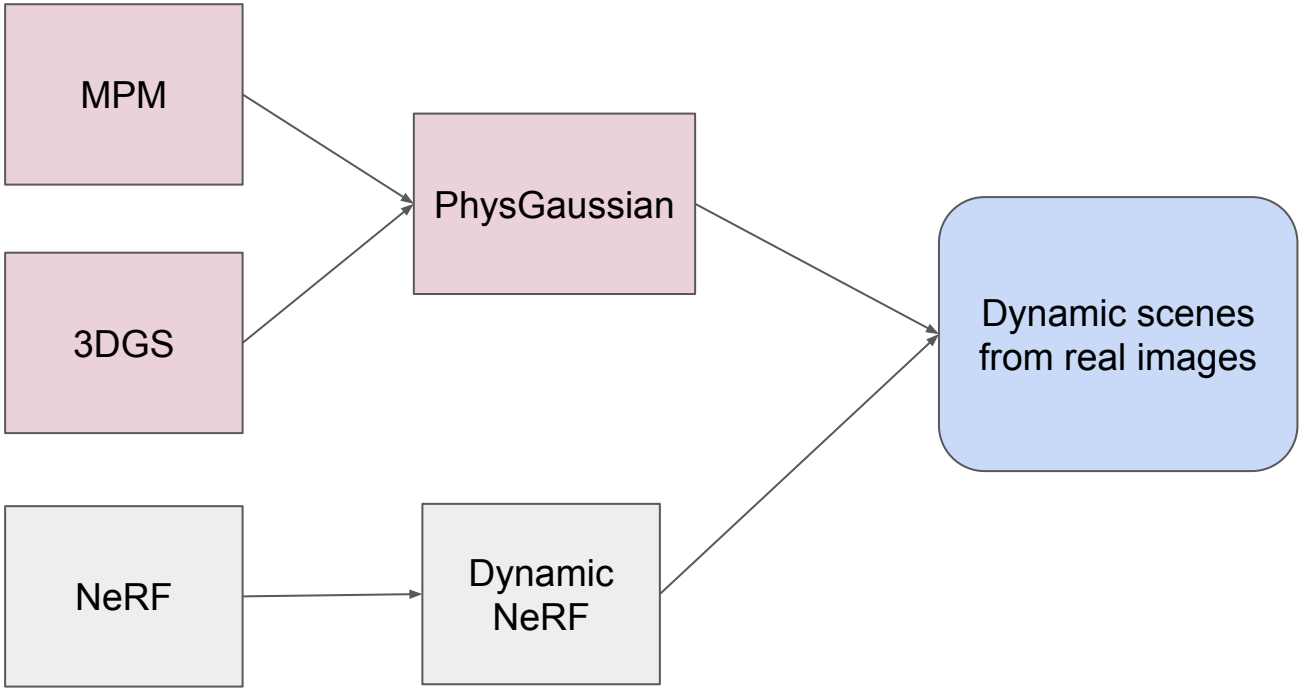
👼 The paper must be accepted ✅

PhysGaussian

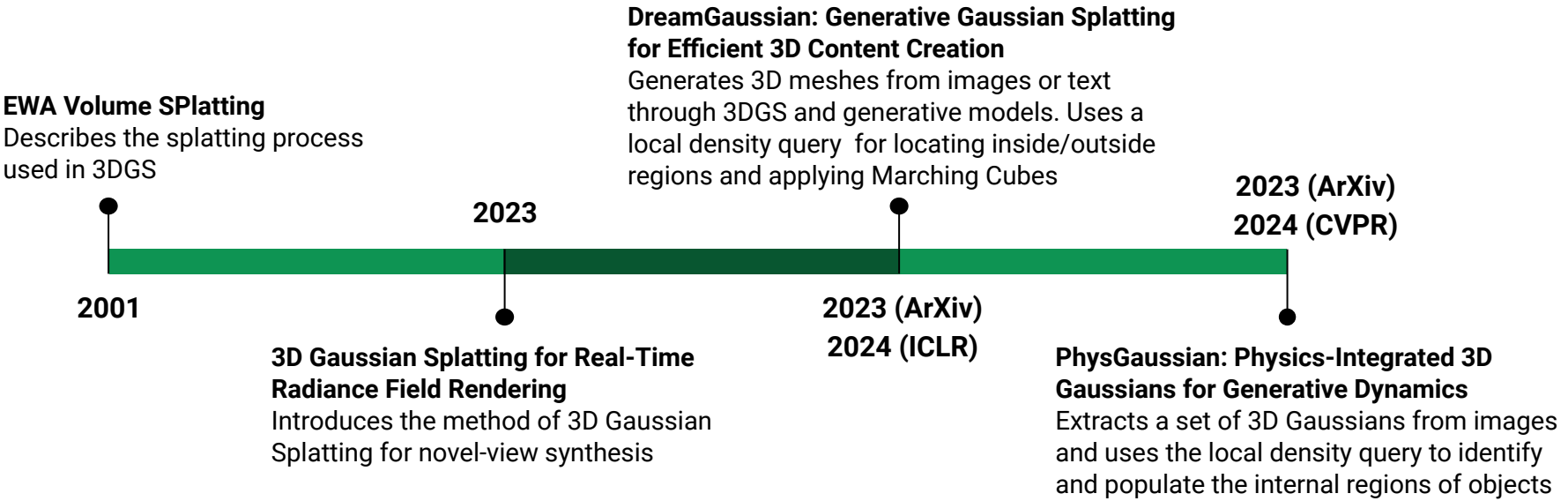
Archaeologist - Leonardo Mendonça

Previous Works

PhysGaussian influences and context



3D Gaussian Splatting Timeline



Material Point Method Timeline

FLIP: A Method for Adaptively Zoned, Particle-in-Cell Calculations of Fluid Flows in Two Dimensions
Physical simulation of fluids via Lagrangian fluid particles, whose properties are interpolated onto an adaptive grid

1986

Application of a particle-in-cell method to solid mechanics
Introduced the Material Point Method, by adapting FLIP to solid mechanics

1995

A material point method for snow simulation
First applied MPM to graphics and animation. Simulation of snow dynamics in the context of Disney's *Frozen* (2013)

2013

The Material Point Method for Simulating Continuum Materials (course notes)
Course ministered in SIGGRAPH 2016. Instructors included the authors of the previous paper

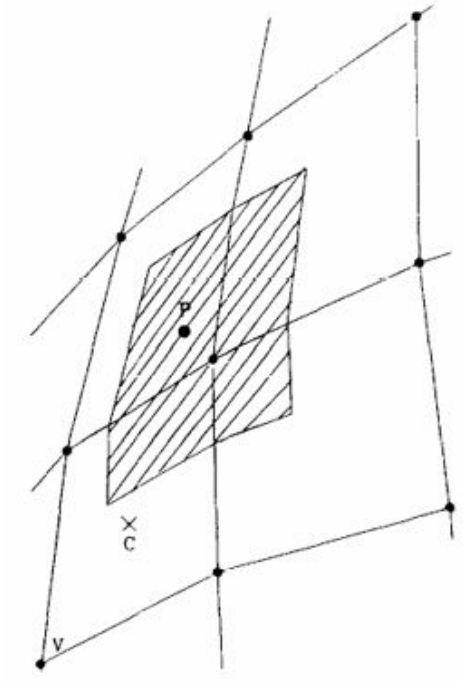
2016

PhysGaussian: Physics-Integrated 3D Gaussians for Generative Dynamics
Leverages MPM and 3DGS for visual representation and physical simulation of dynamic scenes

2023 (ArXiv)
2024 (CVPR)

FLIP: A Method for Adaptively Zoned, Particle-in-Cell Calculations of Fluid Flows in Two Dimensions

- This methodology models a fluid as a set of Lagrangian particles, each with a respective position, velocity, mass and energy;
- The particle properties are projected onto a quadrilateral grid;
- At each time step, the motion equations are solved numerically on the grid, then reprojected back to the particle cloud.



Brackbill, J. U., & Ruppel, H. M. (1986). FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. Journal of Computational physics, 65(2), 314-343.

MPM: Application of a particle-in-cell method to solid mechanics

- MPM extended FLIP to the simulation of solid materials;
- The particle discretization allows for natural modelling of discontinuities, such as fractures or interfaces, as well as large deformations, which would pose a challenge to the Finite Element Method;
- MPM tracks and updates position, velocity, strain and stress for each particle according to the material's equation of motion and the grid values:

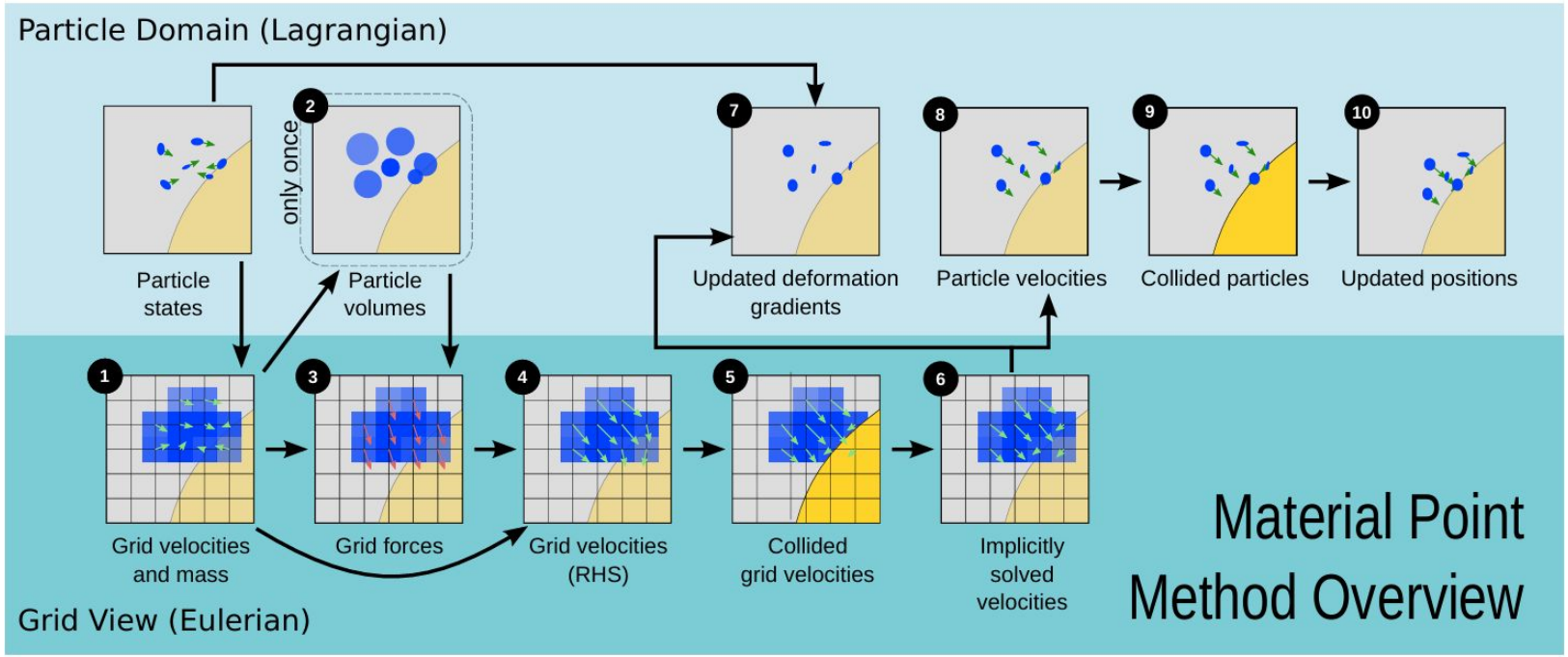
$$\sum_{j=1}^{N_n} m_{ij}^k \mathbf{a}_j^k = \mathbf{f}_i^{\text{int},k} + \mathbf{f}_i^{\text{ext},k},$$

$$\mathbf{X}_p^L = \mathbf{X}_p^k + \Delta t \sum_{i=1}^{N_n} \mathbf{v}_i^L N_i(\mathbf{X}_p^k),$$

$$\mathbf{V}_p^L = \mathbf{V}_p^k + \Delta t \sum_{i=1}^{N_n} \mathbf{a}_i^k N_i(\mathbf{X}_p^k).$$

Sulsky, D., Zhou, S. J., & Schreyer, H. L. (1995). Application of a particle-in-cell method to solid mechanics. Computer physics communications, 87(1-2), 236-252.

A material point method for snow simulation



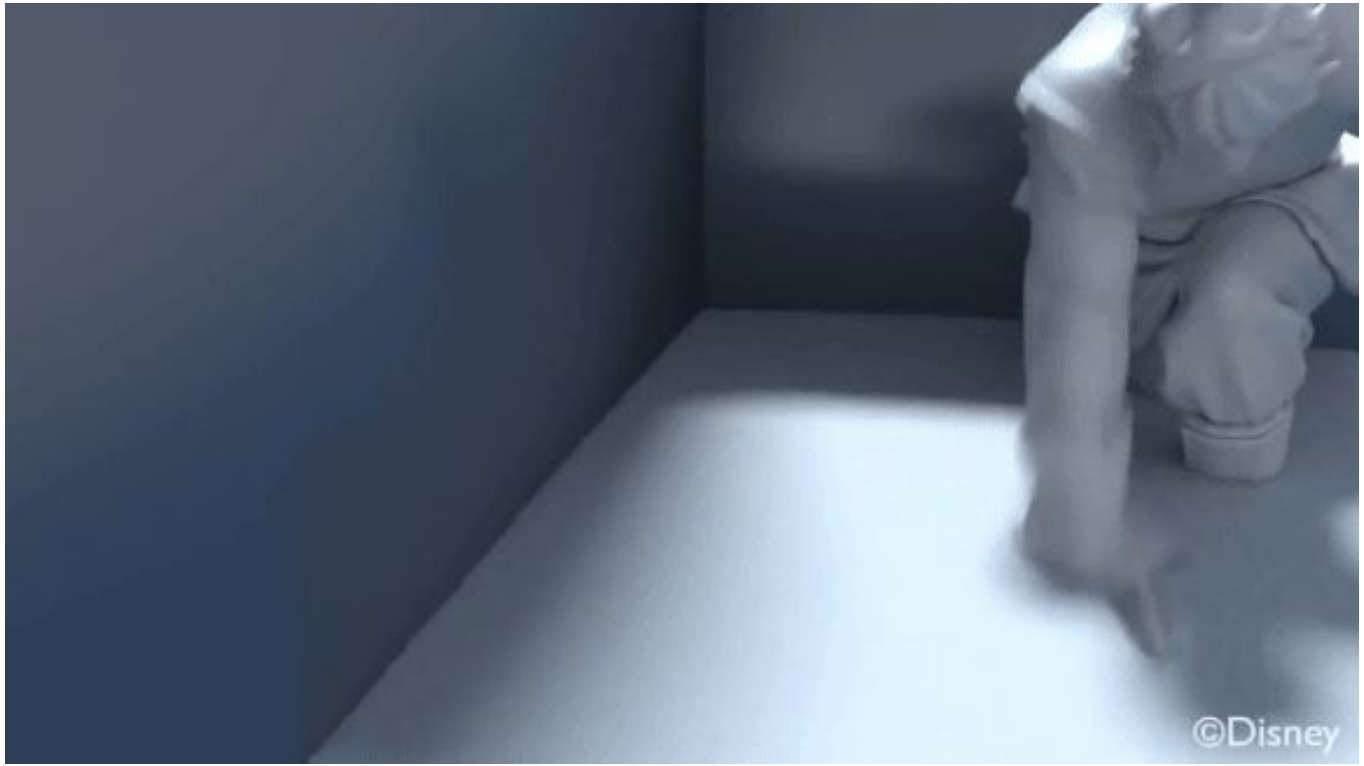
Stomakhin, A., Schroeder, C., Chai, L., Teran, J., & Selle, A. (2013). A material point method for snow simulation. ACM Transactions on Graphics (TOG), 32(4), 1-10.

A material point method for snow simulation

- Pioneer application of MPM to computer graphics;
- MPM is used to model the motion of snow as an elasto-plastic flow;
- The constitutive equation used for snow is derived from engineering models, and calibrated for visually interesting and realistic/verisimilar scenes;
- Some modifications are made to allow for better animator control over snow's visual behavior;
- This method was used in the making of Disney's *Frozen* (2013).

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., & Selle, A. (2013). A material point method for snow simulation. ACM Transactions on Graphics (TOG), 32(4), 1-10.

A material point method for snow simulation



Stomakhin, A., Schroeder, C., Chai, L., Teran, J., & Selle, A. (2013). A material point method for snow simulation. ACM Transactions on Graphics (TOG), 32(4), 1-10.

DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation

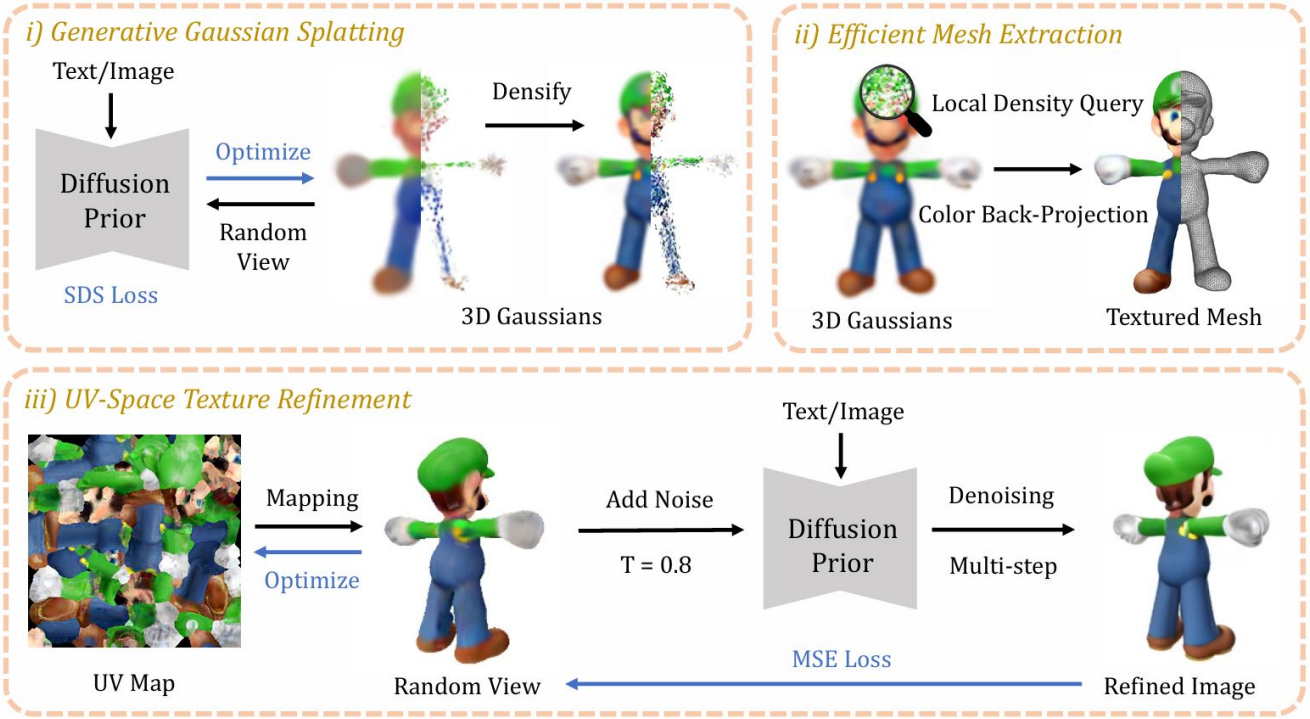
- Employs generative models to create a set of views from a single image or a text prompt;
- A set of 3D Gaussians is then optimized for the generated views;
- Finally, a mesh is extracted through Marching Cubes and a local density query:

$$d(\mathbf{x}) = \sum_i \alpha_i \exp(-\frac{1}{2}(\mathbf{x} - \mathbf{x_i})^T \Sigma_i^{-1}(\mathbf{x} - \mathbf{x_i}))$$

- This density query is the only part of DreamGaussian that's incorporated into PhysGaussian.

Tang, J. et al (2024). "DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation". In: *The Twelfth International Conference on Learning Representations*.

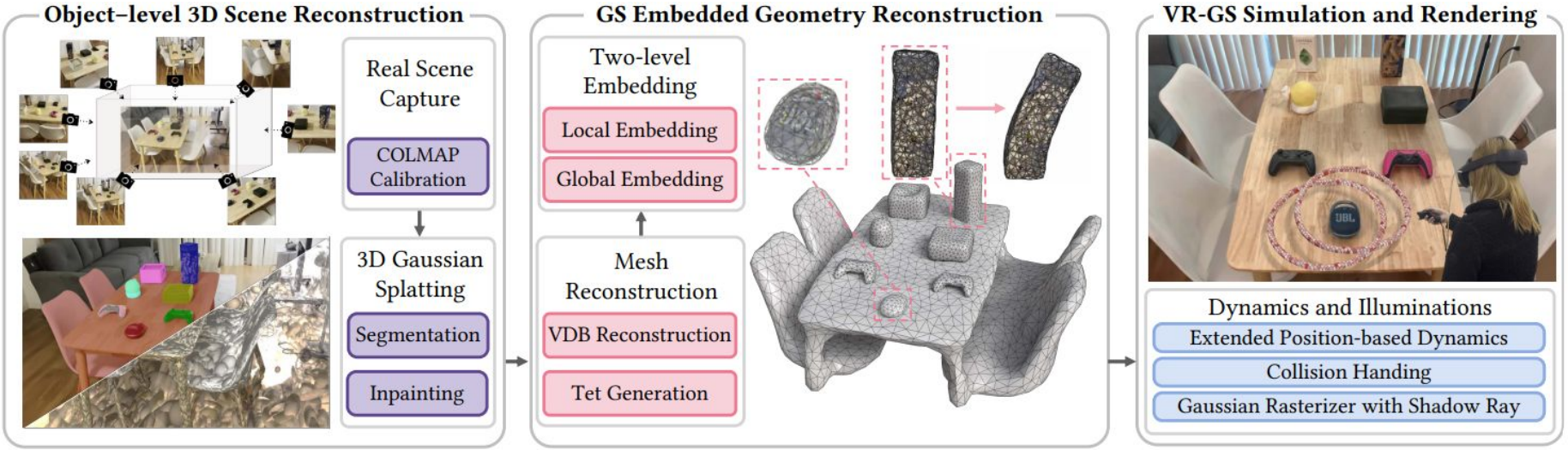
DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation



Tang, J. et al (2024). “DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation”. In: *The Twelfth International Conference on Learning Representations*.

Current Works

VR-GS: A Physical Dynamics-Aware Interactive Gaussian Splatting System in Virtual Reality



Jiang, Y. et al. (2024). Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In ACM SIGGRAPH 2024 Conference Papers (pp. 1-1).

VR-GS: A Physical Dynamics-Aware Interactive Gaussian Splatting System in Virtual Reality

- Y. Jiang *et al*. UCLA; University of Utah; Zhejiang University; Style3D Research; CMU; HKU; Amazon. SIGGRAPH 2024;
- Seeks to allow real-time interactive scenes with physics-based deformation of solids, based on 3DGS;
- Unlike PhysGaussian, VR-GS embeds each Gaussian into a tetrahedral mesh, and then simulates the deformation through XPBD, a preexisting physics simulator;
- Achieves similar visual results as PhysGaussian at a reportedly lower computational cost, however no quantitative experiment was shown in regards to this comparison.

Jiang, Y. et al. (2024). Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In ACM SIGGRAPH 2024 Conference Papers (pp. 1-1).

VR-GS: A Physical Dynamics-Aware Interactive Gaussian Splatting System in Virtual Reality

Jiang, Y. et al. (2024). Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In ACM SIGGRAPH 2024 Conference Papers (pp. 1-1).

PhysGaussian

Hacker - Marcelo de Sousa

Code Pipeline Overview - Initial

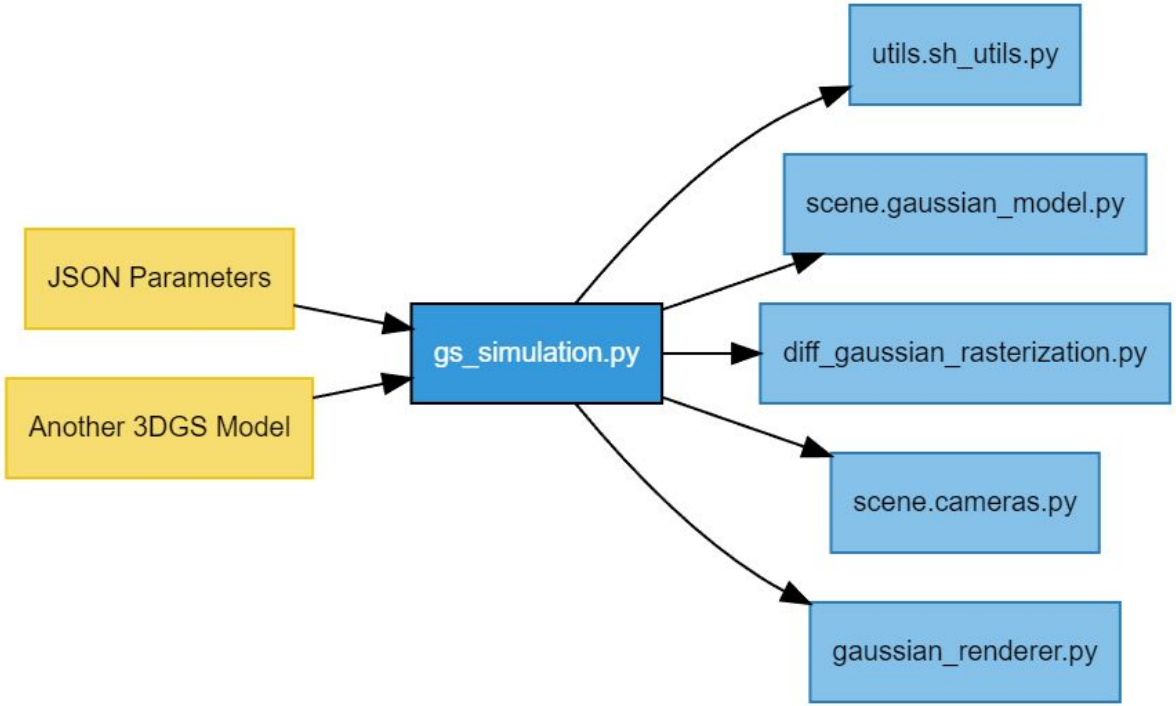
Purpose

Configures and executes simulations using **Gaussian Splatting** and **MPM (Material Point Method)**.

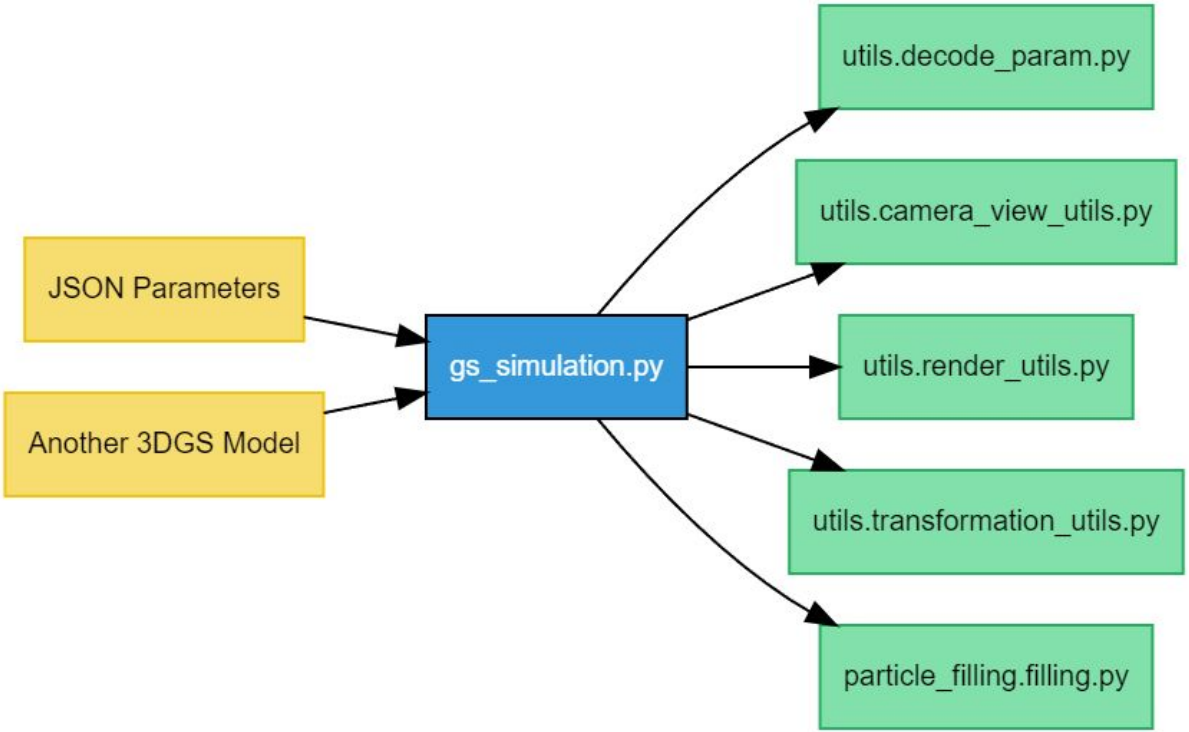
Key Functions

- **Load Configurations**: Loads scene and model settings.
- **Initialize Solver**: Prepares the MPM solver for physical computations.
- **Set Boundary Conditions**: Defines environment constraints.
- **Render Output**: Visualizes results via images or videos.

Code Pipeline Overview - 3DGS

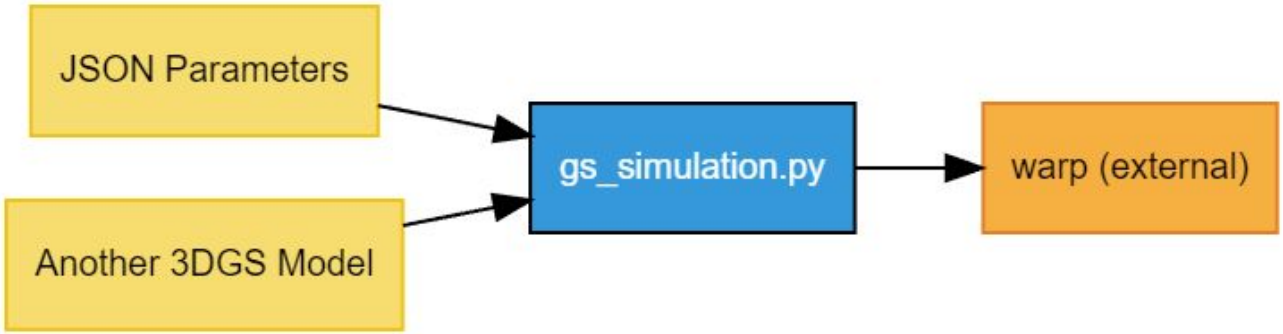


Code Pipeline Overview - Internal Modules Render



Code Pipeline Overview - Internal Modules MPM

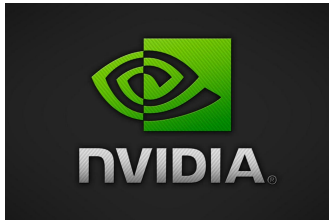
Code Pipeline Overview - External Modules



Integration of Warp in the PhysGaussian Project

What is Warp?

- High-performance library for parallel computing, optimized for GPUs.
- Enables complex physical simulations and efficient graphic rendering.



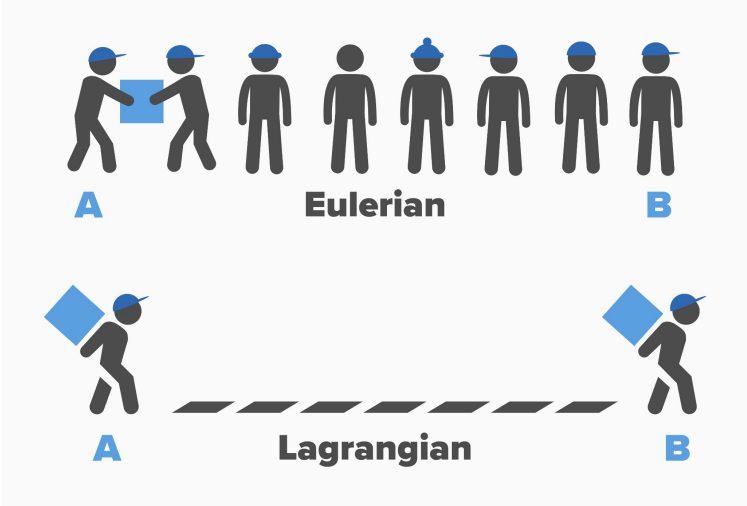
Integration in PhysGaussian:

- **MPM Simulations:**
  - `mpm_solver_warp.py`, `engine_utils.py`, `warp_utils.py`: Optimize physical simulations and parallel calculations.
- **Gaussian Splatting Rendering:**
  - `render_utils.py`, `diff_gaussian_rasterization.py`, `gaussian_renderer.py`: Accelerate Gaussian rendering operations.
- **Graphics and Camera Processing:**
  - `utils.graphics_utils.py`, `camera_view_utils.py`, `transformation_utils.py`: Optimize fov, camera transformations, and data management.

Implementation of the Material Point Method (MPM) in MPM_Simulator_WARP

Objective:

- Leverage MPM's hybrid Lagrangian-Eulerian approach for efficient physical simulations.



Implementation of the Material Point Method (MPM) in MPM_Simulator_WARP

Simulation Initialization

1. Eulerian Grid:

- Defines grid limits, resolution, and spacing (`grid_lim`, `n_grid`, `dx`, `inv_dx`).
- The grid serves as the computational space for properties like stress and velocity.

Implementation of the Material Point Method (MPM) in MPM_Simulator_WARP

Simulation Initialization

2. Particle Properties:

- Initializes Lagrangian particles with positions, velocities, and deformation gradients.

```
self.mpm_state.particle_x = wp.empty(shape=n_particles, dtype=wp.vec3, device=device)
self.mpm_state.particle_v = wp.zeros(shape=n_particles, dtype=wp.vec3, device=device)
self.mpm_state.particle_F = wp.zeros(shape=n_particles, dtype=wp.mat33,
device=device)
```

Implementation of the Material Point Method (MPM) in MPM_Simulator_WARP

Particle to Grid (P2G) Transfer

Kernel: `p2g_apic_with_stress`

- Transfers particle data to the Eulerian grid.
- Includes stress contributions and particle properties for accurate simulations.

Key Steps:

1. Evaluate particle contributions to neighboring grid nodes.
2. Aggregate stress and velocity data for computation.

```
wp.launch(
  kernel=p2g_apic_with_stress,
  dim=n_particles,
  inputs=[self.mpm_state, self.mpm_model, dt],
  device=device
)
```

Implementation of the Material Point Method (MPM) in MPM_Simulator_WARP

Eulerian Grid Update

Kernel: `grid_normalization_and_gravity`

- Normalizes velocities in the grid nodes.
- Applies external forces like gravity or custom-defined forces.

```
wp.launch(
  kernel=grid_normalization_and_gravity,
  dim=grid_size,
  inputs=[self.mpm_state, self.mpm_model, dt],
  device=device
)
```

Implementation of the Material Point Method (MPM) in MPM_Simulator_WARP

Particle Updates and Conservation

Particle Updates:

- **Positions:** Update using velocities:

```
particle_x += dt * particle_v
```

- **Deformation Gradient:** Update to reflect accumulated strain:

```
particle_F = (I + dt * grad_v) @ particle_F
```

Conservation of Mass and Momentum:

- Implements discretized equations ensuring accuracy.

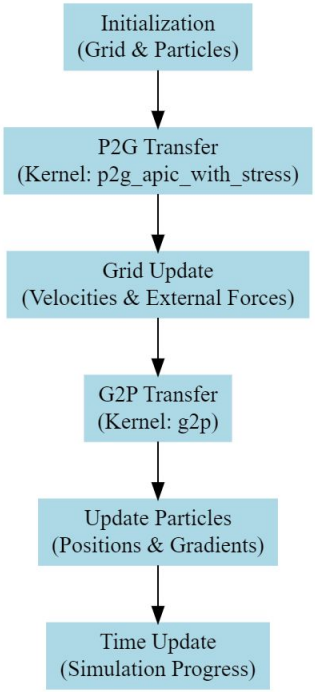
Implementation of the Material Point Method (MPM) in MPM_Simulator_WARP

Boundary Conditions and Time Control

Boundary Conditions:

- Add colliders and custom boundaries to control particle interactions

Implementation of the Material Point Method (MPM) in MPM_Simulator_WARP



Implementation of Physics-Integrated 3D Gaussians

Overview

Objective:

Integrate physics-based updates into 3D Gaussian representations for accurate simulation and rendering.

Key Concepts:

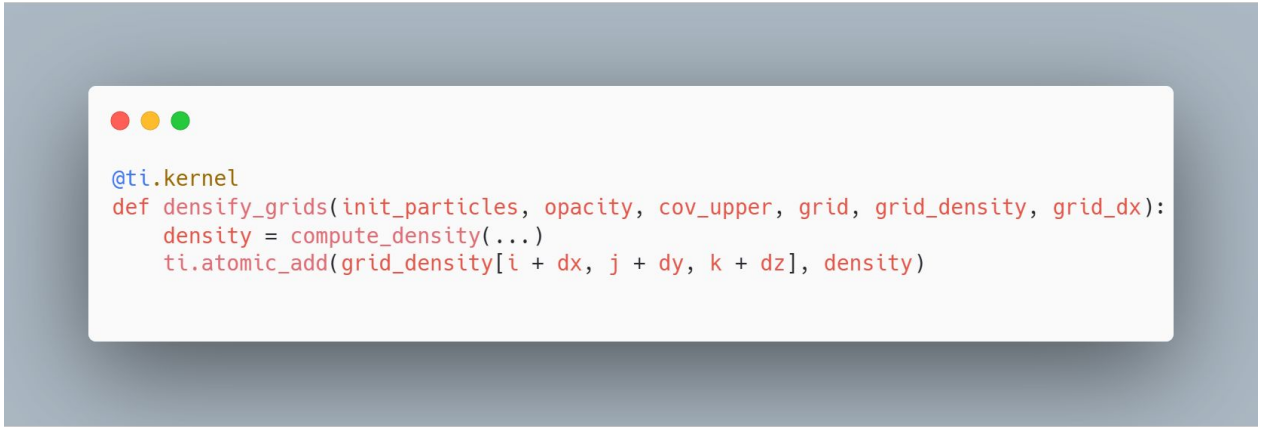
1. **Gaussian Kernels as Particle Clouds:** Represent volumetric objects using Gaussian particles.
2. **Deformation Maps and Affine Transformations:** Track deformation using local transformations.
3. **Dynamic Updates:** Evolve Gaussian properties dynamically during simulation.
4. **Rendering Deformed Gaussians:** Use updated Gaussian parameters to visualize scenes.

Implementation of Physics-Integrated 3D Gaussians

Key Components in the Code

1. Gaussian Kernels as Particle Clouds:

- **Code:** `particle_filling.filling.py`, `diff_gaussian_rasterization.py`.
- **Functionality:**
  - **`densify_grids`:** Calculates particle densities in grids, simulating Gaussian kernel distributions.



```
@ti.kernel
def densify_grids(init_particles, opacity, cov_upper, grid, grid_density, grid_dx):
    density = compute_density(...)
    ti.atomic_add(grid_density[i + dx, j + dy, k + dz], density)
```

Implementation of Physics-Integrated 3D Gaussians

Rendering Deformed Gaussians

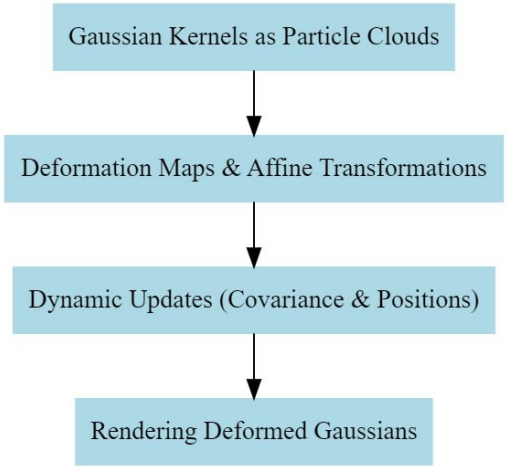
4. Splatting Process:

- **Code:** `gaussian_renderer.py`.
- **Details:**
  - Renders updated Gaussians using particle positions, covariance matrices, and opacity.

Implementation of Physics-Integrated 3D Gaussians

In Summary:

- `mpm_solver_warp.py:` Governs physical updates and deformation gradients.
- `particle_filling.filling.py:` Manages density updates for Gaussian kernels.
- `gaussian_renderer.py:` Handles rendering of dynamically updated kernels.



Implementation in the convert_SH Function

Objective:

Maintain Consistent Visual Quality:

- Adapt spherical harmonics to object rotations to preserve uniform shading and lighting.

```
def convert_SH(shs_view, viewpoint_camera, pc: GaussianModel, position: torch.tensor, rotation:
torch.tensor = None):
    shs_view = shs_view.transpose(1, 2).view(-1, 3, (pc.max_sh_degree + 1) ** 2)
    dir_pp = position - viewpoint_camera.camera_center.repeat(shs_view.shape[0], 1)
    if rotation is not None:
        n = rotation.shape[0]
        dir_pp[:n] = torch.matmul(rotation, dir_pp[:n].unsqueeze(2)).squeeze(2)

    dir_pp_normalized = dir_pp / dir_pp.norm(dim=1, keepdim=True)
    sh2rgb = eval_sh(pc.active_sh_degree, shs_view, dir_pp_normalized)
    colors_precomp = torch.clamp_min(sh2rgb + 0.5, 0.0)

    return colors_precomp
```

Implementation in the convert_SH Function: Implementation Details

- **View Direction Calculation:** Computes direction vectors (`dir_pp`) from the object to the camera center.
- **Inverse Rotation Application:** Aligns spherical harmonics with object orientation if a rotation matrix is provided.
- **Direction Normalization:** Normalizes direction vectors for consistent shading.
- **Spherical Harmonics Evaluation:** Maps coefficients to RGB colors using rotated view directions.
- **Color Value Clamping:** Ensures valid color ranges for rendering.

Incremental Evolution of Gaussians

Current Handling in `MPM_Simulator_WARP`

Incremental Covariance Updates:

- Code uses `particle_cov` and `particle_F` instead of directly updating a_p:

$$\boldsymbol{a}_p^{n+1} = \boldsymbol{a}_i^n + \Delta t(\nabla \boldsymbol{v}_p \boldsymbol{a}_p^n + \boldsymbol{a}_p^n \nabla \boldsymbol{v}_p^T).$$

Rotation Matrix Updates:

- `particle_R` is computed from **F** using:

Incremental Evolution of Gaussians

Overview

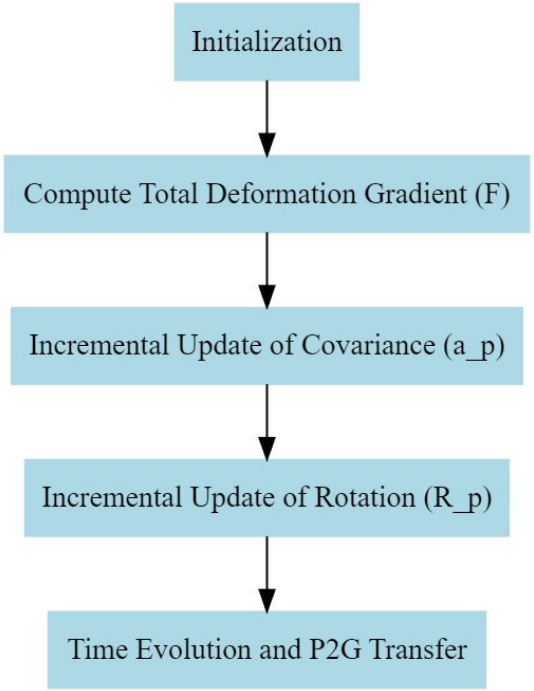
Section 3.6: Incremental Evolution of Gaussians

- The described method avoids dependence on the total deformation gradient **F**.
- Updates the covariance matrix (**a**) and rotation matrix (**R**) incrementally.

Current Implementation:

- Relies on **F** (total deformation gradient) for particle property updates.
- Key kernels: `compute_stress_from_F_trial`, `compute_R_from_F`.

Incremental Evolution of Gaussians



Implementation of Internal Filling

Overview of Internal Filling

Objective:

- Address hollow regions in volumetric objects caused by Gaussian-based reconstruction.
- Ensure accurate volumetric behavior by filling internal gaps with particles.

Key Steps in the Article:

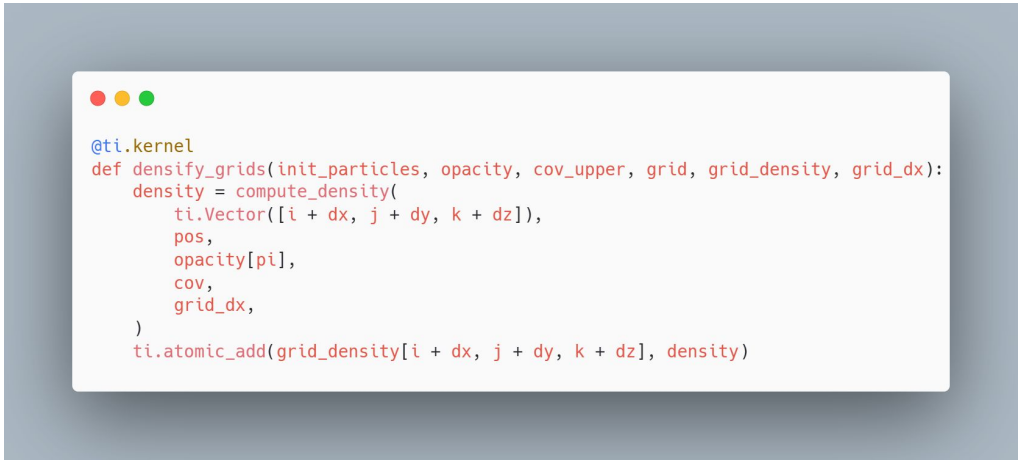
1. Define a 3D opacity field to identify internal regions.
2. Use ray-based methods to detect intersections and internal cells.
3. Fill internal cells with particles inheriting properties from nearby Gaussians.

Implementation of Internal Filling

Code Implementation Highlights

1. 3D Opacity Field:

- Computed as a weighted sum of Gaussians.
- In code: **`densify_grids`** function calculates density for a 3D grid based on Gaussian opacities.



```
@ti.kernel
def densify_grids(init_particles, opacity, cov_upper, grid, grid_density, grid_dx):
    density = compute_density(
        ti.Vector([i + dx, j + dy, k + dz]),
        pos,
        opacity[pi],
        cov,
        grid_dx,
    )
    ti.atomic_add(grid_density[i + dx, j + dy, k + dz], density)
```

Implementation of Internal Filling

Ray-Based Intersection Detection:

- Identifies internal cells using low-to-high opacity transitions.
- In code: **`collision_search`** and **`collision_times`** implement these checks.

```python
@ti.func
def collision_search(grid, grid_density, index, dir_type, size, threshold):
    while ti.max(i, j, k) < size and ti.min(i, j, k) >= 0:
        if grid_density[index] > threshold:
            flag = True
            break
        index += dir
    return flag
```

Implementation of Internal Filling

Internal Filling and Particle Initialization

3. Internal Cell Filling:

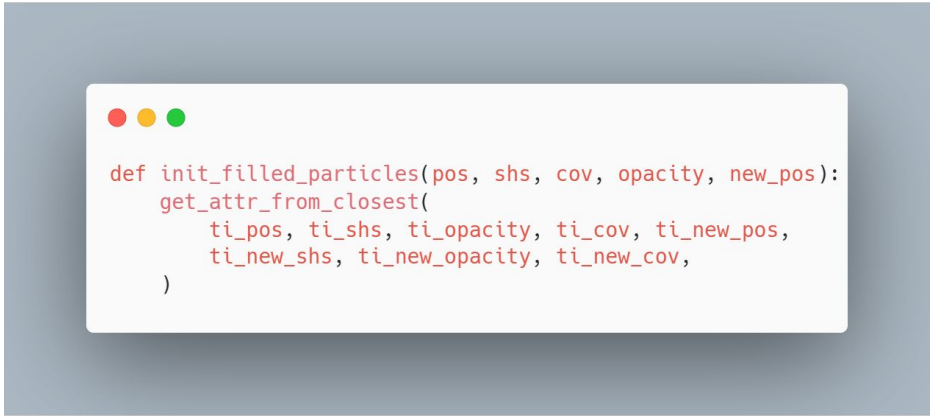
- Internal cells are filled with particles inheriting properties like opacity (**σ_p**) and spherical coefficients (**C_p**).
- In code: `internal_filling` performs the filling.

```
@ti.kernel
def internal_filling(
    grid, grid_density, grid_dx, new_particles, start_idx, max_particles_per_cell,
    exclude_dir, ray_cast_dir, threshold
):
    if collision_hit:
        new_particles[index] = (
            ti.Vector([i + di, j + dj, k + dk]) * grid_dx
        )
```

Implementation of Internal Filling

Particle Property Initialization:

- Covariance matrix initialized as diagonal.
- In code: `init_filled_particles` assigns properties based on the closest Gaussians.

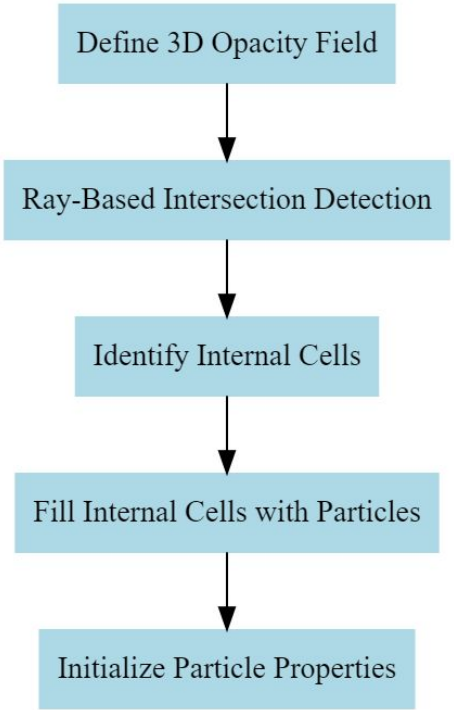


Implementation of Internal Filling

Summary:

- The code aligns with Section 3.7 of the article by:
  - Densifying a 3D grid.
  - Detecting internal regions with ray-based methods.
  - Incrementally filling internal cells with particles.
  - Ensuring accurate volumetric representation for deformable objects.

Implementation of Internal Filling



Simulations

Pros and Cons of Simulation

Pros:

- **Clear and Intuitive Visualization:** Simulations provide an easy-to-understand graphical representation of complex phenomena.
- **Realistic Representation:** The simulation reflects real-world aspects with high fidelity, aiding in analysis and decision-making.

Cons:

- **Challenging Setup:** Creating the simulation, especially assembling the JSON configuration, can be complex and time-consuming.
- **Technical Issues:** Frequent errors in CUDA and Taichi disrupt the workflow and require troubleshooting.
- **Insufficient Documentation:** Limited information on physical parameters and measurement units hampers the accuracy and utility of simulations.

Data Preprocessing Parameters

Purpose:

Defines settings for filtering, aligning, and preparing simulation data.

Key Parameters:

- **Opacity Threshold:** Removes low-opacity Gaussian kernels.
- **Rotation:** Aligns the scene with the grid using rotation degree and axis.
- **Simulation Area:** Selects particles within a defined bounding box.
- **Particle Filling:** Configures cubic regions to fill internal particles, with adjustable density and search thresholds.

Simulation Parameters

Purpose:
Defines the physical and computational settings required for MPM simulations, ensuring realistic and customizable material behavior and environmental conditions.

Key Components:

- **Material Properties:** Specifies material types (e.g., jelly, metal, sand) and their attributes like elasticity and density.
- **Mechanical Parameters:** Includes Young's modulus (E) and Poisson's ratio (nu) for material deformation characteristics.
- **Environment Settings:** Configures gravity and time step size for accurate simulation dynamics.
- **Grid Configuration:** Determines the resolution and structure of the MPM grid.
- **Boundary Conditions:** Enables control over particles or grids, allowing interaction with Gaussian kernels through external forces.

Simulation Parameters (examples)




Export Parameters

Purpose:

Specifies settings for exporting simulation outputs.

Key Parameters:

- **Frame Duration:** Defines the time interval for each exported frame.
- **Total Frames:** Sets the number of frames to export.
- **Camera Index:** Uses a predefined camera view from the training set.

Boundary Conditions

Purpose:

Defines rules to fix or move the reconstructed object in simulations.

Key Types:

- **Bounding Box:** Prevents particles from exiting the simulation area.
- **Cuboid:** Applies conditions on a grid with parameters like center, size, velocity, and time range.
- **Particle Translation:** Enforces movement constraints directly on particles.

Particle Filling Parameters

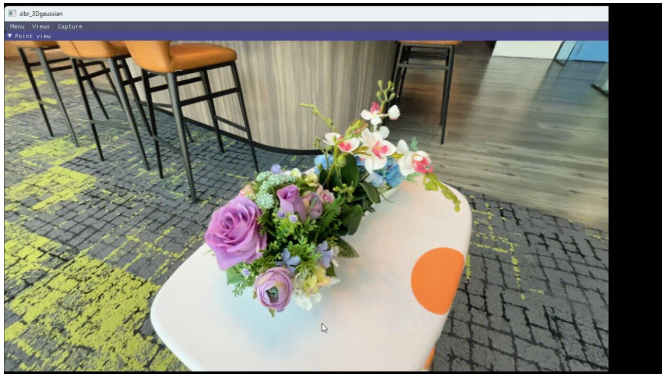
Purpose:

Defines settings for filling inner grids with particles using ray-collision detection.

Key Parameters:

- **Grid Size:** Sets the resolution for particle filling (`n_grid`).
- **Density Threshold:** Identifies surface shell cells for filling.
- **Excluded Directions:** Skips ray casting in specified directions (`search_exclude_direction`).
- **Ray Casting:** Detects collision counts along specified directions (`ray_cast_direction`).
- **Particles per Cell:** Limits particles per grid cell.
- **Boundary:** Specifies the cubic area for filling.

Particle Filling Parameters (examples)



MPM Camera and View Configuration

Overview:
Defines the spatial and visual parameters for camera and viewpoint setup in MPM simulations.

Key Aspects:

- **Vertical Axis Alignment:** Establishes the upward direction for the simulation space.
- **Viewpoint Centering:** Sets the focal point for the camera's initial view.
- **Camera Initialization:** Configures default azimuth, elevation, and radius for optimal framing.
- **Dynamic Camera Movement:** Enables real-time adjustment of camera orientation and zoom during simulation.

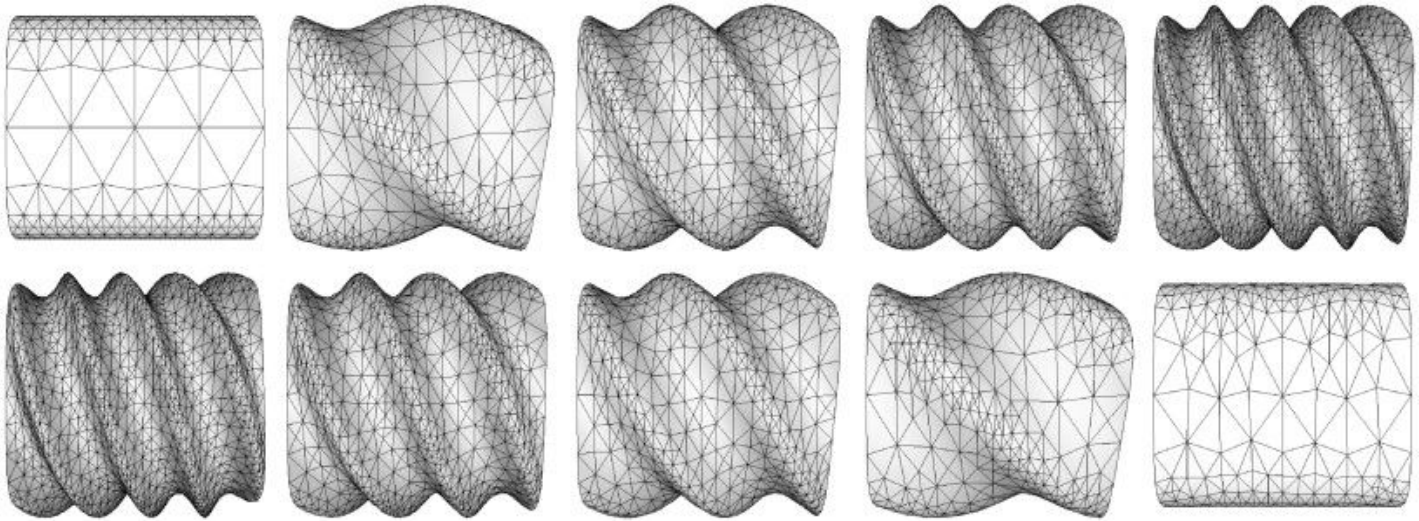
MPM Camera and View Configuration (examples)

PhysGaussian

PhD Student - Marcelo de Sousa

Introduction to the Dynamic Mesh Adaptation Framework

Title: *A Simple and Flexible Framework to Adapt Dynamic Meshes*



Introduction to the Dynamic Mesh Adaptation Framework

Anisotropy Problem in PhysGaussian

Problem:

- Over-skinny Gaussian kernels can form under large deformations.
- These kernels may point outward from the object's surface, causing representations.

Cause:

- Imbalanced scaling of the major and minor axes of Gaussian kernels

Impact:

- Decreases efficiency of 3D representation.
- Leads to visually distorted artifacts in the final output.

$$\mathcal{L}_{aniso} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \max\{\max(\boldsymbol{S}_p)/\min(\boldsymbol{S}_p), r\} - r,$$

Limitations of Regularization and Need for a New Approach

Limitations of Regularization:

- **Global and Static Control:**
  - Imposes uniform restrictions, not adapting to local needs.
- **Insufficient in Extreme Deformations:**
  - May not prevent visual artifacts in complex deformations.
- **Lack of Flexibility:**
  - Does not allow specific control in different regions of the object.
- **Does Not Optimize Resources:**
  - Does not change the number of Gaussians, potentially wasting resources in less important areas.

Limitations of Regularization and Need for a New Approach

Need for a More Robust Solution:

- **Dynamic Local Adaptation:**
  - Adjust the distribution and density of Gaussians according to deformations.
- **Improvement in Visual and Physical Quality:**
  - Preserve important details and reduce artifacts.
- **Optimization of Computational Resources:**
  - Refine or simplify the mesh where necessary.

Integrating the Mesh Adaptation Framework into PhysGaussian

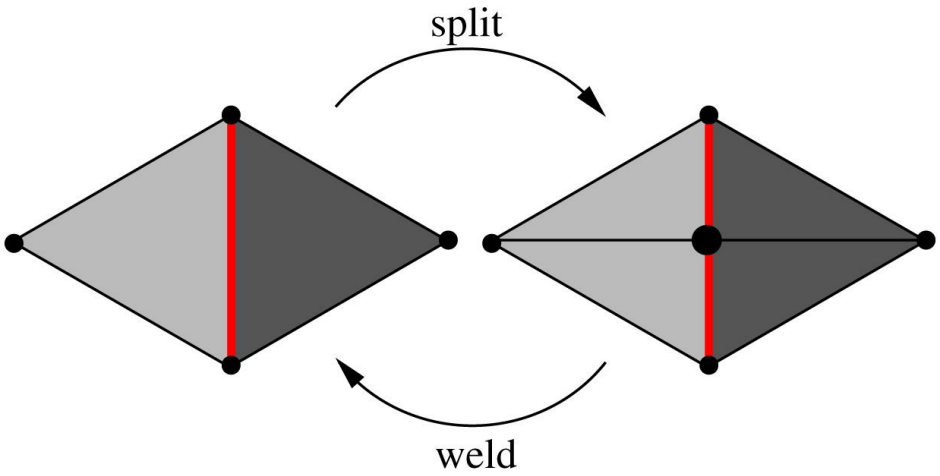
Step 1: Representing Gaussians as a Dynamic Mesh

- Treat the centers of the Gaussians **x** as vertices of a dynamic triangular mesh M.
- Establish connectivity between vertices based on spatial proximity.

Step 2: Structural Operations

- **Refinement (Edge Split):**
  - Split edges in regions with high anisotropy or elevated geometric error.
- **Simplification (Vertex Weld):**
  - Merge vertices in regions of low anisotropy or where density is excessive.

Integrating the Mesh Adaptation Framework into PhysGaussian



Stellar operations on a mesh region.

Integrating the Mesh Adaptation Framework into PhysGaussian

Advantages:

- **Dynamic Adaptation to Deformations:**
  - Adjusts the mesh in response to deformations, improving local representation.
- **Flexibility and Local Control:**
  - Allows incorporating specific criteria, such as curvature or geometric error.

Geometric Operations and Curvature-Sensitive Laplacian Smoothing

Laplacian Smoothing Using the Normal

- **Displacement in the Tangent Plane:** $\delta \mathbf{p} = (\mathbf{c} - \mathbf{p}) - ((\mathbf{c} - \mathbf{p}) \cdot \mathbf{n})\mathbf{n}$
  - **p**: position of the vertex.
  - **c**: centroid of the neighbors.
  - **n**: surface normal at p.
- **Benefits:**
  - **Preservation of Details:** Maintains important surface features.
  - **Reduction of Anisotropy:** Improves the distribution of Gaussians.

Curvature Sensitivity:

- Adjusts the displacement according to local curvature.
- Reduces movement in directions of high curvature to preserve details.

Implementation in PhysGaussian and Additional Benefits

Dynamic Update

- **At Each Time Step:**
  - Apply physical deformations to the vertices.
  - Perform structural operations (refinement/simplification).
  - Apply curvature-sensitive Laplacian smoothing.
  - Reproject vertices onto the deformed surface.

Integration with Physical Properties

- **Updating the Covariance Matrices:** $\mathbf{A}_p(t) = \mathbf{F}_p(t)\mathbf{A}_p(0)\mathbf{F}_p(t)^\top$
- **Rotation of Spherical Harmonics:**
  - Use the rotational part of the polar decomposition of $\mathbf{F}_p(t)$

Implementation in PhysGaussian and Additional Benefits

Additional Benefits:

- **Improvement in Visual Quality and Artifact Reduction:**
  - More accurate representation of deformations.
- **Efficiency in the Use of Computational Resources:**
  - Allocates resources where they are most needed.
- **Improvement in Numerical Stability:**
  - Avoids problems with ill-conditioned elements.

Comparison Between Regularization and Mesh Adaptation

Regularization (Original Solution in PhysGaussian):

- **Advantages:**
  - Simple to implement.
  - Controls anisotropy globally.
- **Limitations:**
  - Does not adapt locally to deformations.
  - May not prevent artifacts in extreme deformations.
  - Does not optimize the distribution of Gaussians.

Comparison Between Regularization and Mesh Adaptation

Mesh Adaptation (Proposed Solution):

- **Advantages:**
  - **Dynamic Local Adaptation:** Responds to local needs during deformations.
  - **Improvement in Visual Quality:** Preserves details and reduces artifacts.
  - **Resource Optimization:** Refines where necessary, simplifies where possible.
  - **Flexibility and Control:** Allows incorporating different adaptation criteria.
- **Challenges:**
  - More complex implementation.
  - Requires additional management of physical properties.

Conclusion and Next Steps

Conclusion:

- **Integration of the Mesh Adaptation Framework:**
  - Provides a robust solution to the anisotropy problem in PhysGaussian.
  - Significantly improves the quality of the simulation and rendering.

Conclusion and Next Steps

Next Steps:

- **Explore Extensions:**
  - Handle topological changes (cuts, merges).
  - Integrate more intuitive user controls.
- **Validation and Testing:**
  - Test the approach in different scenarios and compare results.
- **Optimization:**
  - Improve computational efficiency of the implementation.

Thank you!

Question?