ACG-Simulation: Mid-term Progress Report

Xingyan Chen
chenxy24@mails.tsinghua.edu.cn
IIIS, Tsinghua University
Beijing, China

Zhengyang Zhang
zzy24@mail.tsinghua.edu.cn
IIIS, Tsinghua University
Beijing, China

Abstract

This report summarizes the mid-term progress of our ACG simulation project, focusing on SPH fluid simulation, rigid body dynamics, and rendering pipeline.

Keywords

SPH fluid simulation, rigid body dynamics, Blender rendering

1 Introduction

1.1 Project Topic

Our project aims to simulate the collision of different types of objects and render the simulation outcome with an industrial renderer.

1.2 Project Goals and Technical Points

Our primary goal includes implementing different types of object simulations: rigid body, cloth, fluid and their collision. We also aim to build a rendering pipeline to visualize the simulation results with high quality. Apart from basic implementation, we plan to optimize the simulation performance, provide user-friendly configuration options and support real-time interaction.

The technical points of our project include:

- Implementing rigid body dynamics with collision detection and response.
- Implementing fluid simulation using the Lagrangian point-based WCSPH method [2].
- Implementing fluid and rigid body collision handling with a momentum-conserving two-way coupling method [1].
- Building a rendering pipeline using Blender to visualize simulation results.
- Optimizing simulation performance through algorithmic improvements and multi-threading.
- Providing flexible configuration options for different simulation scenarios.
- Supporting real-time interaction for customized scene setup.

2 Schedule

- week 1-9: Learn basic knowledge about simulation and rendering.
- week 10: Read related papers and documents about simulation and collision implementation methods.
- Weeks 11-12: Implement basic SPH fluid simulation and rigid body dynamics with collision handling. Build up basic rendering pipeline.
- week 13: Implement collision between fluid and rigid bodies. Handleing complex geometry. Try to improve efficiecy of simulation by improve algorithms or using multi-threading.
- week 14: Implement control for customized scene configuration, fixed operation in procudure and interactive in real-time.
- week 15: Test the whole pipeline, fix bugs and improve performance.
- Weeks 16: Finalize rendering pipeline and generate high-quality output videos. Prepare for final demonstration.
- Weeks 17-18: Organize the codebase, complete flexible configuration. Write final report.

3 Methods Completed

3.1 Rigid Body

We have implemented a rigid body dynamics simulator in raw Python, focusing on compact, self-contained math and clear data structures. Each body stores mass, position, linear velocity, orientation (quaternion), and angular velocity. Forces and torques are accumulated per time step and integrated with a semi-implicit Euler scheme. Gravity is applied as a constant force. Linear damping and angular damping are included to stabilize motion.

Orientation is advanced by integrating angular velocity via quaternion updates. We use axis–angle increments constructed from the current angular speed and normalize quaternions for numerical robustness. The inertia tensor is approximated per-shape using the axis-aligned extents to provide diagonal moments consistent with box-like solids.

Collision shapes are convex and represented by either parametric boxes or meshes loaded from OBJ files. For meshes, we parse ‘v‘ and ‘f‘ records, optionally recenter to the centroid, and store triangles and vertices for downstream collision queries. We export simulation frames to OBJ sequences, transforming each body’s local vertices by its current orientation and position for rendering.

3.2 Collision Handling

We employ a Separating Axis Test (SAT) for convex–convex collision detection. Candidate axes include face normals from both shapes and all pairwise cross products of shape edge directions. For each axis, we project both bodies’ world-space vertices and measure interval overlap; a separating axis implies no collision. The contact normal is chosen as the axis with minimal positive overlap, oriented along the bodies’ relative translation.

ACG Mid-term Project Progress Report, Tsinghua University, Beijing, China

Ground contact is handled with a simple but effective scheme. We estimate the lowest support point using transformed vertices and clamp penetration against a flat plane at a configurable height. A normal impulse is applied when the contact point is moving into the plane faster than a small threshold, incorporating both linear and angular contributions via lever arms and inverse inertia. Residual downward velocity is clamped once resting contact is established, and a support-torque term nudges angular velocity toward stable ground interaction.

For body–body response, we compute the relative velocity at the contact points (including rotational components), and apply a restitution-scaled impulse along the contact normal. The impulse denominator includes inverse masses and angular terms derived from lever arms and inverse inertia in local space, transformed back to world space. To correct interpenetration, we add a positional correction along the normal using a fraction of the measured penetration with a small slop to avoid jitter. Restitution and friction-style stabilization are controlled via configuration parameters.

Here is an example of simulation results

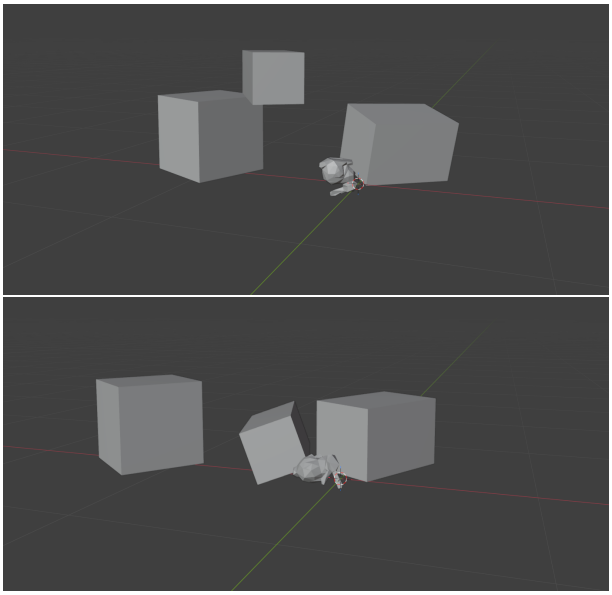

Figure 1: Rigid body simulation frames, we can see that both simple boxes and complex mesh (bunny in the picture) is supported. Also there are interation between different rigid bodies or between rigid body and ground, no penetration is observed.

3.3 Fluid

We have implemented a basic WCSPH fluid simulator using the Taichi programming language. The simulator supports essential SPH operations such as density and pressure computation, viscosity and surface tension forces, and time integration. We have also integrated the SplashSurf library to reconstruct fluid surfaces from particle data for rendering.

The WCSPH method approximates fluid dynamics using a set of particles, where each particle represents a small volume of fluid. The core equations include:

Density computation for particle i:

$$\rho_i = \sum_j m_j W(r_{ij}, h)$$

where m_j is the mass of particle j, W is the cubic smoothing kernel with support radius h, and $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$.

Pressure calculation using the Tait equation:

$$p_i = B\left(\left(\frac{\rho_i}{\rho_0}\right)^\gamma - 1\right)$$

where ρ_0 is the rest density, B is the bulk modulus, and $\gamma = 7$ for water.

The pressure force on particle i:

$$\mathbf{f}_i^{pressure} = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right) \nabla W(r_{ij}, h)$$

Viscosity force (Morris model):

$$\mathbf{f}_i^{viscosity} = \nu \sum_j m_j \frac{(\mathbf{v}_i - \mathbf{v}_j) \cdot \mathbf{r}_{ij}}{r_{ij}^2 + 0.01h^2} \nabla W(r_{ij}, h)$$

where ν is the viscous term.

Surface tension force (approximated):

$$\mathbf{f}_i^{surface} = -\frac{\kappa}{\rho_i} \sum_j \rho_j W(r_{ij}, h) \mathbf{r}_{ij}$$

where κ is the surface tension coefficient.

Time integration uses a semi-implicit Euler method to update positions and velocities.

To efficiently find neighboring particles, we employ a uniform grid data structure. Each particle is assigned to a grid cell based on its position, and only particles in the same or adjacent cells are considered as potential neighbors, reducing the computational complexity from $O(N^2)$ to $O(N)$.

3.4 Rendering Pipeline

We have implemented a complete rendering pipeline using Blender. The pipeline imports reconstructed OBJ mesh sequences, applies Principled BSDF water materials with transparency, sets up lighting and camera, and renders animations. Output includes PNG sequences and MP4 videos generated via ffmpeg.

4 Plan for Remaining Technical Tasks

The remaining tasks focus on refinement and optimization.

First, we plan to read more advanced papers on fluid simulation and collision handling to explore potential improvements of the basic WCSPH method. We will also investigate parallelization techniques to enhance simulation performance.

Second, we will implement the fluid-rigid body collision handling using a momentum-conserving two-way coupling method. This involves calculating interaction forces between fluid particles and rigid body surfaces, ensuring realistic momentum exchange during collisions.

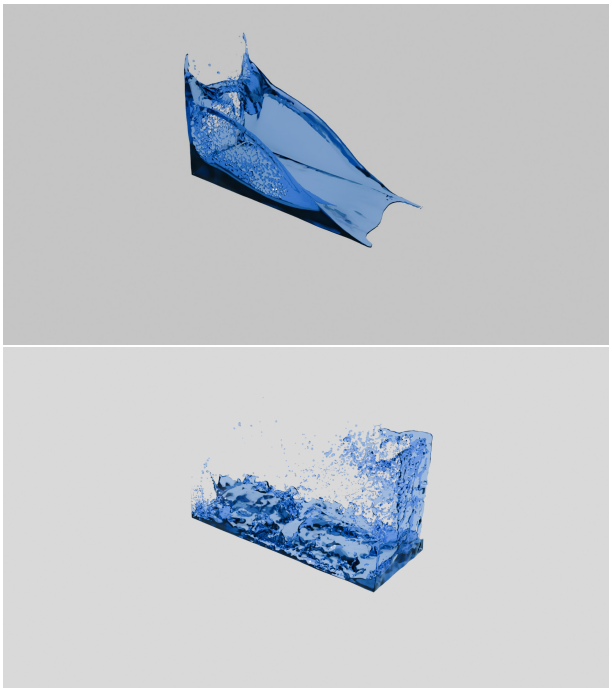

Figure 2: Fluid simulation results

What is more, we will support more complex geometries for rigid bodies, and implement configuration options for different simulation scenarios.

Finally, we will fine-tune the rendering parameters to achieve more realistic water appearance, including material properties and lighting setup.

If time permits, we will also try to implement real-time interaction features, allowing users to modify simulation parameters and visualize results interactively.

5 External Tools

We are using the following external tools and libraries:

- `taichi`: For high-performance physics simulation.
- `Blender`: For 3D rendering and animation.
- `splashsurf`: For surface reconstruction from particle data.
- `ffmpeg`: For video encoding from image sequences.

References

[1] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. 2012. Versatile rigid-fluid coupling for incompressible SPH. *ACM Trans. Graph.* 31, 4, Article 62 (July 2012), 8 pages. doi:10.1145/2185520.2185558

[2] Markus Becker and Matthias Teschner. 2007. Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '07)*. Eurographics Association, Goslar, DEU, 209–217.