Particle Based Simulation

Physics 301, Fall 2022

Ayden Cook

12/9/22

Table of contents

Introduction

Methods

To remain very general purpose, modular, and extensible, the framework has been created in such a way that allows for any type of particle behavior to easily tap into the system's main update loop. Two different APIs have been created to do so.

The first API is called an `Interaction`, and the entrypoint for an Interaction occurs at the beginning of the main loop, while the second is called a `Constraint`, which occurs after the particle integration step (see Listing 0.1).

Interactions

Particle Dynamics

Position based forces, pair forces, field interaction forces.

Smoothed Particle Hydrodynamics (SPH)

The foundational starting point of the SPH method is the following identity:

$$A(\vec{r}) = \iiint_{\mathbb{R}^3} A(\vec{r}')\delta(\vec{r} - \vec{r}')dV' \tag{1}$$

By recognizing that $dm = \rho dV$, and by substituing a kernel approximation to the Dirac-Delta distribution, we arrive at the following continuous formula:

$$A(\vec{r}) = \iiint_{\mathbb{R}^3} \frac{A(\vec{r}')}{\rho(\vec{r}')} W(\vec{r} - \vec{r}', h)dm'$$

The final step is the discretization. Our spatial domain is sampled by a collection of particles, and so the integral over that domain will become the following discrete sum over the particles:

$$A(\vec{r}) = \sum_i^N m_i \frac{A_i}{\rho_i} W(\vec{r} - \vec{r}_i, h) \tag{2}$$

Equation 2 is the central equation to the SPH method, as it allows any set of continuum equations to be discretized and for the interpolatation of any continuous field using a set of discrete sampling particles.

A particularly common value that needs to be calculated is the density, so by setting $A(\vec{r}) = \rho(\vec{r})$, we arrive at the following equation:

$$\rho(\vec{r}) = \sum_i^N m_i W(\vec{r} - \vec{r}_i, h) \tag{3}$$

In a continuum, the particles experience an acceleration due to the local pressure field[1].

$$\frac{d\vec{r}_i}{dt} = -\frac{1}{\rho_i} \nabla_{\vec{r}_i} P$$

Discretizing this force alone will allow us to capture a lot of the behavior of a fluid, and so it will serve as the primary example shown here.

The nice thing about the SPH method is that it allows us to easily calculate gradients by just passing the gradient operator through the summation, leading to:

$$\frac{d\vec{r}_i}{dt} = -\frac{1}{\rho_i} \sum_j^N m_j \frac{P_j}{\rho_j} \nabla W(\vec{r}_i - \vec{r}_j, h)$$

However, upon further inspection one may notice that the force on particle i due to particle j is not antisymmetric as Newton's Third Law requires for momentum conservation $\vec{F}_{ij} \neq -\vec{F}_{ji}$. To fix this, the following identity can be used:

$$\nabla(\frac{P}{\rho}) = \frac{1}{\rho}\nabla P - \frac{P}{\rho^2}\nabla\rho$$

Using this instead, we arrive at the pairwise force that is central to SPH:

[1] They also experience other forces such as viscosity, body forces, elasticity, etc, but we can treat all of them as independent.

$$\vec{F}_{ij}^{pr} = -m_i m_j (\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}) \nabla W(\vec{r}_i - \vec{r}_j, h) \quad (4)$$

Given that $\nabla W(\vec{r}_i - \vec{r}_j, h) = -\nabla W(\vec{r}_j - \vec{r}_i, h)$, this pairwise force will conserve momentum. Finally, to close the equations of motion, an equation of state is needed so that pressure can be calculated. A common one in astrophysics is the polytrope $P = K\rho^{\frac{n+1}{n}}$.

Constraints

Extended Position Based Dynamics (XPBD)

Boundary Conditions

Results

Discussion

Conclusion

Appendix

Relevent Links:

- Project GitHub Repository

Listing 0.1 Update Loop

```
pub fn step_forward(&mut self, dt: f64) {
    if !self.running || dt == 0_f64 {
        return;
    }

    let sub_dt = dt / (self.substeps as f64);
    for _ in 0..self.substeps {
        for interaction in &mut self.interactions {
            interaction.handle(&mut self.particles, sub
        }

        for particle in &mut self.particles {
            particle.integrate(sub_dt);
            particle.forces.clear();
        }

        for constraint in &mut self.constraints {
            constraint.project(&mut self.particles, sub
        }

        for particle in &mut self.particles {

            particle.update_vel(sub_dt);
        }
    }
    self.time += dt;
}
```