Particle system dynamics

Jiří Chmelík, Marek Trtík

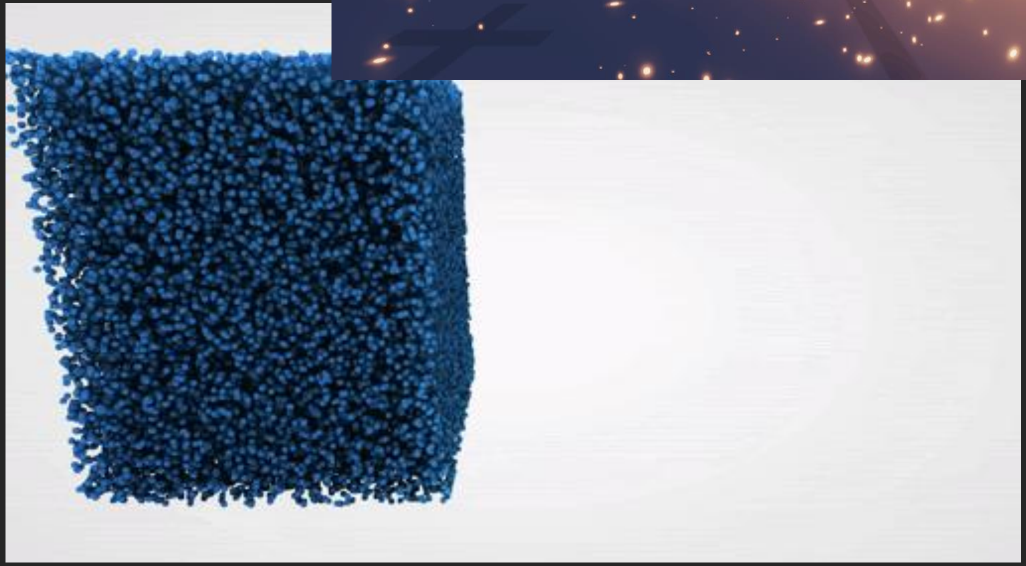
PA199

Outline

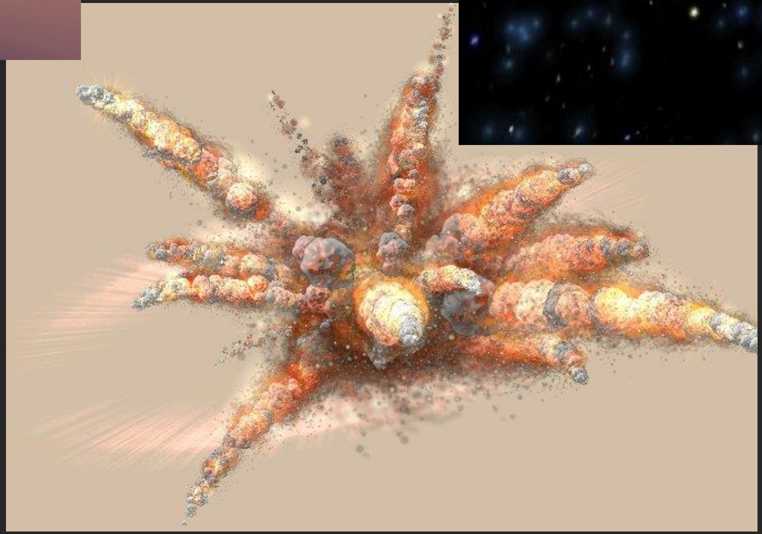
- Motivation
- Motion of a single particle: Equations of motion
  - Use of an ODE solver
- Motion of many particles
- Forces
  - Gravity, drag, spring, local interaction
- Collision: particle vs. plane
  - Detection, response, simple friction

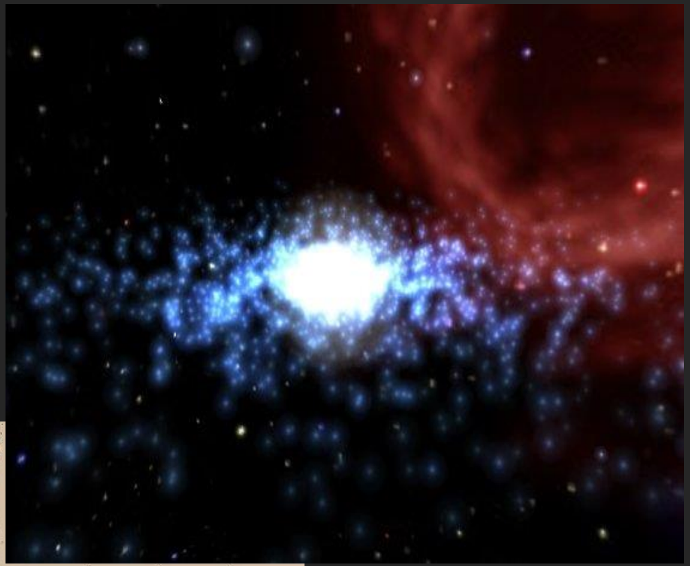
Motivation

https://github.com/Lakshitha
Madushan/Unity-Particle-
System



https://experiments.withgoogle.com/fluid-particles





https://en.wikipedia.org
/wiki/Particle_system

Particle definition

- Particle = an abstract object with these properties:
  - No spatial extent - it is just a point in 3D space
  - Velocity
  - Respond to forces (e.g., gravity)
  - Mass - resistance to changes in motion state
- Particle in math: $\mathcal{P} = (\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{F}, m)$.
- Particle in C++:

```
struct Particle {
        Vector3 position;
        Vector3 velocity;
        Vector3 force;
        float mass;
};
```

Particle equations of motion

- Motion of a particle $\mathcal{P}$ in space is given by a function of time:
  - $\mathcal{P}(t)=(\boldsymbol{x},\boldsymbol{v},\boldsymbol{F},m)(t) = (\boldsymbol{x}(t),\boldsymbol{v}(t),\boldsymbol{F},m)$
    - m is constant (not dependent on time).
    - $\boldsymbol{F}$ is total **external** force (not updated by the particle system).

- To compute $\mathcal{P}(t)$ we need to know how it **changes in time**.
  => We need to compute $\dot{\mathcal{P}}(t) = (\dot{\boldsymbol{x}}(t),\dot{\boldsymbol{v}}(t))$.
  - Newton's second law of motion: $\boldsymbol{F} = m\boldsymbol{a}$
  - Important relations: $\boldsymbol{v} = \dot{\boldsymbol{x}} = \frac{d\boldsymbol{x}}{dt}$, $\boldsymbol{a} = \dot{\boldsymbol{v}} = \frac{d\boldsymbol{v}}{dt}$.

- So, $\mathcal{P}(t)$ is a solution of **Newton's equations of motion**:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{v}(t), \qquad \dot{\boldsymbol{v}}(t) = \boldsymbol{a} = \frac{\boldsymbol{F}}{m}.$$

Solving equations of motion

- There is 6 **ordinary differential equations** (ODE) of the 1st order in the Newton's equations of a single particle.
  - $\boldsymbol{x}(t)$ and $\boldsymbol{v}(t)$ are 3D vector functions.
- In general, a system of n 1st order ODEs has the form:

$$\dot{\boldsymbol{y}} = \boldsymbol{F}(\boldsymbol{y}, t)$$

where $\boldsymbol{y}(\boldsymbol{t}) = \left(y_0(t), \dots, y_{n-1}(t)\right)^{\mathrm{T}}$ and

$$\boldsymbol{F}(\boldsymbol{y}, \boldsymbol{t}) = \left(F_0(y_0(t), \dots, y_{n-1}(t), \mathrm{t}), \dots, F_{n-1}(y_0(t), \dots, y_{n-1}(t), \mathrm{t})\right)^{\mathrm{T}}.$$

Therefore, we have a system:

$$\dot{y}_0 = F_0(y_0, \dots, y_{n-1}, t), \quad \dots \quad , \dot{y}_{n-1} = F_{n-1}(y_0, \dots, y_{n-1}, t)$$

- At each simulation time t_0 we know $\boldsymbol{x}(t_0) = \boldsymbol{X}_0$ and $\boldsymbol{v}(t_0) = \boldsymbol{V}_0$.
- Therefore, we solve the **initial value problem** of 1st order ODEs:

$$\dot{\boldsymbol{y}} = \boldsymbol{F}(\boldsymbol{y}, t), \;\; \boldsymbol{y}(t_0) = \boldsymbol{y}_0$$

Solving equations of motion

- We are given a black-box function ODE solving the initial value problem of 1st order ODEs $\dot{\boldsymbol{y}} = \boldsymbol{F}(\boldsymbol{y}, t),\ \boldsymbol{y}(t_0) = \boldsymbol{y}_0$:

```
using F_y_t = std::function<float(std::vector<float> const&,float)>;

void ODE(
        std::vector<float> const& y0,        // X0,V0 of particle(s)
        std::vector<F_y_t> const& Fyt,       // ẋ,v̇ of particle(s), i.e. v,F/m
        float& t,                            // current time (to be updated)
        float const dt,                      // time step
        std::vector<float>& y                // integrated x,v of particle(s)
        );
```

Comments: $\boldsymbol{X}_0, \boldsymbol{V}_0$ of particle(s); $\dot{\boldsymbol{x}}, \dot{\boldsymbol{v}}$ of particle(s), i.e. $\boldsymbol{v}, \boldsymbol{F}/m$; integrated $\boldsymbol{x}, \boldsymbol{v}$ of particle(s).

NOTE: Implementation of ODE is the topic of next lecture.

Building initial state for ODE

```
void getState(Particle const& p, std::vector<float>& y0) {
	y0.push_back(p.position.x);
	y0.push_back(p.position.y);
	y0.push_back(p.position.z);

	y0.push_back(p.velocity.x);
	y0.push_back(p.velocity.y);
	y0.push_back(p.velocity.z);
}
```

Building derivatives for ODE

```
void getDerivative(Particle const& p, std::vector<F_y_t>& Fyt) {
    Fyt.push_back([&p](std::vector<float> const&,float){ return p.velocity.x; });
    Fyt.push_back([&p](std::vector<float> const&,float){ return p.velocity.y; });
    Fyt.push_back([&p](std::vector<float> const&,float){ return p.velocity.z; });

    Fyt.push_back([&p](std::vector<float> const&,float){ return p.force.x/p.mass; });
    Fyt.push_back([&p](std::vector<float> const&,float){ return p.force.y/p.mass; });
    Fyt.push_back([&p](std::vector<float> const&,float){ return p.force.z/p.mass; });
}
```

- Observation: Parameters of lambda functions are not used.
    - Our functions $\boldsymbol{F}(\boldsymbol{y}, t)$ are simple; ODE solver handles general case.

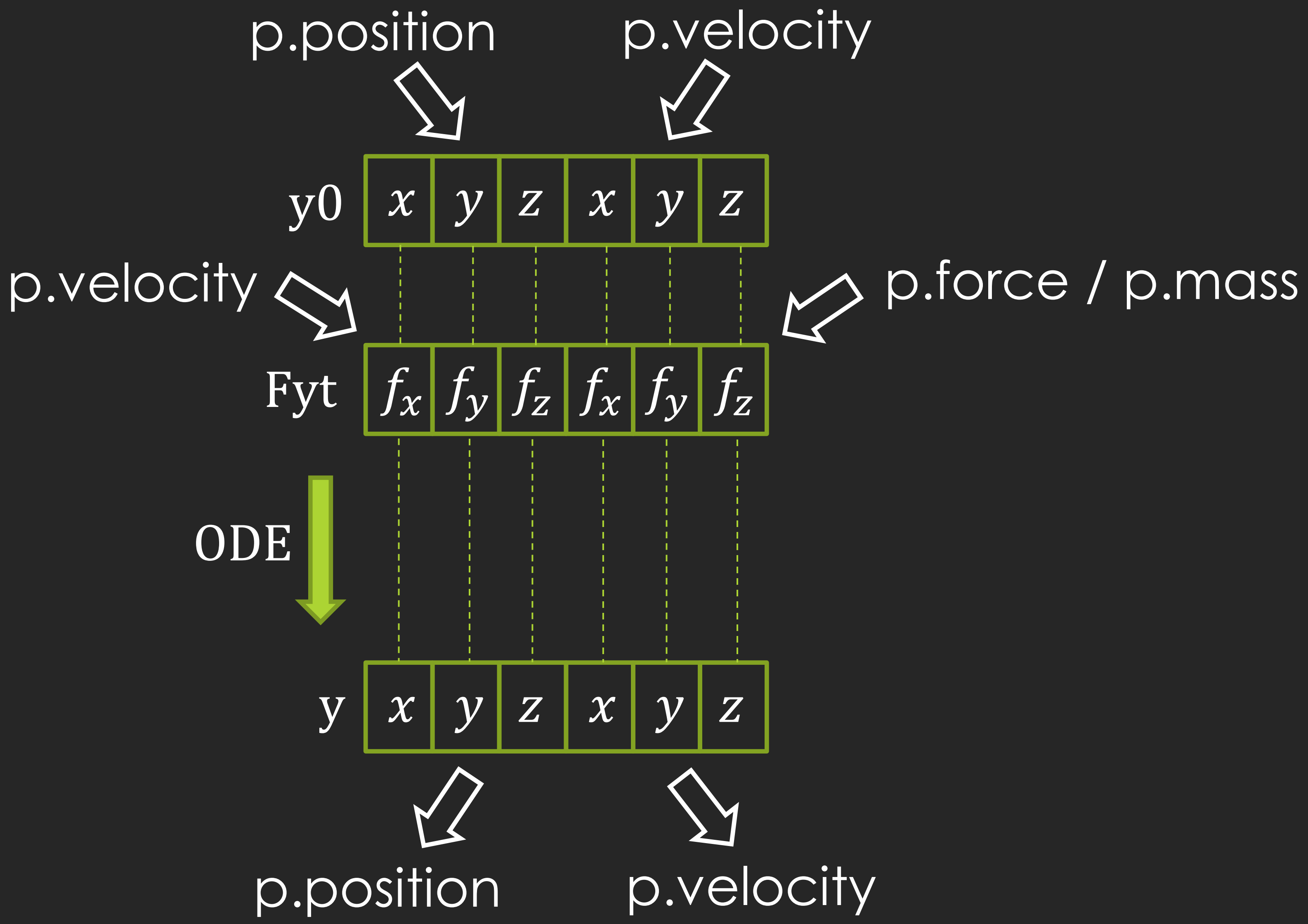
Simulation step for single particle

```
void doSimulationStep(Particle& p, float& t, float const dt) {
        UpdateForce(p,t,dt);        // Applies external forces and impulses.

        std::vector<float> y0, y;
        std::vector<F_y_t> Fyt;
        getState(p, y0);
        getDerivative(p, Fyt);
        ODE(y0, Fyt, t, dt, y);     // Computes y and updates t (t += dt).
        setState(p, y.begin());
}
```

Saving ODE results

```
void setState(Particle& p, std::vector<float>::const_iterator& it) {
        p.position.x = *it; ++it;
        p.position.y = *it; ++it;
        p.position.z = *it; ++it;

        p.velocity.x = *it; ++it;
        p.velocity.y = *it; ++it;
        p.velocity.z = *it; ++it;
}
```

Data flow in simulation step



Particle system

- It is a system consisting of n patricles.

- Particle system in math:

$$\mathcal{P}^n = [\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_{n-1}] = [(\boldsymbol{x}_0, \boldsymbol{v}_0, \boldsymbol{F}_0, m_0), (\boldsymbol{x}_1, \boldsymbol{v}_1, \boldsymbol{F}_1, m_1), \dots, (\boldsymbol{x}_{n-1}, \boldsymbol{v}_{n-1}, \boldsymbol{F}_{n-1}, m_{n-1})].$$

- Particle system in C++:

```
using ParticleSystem = std::vector<Particle>;
```

ODE helper functions

```
void getState(ParticleSystem const& ps, std::vector<float>& y0) {
	for (Particle const& p : ps) getState(p,y0);
}

void getDerivative(ParticleSystem const& ps, std::vector<F_y_t>& Fyt) {
	for (Particle const& p : ps) getDerivatives(p, Fyt);
}

void setState(ParticleSystem& ps, std::vector<float>::const_iterator& it) {
	for (Particle& p : ps) setState(p, it);
}
```

Simulation step for whole system

```
void doSimulationStep(ParticleSystem& ps, float& t, float const dt) {
	UpdateForce(ps,t,dt);	// Applies external forces and impulses.

	std::vector<float> y0, y;
	std::vector<F_y_t> Fyt;
	getState(ps, y0);
	getDerivative(ps, Fyt);
	ODE(y0, Fyt, t, dt, y);		// Computes y and updates t (t += dt).
	setState(ps, y.begin());
}
```

Data flow in simulation step

NOTE: For $\mathcal{P}^n$ we have a system of $6n$ equations.

Forces

```
void UpdateForce(ParticleSystem& ps, float const t, float const dt) {
        clearForce(ps);
        applyForce(ps,t,dt);  // Add all forces and impulses to all particles.
}

void clearForce(ParticleSystem& ps) {
        for (Particle& p : ps) p.force = Vector3(0,0,0);
}
```

- Next we discuss what forces we can add to particles inside the function applyForce().

Gravity

- Homogenous field:
  - For each particle we add the force vector $\boldsymbol{F} = m\boldsymbol{g}$ where
    - m is the mass of the particle.
    - $\boldsymbol{g}$ is a **constant** vector, e.g., $\boldsymbol{g} = \text{Vector3}(0,0,-10)$.

- Radial field:
  - There is a center of gravity $\boldsymbol{S}$ of mass M (it can be one of the particles).
  - For each particle we add the force vector

$$\boldsymbol{F} = G\frac{Mm}{|\boldsymbol{S}-\boldsymbol{x}|^2}\frac{\boldsymbol{S}-\boldsymbol{x}}{|\boldsymbol{S}-\boldsymbol{x}|} = G\frac{Mm}{|\boldsymbol{S}-\boldsymbol{x}|^3}(\boldsymbol{S}-\boldsymbol{x}), \quad \text{where}$$

    - G is the gravitational constant.
    - m is the mass of the particle.
    - $\boldsymbol{x}$ is the position of the particle.
  - We can handle cases when $|\boldsymbol{S}-\boldsymbol{x}|$ is small by not applying the force.

Viscous Drag

- A force of the environment making a particle decrease its velocity relative to the environment.

- A drag force can also enhance numerical stability of simulation.

- For each particle we add the force vector $\boldsymbol{F} = k_d(\boldsymbol{V} - \boldsymbol{v})$, where
  - k_d is the coefficient of drag.
  - $\boldsymbol{V}$ is the velocity of the environment (often $\boldsymbol{V} = \boldsymbol{0}$).
  - $\boldsymbol{v}$ is the velocity of the particle.

Spring

- It is a force between two particles $\mathcal{P}_i$ and $\mathcal{P}_j$ given by Hook's law:

$$\boldsymbol{F}_i = -\left(k_s(|\boldsymbol{d}| - d_0) + k_d \dot{\boldsymbol{d}} \cdot \frac{\boldsymbol{d}}{|\boldsymbol{d}|}\right)\frac{\boldsymbol{d}}{|\boldsymbol{d}|}$$

$$\boldsymbol{F}_j = -\boldsymbol{F}_i \quad \text{(3rd Newton's law – action and reaction)}$$

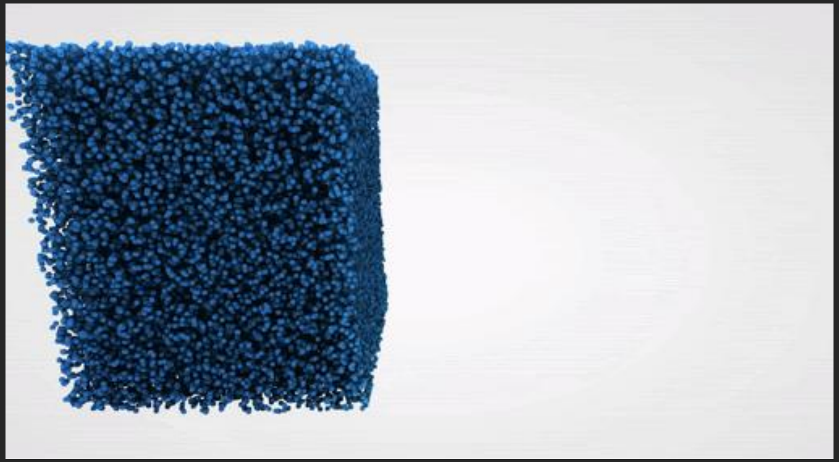
  where
  - k_s is the spring constant.
  - k_d is the damping constant.
  - $\boldsymbol{d} = \boldsymbol{x}_i - \boldsymbol{x}_j$ is the distance vector between the particles.
  - d_0 is the rest length between the particles.
  - $\dot{\boldsymbol{d}} = \boldsymbol{v}_i - \boldsymbol{v}_j$ is the relative velocity between the particles.

Local interaction

- Particles start to interact when they come close.
- Particles stop to interact when they move apart.

- Example: Particle-based fluid simulation.

- Computationally expensive task:
  - $\mathcal{O}(n^2)$ – all pairs of particles are checked.
  - Space partitioning methods (e.g., octree) are essential for performance.



https://experiments.withgoogle.com/fluid-particles

Collision: particle vs. plane

- We often want particles to collide with the ground or a wall. These boundaries can be approximated by planes.

https://github.com/LakshithaMadushan/Unity-Particle-System

- The process consists of two parts:
  - Detection of a collision.
  - Response to the collision.

Collision detection

- Let us consider a particle $\mathcal{P} = (\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{F}, m)$.

- The plane is represented by the equation $\boldsymbol{N} \cdot (\boldsymbol{X} - \boldsymbol{P}) = 0$, where
  - $\boldsymbol{N}$ is the unit normal vector pointing “outside” (above the ground).
  - $\boldsymbol{P}$ is some point in the plane.
  - $\boldsymbol{X}$ is a tested point.

- The particle collides with the plane only if $\boldsymbol{N} \cdot (\boldsymbol{x} - \boldsymbol{P}) \leq 0$.
  - Only in that case we proceed to the collision response.

Collision response

- If the particle increases the penetration with the plane, i.e., when $\boldsymbol{N} \cdot \boldsymbol{v} < 0$, then we change the component of $\boldsymbol{v}$ orthogonal to the plane:
  - The component of $\boldsymbol{v}$ orthogonal to the plane is $\boldsymbol{v}^{\perp} = (\boldsymbol{N} \cdot \boldsymbol{v})\boldsymbol{N}$.
  - The velocity change is then is $\Delta\boldsymbol{v} = -(1+r)\boldsymbol{v}^{\perp} = -(1+r)(\boldsymbol{N} \cdot \boldsymbol{v})\boldsymbol{N}$, where
    - $r \in \langle 0,1 \rangle$ is the coefficient of restitution.
  - We update $\boldsymbol{v}$ to be $\boldsymbol{v} + \Delta\boldsymbol{v}$.
    - NOTE: Formally, we apply an impulse $\boldsymbol{I} = m\Delta\boldsymbol{v}$ to the particle.

- If $\boldsymbol{N} \cdot \boldsymbol{F} < 0$, then we cancel the component of $\boldsymbol{F}$ orthogonal to the plane:
  - We compute $\Delta\boldsymbol{F} = -\boldsymbol{F}^{\perp}$, where $\boldsymbol{F}^{\perp} = (\boldsymbol{N} \cdot \boldsymbol{F})\boldsymbol{N}$.
  - We update $\boldsymbol{F}$ to be $\boldsymbol{F} + \Delta\boldsymbol{F}$.
  - NOTE: This step should be applied **after** all external forces (gravity, etc.) were added to the $\boldsymbol{F}$ field of the particle.

Simple friction

- We build a simplified friction model for particle system:
  - We do not distinguish static and dynamic friction.
  - We ignore variable changes caused by interactions with other particles.

- If $\boldsymbol{N} \cdot \boldsymbol{F} < 0$, then a friction force $\boldsymbol{F}_f$ is acting on the particle:
  - $|\boldsymbol{F}_f|$ is proportional to $|\boldsymbol{F}^{\perp}|$, where $\boldsymbol{F}^{\perp} = (\boldsymbol{N} \cdot \boldsymbol{F})\boldsymbol{N}$.
  - The direction of $\boldsymbol{F}_f$ is opposite to the component $\boldsymbol{v}^{\parallel}$ of $\boldsymbol{v}$ parallel with the plane, where $\boldsymbol{v}^{\parallel} = \frac{\boldsymbol{N} \times \boldsymbol{v} \times \boldsymbol{N}}{|\boldsymbol{N} \times \boldsymbol{v} \times \boldsymbol{N}|}$.

- Therefore, we define the friction force as $\boldsymbol{F}_f = k_f(\boldsymbol{N} \cdot \boldsymbol{F})\boldsymbol{v}^{\parallel}$, where
  - k_f is a friction coefficient.

- Note: We should apply the friction before the collision response.

Summary

- We defined particle and particle system.
- We learned Newton's equations of motion for a particle, i.e., a system of 1^{st} order ODEs.
- We learned how to use ODE solver for the simulation.
- We learned several kinds of forces which we can apply to particles.
- We know how to compute and respond to collision of a particle with a plane, including application of a friction force.

References

- [1] *Andrew Witkin*; Physically Based Modeling: Principles and Practice Particle System Dynamics; Robotics Institute, Carnegie Mellon University, 1997.