Particle system dynamics

Jiří Chmelík, Marek Trtík

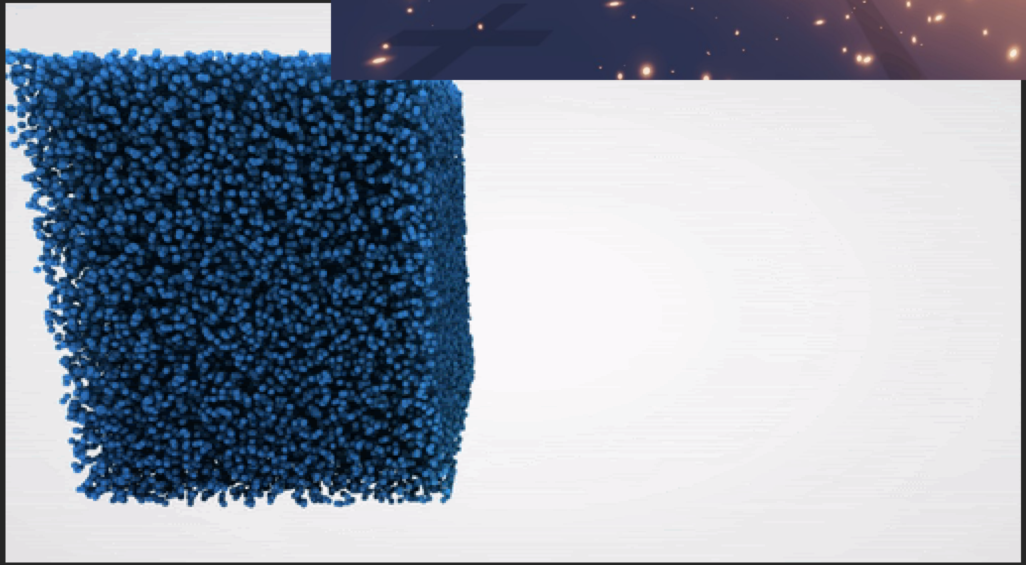
PA199

Outline

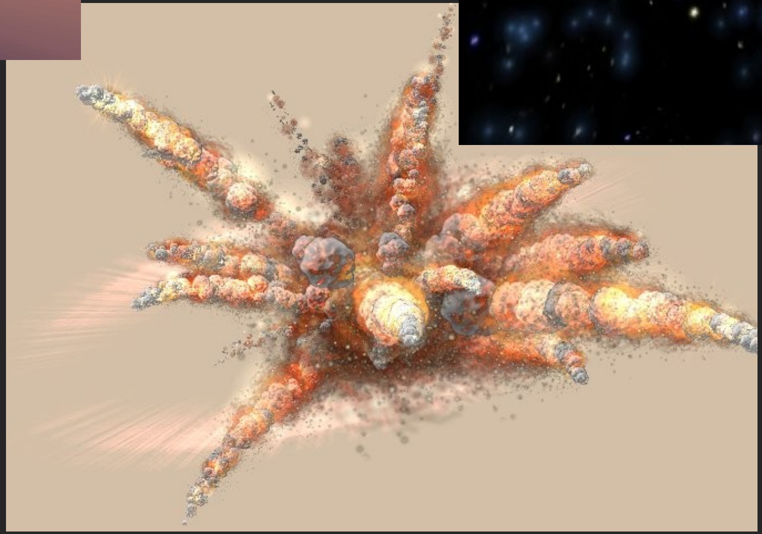
- Motivation
- Motion of a single particle: Equations of motion
  - Use of an ODE solver
- Motion of many particles
- Forces
  - Gravity, drag, spring, local interaction
- Collision: particle vs. plane
  - Detection, response, simple friction

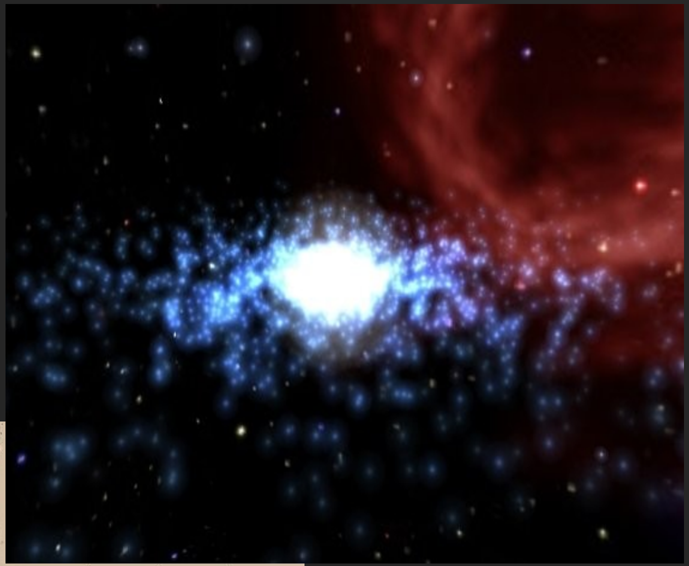
Motivation

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: $\mathcal{P} = (x, v, F, m)$.
- 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:
  - $\mathcal{P}(t)$=$(x, v, F, m)(t) = (x(t), v(t), F, m)$
    - m is constant (not dependent on time).
    - 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 $\dot{\mathcal{P}}(t) = (\dot{x}(t), \dot{v}(t))$.
  - Newton's second law of motion: $F = ma$
  - Important relations: $v = \dot{x} = \frac{dx}{dt}$, $a = \dot{v} = \frac{dv}{dt}$

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

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

Solving equations of motion

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

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

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

$$F(y,t) = (F_0(y_0(t), \dots, y_{n-1}(t), t), \dots, F_{n-1}(y_0(t), \dots, y_{n-1}(t), t))$$

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 $x(t_0) = x_0$ and $v(t_0) = v_0$
- Therefore, we solve the initial value problem of 1st order ODEs:

$$\dot{y} = F(y,t), \quad y(t_0) = y_0$$

Solving equations of motion

- We are given a black-box function ODE solving the initial value problem of 1st order ODEs $\dot{y} = F(y,t)$, $y(t_0) = y_0$.

```
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,                       // timestep
        std::vector<float> & y                // integrated x, 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                   const
                         float   const   float   return
                         float   const   float   return
                         float   const   float   return

                         float   const   float   return
                         float   const   float   return
                         float   const   float   return
```

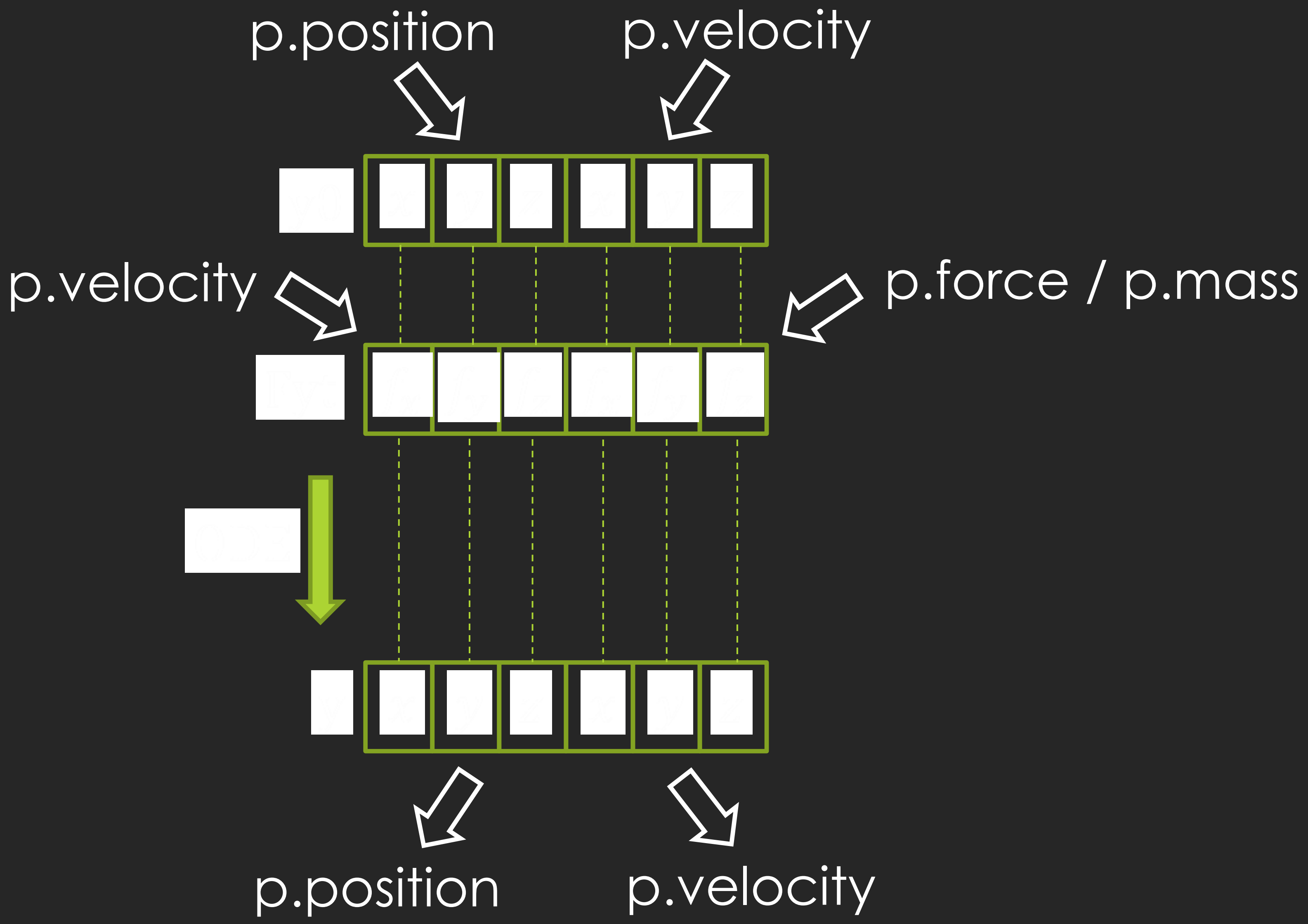
Simulation step for single particle

```
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 particles.
- Particle system in math:

$$\mathcal{P}^n = [\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_{n-1}] = [(x_0, v_0, F_0, m_0), (x_1, v_1, F_1, m_1), \dots, (x_{n-1}, v_{n-1}, F_{n-1}, m_{n-1})]$$

- 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

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 $F = mg$ where
    - m is the mass of the particle.
    - g is a constant vector, e.g., $g = \text{Vector3}(0,0,-10)$.

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

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

    - G is the gravitational constant.
    - m is the mass of the particle.
    - x is the position of the particle.
  - We can handle cases when $|S-x|$ 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 $\vec{f} = k_d(\vec{V} - \vec{v})$, where
  - k_d is the coefficient of drag.
  - $\vec{V}$ is the velocity of the environment (often $\vec{V} = \vec{0}$).
  - $\vec{v}$ is the velocity of the particle.

Spring

- It is a force between two particles p_i and p_j given by Hook's law:

$$F_i = -\left(k_s(|\vec{d}| - d_0) + k_d \dot{\vec{d}} \cdot \frac{\vec{d}}{|\vec{d}|} \right) \frac{\vec{d}}{|\vec{d}|}$$

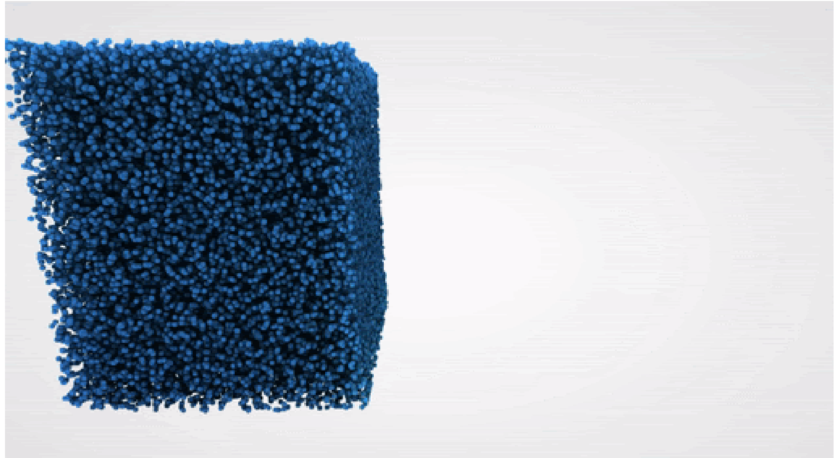
$$F_j = -F_i \quad \text{(3}^{\text{rd}}\text{ Newton's law – action and reaction)}$$

where
  - k_s is the spring constant.
  - k_d is the damping constant.
  - $\vec{d} = x_i - x_j$ is the distance vector between the particles.
  - d_0 is the rest length between the particles.
  - $\dot{\vec{d}} = v_i - v_j$ 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:
    - $O(n^2)$ – all pairs of particles are checked.
    - Space partitioning methods (e.g., octree) are essential for performance.

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 $\mathcal{P} = (x, v, F, m)$.

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

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

Collision response

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

- If $\vec{N} \cdot \vec{F} < 0$, then we cancel the component of $\vec{F}$ orthogonal to the plane:
  - We compute $\Delta \vec{F} = -\vec{F}^{\perp}$, where $\vec{F}^{\perp} = (\vec{N} \cdot \vec{F})\vec{N}$.
  - We update $\vec{F}$ to be $\vec{F} + \Delta \vec{F}$.
  - NOTE: This step should be applied after all external forces (gravity, etc.) were added to the $\vec{F}$ 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 $N \cdot F < 0$, then a friction force F_f is acting on the particle:
  - $|F_f|$ is proportional to $|F^\perp|$, where $F^\perp = (N \cdot F)N$
  - The direction of F_f is opposite to the component v^t of v parallel with the plane, where $v^t = \frac{N \times v \times N}{|N \times v \times N|}$

- Therefore, we define the friction force as $F_f = k_f (N \cdot F) v^t$, 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 1st 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.