Computer Graphics
Practical 5: Physically-based animation

INSA Fourth Year - 2023/2024
Maud Marchal, Glenn Kerbiriou

About this practical

The aim of this practical is to animate a scene using a simple physics-based system. It relies on particles only, which are updated via the application of forces (gravity, springs, etc.). These particles can be rendered directly, or used as control positions by more complex objects.

1 Particle system

A particle is defined at time t by its position $\vec{x}(t)$ and its velocity $\vec{v}(t)$, as well as its mass m.

1.1 Law of motion and system update

Following **Newton's second law**, each particle is accelerated via all the *forces* exerted on it, with the relation:

$$m\frac{d\vec{v}(t)}{dt} = \sum \vec{f(t)}$$

In a discrete system, the state of a particle at time $t + dt$ can then be computed out of its previous state and all the forces applied at time t. This yields to a differential system, which must be solved using an **integration scheme**. In this practical, a simple *explicit Euler* scheme is used to successively compute the new velocity then the new position of each particle:

$$\begin{cases} \vec{v}(t+dt) &= \vec{v}(t) + \frac{1}{m}dt\sum \vec{f} \\ \vec{x}(t+dt) &= \vec{x}(t) + dt\vec{v}(t+dt) \end{cases}$$

1.2 Models of forces

Various forces will be applied in our system, especially gravity and global damping. Two particles could also be linked by a spring, which generates opposite forces on each of its extremities.

Gravity

On Earth, gravity g is uniform and exerts a force, called weight, proportional to the mass. For each particle i, the force is following:

$$\vec{f}_{G\to i} = m_i \vec{g}$$

Damped spring

Springs are used to simulate an elastic behavior that tends to bring two particles back to a given distance from each other, the *equilibrium length*. An ideal spring between particles i and j is parametrized by its equilibrium length l_0 and its stiffness k. The force exerted on particle i is then:

$$\vec{f^k}_{j\to i} = -k\left(\|\vec{x_i} - \vec{x_j}\| - l_0\right)\frac{\vec{x_i} - \vec{x_j}}{\|\vec{x_i} - \vec{x_j}\|}$$

Ideal springs quickly yield to unstable simulation. To prevent this, *damping* is used to reduce the relative motion between two particles. The action of a damper is proportional to the viscous damping coefficient k_c and the relative velocity (difference of velocities of the two particles projected on the spring direction):

$$\vec{f^{k_c}}_{j\to i} = -k_c\left((\vec{v_i} - \vec{v_j})\cdot\frac{\vec{x_i} - \vec{x_j}}{\|\vec{x_i} - \vec{x_j}\|}\right)\frac{\vec{x_i} - \vec{x_j}}{\|\vec{x_i} - \vec{x_j}\|}$$

The total action of a damped spring is computed by summing the different contributions of an ideal spring and a damper:

$$\vec{f}_{j\to i} = \vec{f^k}_{j\to i} + \vec{f^{k_c}}_{j\to i}$$

From **Newton's third law**, the force generated at the other extremity of the spring is simply the opposite :

$$\vec{f}_{i\to j} = -\vec{f}_{j\to i}$$

1.3 Global damping: viscosity of the medium

The movements of all bodies moving within a medium are damped according to its global viscosity. This is mostly relevant in fluids or windy environments. This phenomenon can be modeled by an action proportional to the medium viscosity coefficient c exerted on the opposite direction of the velocity:

$$\vec{f}_{C\to i} = -c\vec{v_i}(t)$$

In animation, global damping can also be used to simply slow the system down thus increasing its numerical stability.

1.4 Collisions

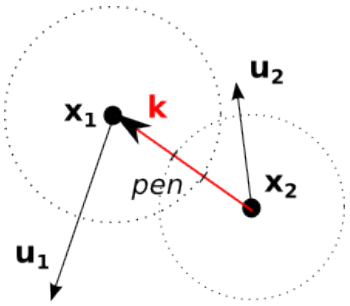
Two kind of collisions are handled in our dynamic system: between a particle and a (fixed) plane, and between two particles. In all cases, two steps are required:

- **Collision detection**. Usually, a broad phase first detects (in an efficient way) if two objects *could* intersect. In that case, a narrow phase actually computes whether the two objects are really inter-penetrating or not. Since we only deal with few and very simple objects, the broad phase will not be implemented in this practical.
- **Response to collision**. In our case, the position and velocity of each particle will be modified to correct the penetration. Part of the energy could be absorbed during the impact, which would slow down the particles.

Let's consider two particles p_1 and p_2 with a position $\{\vec{x_i}\}_{i=1,2}$, a velocity pre-collision $\{\vec{u_i}\}_{i=1,2}$, a velocity post-collision $\{\vec{v_i}\}_{i=1,2}$, a mass $\{m_i\}_{i=1,2}$ and a radius $\{r_i\}_{i=1,2}$.

Collision between two particles

- **Detection**: two particles intersect each other if the distance between their center is less than the sum of their radius:
$$\|\vec{x_1} - \vec{x_2}\| \leq r_1 + r_2$$
In that case, let's set $\vec{k}$ the vector between both centers
$$\vec{k} = \frac{\vec{x_1} - \vec{x_2}}{\|\vec{x_1} - \vec{x_2}\|}$$
and *pen* the interpenetration distance along $\vec{k}$
$$pen = r_1 + r_2 - \|\vec{x_1} - \vec{x_2}\|$$



- **Correction of the positions**: each particle is *moved* along $\vec{k}$, to be in contact but without interpenetration. Both particles move half of the interpenetration distance, with a ponderation if their masses differ (the heavier moves less). If one of the particle is fixed, only the other is displaced.
$$\begin{cases} \vec{x_1} & \leftarrow & \vec{x_1} + \left(\frac{m_2}{m_1+m_2} pen\right)\vec{k} \\ \vec{x_2} & \leftarrow & \vec{x_2} - \left(\frac{m_1}{m_1+m_2} pen\right)\vec{k} \end{cases}$$
- **Correction of the velocities**:
$$\begin{cases} \vec{v_1} & = & \vec{u_1} - \frac{a}{m_1}\vec{k} \\ \vec{v_2} & = & \vec{u_2} + \frac{a}{m_2}\vec{k} \end{cases}$$

with:

$$a = \frac{(1+e)\vec{k} \cdot (\vec{u_1} - \vec{u_2})}{\left(\frac{1}{m_1} + \frac{1}{m_2}\right)}$$

e is the *restitution coefficient*, which ranges from 0.0 (full absorption of the energy by the impact) to 1.0 (full elastic response).

The new velocities of the particles are computed so that the linear momentum and kinetic energy are conserved (in the elastic case).

- Conservation of linear momentum:

$$m_1\vec{u_1} + m_2\vec{u_2} = m_1\vec{v_1} + m_2\vec{v_2}$$

- Conservation of kinetic energy:

$$\frac{1}{2}m_1\vec{u_1} \cdot \vec{u_1} + \frac{1}{2}m_2\vec{u_2} \cdot \vec{u_2} = \frac{1}{2}m_1\vec{v_1} \cdot \vec{v_1} + \frac{1}{2}m_2\vec{v_2} \cdot \vec{v_2}$$

The resolution of this system (2 equations, 2 unknowns) leads to the above result.

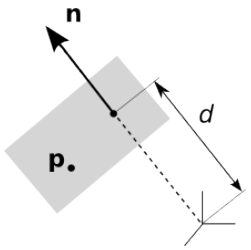
Collision between a particle and a plane

A plane π (supposed infinite and fixed) is defined by the equation:

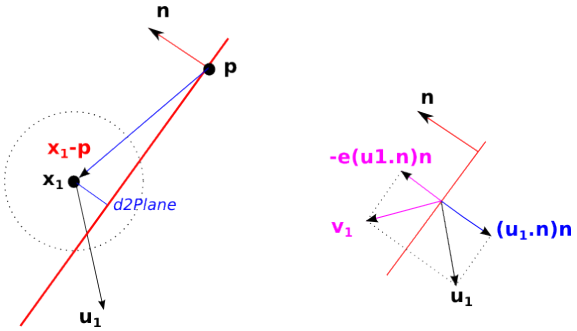
$$ax + by + cz + d = 0 \quad \Leftrightarrow \quad \vec{n} \cdot \vec{p} = d \quad (\pi)$$

where $\vec{n} = (a, b, c)$ is the normal of the plane (normalized), $\vec{p} = (x, y, z)$ is a point of the plane and d is the distance from the plane to the origin.

The distance between a point $\vec{q}$ and the plane is given by $|(\vec{q} - \vec{p}) \cdot \vec{n}|$



- **Detection**: there is penetration if the distance between the particle center and the plane is less than the radius of the particle: $|(\vec{x_1} - \vec{p}) \cdot \vec{n}| \leq r_1$. Let's call this distance *d2Plane*.



- **Correction of the position**: the particle is projected as if the plane fully absorbed the impact, along the plane normal:

$$\vec{x_1} \leftarrow \vec{x_1} - (d2plane - r_1)\,\vec{n}$$

- **Correction of the velocity**:

$$\vec{v_1} = \vec{u_1} - (1+e)\,(\vec{u_1}.\vec{n})\,\vec{n}$$

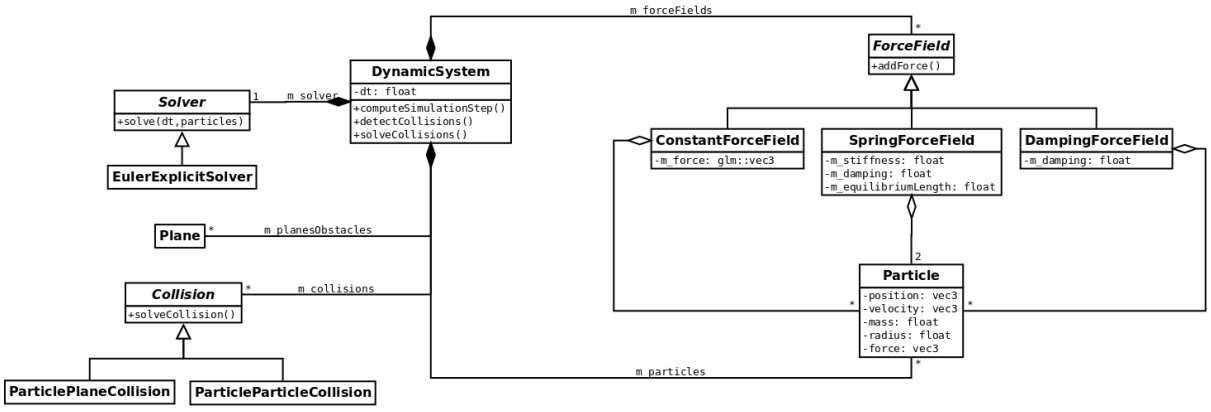
The plane case is actually similar to the particle/particle collision. Just consider that the plane is particle 2, with a null velocity (fixed) and an infinite mass. Vector $\vec{k}$ is colinear with the plane normal.

2 About the provided code

The provided code enables to model a system composed of particles linked together with damped springs, within a medium with gravity and viscosity. Particles are represented by small spheres, with a radius and a mass. Collisions are handled to ensure non-penetration between particles and infinite planes, and between pairs of particles.

All classes are gathered in a `dynamics` subdirectory. The first group of class is the core of the dynamics system:

- The main class is `DynamicSystem`, which gathers all components and run the system resolution over time;
- `Particle` represent a single particle, with its own properties (position, velocity, mass, radius) and the forces applied to it;
- Forces are modeled in the `ForceField` hierarchy, specialized with classes `ConstantForceField` (e.g. for gravity), `SpringForceField` or `DampingForceField` (e.g. for the viscosity of the medium);
- Collisions between objets are detected and solved in the `Collision` hierarchy, specialized by the
  `ParticlePlaneCollision` and `ParticleParticleCollision` classes;
- Finally the integration scheme is delegated to a solver, with a single implementation in `EulerExplicitSolver`.
- This physics code is incomplete, and you will need to write part of it.



The second group of class are `Renderable` objects used for vizualisation:

- `DynamicSystemRenderable` handles the animation loop and control keys, but does not draw anything. Dynamics renderables should be added as children of the DynamicSystemRenderable.
- All particles can be rendered using a single `ParticleListRenderable` object. Rather than looping through all the particles and draw them using `glDrawArrays`, we use *instancing* (see https://learnopengl.com/Advanced-OpenGL/Instancing for more information).
- Each plane is rendered using a `QuadRenderable` instance.
- Finally, each force field (springs, constants, damping) can be displayed using its associated renderable class.

3 Exercise 1: simple particles

All scenes of these exercises are built in `practical5.cpp`. Let's start with the `initialize_scene` function.

- Study how the dynamics system is created, containing two particles and the gravity force field.
- To understand how the dynamic system works, start with the function `computeSimulationStep()`. This is the heart of the system. Now compile and run the animation. Deduce from the function `computeSimulationStep()` why the particles stay put.
- You must first complete the integration scheme in `EulerExplicitSolver::do_solve(...)` to update the velocity and position of each particle. Now, the particles should fall down as expected.

  Several keys are available to control the dynamic system animation (the code is in `DynamicSystemRenderable::do_keyPressedEvent(sf::Event &e)`):

  - `F4`: play/pause the animation
  - `F5`: reset the animation
  - `t`: "tilt" all particles, a brutal way to randomly animate the system...

  More generally, press the `F1` key to display all the available options.

4 Exercise 2: it's spring time

In the `initialize_scene` function, load the scene defined in `springs` function.

- Look at the created system, a 2D net of particles linked with springs. Border particles are fixed.
- Complete the `SpringForceField::do_addForce()` method to implement a damped spring. Forces must be computed then added to the particles at each extremity.
- Run the simulation: the system should dangle... Does it stop? How can you explain this?

- In this practical, a global damping is added to dissipate the velocity of all particles in a medium viscosity (like air friction). Tune the `dampingCoefficient` value of the global damping and the `damping` value of the springs to get a realistic simulation (a balance must be found between stable and too slow...)

5 Exercise 3: collisions

Load the scene `collisions`, containing several particles and a plane. They will collide each other... once you will have done with this exercise.

- Run the animation a first time: nothing is treated...
- Do it again after activating the collision detection in the scene creation using `DynamicSystem::setCollisionDetection(true)`.
  So far, only collisions between particles are implemented. Study the code in `ParticleParticleCollision::do_solveCollision()`, corresponding to the collisions theory.
- Vary the restitution coefficient of the dynamics system between 0.0 and 1.0 to observe its effect.
  Set it to 1.0 and let the simulation run at least 30 seconds: what happens? How could you explain it?
- Based on the above theory and the previous particle/particle case, complete the `ParticlePlaneCollision` class to detect the collision and solve the contact.
- Initialize the horizontal velocity of the particles with a non null value. Run the animation and check if the system behaves as expected (be logical).

Collisions are resolved by pair, between particles and planes then among particles. When many collisions occur, is there any guarantee that they will not be any inter-penetration at the end of the iteration? No! Any idea why? (*nb: this is a very difficult problem to resolve!*)
This may be observed by adding one or more planes to the scene, and increasing the number of particles (in our case however, the double loop is quite efficient...).

6 Exercise 4: let's play pool

- In this last exercise, load the scene `playPool` by calling the corresponding function in `practical5.cpp`: two particles are set on an horizontal plane, and bounded by four vertical planes. The first particle can be moved (accelerated) interactively with the arrows keys. Lets' play!
- Carefully study the scene creation as well as the `ControlledForceFieldRenderable` class.

7 Project related

Start thinking about how you could use particle-based physics simulation in your project. For example, you can simulate fabric with a net of particles linked with springs. Be creative !