Homework 3
Particle physics simulation

Advanced computer graphics 2022/23

1 Introduction

The goal of this homework is to get familiar with particle dynamics in physically based animation. Your task is to implement a particle dynamics simulation and visualization.

1.1 Particle dynamics

Particles will be modeled as point masses with position $\mathbf{x}(t)$, velocity $\mathbf{v}(t)$, and mass m. We will use the following system of differential equations (Newton's second law) to model the particles' motion:

$$\dot{\mathbf{x}} = \mathbf{v}, \tag{1}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{F}(\mathbf{x}, \mathbf{v}, m, t)}{m}, \tag{2}$$

where $\mathbf{F}$ is the sum of all forces acting on a particle. You can experiment with a lot of different types of forces, but for this assignment you are only required to implement the following ones:

- $\mathbf{F} = \mathbf{c}$ – a constant force.
- $\mathbf{F} = m\mathbf{g}$ – a gravitational force with the acceleration vector $\mathbf{g}$ with the units $\mathrm{m\,s^{-2}}$.
- $\mathbf{F} = -b(\mathbf{v} - \mathbf{v}_f)$ – a linear drag force[1], where $\mathbf{v}_f$ is the velocity of the fluid, and $b \geq 0$ is the "linear drag coefficient" with the units $\mathrm{kg\,s^{-1}}$.
- $\mathbf{F} = -s\frac{\mathbf{x}-\mathbf{x}_r}{\|\mathbf{x}-\mathbf{x}_r\|^3}$ – a radial force[2] with a position $\mathbf{x}_r$, and strength s with the units $\mathrm{N\,m^2}$.

1.2 Emitter types

This homework specifies two types of emitters:

- **Point emitters**. They have a fixed position in space and emit particles uniformly in all directions.
- **Disk emitters**. They have a fixed position in space, a direction (normal vector) and a radius. They emit particles uniformly across the surface of the disk in the direction of the normal.

1.3 Poisson process

Particle emitters will emit particles with a certain rate $\lambda > 0$ (given in expected particles emitted per second, or $\mathrm{s^{-1}}$). It is important to note that $\lambda = 10$ does not mean that exactly 10 particles will be emitted in every one second interval. To see this even more clearly, consider $\lambda = \frac{1}{\pi}$. How do we emit $\frac{1}{\pi}$ particles every second? Maybe we should emit 1 particle every π seconds? Or 10 particles every 10π seconds? Or generally, N particles in $\frac{N}{\lambda}$ seconds? For this to work, the emitter would have to have a memory of the last burst of particles. It would have to measure the time from the last burst, and whenever it would hit a certain threshold, it would emit a certain number of particles.

We want to model the real world (e.g. particle emission during radioactive decay), where emitters do not have a memory – they are *memoryless*. The only way this works out mathematically is to model the interarrival times – the times between emissions of consecutive particles – as exponentially distributed random variables. This also implies that the number of particles emitted in a given time interval follows a Poisson distribution[3], which gives the Poisson process its name.

[1]https://en.wikipedia.org/wiki/Drag_(physics)
[2]https://en.wikipedia.org/wiki/Coulomb's_law
[3]https://en.wikipedia.org/wiki/Poisson_distribution

To draw samples from the Poisson distribution (in other words, count the number of particles emitted in a given time interval), we can generate interarrival times Δt_i until their sum exceeds the given time interval t:

$$N = \max \left\{ k : \sum_{i=1}^{k} \Delta t_i < t \right\}. \tag{3}$$

To draw samples from the exponential distribution to generate the interarrival times, you can use inversion sampling:

$$\Delta t_i = -\frac{1}{\lambda} \log(1 - \xi_i), \tag{4}$$

where ξ_i are uniform random variables on the unit interval.

You will have to be extra careful with large λ. See the Wikipedia link for a better algorithm.

2 Input

The input to your simulation will be a JSON file, containing descriptions of the particles, emitters, and forces that act on the particles. All units in the input file will be SI units (meters, seconds, kilograms, Newtons, etc.). Here is an example input file:

```
{
  "particles": [
    {
      "mass": [1, 2],            // in kg, uniformly distributed between 1 and 2
      "lifetime": [2, 3]         // in s, uniformly distributed between 2 and 3
    },
    {
      "mass": [5, 6],
      "lifetime": [0.1, 3]
    }
  ],
  "emitters": [
    {
      "type": "point",
      "particles": [0, 1],       // indices of the "particles" list
      "rate": 50,                // expected number or particles in s^-1
      "limit": 10000,            // max number of particles for this emitter
      "velocity": [0, 2],        // initial velocity in m/s,
                                 // uniformly distributed between 0 and 2
      "parameters": {
        "position": [0, 1, 2]
      }
    },
    {
      "type": "disk",
      "particles": [1],
      "rate": 100,
      "limit": 10000,
      "velocity": [3, 4],
      "parameters": {
        "position": [1, 1, 2],
        "direction": [0, 1, 1],
        "radius": 1,
      }
    }
  ],
  "forces": [
    {
      "type": "constant",
      "parameters": {
        "force": [1, 1, 1]
      }
    },
    {
      "type": "gravity",
      "parameters": {
        "acceleration": [0, -9.81, 0] // in m/s^2
      }
    },
    {
      "type": "drag",
      "parameters": {
        "wind": [1, 1, -2],     // fluid velocity in m/s
        "drag": 10              // "drag coefficient" in kg/s
      }
    },
    {
      "type": "radial",
      "parameters": {
        "position": [-1, -1, 0],
        "strength": 3           // strength in N m^2
      }
    }
  ]
}
```

Emitters should choose the particle type randomly from the given list of particle types. The parameters for each type are given in the `particles` list. Some values in the input file are given as pairs of numbers, which means that they are uniformly distributed on the interval bounded by the given pair of numbers.

The `limit` property of an emitter limits the maximum number of particles that can exist at any point in time for that emitter. When a particle dies, the emitter is free to emit a new one.

Other inputs should be clear from the comments and the description in Section 1. If you have any questions, ask in the lab or in the forum.

3 Tasks

3.1 Particle dynamics simulation

Implement a particle dynamics solver that takes the simulation definition from the input file as defined in Section 2 and simulates the creation and movement of particles as described in Section 1. Use the explicit Euler integrator. The simulation should run in real time. This means that you will have to measure the time between consecutive frames and adjust the simulation (the integration step and the Poisson process).

3.2 Visualization

Implement a simple visualization of the simulation. This can be a simple orthogonal projection or a perspective projection. Vary the size of the particles according to their distance from the camera to convey the depth. Vary the color of the particles according to their age.

3.3 *Optional: planar colliders*

Add planar colliders, from which the particles would bounce according to the reflection law. The collisions can be elastic, but you can also add a restitution coefficient. The planes should be given as a list in the input JSON file, where each plane is defined with a position and a normal:

```
{
  "particles": ...
  "emitters": ...
  "forces": ...
  "colliders": [
    {
      "type": "plane",
      "parameters": {
        "position": [0, -5, 0],
        "normal": [0, 1, 0],
      }
    }
  ]
}
```

3.4 *Optional: higher-order integrator*

Use a higher-order integrator, such as the 4th order Runge-Kutta, instead of the explicit Euler.

3.5 *Optional: collisions between particles*

Implement collisions between particles. For this to work, extend the particle definitions with the radius, which will be drawn from a uniform distribution on a given interval. The input file will change as follows:

```
{
  "particles": [
    "mass": [1, 2],
    "lifetime": [5, 10],
    "radius": [0.1, 0.5]        // in m, uniformly distributed between 0.1 and 0.5
  ],
  ...
}
```

The collisions between particles can also be completely elastic, or you can add a restitution coefficient. Take into account the momentum interchange between the colliding particles. Optimize the collision detection with an acceleration structure such as a regular grid, octree or k-d tree.

4 Outputs

The expected output of this homework is a real-time simulation and visualization of a particle system.

5 Grading

This assignment is worth 10 points:

- 2 points for the emitters and Poisson process,
- 2 points for the forces,
- 3 points for the simulation,
- 3 points for the visualization.