c3particles:
Modeling a Particle System in C++

Rosalie Kletzander

Practical Course "Advanced Software Development with Modern C++"
Summer Term 2018
Institute for Computer Science
Ludwig-Maximilians-Universität München

1 Introduction

Particle systems are used in many different areas: most prominently in the entertainment industry in games and movies and for simulations and visualizations scientific research. No matter the area of application, the basic rules governing these systems are the same: the laws of physics. c3particles (cpp particles) implements a model of a particle system in C++ that separates the physical concepts and laws from the underlying graphics library. This enables a mathematical formulation of the forces influencing the particles.

2 Short Recap of Physics

In order to be able to model the physics of a particle system, it is necessary to first understand the basic rules of motion.

Newton's First Law of Motion states:

> Every object in a state of uniform motion tends to remain in that state of motion unless an external force is applied to it.

This means that an object will not move unless it is accelerated by a force, which brings us to Newton's Second Law of Motion:

> The relationship between an object's mass m, its acceleration a, and the applied force F is $F = m * a$.

With this information it is possible to calculate the acceleration of an object by dividing the applied force by the object's mass. The next step is deriving the velocity and location of the object per time step by integration.

The velocity of an object can be calculated by integrating the acceleration over time t.

$$\overrightarrow{v}(t) = \int (\overrightarrow{a})\mathrm{d}t = \overrightarrow{a} * t + C_v \tag{1}$$

C is the integration constant, in this case it is equal to the velocity of t-1 for discrete time steps. Integrating the velocity over t yields the location.

$$\overrightarrow{s}(t) = \int (\overrightarrow{v})\mathrm{d}t = \int (\overrightarrow{a} * t + C_v)\mathrm{d}t = \frac{\overrightarrow{a} * t^2}{2} + C_s \tag{2}$$

Analogous to C_v, C_s is equal to the location at t-1.

These formulas serve as the foundations of the physical model of the particle system.

3 Modeling the Particle System

what are elements of the system and how do we get them (especially the forces)

- elements of the system: particle (newtonian object)
- force
- particle is passive, it doesn't "care" where the force is coming from → need some way to apply a force to a particle
- update at each time step

3.1 Concepts

Particle ≪ *Force* applies a force to a particle using the formulas 1 and 2

Force calc_force(Particle, Particle, Function) calculates a force between two particles using the supplied function, when given the same particle twice, it returns the additive identity

Force gravity(Particle, Particle, List params) a specialization of calc_force for calculating the gravitational force between two particles (is not actually a concept, but shows what calc_force is capable of)

Force accumulate(Particle, ParticleContainer, Function) calc_force for p with each other p in the container and reduce (addition) to one force

Force accumulate(Particle, ParticleContainer, List params, Function) use a predefined function to calculate the forces, pass initializer list for parameters

paragraphForce accumulate(List forces) sum up a set of forces

4 Implementation

- system diagram
- particle system would be fairly useless without visualization, however, this is not part of the formal model and so it is also a separate module in the code
- OpenGL is used, however any graphics library could be utilized with an appropriate interface (Particle Renderer) (also would have to use something other than glm::vec3)

```
void ParticleSystem::update()
{
  // deltaT will always be 1.0 because calculation is based on frames
  for (Particle &p : _particles)
    {
      //v(t) = a*t + v(t-1)
      p.velocity = p.acceleration * 1.0f + p.velocity; //deltaT = 1.0

      //s(t) = (a*t^2)/2 + v(t) + s(t-1)
      p.location = (p.acceleration * 1.0f) / 2.0f + p.velocity +
          p.location;

      //acceleration is not accumulative, but recalculated at each time
          step
      p.acceleration = {0, 0, 0};
    }
}
```

Fig. 1: ParticleSystem::update()

- for each time step (signaled by OpenGL), the Particle System module calculates the new values for all the particles based on the input given by the user control window. These values are then read by the Particle Renderer and used to fill the vertex buffers depending on the visualization selected (e.g as points or cubes)
- then, the OpenGL Rendering Pipeline renders the frame, using the provided shaders
- as soon as the screenbuffer is swapped, the calculation for the next frame is started

4.1 Particle System

The physical model contains algorithms for calculating the forces on the particles, and the particles themselves. For each frame, the old values of the particles are read and used to update to the new values. The functions used to update the values are taken directly from 1 2

Algorithms

- concepts and expressions come to fruition here
- differentiation between "external" forces and inter-particle forces
- apply force " << "
- calc_force
- accumulate
- specializations of calc_force, e.g. gravity

Particle Container The particles need to be stored in some data structure. At the moment, this is a simple std::vector¡Particle¿ which provides all the needed operations.

4.2 User Control Window

The user controls are implemented with gtk. The control window runs in a different thread that fills a C struct with the values set by the user. These values are then read by the system in order to calculate the desired forces.

4.3 Particle Renderer

4.4 Shaders

4.5 Graphics Engine (OpenGL)