c3particles:
Modeling a Particle System in C++

Rosalie Kletzander

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.

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

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

In order to model a particle system, it is necessary to understand the different underlying concepts.

3.1   Concepts

A particle system is actually a fairly simple construct. It contains objects, its "particles", which are understood to be objects that behave according to newtonian physics, i.e. "newtonian objects". They have a location, velocity, acceleration and mass. According to Newton's laws, an object will not change its motion unless it is accelerated by a *force*. As a result, it is necessary to have the concept of a force.

In fact, these two concepts, "newtonian object" and "force" are sufficient for modeling a particle system. The basic .. expressions ..:

Particle $\ll$ 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

3.2   Time

- – discrete time steps
- – each time step (frame) triggers a new set of calculations

4   Implementation

c3particles contains several separate modules (Figure 1. The Particle System module contains the physical model and uses input given by the user to select forces. It updates the particles that are then read by the Particle Renderer, which uses the location to calculate the vertices and faces that need to be drawn. It passes the vertex buffers of all the particles to the OpenGL Rendering Pipeline, which processes them accordingly. It then writes the framebuffer to the screen and triggers a new calculation.
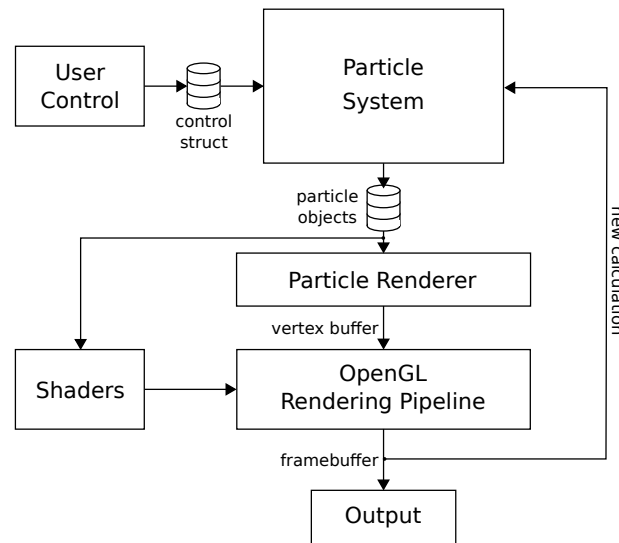

Fig. 1: c3particles system diagram

4.1   Particle System

The Particle System module contains the physical model of the particle system and the particle objects (Figure 2) 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

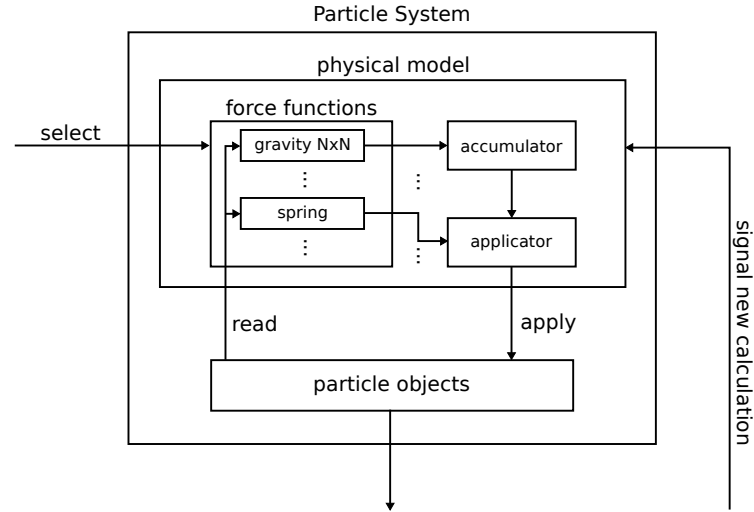– concepts and expressions come to fruition here

Fig. 2: detailed diagram of the particle system

- differentiation between "external" forces and inter-particle forces
- apply force " $<<$ "
- calc_force
- accumulate
- specializations of calc_force, e.g. gravity

The concepts and expressions defined above are incorporated into the system's algorithms. The core algorithm is calc_force (Listing **??**), which takes a lamda defining the calculation of a force between two particles.

This enables very straightforward definition of any force between two particles. It can be used to describe forces on the fly, or to create pre-defined force functions, such as gravitational force

For the calculation of the forces it is helpful to split them logically into "inter-particle forces" and "external forces". The former refers to the forces that exist between all pairs of particles (e.g. gravitational forces) and the latter refers to forces that are applied to each particle independently of all others (e.g. wind).

Particle Container The particles need to be stored in some data structure. At the moment, this is a std::vector 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.

```
1  void ParticleSystem::update()
2  {
3    // deltaT will always be 1.0 because calculation is based on frames
4    for (Particle &p : _particles)
5      {
6        //v(t) = a*t + v(t-1)
7        p.velocity = p.acceleration * 1.0f + p.velocity; //deltaT = 1.0
8
9        //s(t) = (a*t^2)/2 + v(t) + s(t-1)
10       p.location = (p.acceleration * 1.0f) / 2.0f + p.velocity +
11           p.location;
12       //acceleration is not accumulative, but recalculated at each time
             step
13       p.acceleration = {0, 0, 0};
14     }
15 }
```

Fig. 3: ParticleSystem::update()

4.3   Particle Renderer

The particle renderer iterates over the particles in the particle system and fills the vertex buffers for each one. How the vertex buffers are filled depends on the function called. At the moment, it is possible to render the particles as pixels with no depth perception and uniform size (*c3p::ParticleRenderer::renderPoints* or as cubes (*c3p::ParticleRenderer::renderCubes*). The size of the cubes depends on the size of the particle (which, at the moment is dependent on its mass). The vertex buffers are then passed to the OpenGL Rendering Pipeline.

4.4   Shaders

4.5   Graphics Engine

The graphics engine is only secondary for this project. OpenGL was chosen because of its wide-spread use, however, with an appropriate particle renderer, any one could be used.

4.6   Optimization

– force matrix?
– parallelize particle calculation (works very well, because applyforce only updates acceleration, so location and velocity are unchanged until all forces have been calculated)

```
1   // calculates a force between two particles using the function ff
2   // when given the same particle twice, it returns the identity
3   Force calc_force(
4       const Particle &p1,
5       const Particle &p2,
6       std::function<Force(const Particle &p1, const Particle &p2)> ff)
7   {
8     if (&p1 == &p2)
9       { return glm::vec3(0, 0, 0); }
10    return ff(p1, p2);
11  }
```

Fig. 4: calc_force function

5   Future Work

– binary space partitioning

```cpp
// uses calc_force to calculate gravitational force between two particles
Force gravity(const Particle &p1,
         const Particle &p2,
            std::initializer_list<float> params)
{
  Force result = calc_force(p1,
                  p2,
                   [p1, p2](const Particle &, const Particle &) {
      glm::vec3 direction = p2.location - p1.location;
      float gforce = (p1.mass * p2.mass) / pow(glm::length(direction),
          2);
      glm::normalize(direction);

      return (gforce * direction);
    });
  // multiply by gravity constant
  for (auto c : params)
    {
      result *= c;
    }
  return result;
}
```

Fig. 5: gravity function