ГБОУ «Президентский ФМЛ №239»

Симуляция жидкости как множества частиц

Годовой проект по информатике

Капитульский Павел, 10-2

1. Постановка задачи

Создать большое количество частиц. Отобразить на экране анимированное взаимодействие между ними.

2. Исходные и выходные данные

2.1. Входные данные

Данные вводятся непосредственно в код программы. Они состоят из количества частиц (натуральное число), полного времени симуляции (положительное число), временного шага (положительное число, меньшее полного времени) и коэффициента потерь (число от 0 до 1). Стоит заметить, что при слишком большом числе частиц вся симуляция замедляется и теряет функциональность. Кроме того, теоретически допускаются числа вне указанных диапазонов, но симуляция в таком случае теряет практический смысл, т.к. она либо пуста, либо просчитывает один шаг и завершается, либо очевидно не соответствует реальности.

2.2. Выходные данные

На экран выводится квадратное окно со стороной 500 пикс., на котором и изображается симулированное поведение частиц.

3. Математическая модель

При расчёте используются четыре формулы: скорости и координат при равноускоренном движении, столкновения с частицей и столкновения со стенкой:

$\boldsymbol{x = x_0 + v_x}$, где $\boldsymbol{x}$ и $\boldsymbol{x_0}$ – координаты по оси $\boldsymbol{x}$ соответственно в конце и начале шага, $\boldsymbol{v_x}$ – скорость по той же оси;

$\boldsymbol{v_x = v_{0x} + a_x}$, где $\boldsymbol{v_x}$ и $\boldsymbol{v_{0x}}$ -- скорости по оси $\boldsymbol{x}$ соответственно в конце и начале шага, $\boldsymbol{a_x}$ – ускорение по той же оси;

$\boldsymbol{\Delta v_x = v'_x \times \sin\alpha + v'_y \times \cos\alpha}$, где $\boldsymbol{\Delta v_x -}$ изменение скорости по оси $\boldsymbol{x}$, $\boldsymbol{v'_x}$ и $\boldsymbol{v'_y}$ -- скорости второй частицы по осям $\boldsymbol{x}$ и $\boldsymbol{y}$ соответственно, $\boldsymbol{\alpha -}$ угол между осью $\boldsymbol{x}$ и отрезком, соединяющим центры сталкивающихся частиц;

$\boldsymbol{v_x = -v_{0x}}$, где обозначения те же, что и в ф. для расчёта скорости.

Если во всех формулах поменять Х на У, то они будут описывать те же процессы, но по оси У. Важно, что горизонтальное ускорение отсутствует, а вертикальное постоянно и равно 10.

4. Структура данных

Так как предусмотрено произвольное количество одинаковых частиц, они реализованы с помощью собственного класса. Объект класса «частица» несёт информацию о координатах и скорости по двум осям. Все частицы хранятся в списке частиц, по которому каждый шаг выполняется прохождение, в процессе которого положения и скорости частиц обновляются в соответствии с мат. моделью и на экране изображаются круги на основании координат частиц.

5. Метод решения

Сначала программа при помощи случайных чисел создаёт заданное кол-во частиц вдоль вертикальной средней линии экрана с небольшими горизонтальными отклонениями. Все частицы в момент создания заносятся в список частиц. Затем скорости всех частиц пересчитываются в соответствии с формулами и умножаются на коэффициент потерь, затем координаты частиц меняются с учётом новой скорости, после чего к некой переменной учёта времени прибавляется временной шаг. Затем эти

действия повторяются, пока переменная учёта времени меньше полного времени симуляции.

6. Листинг программы

```
from tkinter import *
from random import *
scale = 500
g = 10
step = 0.02                    # временной шаг в секундах
size = 1
r = scale / 100 * size
n = 200                        # всего частиц
total = 10                     # полное время в секундах
things = []                    # пустой список частиц
loss = 0.9                     # коэффициент потерь



class Particle:
    def __init__(self, x, y):
        self.shape = canvas.create_oval(x - r, y - r, x + r, y + r, fill='black')
        self.x, self.y = x, y
        self.vx, self.vy = 0, 0


    def move(self):            # метод расчёта позиции
        self.vy += g * step
        if (self.x > scale - r) or (self.x < r):
            self.vx *= -1 * loss
        if (self.y > scale - r) or (self.y < r):
            self.vy *= -1 * loss
        for i in things:
            d = (self.x - i.x) ** 2 + (self.y - i.y) ** 2
            if (d > 0) and (d < 4 * r * r):
                self.vx += i.vx * (self.y - i.y) / d + i.vy * (self.x - i.x) / d
                self.vy += i.vy * (self.x - i.x) / d + i.vx * (self.y - i.y) / d
                self.vx *= loss
                self.vy *= loss
        self.x += self.vx
        self.y += self.vy
```

```
def move(t):                    # цикл, следящий за временем
    canvas.delete(ALL)
    for i in things:
        i.move()
        canvas.create_oval(i.x - r, i.y - r, i.x + r, i.y + r, fill='black')
    if t < total:
        canvas.after(int(step * 1000), lambda: move(t + step))



frame = Tk()
canvas = Canvas(frame, width=scale, height=scale, background="white")
canvas.grid()
for g in range(n):
    things.append(Particle(randint(scale / 2 - r, scale / 2 + r), randint(r, scale
- r)))
move(0)
frame.mainloop()
```

7. Пример работы

Входные данные:
шаг 0,02 сек
частиц 200
симуляция 10 сек
коэфф. Потерь 0,9
Вывод:
https://github.com/interkaiser/school_project/blob/master/results/ezgif-2-fc3358fdac.mp4

8. Анализ правильности решения

«Жидкость» не только разливается, но и собирается в капли и даже испаряется. Задачу совпадения с реальными наблюдениями программа выполнила. Хотя такое решение и соответствует действительности, оно далеко не оптимально. При количестве частиц больше 300 даже мощный компьютер начинает испытывать трудности в их перемещении.