Programming Test

Time: 3h

1. Introduction

The subject of this programming test is to simulate the motion of a solid sphere through air and its collision with the ground.

The position **r** of the center of the sphere will be given by its 3d coordinates:

ri where i=1,2,3

By convention we assume that the indices 1, 2, 3 correspond to the x, y, z coordinates respectively.

Similarly its velocity **v** and acceleration **a** will be:

vi, ai where i=1,2,3

The well-known equations of motion are:

vi(t+dt) = vi(t) + ai(t)*dt

ri(t+dt) = ri(t) + vi(t)*dt

where:

t = the current simulation time

t+dt = the next simulation time

dt = the time difference between the current and the next simulation time

The gravitational acceleration **g** is a vector given by:

g1 = 0

g2 = 0

g3 = -gz

where

gz = a positive real number constant given as input to your code

The aerodynamic drag **d** is a vector given by:

di = -c*vi*vi where i=1,2,3

where:

c = a positive real number constant given as input to your code

The acceleration is the sum of the gravitational acceleration and the aerodynamic drag:

ai=gi+di

2. Directions for the programming tasks

Here are some directions for the programming tasks below:

-you can use the standard C++ libraries including STL with the exceptions described below

-you can use all the containers in the STL library (vector, map, list, etc.) but not the algorithms

-you cannot use any other library like “boost”, etc.

-there is a code skeleton provided to you in the file:

MotionSimulation.cpp

that you should use.

For tall the tasks below, the simulation starts at

t=0

The following C++ definitions will be used in the tasks below:

```
struct SimulationConfiguration
{
	double gz;			// gravitational constant
	double c;			// aerodynamic coefficient
	double dt;			// time step
	double stopTime;		// stop time of the simulation
	double ballRadius;		// radius of the ball
	double timeAccuracy;		// time accuracy of the collision detection
};

typedef vector<double> Vect3D;
```

All the positions, speeds, accelerations and other 3d vectors will be represented by the type Vect3d defined above.

There might be some information missing from the tasks descriptions below. This is intended, especially for the later tasks since this is similar to the real life. You should make reasonable assumptions for the missing information.

If you are unsure of your assumption, please put a comment at the code affected and state your assumption.

The code should be easy to read. The performance should be reasonable but it’s not required that the code is optimized for performance.

The tasks below are designed to be implemented in order.

3. Programming tasks

1. Basic motion simulation.

Write a function that takes as input a SimulationConfiguration and the initial position of the ball

and simulates the motion of the ball from time = 0 until the stopTime defined in the SimulationConfiguration.

The time is incremented by dt for each new time step of the simulation.

We assume that the ball will not collide with any object during the simulation.

The function should write at the standard output (std::cout) the following values separated by comma:

time,r0,r1,r2,v0,v1,v2,a0,a1,a2

Modify the function "Task1" to call your simulation function using the predefined entities "simConfig1" and "GetInitialPosition1".

2. Wind simulation.

We want to model the influence of the wind on the motion of the ball through the air.

The wind is a 3D vector denoted by **w** that depends on the position of the ball. It affects the aerodynamic drag which now becomes:

di = -c*(vi+wi)*(vi+wi)   where i=1,2,3

The wind dependence on the position is not known when your code will be compiled. You should assume that someone in your team will provide your simulation code as a DLL to external users. The users will not have access to your simulation code but will need to define the wind dependence of the position.

2.1. Propose a C++ representation of the wind which, as mentioned before, will depend of the position of the ball. You do not need to write the code for the DLL provided to the external users.

2.2 Create a new simulation function similar to the one for the first task which will take as additional input your proposed C++ representation of the wind and will simulate the motion of the ball.

3. **Collision with the ground.**

We assume that the ground is at height equal to 0 and is perfectly elastic, meaning that the horizontal velocity of the ball is conserved through the collision to the ground.

When the ball collides with the ground during a simulation step the code should:

-determine the collision time with an accuracy given by:

SimulationConfiguration::timeAccuracy

-insert a new time step at the collision time. Add to the output a new column with the string "ground collision" at the collision time step.

If the motion equations imply several collisions with the ground during the same time step, only one should be reported.

4. **Multiple balls.**

This is a bonus task that should be done if the time allows.

Write a code that simulates N balls simultaneously. There will be a single configuration for all of them. The initial positions will be given by a vector.

The balls can occupy the same space during their motion but they will not affect each other.

The output will modified as follows:

-the first column will be the no. of the ball

-there will be a new column at the end that will contain the string "ball_intersections:" when the current ball intersects other balls, followed by the indexes of those balls.

-the intersection detection can be approximate: we assume that during the time step the balls occupy only the position at the beginning of the time step.

-the time in the time column should be increasing.

Note: for this task you can use "std::sort" or other sorting algorithms from the "STL" library.