Name: **SID:** **LAB Section:**

Lab 9 - Part 1: Particle Simulations

In particle simulations, each particle's dynamic state (position, velocity, acceleration, etc) is modeled independently of the particle's visual state (color, shape, texture, etc). The frame rate of the dynamic update may be different from the rendering frame rate.

Read the accompanied document (particle.pdf) and answer the questions.

You may assume the following variables are available for each particle.

- **m**: mass of a particle
- **x**: position of a particle
- **v**: velocity of a particle
- **f**: force applied on a particle

1. Write the explicit Euler formulation for a particle with properties given above:

a. What does h represent in this equation?

b. Select <u>more</u> or <u>less</u>: Smaller steps typically result in <u>more</u>/<u>less</u> physically accurate and stable solutions, but require <u>more</u>/<u>less</u> iterations.

c. Write the pseudo-code the explicit Euler update, you may assume the availability of the particle variables

```
void Euler_Step(float h)
{



}
```

2. Write down the 3D gravity force applied on a particle in terms of g=9.8 and particle variables.

$F_{gravity}$= [, ,]

3. According to the collision handling definition in the accompanying document, the particles should be reflected when they hit the ground. Select True or False, correct the sentence if False.

(T/F) The y-coordinate of a particle's position can be used to detect the collision with the ground.

(T/F) If the particle is above the ground level, the y-coordinate of the particle's position should be set to 0.

(T/F) The z-coordinate of the particle's velocity should be inverted (v_z=-v_z) if the particle is below the ground and it's v_z is less than 0.

(T/F) The damping coefficient is used to control the bounciness of particles when they hit the ground. The y-component of the velocity should be changed according to this coefficient.

(T/F) The coefficient of restitution is applied to the tangential velocity of the particles to create an effect of friction.

(T/F) Damping and restitution should only be applied if the particle is below the ground and its velocity is pointing downwards ($v_y < 0$).

(T/F) Both damping and restitution coefficients are selected to be between -1 and 1.

4. We can draw a line showing the particle trail the simulation. For this purpose, one can trace the earlier positions of a triangle or find a point in the direction of the velocity of a particle and draw a line from this point to the particle position. Given the particle variables above find a point x_{old} that is s*|v| away from the position x of the particle in the direction of its velocity v.

x_{old} =

5. Given that x and x_old are vec3, write the OpenGL code that draws a line from x to x_old:

```
glBegin(__________________________);



glEnd();
```

Lab 9 - Part 2: Implementation

Here is a brief outline of what you'll need to do in this lab. See next pages for details.

- ❏ Download the skeleton code and compile/run it.
- ❏ Create a particle class/struct.
- ❏ Add member functions to simulate particles and handle collisions.
- ❏ Add global variable to keep a list of particles.
- ❏ Add helper functions to add randomly initialized particles.
- ❏ Use the helper functions to generate some initial particles in **init_event** function.
- ❏ Modify draw_event function to draw particles.
- ❏ Run and see if the particles are properly created and drawn. (you can hide the volcano by pressing 'v' in the executable.)
- ❏ Simulate the particles and handle collisions in **draw_event** function. Run and test again.
- ❏ Modify **draw_event** function so that it will generate new particles at every call.
- ❏ Play with the coefficient of restitution and damping to get different collision effects.
- ❏ Add a time variable and update the color of the particle according to time in draw_event function.

Complete the exercises below and update your code accordingly.

1. Fill the 'Particle' struct definition below with the required variables for its dynamics and its visual state (color) and add it to application.cpp or a new file :particle.h.

```
struct Particle
{









};
```

2. Add the following member functions to the particle class/struct and implement them according to the documentation.

function Euler_Step(float h) # update v and x with an Forward Euler Step (see particle.pdf and Part 1.1)

function Reset_Forces() # reset force to 0 vector;

function Handle_Collision(damping,coeff_resititution) # reflect particle on ground and apply damping and restitution (see Force Sources section of the document and Part 1.3)

3. Create a vector that stores a list of particles globally in application.cpp.

4. Add these global helper function to application.cpp

function Add_Particles(n) # generates n random particles, and **appends to the particle vector.**

Suggestions:

mass of particle: 1

start position of a particle, x: (random(0,0.2) , 0.05, random(0,0.2)

start velocity of a particle, v: (10*x.x, random(1,10),10*x.z)

color of the particle: yellow

* *Play with the numbers to take the simulation to your liking*

function random(k,l) #returns a random **float** between floats k,l (Google 'c++ random float number')

5. At every draw_event call (in application.cpp), your code should:

Step 1. Create new particles. *Implement a helper function, (see 5).*

Step 2. Iterate each particle and update its dynamics according to (correctly ordered) the Table (see 6).

Step 3. Draw each particle p as a line from p.x to p.x+0.04*p.v (with color of the particle).

5.a. Create 10 new particles in the init_event.

5.b. Draw your particles in draw_event function (see comments in the code for exact location). Test your code.

5.c. Create 20 new particles in the draw_event (in the beginning of the 'if not paused' block). This will add 20 new particles at every h seconds. Change the value if you want more.

*You'll implement the 2nd step in the following part.

6. Order the code below so that it will update the particle dynamics at each frame. Implement this as the 2nd step of the algorithm in Part 2.4 in your code

Order: ____

A	**Add forces**
B	**Handle the collisions:** correct velocity and position if it hits the ground.
C	**Set total/accumulated force to 0**
D	**For each particle p:**
E	**Use explicit Euler step** to update the position and the velocity

7. Test your code with the following values and briefly describe their effect in the simulation

Damping:	0	0.5	1
Restitution:	0	0.5	1

Play with these parameters so that the simulation would look as you like.

8. Change color dynamically.

a. Add a new variable duration (d) in the particle class.
b. Initially, the d value of every particle should be set to 0. Change the Add_Particles function accordingly.
c. Update d with the time-step h: d = d + h before you draw the particles.
d. Add a global helper function that returns the interpolated color:

vec3 Get_Particle_Color(float d)
  Your function should return a color according to:

     if d < 0.1:   return **yellow**
     else if d < 1.5: return an interpolated value from **yellow** to **red**.
     else if d < 2: return **red**
     else if d < 3: return an interpolated value from **red** to **grey**.
     else return **grey**.

* You can use (0.5, 0.5, 0.5) for grey.

e. After you update d value of the particle (8.c), update each particle's color with the return value of Get_Color function, called with the particle's duration d as the input parameter.