CS130 - LAB - Particle Simulations

Name: Jianeng Yang SID:

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 accompaning document (particle.pdf) and answer the following 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 update formula for a particle with the properties given above:

■

$x_{i+1} = x_i + hv_i$
$v_{i+1} = v_i + \frac{h}{m}f_i$

2. What does h represent in this equation?

■h is related to the accuracy and stability of the numerical integrator.(STEP)e.g t = ih, where i is the i-th integration iteration that depend on the value of h.

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

■ Smaller steps typically result in more physically accurate and stable solutions, but require more iterations.

4. Write the pseudocode for the explicit Euler update. You may assume the availability of the particle variables.

```
void Euler_Step(float h)
{
    // position step(update)
```

```
    p += h * v;
    // velocity step(update)
    v += (h / m) * f;

}
```

5. Write down the 3D gravity force applied on a particle in terms of $g = 9.8$ and particle variables. ■ $F_{gravity} = [0, -mg, 0]$

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

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

■(F) If the particle is above the ground level, the y-coordinate of the particle's position should be set to 0.
If the particle is below the ground level, the y-coordinate of the particle's position should be set to 0.

■(F) The z-coordinate of the particle's velocity should be inverted ($v_z = -v_z$) if the particle is below the ground and its v_z is less than 0.
The z-coordinate of the particle's velocity should be updated ($v_z = \alpha v_z$) if the particle is below the ground and its v_y is less than 0.
■(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.

The restitution 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.

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

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

■(F) Both damping and restitution coefficients are selected to be between -1 and 1.
Both damping and restitution coefficients are selected to be between 0 and 1.

Important Note: In the particle document, section 4 paragraph 4, it says to apply the force representing friction to the x and y velocities of the particle. It should instead be applied to the x and z velocities when appropriate.

7. We can draw a line showing the particle trail in the simulation. For this purpose, one can trace the earlier positions of a particle 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.
$x_{old} = x + s * v$ or $(x[0] + s * v[0], x[1] + s * v[1], x[2] + s * v[2])$

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

```
glBegin(GL_LINES);
    glVertex3f(x[0], x[1], x[2]); //starting vector
    glVertex3f(x_old[0], x_old[1], x_old[2]); //ending vector
glEnd();
```

or

```
float s = 0.04; // length of line relative to velocity(said work well in part
glBegin(GL_LINES);
    glVertex3f(x[0], x[1], x[2]); //starting vector
    glVertex3f(x[0]+s*v[0], x[1]+s*v[1], x[2]+s*v[2]); //ending vector
glEnd();
```

Part 2: Implementation

Here is a brief outline of what you'll need to do in this lab. See the 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(s) to keep a list of particles.
- Add helper functions to add randomly initialized particles.
- Use the helper functions to generate some initial particles in the init_event function.
- Modify the draw_event function to draw particles.
- Run and test if the particles are properly created and drawn. (you can hide the volcano by pressing the 'v' key while the program window is in focus)

- Simulate the particles and handle collisions in the draw_event function. Run and test again.
- Modify the draw_event function so that it will generate new particles at every call.
- Play with the coefficients of restitution and damping to get different collision effects.
- Add a time variable and update the color of the particles according to time in the draw_event function.

Complete the exercises below and update your code accordingly:

9. Fill the Particle struct definition below with the required variables for its dynamics and its visual state (color) and add it to application.cpp.

```
struct Particle
{
    float m = 0.0; // for the mass of a partcle
    vec3 p; // for the positive of a particle
    // below are varibale have direction ,so its vector
    vec3 v; // for the velocity of a particle
    vec3 f; // for the force applied on a particle
    vec3 c; // for the color of the particle

};
```

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

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

void Reset_Forces() // reset force to 0 vector;

void Handle_Collision(float damping, float coeff_resititution)
// reflect particle on ground and apply damping and restitution
// (see Force Sources section of the document and Part 1.3)
```

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

Add these global helper functions to application.cpp

```
void Add_Particles(int n)
// generates n random particles, and appends to the particle vector.

float random(float low, float high)
// returns a random float between low and high
```

Particle initial value suggestions:

- mass of particle: 1
- start position of a particle, x: (random(-0.2,0.2), 0.05,random(-0.2,0.2))
- 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

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

- Create new particles. Use the Add_Particles helper function.
- Iterate over each particle and update its dynamics according to the Table (correctly ordered).
- Draw each particle p as a line from p.x to p.x+0.04*p.v (with color of the particle).

Create 10 new particles in the init_event.

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

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

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

10. 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: D E C A B

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

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

Damping (0, 0.5, 1):

If the particle is below the ground and $v_y < 0$.
■v_x and v_z will become zero. (the grey color not spread out/spread out the less)

■v_x and v_z will become half of of itself. (the grey color are spread out)

■v_x and v_z will not change.(the grey color are most spread out/spread out faster)

Restitution (0, 0.5, 1):

If the particle is below the ground and $v_y < 0$.
■v_y will become zero (no bouncing)

■v_y will become half of itself in opposite direction.(bouncing)

■v_y will become in opposite direction.(the most bouncing)

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

Change color dynamically.

- Add a new variable duration (d) in the particle class.
- Initially, the d value of every particle should be set to 0. Change the Add_Particles function accordingly.
- Update d with the time-step h: d = d + h before you draw the particles.
- 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.
- After you update the d value of the particle, update each particle's color with the return value of the Get_Particle_Color function, called with the particle's duration d as the input parameter.