Introduction to Visual Computing

Assignment# 4

Physics Engine

March 9, 2020

Description

In this session you should implement a physics engine that simulates objects' movements considering acceleration due to gravity and friction, as well as bouncing due to object collisions.

Objectives

- Put a sphere on the moving plate that you implemented last week
- Change the sphere position according to the plate angle.

Specific Challenges

To compute forces that apply to the sphere and translate them into acceleration, velocity and finally the position of the sphere.

Preliminary steps

In Moodle we've shared a zip file called Week3 Goal Video. It demonstrates what you need to have from the last week's development, to be able to start this week's assignment. Make sure that your code implements the rotation around the X and Z axes as it is shown in the demo. In addition, the speed of tilting should change with the mouse wheel.

Part I

Mastering PVector

Make sure that you have done the preliminary step given in the first page.

The Euclidean vector as an entity that has both magnitude and direction is implemented in Processing with the `PVector` datatype. It stores the components of the vector (x, y for 2D, and x, y, z for 3D). The magnitude and direction, however, can be accessed via `mag()` and `heading()`. Read carefully the specification of `PVector` to master its concept and functionalities: `https://processing.org/reference/PVector`.

The sample code below uses `PVector` to describe the position and velocity of a circle in 2D mode. Run the code and try to understand what each line does.

```
// The Nature of Code
// Daniel Shiffman
// http://natureofcode.com

PVector location;
PVector velocity;

void settings(){
  size(200, 200);
}

void setup() {
  background(255);
  location = new PVector(100, 100);
  velocity = new PVector(2.5, 5);
}

void draw() {
  noStroke();
  fill(255, 10); // 10 is the level of opacity (to get the "trace" effect)
  rect(0, 0, width, height);

  // Add the current speed to the location.
  location.add(velocity);

  if ((location.x > width) || (location.x < 0)) {
    velocity.x = velocity.x * -1;
  }
  if ((location.y > height) || (location.y < 0)) {
    velocity.y = velocity.y * -1;
  }

  // Display circle at its location
  stroke(0);
  fill(175);
  ellipse(location.x, location.y, 16, 16);
}
```

Part II

2D Mover

Step 1 – Basic Structure

The main objective of this week's assignment is to implement and add a physics engine to your game. We propose a program structure that encapsulates this part within a class called `Mover` with `update()`, `display()`, and `checkEdges()` functions. The code below is an example, illustrating the expected structure.

Make a new Processing project and copy the first piece of code given below in it. Then make a new tab in the same project, call it Mover, and copy the second piece of code given below in it. Run the code and try to understand how it is organized.

```
Mover mover;

void settings() {
  size(800, 200);
}

void setup() {
  mover = new Mover();
}

void draw() {
  background(255);

  mover.update();
  mover.checkEdges();
  mover.display();
}
```

```
class Mover {

  PVector location;
  PVector velocity;

  Mover() {
    location = new PVector(width/2, height/2);
    velocity = new PVector(1, 1);
  }

  void update() {
    location.add(velocity);
  }

  void display() {
    stroke(0);
    strokeWeight(2);
    fill(127);
    ellipse(location.x, location.y, 48, 48);
  }

  void checkEdges() {
    if (location.x > width) {
      location.x = 0;
    }
    else if (location.x < 0) {
      location.x = width;
    }

    if (location.y > height) {
      location.y = 0;
    }
    else if (location.y < 0) {
      location.y = height;
    }
  }
}
```

Step 2 – Add Bouncing

Now modify the `checkEdges()` function, so that the ball bounces when hitting the edges (similar to the one in Part I).

Note

The bouncing that you implement for this part is particularly simple, since the edges are straight lines and are aligned with the X and Y axes. Next week you will have to implement a bouncing effect in a more general setting and you will need to build up a rigid understanding of bouncing in geometry terms.

Step 3 – Add Gravity

Try to add gravity to your world. Define it as a `PVector` with `x = 0` and constant Y value, and then add it to the velocity (at each frame).

Part III

3D Mover

Now that you know how to implement a 2D physics engine, the only remaining challenge is to extend it to 3D. In Moodle we have shared a video, called Week 4 - Goal Video. It demonstrates what you are expected to achieve by the end of this part. More precisely you should go through the following steps:

Step 1 – Rolling Ball

Open your `Game.pde` project. Put a sphere on the plate, at its center. When tilting the plate the sphere should change position on it, considering gravity and friction forces.

The block of code below does not fit into your code as it is, but it gives an idea of how gravity and friction should be considered.

```
gravityForce.x = sin(rotZ) * gravityConstant;
gravityForce.z = sin(rotX) * gravityConstant;

float normalForce = 1;
float mu = 0.01;
float frictionMagnitude = normalForce * mu;
PVector friction = velocity.copy();
friction.mult(-1);
friction.normalize();
friction.mult(frictionMagnitude);
```

Note

Here we are implementing the realistic kinetic friction force between two solid bodies, namely the sphere and the plate: $\overrightarrow{F_k} = \mu\overrightarrow{N}$, where the direction of $\overrightarrow{F_k}$ is the opposite of the direction of the velocity of the sphere.

Step 2 – Bouncing Ball

The ball should bounce back naturally when it hits a plate border (as shown in the video Week4 Goal Video).

Note

To go further: You can add an elasticity factor influencing the bouncing velocity of your ball.