Introduction to Visual Computing

Assignment# 5

Physics Engine (II)

March 22, 2016

Description

In this session you should implement a new mode in your game where the user can add obstacles in the shape of cylinders on the plate. Then you implement the bouncing effect when the sphere hits one of the cylinders.

Objectives

- Create a mode where the user can add cylinders at the click location.
- Change the Mover class so that each cylinder becomes an obstacle.

Specific Challenges

To understand the concept of bouncing and implement it. To create a user interaction that allows for transitioning between modes.

Preliminary steps

In Moodle we've shared a zip file called Week4-Goal-Demo. There you find an application that demonstrates what you need to have from the last week's development, to be able to start this week's assignment.

Part I

Crafting a Cylinder Shape

One of the main tasks of this week is to add a cylinder onto your moving plane. The first step is to be able to draw a cylinder, and since it is not among Processing primitive shapes, you need to make it yourself. There are a few methods for crafting a new shape in Processing. To grasp the main trends and learn when to use which method, consider the following example. Below two code snippets are given. While both of them make the same shape (a triangle), one is more efficient than the other. Try to understand the code and analyze the the differences.

```
void settings() {
  size(400, 400, P3D);
}

void setup() {
}

void draw() {
  background(0);
  translate(mouseX, mouseY);
  beginShape(TRIANGLES);
    vertex(0, 0);
    vertex(50, 0);
    vertex(50, 50);
  endShape();
}
```

```
PShape triangle = new PShape();

void settings() {
  size(400, 400, P3D);
}

void setup() {
  triangle = createShape();
  triangle.beginShape(TRIANGLES);
    triangle.vertex(0, 0);
    triangle.vertex(50, 0);
    triangle.vertex(50, 50);
  triangle.endShape();
}

void draw() {
  background(0);
  translate(mouseX, mouseY);
  shape(triangle);
}
```

In the first sketch, the triangle is constructed (with vertices) *every frame*, whereas the second sketch avoids this overhead by constructing the model once (in setup function), and keeping it in the graphic card memory (as Vertex Buffer Objects), so that rendering can be much faster. The first method is called "Immediate" and the second "Retained".

Immediate and retained modes in P3D are complementary: when the geometry is highly dynamic and changes every frame, the former is a better approach as the PShape objects would need to be updated often anyways. In contrast, big static geometries are good candidates for retained mode rendering.

For your application since you only need to make the cylinder shape once and do not need to modify it afterwards, the retained mode is the preferable choice. In the next step you make a cylinder in retained mode.

Step 1 – Open cylinder

Try the code below to get an open cylinder (a pipe). As you see, following the retained method, we create and save the shape once and then load it in draw function every frame.

```
float cylinderBaseSize = 50;
float cylinderHeight = 50;
int cylinderResolution = 40;

PShape openCylinder = new PShape();

void settings() {
  size(400, 400, P3D);
}

void setup() {
  float angle;
  float[] x = new float[cylinderResolution + 1];
  float[] y = new float[cylinderResolution + 1];

  //get the x and y position on a circle for all the sides
  for(int i = 0; i < x.length; i++) {
    angle = (TWO_PI / cylinderResolution) * i;
    x[i] = sin(angle) * cylinderBaseSize;
    y[i] = cos(angle) * cylinderBaseSize;
  }

  openCylinder = createShape();
  openCylinder.beginShape(QUAD_STRIP);
    //draw the border of the cylinder
    for(int i = 0; i < x.length; i++) {
      openCylinder.vertex(x[i], y[i] , 0);
      openCylinder.vertex(x[i], y[i], cylinderHeight);
    }
  openCylinder.endShape();
}

void draw() {
    background(255);
```

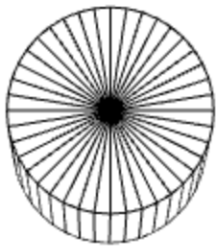
```
    translate(mouseX, mouseY, 0);
    shape(openCylinder);
}
```

Detailed information about how to use `PShape` is available here:

- `https://processing.org/reference/PShape.html`

Step 2 – Closed cylinder

Similarly, create the top and bottom surfaces to close the cylinder. For making surfaces you may need to use `TRIANGLES` or `TRIANGLE_FAN` instead of `QUAD_STRIP` (`https://processing.org/reference/beginShape_.html`).



Note

Alternatively you could create the whole object in a 3D modeling software such as Sketchup or Blender, export it as ".obj" and then import it to Processing as a `PShape` object. You will use this method in a couple of weeks once you have lectures on a 3D modeling software.

Part II

Click & Add Cylinders

In this part you add a new feature to your game: the user can position cylinders on the plate, by simply clicking. The cylinders are added at run time, when the plate may be tilted.

The direct solution is to put the mouse cursor on the plate where you want to position a cylinder and single-click. This solution brings about two remarkable challenges:

- If the plate is already tilted, `mouseX` and `mouseY` values should be transformed to the coordinate of the plate (which is not the same as coordinate of the viewing window anymore). This would require computing a "reverse projection transformation"!
- If the plate is not tilted enough, it would be visually uncomfortable (if not impossible) for the user to click on the plate's surface.

Step 1 – A new interaction mode

To circumvent the above mentioned problems, we recommend another solution that entails making a new mode, "adding-cylinders mode". This is how it should go:

1. When the user holds the SHIFT key, the game stops, and one rectangle at the size of the plate's surface appears.
2. While holding the SHIFT key, the user clicks on the rectangle to position the cylinders.
3. Once releasing the SHIFT key, the game goes back to its state where it was stopped, except that the cylinders appear on the plate and move with it.

A video published on Moodle, called "SHIFT-Cylinders" demonstrates, what is expected from you to achieve in this part.

To store the cylinders' position, we recommend that you use a data structure such as `ArrayList<PVector>`. The PVector in this case is 2D since we only need the center of the bottom surface which is always laying on the plate.

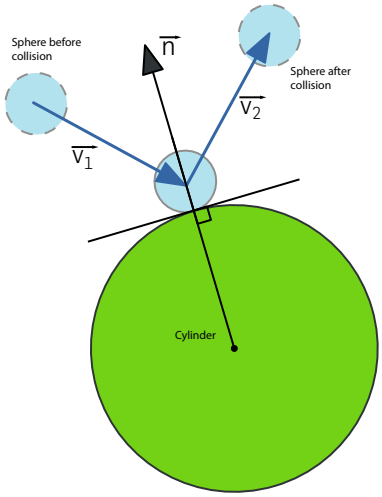
Part III

Bouncing Off Cylinders

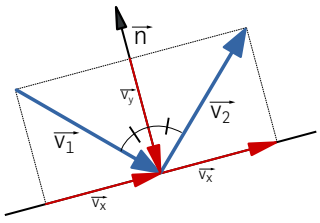
Step 1 – Theory of bouncing

When the sphere hits one of the cylinders it should bounce back. The main challenge here is to compute the velocity vector of sphere after hitting a cylinder, which depends on (1) the velocity vector of sphere before hitting, and (2) the relative positioning of cylinder and sphere, more precisely the vector from cylinder's center to the spheres center at the moment of hitting.

Figure below illustrates the bouncing effect in 2D geometric terms, (which is enough for us since the sphere moves only on the 2D surface of the plate.) In this figure, V_1 is the velocity before collision, V_2 is the velocity after collision, and n is the surface normal ($|n| = 1$).



Bouncing follows the law of reflection which states that the angle of incidence is equal to the angle of reflection. Figure below, shows the law of reflection.



To compute V_2 from V_1 and n, you can use the following formula: $V_2 = V_1 - 2(V_1 \cdot n)n$

Recall: Definition of dot product: $V_1 \cdot n = |V_1||n|\cos(\theta)$, in which θ is the angle between V_1 and n.

Step 2 – Check collision and compute V_2

Now that you know the theory of bouncing, you only need to integrate it into your game. Make a new function in your `Mover` class and call it `checkCylinderCollision`. This function should look at the current position of the sphere and the position of cylinders (as they are already stored in an `ArrayList<PVector>`) to check whether there is a collision. In this case, the velocity after collision should be computed using the formula given in the last step. To implement the formula take advantage of `PVector()` methods such as `dot()`, `mult()`, `normalize()`, and `sub()`.