Introduction to Visual Computing

Assignment# 1.1

Processing: Walking Through Basic Features

February 17, 2020

Description

This session makes you comfortable with the tools that you will use for this course. In this first part you will try some basic elements of Processing.

Objectives

A walk through the basic elements of Processing in 2D rendering mode.

Specific Challenges

Let's start with less challenging tasks, we have time for that.

Preliminary steps

If not already done, download Processing from `https://processing.org/download/` and install it on your laptop.

Part I

Coordinates, Primitives

Let's start by understanding the dimensions and the rendering qualities of the surface on which you would be drawing.

Step 1 – settings(), setup(), and draw()

- Open Processing IDE and try the following piece of code. The `settings()` function is used to define the screen size and the render mode. The `setup()` function is used to initialize the other general parameters such as the background color. We strongly recommend that you use the `size()` function only inside the `settings()`, even though in certain cases it is possible to use it inside the `setup`;

```
void settings() {
  size(200, 200, P2D);
}

void setup() {
  background(255, 200, 0);
}
```

Note

In Processing 3, there are five render modes: the default renderer, P2D, P3D, PDF, and FX2D. In this assignment you can use either the default mode, or P2D which makes use of OpenGL-compatible graphics hardware, runs faster and provides a few extra features such as lighting.

- Add the `draw()` function. The code inside this function runs continuously at every frame, which you can modify using the `frameRate()` function.

```
void settings() {
  size(200, 200, P2D);
}

void setup() {
  background(255, 200, 0);
  frameRate(30);
}

void draw() {
  ellipse(width/2, height/2, 40, 40);
}
```

- Now let's add some action; copy the code below and give it a try.

```
float x = 0.0;

void settings() {
  size(400, 300, P2D);
}

void setup() {
  frameRate(30);
}

void draw() {
  background(255, 200, 0);
  ellipse(x, height/2, 40, 40);
  x += 2;
  if (x > width + 40) {
    x = -40.0;
  }
}
```

Taking it further (optional)

Have you noticed that we called the `background()` function inside `draw()`, and not inside `setup()`? Why do you think we did that? Try to see what would happen otherwise.

Step 2 – noLoop(), loop()

The way to stop the continuous `draw()` function call is to call `noLoop()`, which can be undone with `loop()`. To test, let's add some interactivity to our rolling ball.

```
boolean isMoving = true;

// mousePressed is a built-in Processing function, called every time a mouse button is pressed
// The descriptions of built-in functions are available here: processing.org/reference
void mousePressed () {
  if (isMoving) {
    noLoop();
    isMoving = false;
  }
  else {
     loop();
     isMoving = true;
  }
}
```

Part II

Vertex and Shape

Sometimes you want to be more creative and go beyond the geometric primitives. To create new visual forms you can use a series of coordinates called `vertex` and then group them in a block of code that starts with `beginShape()` and ends with `endShape()`.

Step 1 – Draw a zigzag line

Try the following code:

```
noFill(); // a Processing built-in function to avoid filling the shape
beginShape();
for (int i = 0; i < 20; i++) {
    int y = i % 2;
    vertex(i * 10, 50 + y * 10);
}
endShape();
```

Note

As you saw in the example above, you could skip the `settings()`, `setup()`, and `draw()` functions. In this case the default values are set and the rendering happens only once. However, this practice is not recommended, unless for quickly trying a single feature.

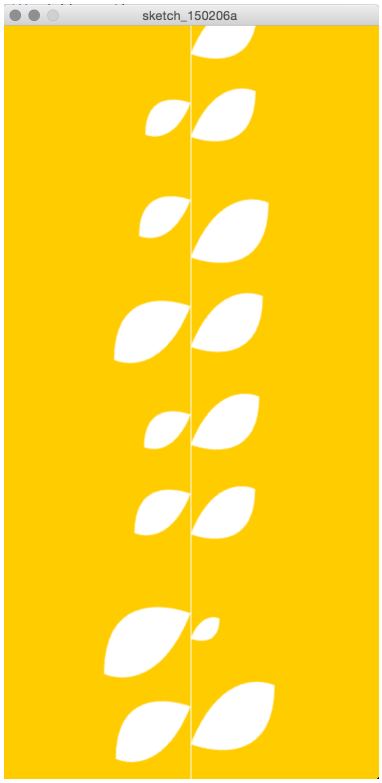
Step 2 – Draw a leaf

You can do better than that and draw curvy shapes like a leaf. Run the following code (don't worry about the `bezierVertex()` function for now).

```
void settings() {
  size(400, 800, P2D);
}

void setup() {
  background(255, 204, 0);
```

```
  noLoop();
}

void draw() {
  translate(width/2, height/2); // position your leaf at the center of the window
  leaf();
}

void leaf () {
  beginShape();
    vertex(100.0, -70.0);
    bezierVertex(90.0, -60.0, 40.0, -100.0, 0.0, 0.0);
    bezierVertex(0.0, 0.0, 100.0, 40.0, 100.0, -70.0);
  endShape();
}
```

In the next part you will see how to draw a plant as in the figure below. Take a moment now to think about how you would do it: what challenges can you foresee?



Part III

2D Transformation

In the last part of this preliminary journey you will learn how to control the coordinate system using functions such as `translate()`, `rotate()`, `scale()`, `pushMatrix()` and `popMatrix()`. Try to complete the code below to get something looking like the plant figure shown in the previous page. You need to use `leaf()`, `rotate()`, `translate()`, and `scale()` functions and put them in the right order.

```
void settings() {
  size(400, 800, P2D);
}
void setup() {
  background(255, 200, 0);
  noLoop();
}
void draw() {
  plant(15, 0.4, 0.8);
}
void leaf() {
  beginShape();
    vertex(100.0, -70.0);
    bezierVertex(90.0, -60.0, 40.0, -100.0, 0.0, 0.0);
    bezierVertex(0.0, 0.0, 100.0, 40.0, 100.0, -70.0);
  endShape();
}
void plant(int numLeaves, float minLeafScale, float maxLeafScale) {
  line(width / 2, 0, width / 2, height); // the plant's stem
  int gap = height/numLeaves; // vertical spacing between leaves
  float angle = 0;

  for (int i = 0; i < numLeaves; i++) {
    int x = width / 2;
    int y = gap * i + (int)random(gap);
    float scale = random(minLeafScale, maxLeafScale);

    pushMatrix();
        // Complete the code!
    popMatrix();

    angle += PI; // alternate the side for each leaf
  }
}
```

Taking it further (optional)

Why are the `pushMatrix()` and `popMatrix()` functions needed? What happens if you change the order of the `translate()`, `rotate()` and `scale()` functions? Try to gain a clear understanding on the concept of Matrix Stack: you will use it a lot, at least in the remainder of this course.