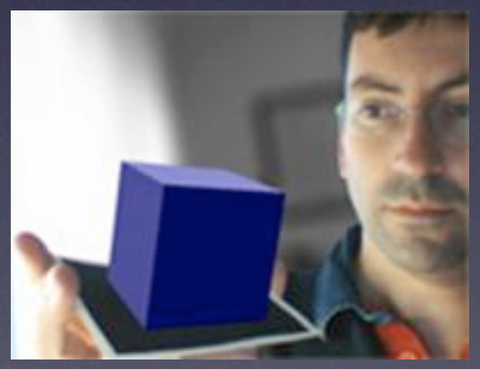
Computer Graphics

Interaction



Jordi Linares i Pellicer
Escola Politècnica Superior d'Alcoi
Dep. de Sistemes Informàtics i Computació
jlinares@dsic.upv.es
http://www.dsic.upv.es/~jlinares

Interaction

- *processing* allows two methods in order to control user interaction:
  - Using a set of system variables
  - Defining a set of callback functions as an answer to certain events
- The use of one method or another depends on the application needs and may even be combined
  - We generally consult system variables inside the `draw()` function. Although the function is continuously executed, the detection of all the events is not, following this mechanism, granted.
  - Using callback functions, thanks to the fact they are managed using a queue, guarantees the consideration of all the events.

Interaction

`mouseX, mouseY`

- They always storage the current mouse position:
  - `mouseX` => x coordinate
  - `mouseY` => y coordinate

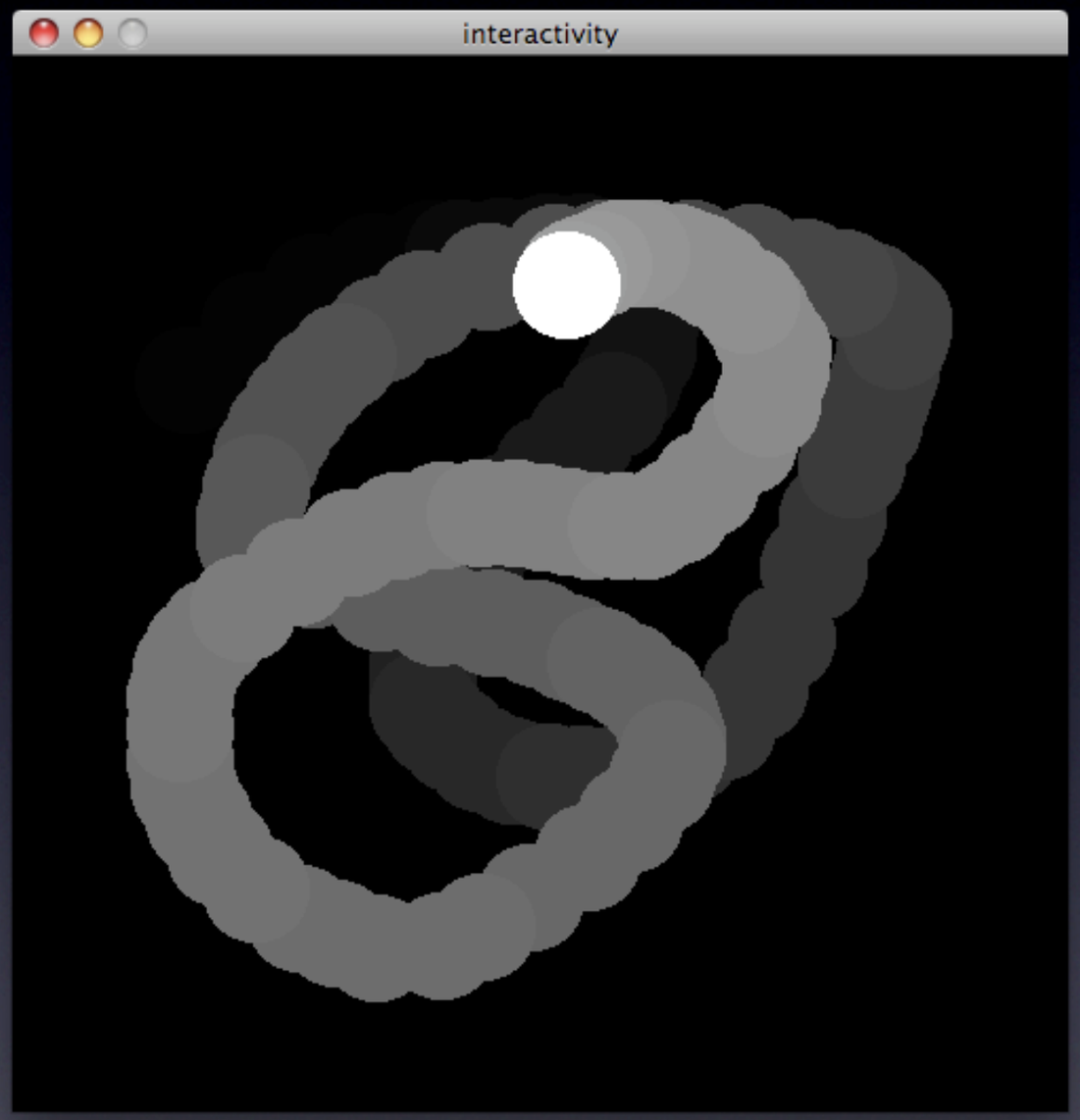
Interaction

```
// A ball that follows the mouse
void setup()
{
  size(500, 500);
  noStroke();

  // Background
  background(0);
}

void draw()
{
  // Fading
  fill(0, 1);
  rect(0, 0, width, height);

  // We draw the ball
  // following mouse position
  fill(255);
  ellipse(mouseX, mouseY, 50, 50);
}
```



Interaction

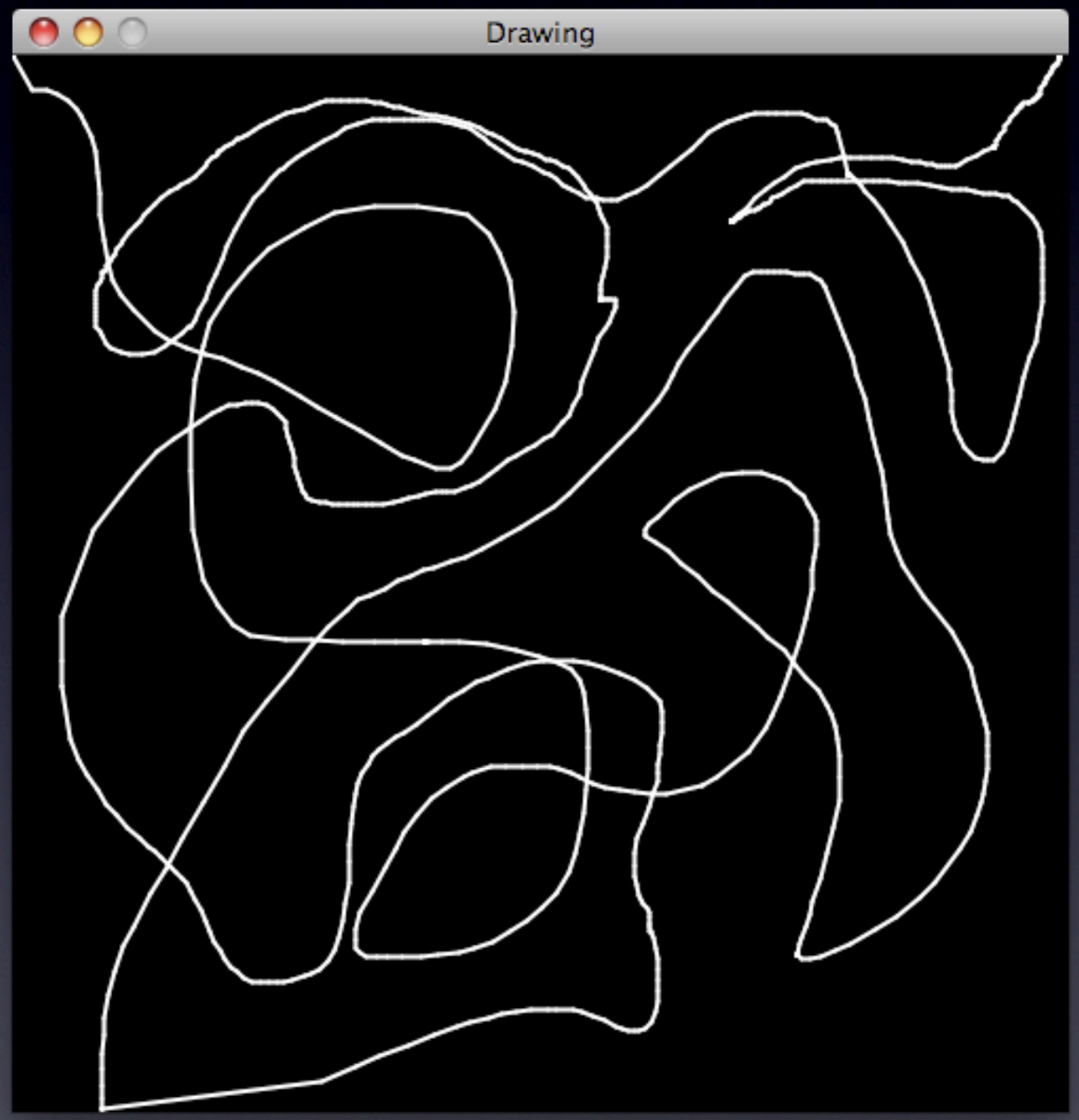
`pmouseX, pmouseY`

- Storage mouse position in the previous frame:
  - `pmouseX` => x coordinate
  - `pmouseY` => y coordinate

Interaction

```
// Free drawing
void setup()
{
  size(500, 500);
  background(0);
  stroke(255);
  strokeWeight(2);
  smooth();  // Uses antialiasing techniques
}

void draw()
{
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```



Interaction

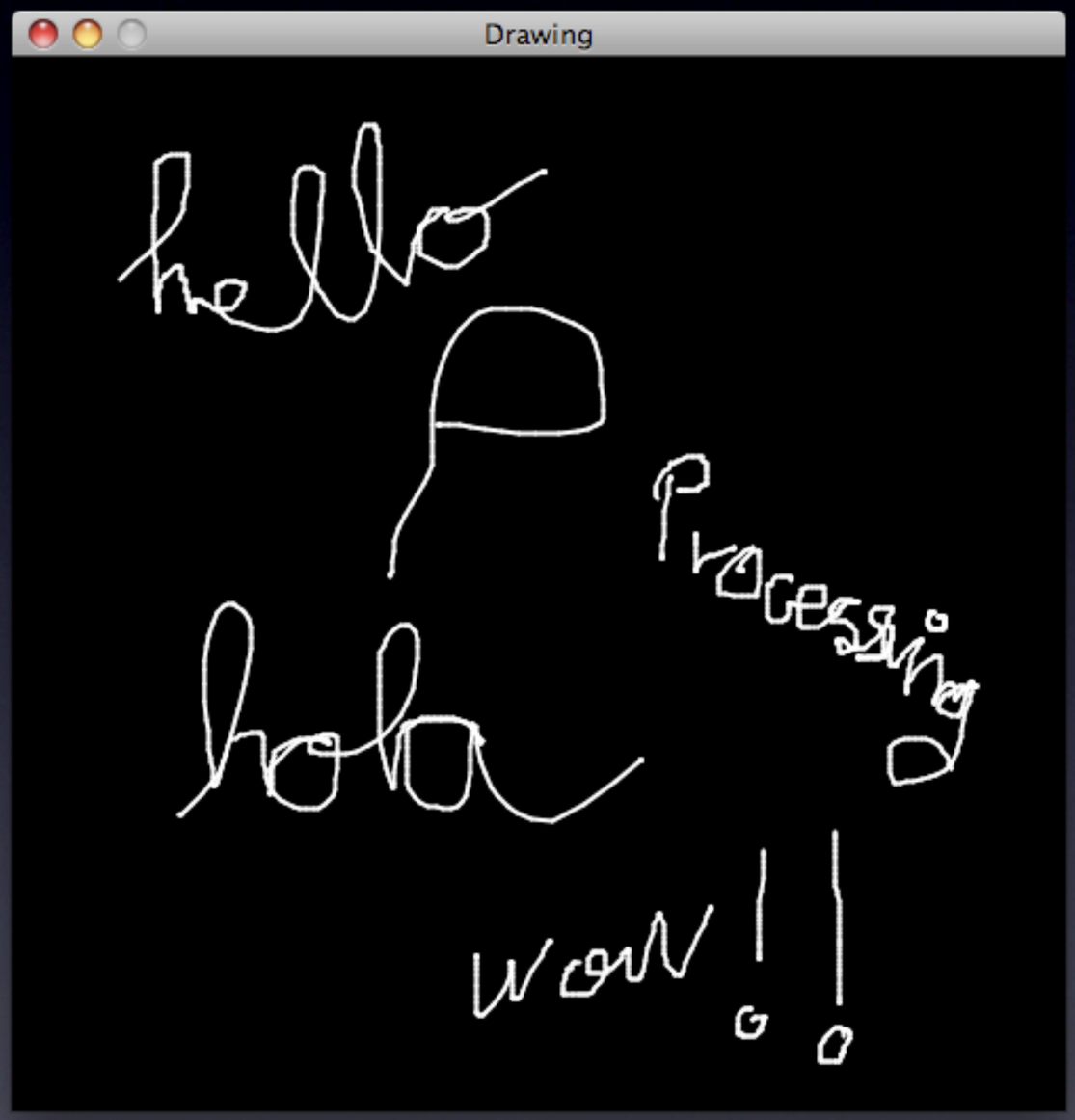
`mousePressed`

- It will be `TRUE` if any button of the mouse is pressed and `FALSE` otherwise

Interaction

```
// Free drawing
void setup()
{
  size(500, 500);
  background(0);
  stroke(255);
  strokeWeight(2);
  smooth();  // Uses antialiasing techniques
}

void draw()
{
  if (mousePressed)
    line(pmouseX, pmouseY, mouseX, mouseY);
}
```



Interaction

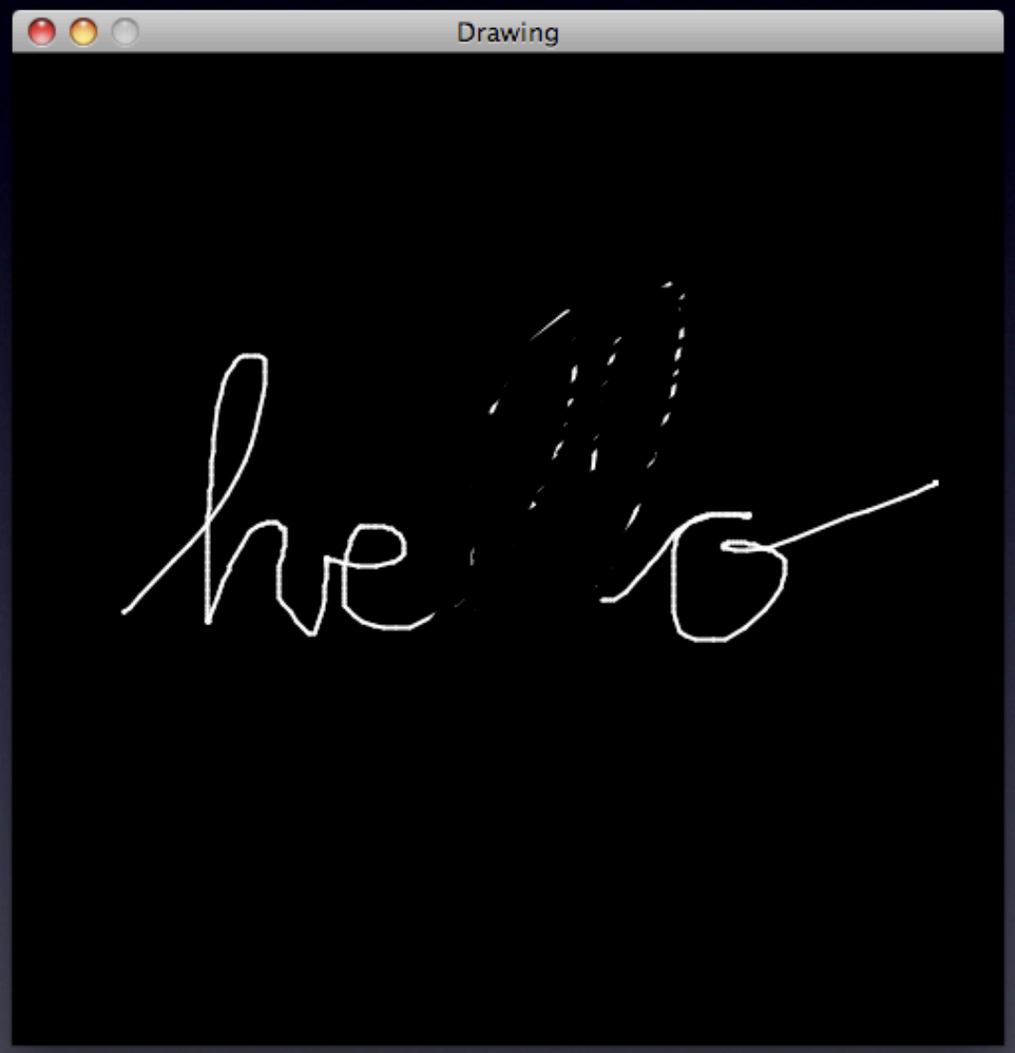
`mouseButton`

- It will be either `LEFT`, `RIGHT` or `CENTER` depending on the button pressed
- Its use must be combined with `mousePressed`

Interaction

```
// Free drawing
void setup()
{
  size(500, 500);
  background(0);
  strokeWeight(2);
  smooth();  // Uses antialiasing techniques
}

void draw()
{
  if (mousePressed) {
    if (mouseButton == LEFT) {  // Draw
      stroke(255);
      strokeWeight(2);
    }
    else {                      // Erase
      stroke(0);
      strokeWeight(4);
    }
    line(mouseX, mouseY, pmouseX, pmouseY);
  }
}
```



Interaction

`keyPressed`

- It will be `TRUE` is a key is pressed

`key`

- It will storage the character of the pressed key
- It could also be one of the following values: `BACKSPACE`, `TAB`, `ENTER`, `RETURN`, `ESC`, and `DELETE`
- If its value is equal to the constant `CODED`, the system variable `keyCode` will inform us which special key has been pressed: `UP`, `DOWN`, `LEFT`, `RIGHT`, `ALT`, `CONTROL`, `SHIFT`.
- Its use must be combined with `keyPressed`

Interaction

Defining callback functions as answer to events

- It consists of adding new functions, called callback functions, that will be executed whenever the associated event is generated
- These functions have specific names, without parameters and return `void`
- In their body, it could be interesting to consult the system variables previously described

Interaction

```
void mousePressed()
```

- It is always called after a button mouse is pressed
- With `mouseButton` we could find out which button has been pressed
- It detects when the user presses (the first click, before releasing)

```
void mouseReleased()
```

- It is always called after a mouse button is released, that is, the second click after it was previously pressed
- With `mouseButton` we could find out which button has been released

Interaction

```
void mouseClicked()
```

- It is always called after finish a complete click, i.e., after pressing and releasing
- With `mouseButton` we could find put the corresponding button
- Before this event `mousePressed()` and `mouseReleased()` will be generated

Interaction

```
void mouseMoved()
```

- It is always called after a mouse movement and with no mouse button pressed
- It is common to used `mouseX` and `mouseY` in its body to consult mouse position

```
void mouseDragged()
```

- It is called after a mouse movement with a mouse button pressed (`mouseButton` can say which one)
- It is common to use `mouseX` and `mouseY` in its body to consult mouse position

Interaction

```
void keyPressed()
```

- It is always called after a key is pressed
- It is common to use `key` in its body to consult which key was pressed (`keyCode` can also be used)

```
void keyReleased()
```

- It is always called after a key is released
- It is common to use `key` in its body to consult which key was released (`keyCode` can also be used)

Interaction

`void keyTyped()`

- It is always called after a typing a key (after pressing and releasing)
- The events `keyPressed()` and `keyReleased()` will be previously invoked
- The continuous pressing of a button will send repeated `keyTyped()` events following what was established in the operative system
- The keys Control, Alt and Shift are ignored by this event

Interaction

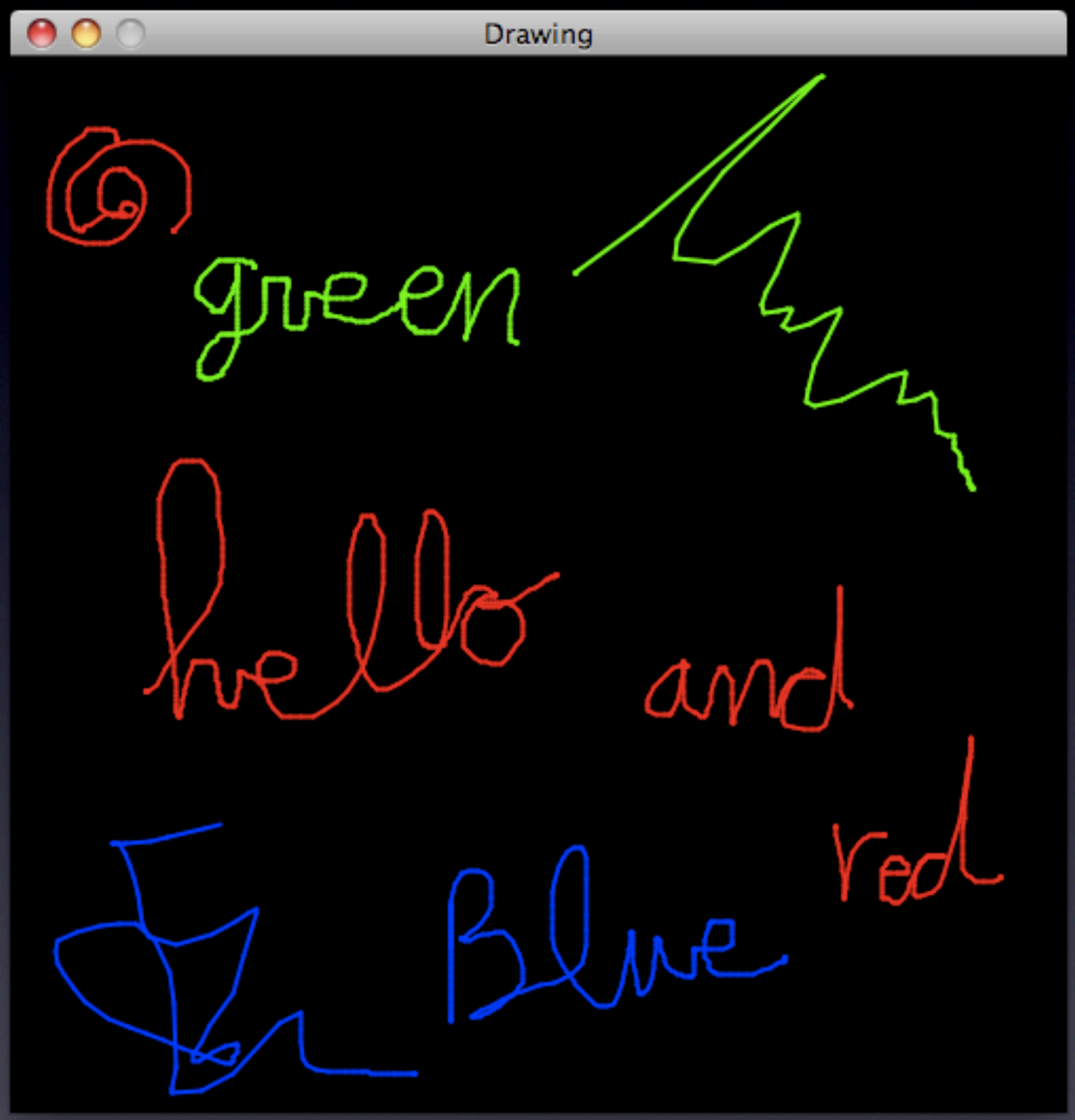
```
// Free drawing
color colorStroke = color(255, 0, 0);

void setup()
{
  size(500, 500);
  background(0);
  strokeWeight(2);
  smooth();  // Uses antialiasing techniques
}

void draw()
{
  if (mousePressed) {
    if (mouseButton == LEFT) {  // Draw
      stroke(colorStroke);
      strokeWeight(2);
    }
    else {
      stroke(0);                // Erase
      strokeWeight(4);
    }

  line(mouseX, mouseY, pmouseX, pmouseY);
  }
}

void keyPressed()
{
      switch (key) {
      case 'r':
      case 'R':
                colorStroke = color(255,0,0);
                break;
      case 'g':
      case 'G':
                colorStroke = color(0,255,0);
                break;
      case 'b':
      case 'B':
                colorStroke = color(0,0,255);
                break;
    }
}
```



Practice 6-1

- Develop the game 'Lunar Landing'

Engine (r)

$F = m \cdot (g + r)$

Gravity (g)

$a = r + g$

$v(t+1) = v(t) + a$

$e(t+1) = e(t) + v(t+1)$

Practice 6-1

- The ship has to be initially placed in the top of the window and with a random horizontal position
- At the top, draw the moon surface as just a rectangle where, in the middle, a landing area has to be drawn
- The ship will be affected by the gravity (as it was the case in our previous examples in the previous chapter)
- When user presses UP, and only in that case, the final force will be the gravity and the engine opposite force (only engine on/off, no intermediate levels)
- By pressing LEFT and RIGHT the ship will be move towards left or right (a simple translation, no additional considerations)

Practice 6-1

- The landing will be successful if:
  - The ship is on the landing platform
  - Its velocity in this moment is inferior to a particular threshold. If this threshold is exceeded, it will be considered that the ship has crashed
- After a successful or unsuccessful landing, the game will be restarted, placing the ship again to its initial position
- Draw the ship as you want (it is advised to use a translation to place it in the screen)
- If the engine is on, a fire can be drawn
- A crashing effect can also be drawn
- Adjust gravity and engine force values as well as the horizontal movements in order to provide playability to the game
- Game interaction can be completely solved inside the `draw()` function by consulting the appropriate system variables

Practice 6-1