Design Programming

DECO1012 & DECO2011

Rob Saunders
Rob Saunders

web: http://www.arch.usyd.edu.au/~rob

e-mail: rob@arch.usyd.edu.au

office: Room 274, Wilkinson Building
Images & Recursion
Parameterised Drawing
void setup() {
  size(100, 100);
  smooth();
  fill(0);
  face(20, 80, 26);
}

void face(int x, int y, int gap) {
  line(x, 0, x, y);               // Nose Bridge
  line(x, y, x+gap, y);           // Nose
  line(x+gap, y, x+gap, height);  // Nose
  int mouthY = (height+y)/2;
  line(x, mouthY, x+gap, mouthY); // Mouth
  ellipse(x-gap/2, y/2, 5, 5);    // Left eye
  ellipse(x+gap, y/2, 5, 5);      // Right eye
Images

- Processing can load images, display them on the screen, and change their size, position, opacity, and tint.
  - The data type for images is called `PImage`.
  - Images are loaded from file using `loadImage()`.
  - Images are displayed using `image()`.
  - Images can be manipulated with `tint()`.
Images can be loaded from files using the `loadImage()` function.

- The single parameter specifies the filename.
  - The filename should be enclosed in quotes and include the format extension, e.g., “pup.gif”, “kat.jpg”, “fsh.png”.
- The image must be in your sketch’s data folder:
  - Add the image using Sketch > Add File; or,
  - Drag the image onto the Processing window.
Displaying Images

- The **image()** function is used to display images on screen.
  - The name parameter must be a **PImage** variable, created using **loadImage()**.
  - The **x** and **y** parameters set the position relative to the current origin.
  - The optional **width** and **height** parameters allow the image to be scaled.

```
image(name, x, y)
image(name, x, y, width, height)
```
Displaying Images

PImage img;
// Image must be in the sketch's "data" folder
img = loadImage("puppy.jpg");
image(img, 0, 0);

PImage img;
// Image must be in the sketch's "data" folder
img = loadImage("puppy.jpg");
image(img, 20, 20, 60, 60);
Tinting Images

- Images are coloured using `tint()`.
  - All images drawn after running `tint()` will be tinted by the colour specified in the parameters.
  - This has no permanent effect on the images.
  - Running the `noTint()` function disables the coloration for all images drawn after it is run.

```
tint(gray)
tint(gray, alpha)
tint(value1, value2, value3)
tint(value1, value2, value3, alpha)
tint(color)
```
Tinting Images

PImage img;
img = loadImage("puppy.jpg");
tint(102);          // Tint gray
image(img, 0, 0);
noTint();          // Disable tint
image(img, 50, 0);

PImage img;
img = loadImage("puppy.jpg");
tint(0,153, 204);  // Tint blue
image(img, 0, 0);
noTint();          // Disable tint
image(img, 50, 0);
Tinting Images

- Like `fill()` and `stroke()` the parameters for `tint()` are based in the current colour space determined by `colorMode()`.
- For example, using HSB tints can be specified in terms of hue, saturation and brightness.
Changing Opacity

- The opacity of an image can also be changed using \texttt{tint()}.
- To make an image transparent without changing its colour, set the tint to white.

```java
PImage img;
img = loadImage("puppy.jpg");
background(255);
tint(255, 51);
// Draw the image 10 times
for (int i = 0; i < 10; i++) {
    image(img, i*10, 0);
}
```
Transparency in Images

- GIF & PNG images keep their transparency when displayed in Processing.
  - GIF images have only 1-bit transparency
  - PNG format supports 8-bit transparency

```java
// Loads a PNG image transparency
PImage img;
img = loadImage("puppy_sprite.png");
background(255);
image(img, 0, 0);
image(img, -20, 0);
```
Recursion
Recursion

- Recursion is a common programming technique, where a function calls itself
  - Recursion is commonly used to process tree-like structures, e.g., HTML files
- Recursion can be used to draw complex fractal-like structures
  - Recursion makes drawing structures with self-similarity simple
void setup() {
    background(255);
    drawV(width/2, height, height/2, 10);
}

void drawV(int x, int y, int size, int num) {
    line(x, y, x - size/2, y - size);
    line(x, y, x + size/2, y - size);
    if (num > 0) {
        drawV(x - size/2, y - size, size/2, num-1);
        drawV(x + size/2, y - size, size/2, num-1);
    }
}

Drawing with Recursion
void setup() {
  size(200, 200);
  background(255);
  fill(0, 32);
  noStroke();
  smooth();
  drawCircle(width/2, height/2, width/2, 10);
}

void drawCircle(int x, int y, int r, int num) {
  ellipse(x, y, r*2, r*2);
  if (num > 0) {
    drawCircle(x - r/2, y, r/2, num-1);
    drawCircle(x + r/2, y, r/2, num-1);
  }
}
void drawCircle(float x, float y, int r, int num)
{
    ellipse(x, y, r*2, r*2);
    if (num > 0) {
        int branches = int(random(2, 6));
        for (int i = 0; i < branches; i++) {
            float a = random(0, TWO_PI);
            float newx = x + cos(a) * 6.0 * num;
            float newy = y + sin(a) * 6.0 * num;
            drawCircle(newx, newy, r/2, num-1);
        }
    }
}
Time & Date
Seconds, Minutes, Hours

- Processing programs can read the value of the computer’s clock.
  - The current second is read with the `second()` function, which returns values from 0 to 59.
  - The current minute is read with the `minute()` function, which also returns values from 0 to 59.
  - The current hour is read with the `hour()` function, which returns values from 0 to 23.
Telling the Time

int s = second(); // Returns values from 0 to 59
int m = minute(); // Returns values from 0 to 59
int h = hour();   // Returns values from 0 to 23
println("The time is " + h + ":" + m + ":" + s);

The time is 14:31:27
Telling the Time

```java
int lastSecond = second();

void draw() {
    if (second() != lastSecond) {
        println(hour() + ":" + minute() + ":" + second());
        lastSecond = second();
    }
}
```

14:32:15
14:32:16
14:32:17
void setup() {
    textFont(loadFont("Monaco-14.vlw"));
    textAlign(CENTER, CENTER);
}

void draw() {
    background(0);
    int s = second();
    int m = minute();
    int h = hour();
    // The nf() function spaces the numbers nicely
    String t = nf(h,2) + ":" + nf(m,2) + ":" + nf(s,2);
    text(t, width/2, height/2);
    saveFrame();
}
void draw() {
    translate(width/2, height/2);
    background(0);
    noStroke();
    fill(80);
    // Angles for sin() and cos() start a 3 o'clock,
    // subtract HALF_PI to make them start at the top
    ellipse(0, 0, 80, 80);
    stroke(255);
    float s = map(second(), 0, 60, 0, TWO_PI) - HALF_PI;
    float m = map(minute(), 0, 60, 0, TWO_PI) - HALF_PI;
    float h = map(hour() % 12, 0, 12, 0, TWO_PI) - HALF_PI;
    line(0, 0, cos(s) * 38, sin(s) * 38);
    line(0, 0, cos(m) * 30, sin(m) * 30);
    line(0, 0, cos(h) * 25, sin(h) * 25);
}
millis()

- Each Processing program counts the time passed since the program started.
  - This time is stored in milliseconds (thousandths of a second) and is obtained with the `millis()` function.
    - 2000 milliseconds is 2 seconds
    - 200 milliseconds is 0.2 seconds

- Use the `millis()` function to trigger events and calculate the passage of time.
Using \textbf{millis()} to Animate

// Uses millis() to start a line in motion
// three seconds after the program starts
int x = 0;

void draw() {
    if (millis() > 3000) {
        x++;
    }
    line(x, 0, x, 100);
}

Using *millis()* to Animate

// Uses millis() to start a line in motion
// three seconds after the program starts
int x = 0;

void draw() {
    if (millis() > 3000) {
        x++;
    }
    line(x, 0, x, 100);
}
Using **millis()** to Animate

// Uses millis() to start a line in motion
// three seconds after the program starts
int x = 0;

void draw() {
    float sec = millis() / 1000.0;
    if (sec > 3.0) {
        x++;
    }
    line(x, 0, x, 100);
}
Using `millis()` to Animate

// Uses millis() to start a line in motion
// three seconds after the program starts
int x = 0;

void draw() {
    float sec = millis() / 1000.0;
    if (sec > 3.0) {
        x++;
    }
    line(x, 0, x, 100);
}
Timing Functions

```
void drawCircle(float x, float y, int r, int num) {
    int startMillis = millis();
    ellipse(x, y, r*2, r*2);
    if (num > 0) {
        int branches = int(random(2, 6));
        for (int i = 0; i < branches; i++) {
            float a = random(0, TWO_PI);
            float newx = x + cos(a) * 6.0 * num;
            float newy = y + sin(a) * 6.0 * num;
            drawCircle(newx, newy, r/2, num-1);
        }
    }
    println("drawCircle() took "+ millis() - startMillis + "ms");
}
```
Lab Exercises
Lab Exercises

› Create a function for drawing a face.
  › Use two parameters to change its position and two more to change the shape. Using your function, draw 9 faces in the display window in a regular 3 * 3 matrix. Use different parameters to give each face drawn a unique shape.

› Draw two images in the display window.
  › Draw three images, each with a different tint.

› Load a GIF or PNG image with transparency and create a collage by layering the image.
Lab Exercises

- Modify the `drawCircle()` example to draw a complex design using recursion.
  - Add colour to your sketch.
- Make a simple clock to run an animation for two seconds at the beginning of each minute.
- Create an abstract clock that communicates the passage of time through graphical quantity rather than numerical symbols.