7/Media

Processing is capable of drawing more than simple lines and shapes. It's time to learn how to load raster images, vector files, and fonts into our programs to extend the visual possibilities to photography, detailed diagrams, and diverse typefaces.

Processing uses a folder named *data* to store such files, so that you never have to think about their location when moving sketches around and exporting them. We've posted media files online for you to use in this chapter's examples: *http://www.processing.org/learning/books/media.zip*.

Download this file, unzip it to the desktop (or somewhere else convenient), and make a mental note of its location.

To unzip on Mac OS X, just double-click the file, and it will create a folder named *media*. On Windows, double-click the *media.zip* file, which will open a new window. In that window, drag the *media* folder to the desktop.

Create a new sketch, and select Add File from the Sketch menu. Find the *lunar.jpg* file from the *media* folder that you just unzipped and select it. If everything went well, the Message Area will read "One file added to the sketch."

To check for the file, select Show Sketch Folder in the Sketch menu. You should see a folder named *data*, with a copy of *lunar.jpg* inside. When you add a file to the sketch, the data

folder will automatically be created. Instead of using the Add File menu command, you can do the same thing by dragging files into the editor area of the Processing window. The files will be copied to the *data* folder the same way (and the *data* folder will be created if none exists).

You can also create the *data* folder outside of Processing and copy files there yourself. You won't get the message saying that files have been added, but this is a helpful method when you're working with large numbers of files.

On Windows and Mac OS X, extensions are hidden by default. It's a good idea to change that option so that you always see the full name of your files. On Mac OS X, select Preferences from the Finder menu, and then make sure "Show all filename extensions" is checked in the Advanced tab. On Windows, look for Folder Options, and set the option there.

Images

There are three steps to follow before you can draw an image to the screen:

1. Add the image to the sketch's *data* folder (instructions given previously).
2. Create a `PImage` variable to store the image.
3. Load the image into the variable with `loadImage()`.

Example 7-1: Load an Image

After all three steps are done, you can draw the image to the screen with the `image()` function. The first parameter to `image()` specifies the image to draw; the second and third set the *x* and *y* coordinates:

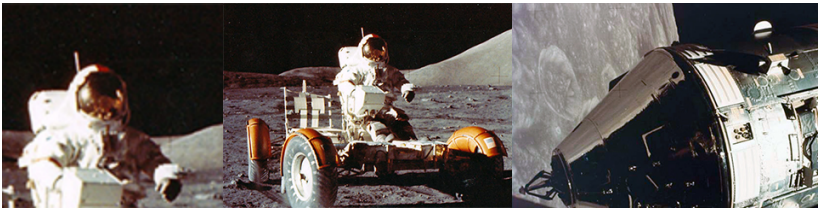
```
PImage img;

void setup() {
  size(480, 120);
  img = loadImage("lunar.jpg");
}

void draw() {
  image(img, 0, 0);
}
```

Optional fourth and fifth parameters set the width and height to draw the image. If the fourth and fifth parameters are not used, the image is drawn at the size at which it was created.

These next examples show how to work with more than one image in the same program and how to resize an image.

Example 7-2: Load More Images

For this example, you'll need to add the *capsule.jpg* file (found in the *media* folder you downloaded) to your sketch using one of the methods described earlier:

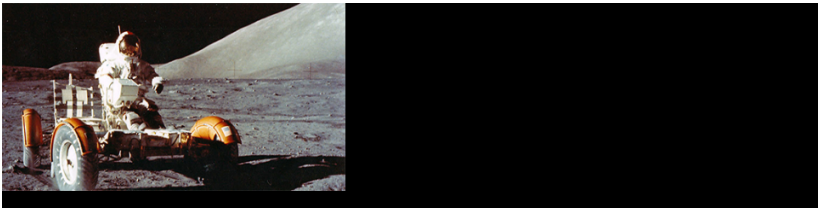


```
PImage img1;
PImage img2;

void setup() {
  size(480, 120);
  img1 = loadImage("lunar.jpg");
```

```
  img2 = loadImage("capsule.jpg");
}

void draw() {
  image(img1, -120, 0);
  image(img1, 130, 0, 240, 120);
  image(img2, 300, 0, 240, 120);
}
```

Example 7-3: Mousing Around with Images

When the `mouseX` and `mouseY` values are used as part of the fourth and fifth parameters of `image()`, the image size changes as the mouse moves:



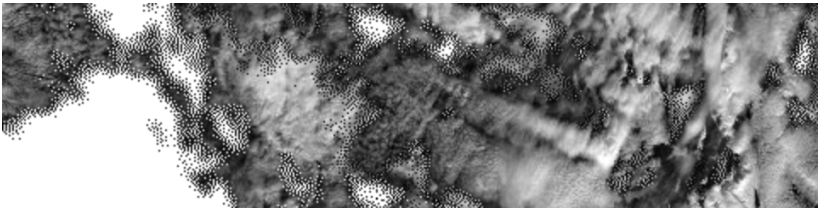
```
PImage img;

void setup() {
  size(480, 120);
  img = loadImage("lunar.jpg");
}

void draw() {
  background(0);
  image(img, 0, 0, mouseX * 2, mouseY * 2);
}
```

When an image is displayed larger or smaller than its actual size, it may become distorted. Be careful to prepare your images at the sizes they will be used. When the display size of an image is changed with the `image()` function, the actual image on the hard drive doesn't change.

Processing can load and display raster images in the JPEG, PNG, and GIF formats. (Vector shapes in the SVG format can be displayed in a different way, as described in "Shapes" on page 97 later in this chapter.) You can convert images to the JPEG, PNG, and GIF formats using programs like GIMP and Photoshop. Most digital cameras save JPEG images that are much larger than the drawing area of most Processing sketches, so resizing such images before they are added to the *data* folder will make your sketches run more efficiently.

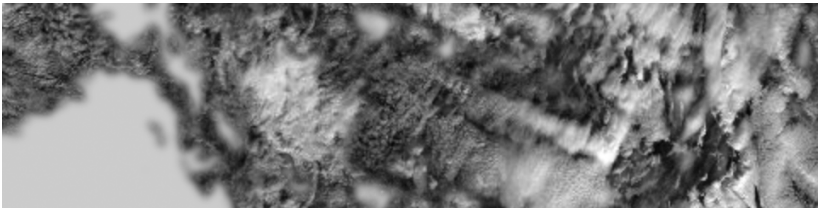
GIF and PNG images support transparency, which means that pixels can be invisible or partially visible (recall the discussion of `color()` and alpha values in Example 3-17 on page 26). GIF images have 1-bit transparency, which means that pixels are either fully opaque or fully transparent. PNG images have 8-bit transparency, which means that each pixel can have a variable level of opacity. The following examples show the difference, using the *clouds.gif* and *clouds.png* files found in the *media* folder that you downloaded. Be sure to add them to the sketch before trying each example.

Example 7-4: Transparency with a GIF



```
PImage img;

void setup() {
  size(480, 120);
  img = loadImage("clouds.gif");
}
void draw() {
  background(255);
  image(img, 0, 0);
  image(img, 0, mouseY * -1);
}
```

Example 7-5: Transparency with a PNG



```
PImage img;

void setup() {
  size(480, 120);
  img = loadImage("clouds.png");
}

void draw() {
  background(204);
  image(img, 0, 0);
  image(img, 0, mouseY * -1);
}
```

Remember to include the file extensions *.gif*, *.jpg*, or *.png* when you load the image. Also, be sure that the image name is typed exactly as it appears in the file, including the case of the letters. And if you missed it, read the note earlier in this chapter about making sure that the file extensions are visible on Mac OS X and Windows.

Fonts

The Processing software can display text using TrueType (*.ttf*) and OpenType (*.otf*) fonts, as well as a custom bitmap format called VLW. For this introduction, we will load a TrueType font from the *data* folder, the *SourceCodePro-Regular.ttf* font included in the *media* folder that you downloaded earlier.

The following websites are good places to find fonts with open licenses to use with Processing:

- Google Fonts
- The Open Font Library
- The League of Moveable Type

Now it's possible to load the font and add words to a sketch. This part is similar to working with images, but there's one extra step:

1. Add the font to the sketch's *data* folder (instructions given previously).
2. Create a `PFont` variable to store the font.
3. Create the font and assign it to a variable with `createFont()`. This reads the font file, and creates a version of it at a specific size that can be used by Processing.
4. Use the `textFont()` function to set the current font.

Example 7-6: Drawing with Fonts

Now you can draw these letters to the screen with the `text()` function, and you can change the size with `textSize()`:


```
PFont font;

void setup() {
  size(480, 120);
  font = createFont("SourceCodePro-Regular.ttf", 32);
  textFont(font);
}
```

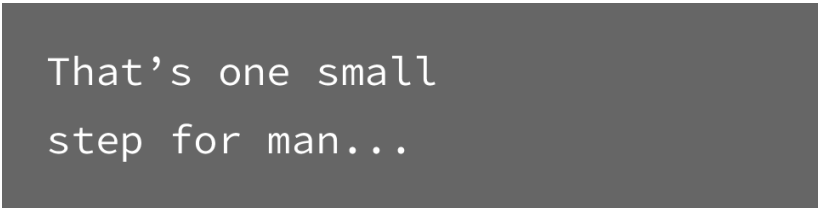
```
void draw() {
  background(102);
  textSize(32);
  text("That's one small step for man...", 25, 60);
  textSize(16);
  text("That's one small step for man...", 27, 90);
}
```

The first parameter to `text()` is the character(s) to draw to the screen. (Notice that the characters are enclosed within quotes.) The second and third parameters set the horizontal and vertical location. The location is relative to the baseline of the text (see Figure 7-1).

Figure 7-1. *Typography coordinates*

Example 7-7: Draw Text in a Box

You can also set text to draw inside a box by adding fourth and fifth parameters that specify the width and height of the box:




```
PFont font;

void setup() {
  size(480, 120);
  font = createFont("SourceCodePro-Regular.ttf", 24);
  textFont(font);
}
```

```
void draw() {
  background(102);
  text("That's one small step for man...", 26, 24, 240, 100);
}
```

Example 7-8: Store Text in a String

In the previous example, the words inside the `text()` function start to make the code difficult to read. We can store these words in a variable to make the code more modular. The `String` data type is used to store text data. Here's a new version of the previous example that uses a `String`:

```
PFont font;
String quote = "That's one small step for man...";

void setup() {
  size(480, 120);
  font = createFont("SourceCodePro-Regular.ttf", 24);
  textFont(font);
}

void draw() {
  background(102);
  text(quote, 26, 24, 240, 100);
}
```

There's a set of additional functions that affect how letters are displayed on screen. They are explained, with examples, in the Typography category of the *Processing Reference*.

Shapes

If you make vector shapes in a program like Inkscape or Illustrator, you can load them into Processing directly. This is helpful for shapes you'd rather not build with Processing's drawing functions. As with images, you need to add them to your sketch before they can be loaded.

There are three steps to load and draw an SVG file:

1. Add an SVG file to the sketch's *data* folder.
2. Create a `PShape` variable to store the vector file.

3. Load the vector file into the variable with `loadShape()`.

Example 7-9: Draw with Shapes

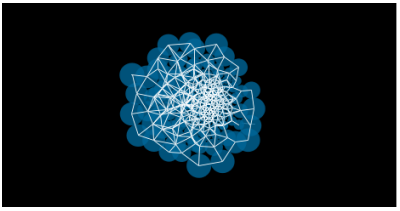
After following these steps, you can draw the image to the screen with the `shape()` function:

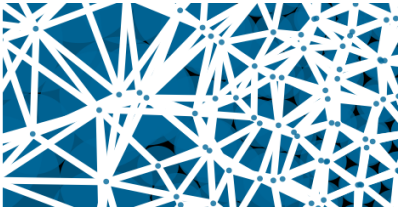
```
PShape network;

void setup() {
  size(480, 120);
  network = loadShape("network.svg");
}

void draw() {
  background(0);
  shape(network, 30, 10);
  shape(network, 180, 10, 280, 280);
}
```

The parameters for `shape()` are similar to `image()`. The first parameter tells `shape()` which SVG to draw and the next pair sets the position. Optional fourth and fifth parameters set the width and height.

Example 7-10: Scaling Shapes

Unlike raster images, vector shapes can be scaled to any size without losing resolution. In this example, the shape is scaled based on the `mouseX` variable, and the `shapeMode()` function is used to draw the shape from its center, rather than the default position, the upper-left corner:



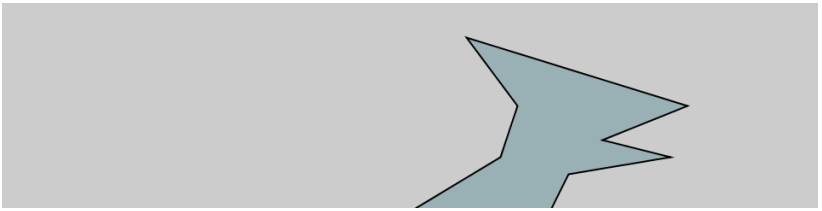


```
PShape network;

void setup() {
  size(240, 120);
  shapeMode(CENTER);
  network = loadShape("network.svg");
}

void draw() {
  background(0);
  float diameter = map(mouseX, 0, width, 10, 800);
  shape(network, 120, 60, diameter, diameter);
}
```

Processing doesn't support all SVG features. See the entry for `PShape` in the *Processing Reference* for more details.

Example 7-11: Creating a New Shape

In addition to loading shapes through the *data* folder, new shapes can be created with code through the `createShape()` function. In the next example, one of the creatures from Example 3-21 on page 29 is built in the `setup()` function. Once this happens, the shape can be used anywhere in the program with the `shape()` function:



```
PShape dino;

void setup() {
  size(480, 120);
  dino = createShape();
  dino.beginShape();
  dino.fill(153, 176, 180);
  dino.vertex(50, 120);
  dino.vertex(100, 90);
  dino.vertex(110, 60);
  dino.vertex(80, 20);
  dino.vertex(210, 60);
  dino.vertex(160, 80);
  dino.vertex(200, 90);
  dino.vertex(140, 100);
  dino.vertex(130, 120);
  dino.endShape();
}

void draw() {
  background(204);
  translate(mouseX - 120, 0);
  shape(dino, 0, 0);
}
```

Making a custom `PShape` with `createShape()` can make sketches more efficient when the same shape is drawn many times.