Make:

Getting Started with p5.js

Making Interactive Graphics in JavaScript and Processing

Lauren McCarthy,
Casey Reas & Ben Fry

7/Media

p5.js is capable of drawing more than simple lines and shapes. It's time to learn how to create images and text in our programs to extend the visual possibilities to photography, detailed diagrams, and diverse typefaces.

Before we do that, we first need to talk a little bit about servers. Up to this point, we've just been viewing the *index.html* file directly in the browser. This works fine for things like running simple animations. However, if you want to do things like load an external image file into your sketch, you might find that your browser doesn't allow this. If you look in the console, you may see an error containing the term *cross-origin*. For loading external files, you will need to run a *server*. A *server* is a program that works as a handler layer. It responds when you type a URL into the address bar, and *serves* the corresponding files to you for you to view.

There are several different ways to run servers. Visit *https://github.com/processing/p5.js/wiki/Local-server* for instructions on how to run a server on Mac OS X, Windows, and Linux systems. Once you have that set up, you are ready to load media!

We've posted some media files online for you to use in this chapter's examples: *http://p5js.org/learn/books/media.zip*.

Download this file, unzip it to the desktop (or somewhere else convenient), and make a 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 copy the *lunar.jpg* file from the *media* folder that you just unzipped into your *sketch* folder.

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 folder.
2. Create a variable to store the image.
3. Load the image into the variable with `loadImage()`.

Example 7-1: Load an Image

In order to load an image, we will introduce a new function called `preload()`. The `preload()` function runs once before the `setup()` function runs. You should generally load your images and other media in `preload()` in order to ensure they are fully loaded before your program starts. We will discuss this in more depth later in the chapter.

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:

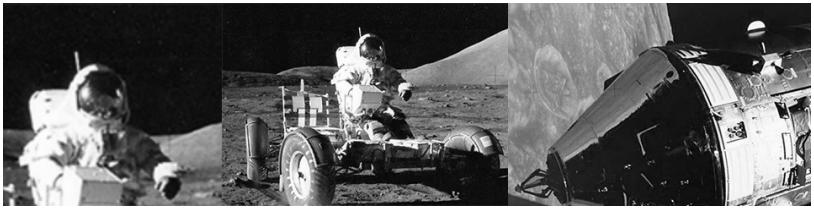
```
var img;

function preload() {
  img = loadImage("lunar.jpg");
}

function setup() {
  createCanvas(480, 120);
}

function 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

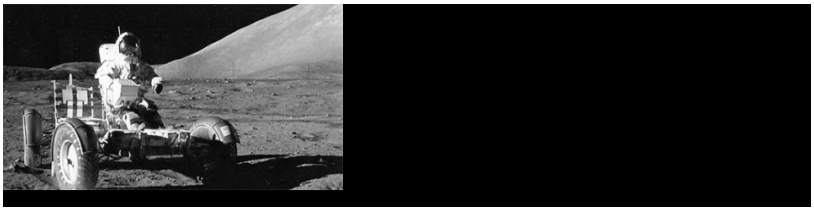
For this example, you'll need to add the *capsule.jpg* file (found in the *media* folder you downloaded) to your *sketch* folder:



```
var img1;
var img2;

function preload() {
  img1 = loadImage("lunar.jpg");
  img2 = loadImage("capsule.jpg");
}

function setup() {
  createCanvas(480, 120);
}

function 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:



```
var img;

function preload() {
  img = loadImage("lunar.jpg");
}

function setup() {
  createCanvas(480, 120);
}

function draw() {
```

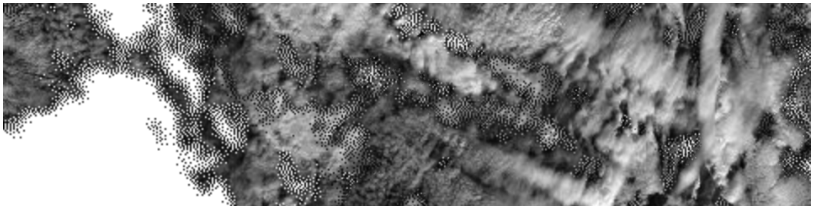
```
  background(0);
  image(img, 0, 0, mouseX * 2, mouseY * 2);
}
```

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 in your *sketch* folder doesn't change.

p5.js can load and display raster images in the JPEG, PNG, and GIF formats, and vector images in the SVG format. You can convert images to the JPEG, PNG, GIF, and SVG formats using programs like GIMP, Photoshop, and Illustrator. Most digital cameras save JPEG images, but they usually need to be reduced in size before being used with p5.js. A typical digital camera creates an image that is several times larger than the drawing area of most p5.js sketches. Resizing these images before they are added to the *sketch* folder makes sketches load faster, run more efficiently, and can save disk space.

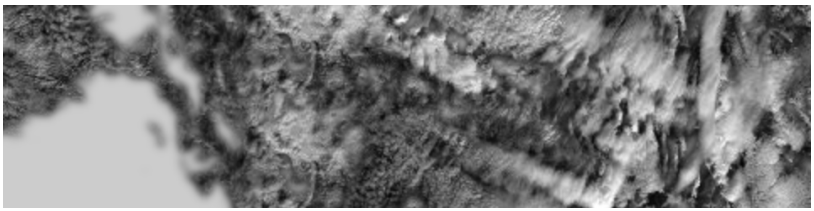
GIF, PNG, and SVG images support transparency, which means that pixels can be invisible or partially visible (recall the discussion of `color()` and alpha values in Example 3-17 on page 33). GIF images have 1-bit transparency, which means that pixels are either fully opaque or fully transparent. PNG images have 8-bit transparency, meaning each pixel can have a variable level of opacity. The following examples use *clouds.gif* and *clouds.png* to show the differences between the file formats. These images are in the *media* folder that you downloaded previously. Be sure to add them to the sketch before trying each example.

Example 7-4: Transparency with a GIF



```
var img;

function preload() {
  img = loadImage("clouds.gif");
}

function setup() {
  createCanvas(480, 120);
}
function draw() {
  background(204);
  image(img, 0, 0);
  image(img, 0, mouseY * -1);
}
```

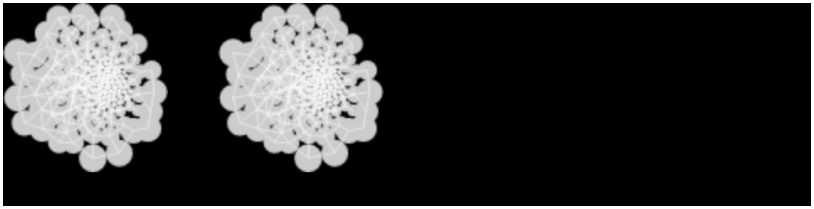
Example 7-5: Transparency with a PNG



```
var img;

function preload() {
  img = loadImage("clouds.png");
}

function setup() {
  createCanvas(480, 120);
```

```
}

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

Example 7-6: Displaying an SVG Image



```
var img;

function preload() {
  img = loadImage("network.svg");
}

function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(0);
  image(img, 0, 0);
  image(img, mouseX, 0);
}
```

Remember to include the appropriate file extension (*.gif*, *.jpg*, *.png*, or *.svg*) 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.

Asynchronicity

Why do we need to load images in `preload()`? Why not use `setup()`? Up until this point, we've been assuming that our programs run from top to bottom, with each line completing before going on to the next one. Although this is generally true, when it comes to certain functions like loading images, your browser will begin the process of loading the image, but skip onto the next line before the image finishes loading! This is known as *asynchronicity*, or an *asynchronous function*. It's a little bit unexpected at first, but this ultimately allows your pages to load and run faster on the Web.

To see this more clearly, consider the following example. It is identical to Example 7-1 on page 104, except we use `loadImage()` in `setup()` instead of `preload()`.

Example 7-7: Demonstrating Asynchronicity


```
var img;

function setup() {
  createCanvas(480, 120);
  img = loadImage("lunar.jpg");
  noLoop();
}

function draw() {
  background(204);
  image(img, 0, 0);
}
```

When you run this program, you'll notice that the drawing canvas is gray with no image displayed. The sketch runs the `setup()` function first, then it runs the `draw()` function. At the `load`

`Image()` line, it begins to load the image, but continues on through the rest of `setup()` and on to `draw()` before the image has completely loaded. The `image()` function is unable to draw the not yet existing image.

To help with this issue, p5.js has the `preload()` function. Unlike `setup()`, `preload()` forces the program to wait until everything has loaded before moving on. It's best to only make load calls in `preload()`, and do all other setup in `setup()`.

Alternatively, instead of using `preload()`, you could use something called a *callback function*. A callback function is a function that is passed as an argument to a second function, and runs after the second function has completed. The following example illustrates this technique.