6/Translate, Rotate, Scale

An alternative technique for positioning and moving things on screen is to change the screen coordinate system. For example, you can move a shape 50 pixels to the right, or you can move the location of coordinate (0,0) 50 pixels to the right—the visual result on screen is the same.

By modifying the default coordinate system, we can create different *transformations* including *translation*, *rotation*, and *scaling*.

Translate

Working with transformations can be tricky, but the `translate()` function is the most straightforward, so we'll start with that. As Figure 6-1 shows, this function can shift the coordinate system left, right, up, and down.

```
translate(40, 20);
rect(20, 20, 20, 40);
```

```
translate(60, 70);
rect(20, 20, 20, 40);
```

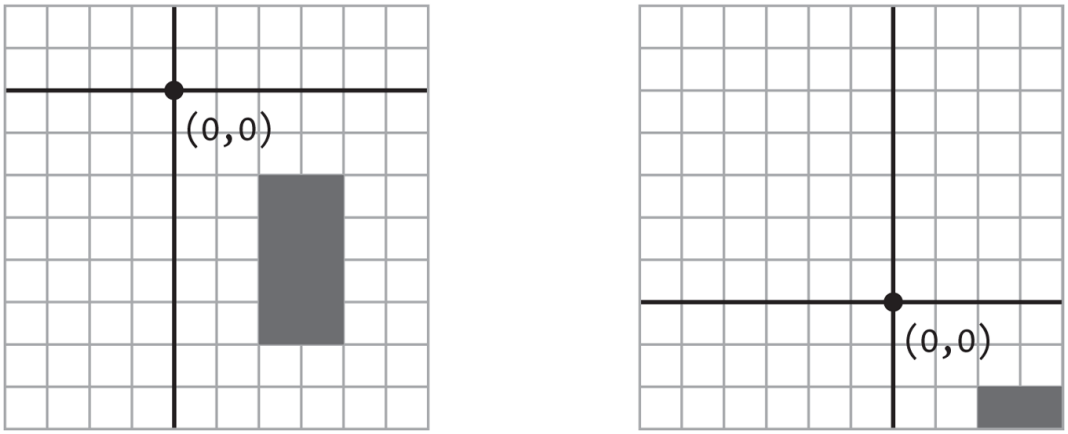
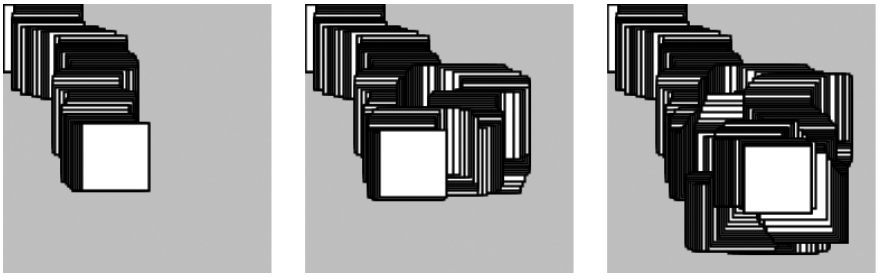


Figure 6-1. *Translating the coordinates*

Example 6-1: Translating Location

In this example, notice that the rectangle is drawn at coordinate (0,0), but it is moved around on the canvas, because it is affected by `translate()`:



```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  rect(0, 0, 30, 30);
}
```

The `translate()` function sets the (0,0) coordinate of the screen to the mouse location (`mouseX` and `mouseY`). Each time the `draw()` block repeats, the `rect()` is drawn at the new origin, derived from the current mouse location.

Example 6-2: Multiple Translations

After a transformation is made, it is applied to all drawing functions that follow. Notice what happens when a second `translate` function is added to control a second rectangle:

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  rect(0, 0, 30, 30);
  translate(35, 10);
  rect(0, 0, 15, 15);
}
```

The values for the `translate()` functions are added together. The smaller rectangle was translated the amount of `mouseX` + 35 and `mouseY` + 10. The *x* and *y* coordinates for both rectangles are (0,0), but the `translate()` functions move them to other positions on the canvas.

However, even though the transformations accumulate within the `draw()` block, they are reset each time `draw()` starts again at the top.

Rotate

The `rotate()` function rotates the coordinate system. It has one parameter, which is the angle (in radians) to rotate. It always rotates relative to (0,0), known as rotating around the *origin*. Figure 3-2 shows the radians angle values. Figure 6-2 shows the difference between rotating with positive and negative numbers.

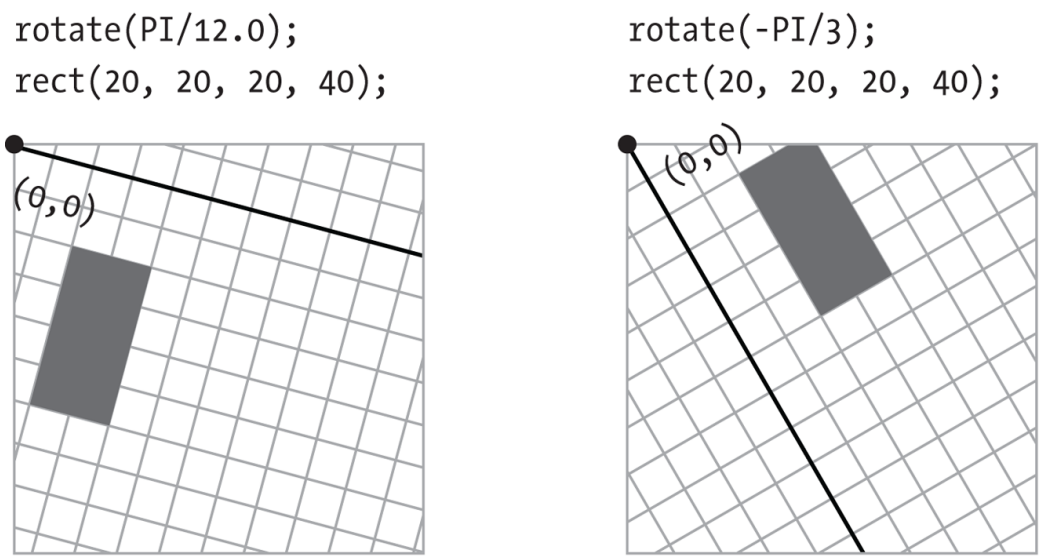
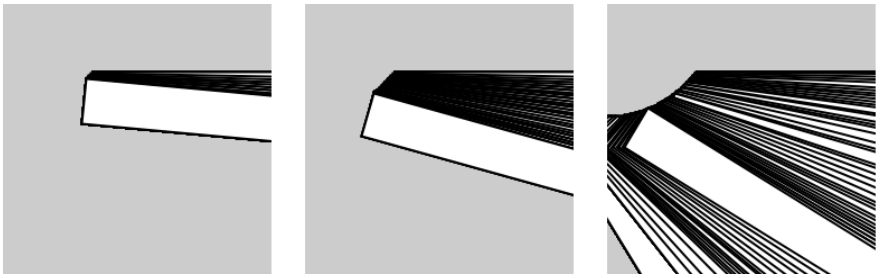


Figure 6-2. *Rotating the coordinates*

Example 6-3: Corner Rotation

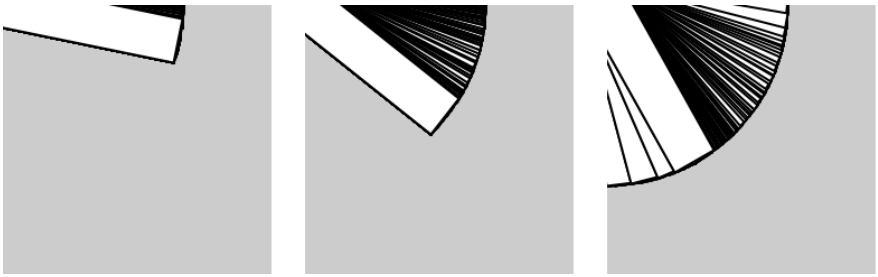
To rotate a shape, first define the rotation angle with `rotate()`, then draw the shape. In this sketch, the parameter to rotate (`mouseX / 100.0`) will be between 0 and 1.2 to define the rotation angle because `mouseX` will be between 0 and 120, the width of the canvas as defined in `createCanvas()`:



```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  rotate(mouseX / 100.0);
  rect(40, 30, 160, 20);
}
```

Example 6-4: Center Rotation

To rotate a shape around its own center, it must be drawn with coordinate (0,0) in the middle. In this example, because the shape is 160 wide and 20 high as defined in `rect()`, it is drawn at the coordinate (-80, -10) to place (0,0) at the center of the shape:

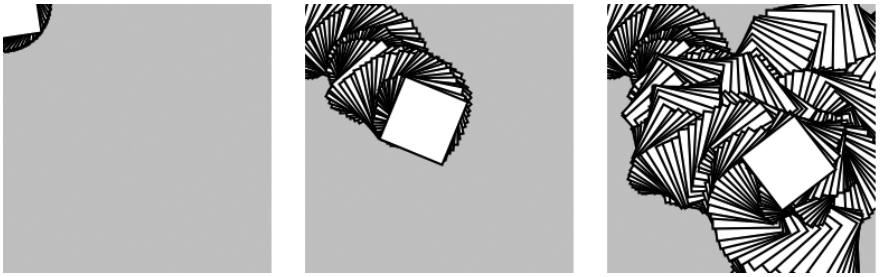


```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  rotate(mouseX / 100.0);
  rect(-80, -10, 160, 20);
}
```

The previous pair of examples show how to rotate around coordinate (0,0), but what about other possibilities? You can use the `translate()` and `rotate()` functions for more control. When they are combined, the order in which they appear affects the result. If the coordinate system is first moved and then rotated, that is different than first rotating the coordinate system, then moving it.

Example 6-5: Translation, Then Rotation

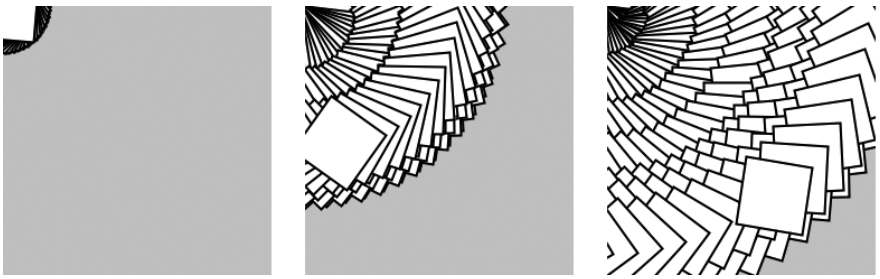
To spin a shape around its center point at a place on screen away from the origin, first use `translate()` to move to the location where you'd like the shape, then call `rotate()`, and then draw the shape with its center at coordinate (0,0):



```
var angle = 0.0;

function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  rotate(angle);
  rect(-15, -15, 30, 30);
  angle += 0.1;
}
```

Example 6-6: Rotation, Then Translation

The following example is identical to Example 6-5 on page 93, except that `translate()` and `rotate()` are reversed. The shape now rotates around the upper-left corner of the canvas, with the distance from the corner set by `translate()`:



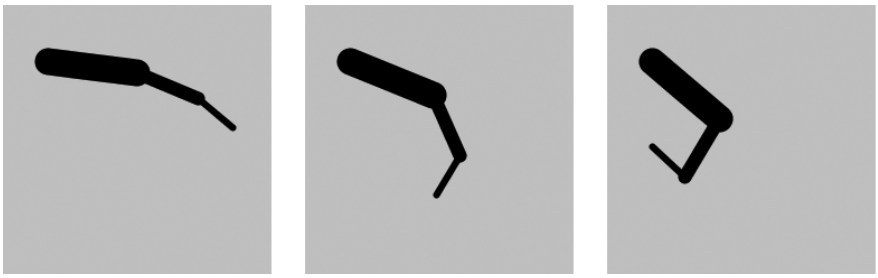
```
var angle = 0.0;

function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
```

```
  rotate(angle);
  translate(mouseX, mouseY);
  rect(-15, -15, 30, 30);
  angle += 0.1;
}
```

You can also use the `rectMode()`, `ellipseMode()` and `imageMode()` functions to make it easier to draw shapes from their center. You can read about these functions in the *p5.js Reference*.

Example 6-7: An Articulating Arm

In this example, we've put together a series of `translate()` and `rotate()` functions to create a linked arm that bends back and forth. Each `translate()` further moves the position of the lines, and each `rotate()` adds to the previous rotation to bend more:



```
var angle = 0.0;
var angleDirection = 1;
var speed = 0.005;

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

function draw() {
  background(204);
  translate(20, 25);  // Move to start position
  rotate(angle);
  strokeWeight(12);
  line(0, 0, 40, 0);
  translate(40, 0);   // Move to next joint
  rotate(angle * 2.0);
  strokeWeight(6);
```

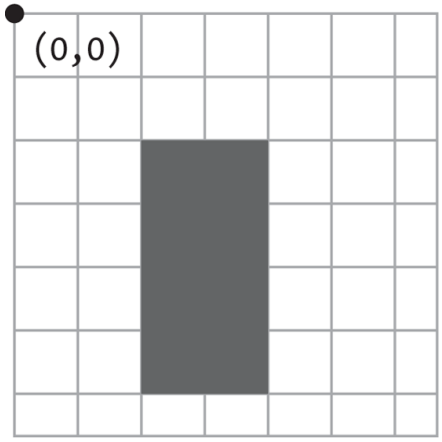
```
    line(0, 0, 30, 0);
    translate(30, 0);   // Move to next joint
    rotate(angle * 2.5);
    strokeWeight(3);
    line(0, 0, 20, 0);

    angle += speed * angleDirection;
    if ((angle > QUARTER_PI) || (angle < 0)) {
      angleDirection *= -1;
    }
  }
```

The `angle` variable grows from 0 to `QUARTER_PI` (one quarter of the value of pi), then decreases until it is less than zero, then the cycle repeats. The value of the `angleDirection` variable is always 1 or –1 to make the value of `angle` correspondingly increase or decrease.

Scale

The `scale()` function stretches the coordinates on the canvas. Because the coordinates expand or contract as the scale changes, everything drawn to the canvas increases or decreases in dimension. The amount to scale is written as decimal percentages. Therefore, the parameter 1.5 to `scale()` is 150% and 3 is 300% (Figure 6-3).

```
scale(1.5);
rect(20, 20, 20, 40);
```




```
scale(3);
rect(20, 20, 20, 40);
```

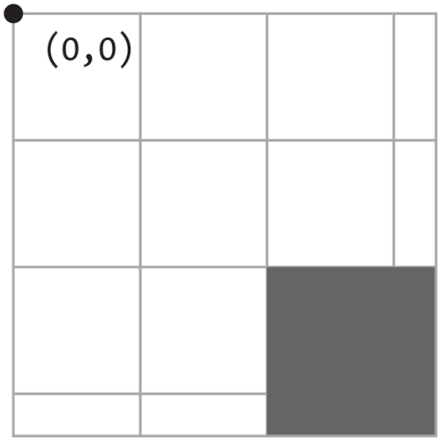
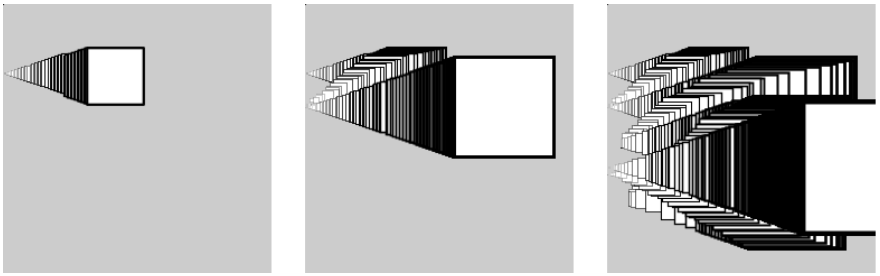


Figure 6-3. *Scaling the coordinates*

Example 6-8: Scaling

Like `rotate()`, the `scale()` function transforms from the origin. Therefore, as with `rotate()`, to scale a shape from its center, translate to its location, scale, and then draw with the center at coordinate (0,0):



```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  scale(mouseX / 60.0);
  rect(-15, -15, 30, 30);
}
```

Example 6-9: Keeping Strokes Consistent

From the thick lines in Example 6-8 on page 97, you can see how the `scale()` function affects the stroke weight. To maintain a consistent stroke weight as a shape scales, divide the desired stroke weight by the scalar value:

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  var scalar = mouseX / 60.0;
  scale(scalar);
  strokeWeight(1.0 / scalar);
```

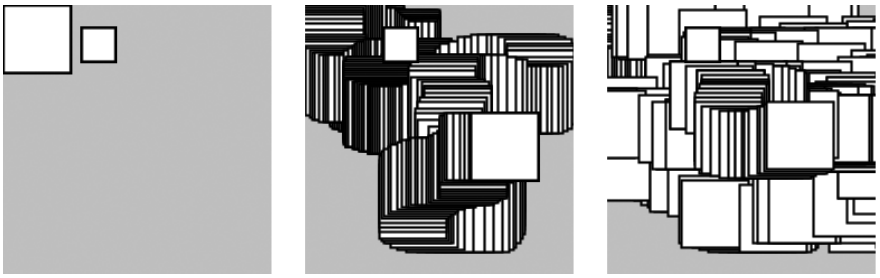
```
  rect(-15, -15, 30, 30);
}
```

Push and Pop

To isolate the effects of transformations so they don't affect later functions, use the `push()` and `pop()` functions. When `push()` is run, it saves a copy of the current coordinate system and then restores that system after `pop()`. This is useful when transformations are needed for one shape, but not wanted for another.

Example 6-10: Isolating Transformations

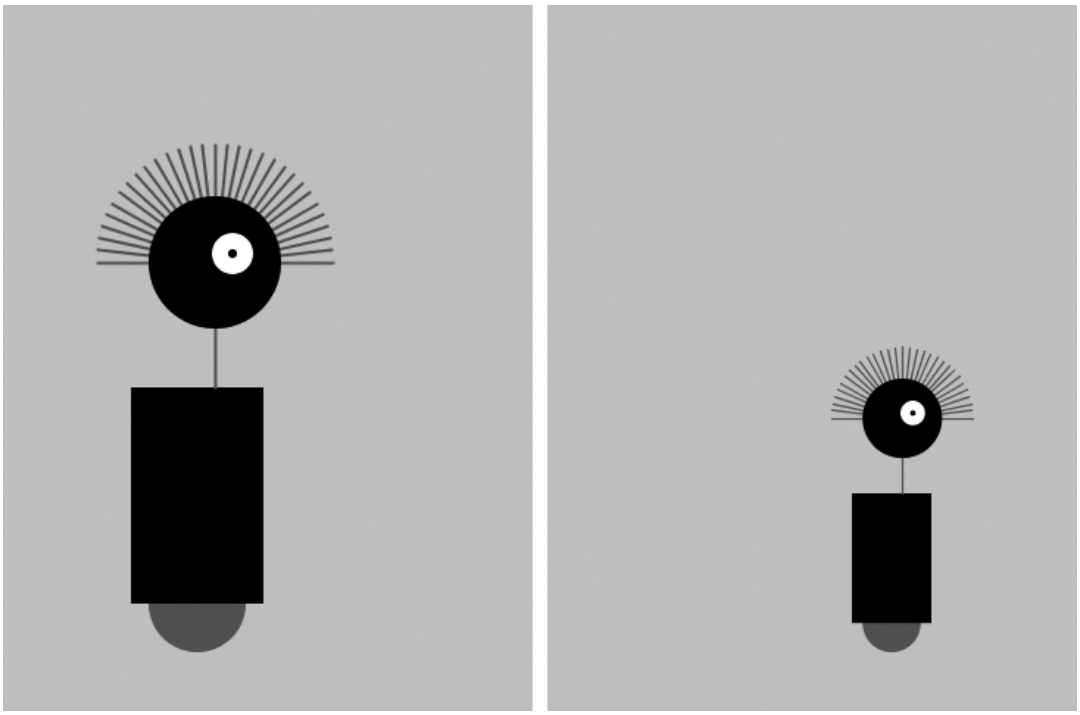
In this example, the smaller rectangle always draws in the same position because the `translate(mouseX, mouseY)` is cancelled by the `pop()`:



```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  push();
  translate(mouseX, mouseY);
  rect(0, 0, 30, 30);
  pop();
  translate(35, 10);
  rect(0, 0, 15, 15);
}
```

The `push()` and `pop()` functions are always used in pairs. For every `push()`, you need to have a matching `pop()`.

Robot 4: Translate, Rotate, Scale



The `translate()`, `rotate()`, and `scale()` functions are all utilized in this modified robot sketch. In relation to "Robot 3: Response" on page 84, `translate()` is used to make the code easier to read. Here, notice how the `x` value no longer needs to be added to each drawing function because the `translate()` moves everything. Similarly, the `scale()` function is used to set the dimensions for the entire robot. When the mouse is not pressed, the size is set to 60%, and when it is pressed, it goes to 100% in relation to the original coordinates. The `rotate()` function is used within a loop to draw a line, rotate it a little, then draw a second line, then rotate a little more, and so on until the loop has drawn 30 lines half-way around a circle to style a lovely head of robot hair:

```
var x = 60;          // x coordinate
var y = 440;         // y coordinate
```

```
var radius = 45;       // Head radius
var bodyHeight = 180;  // Body height
var neckHeight = 40;   // Neck height

var easing = 0.04;

function setup() {
  createCanvas(360, 480);
  strokeWeight(2);
  ellipseMode(RADIUS);
}

function draw() {

  var neckY = -1 * (bodyHeight + neckHeight + radius);

  background(204);

  translate(mouseX, y);  // Move all to (mouseX, y)

  if (mouseIsPressed) {
    scale(1.0);
  } else {
    scale(0.6);  // 60% size when mouse is pressed
  }

  // Body
  noStroke();
  fill(102);
  ellipse(0, -33, 33, 33);
  fill(0);
  rect(-45, -bodyHeight, 90, bodyHeight-33);

  // Neck
  stroke(102);
  line(12, -bodyHeight, 12, neckY);

  // Hair
  push();
  translate(12, neckY);
  var angle = -PI/30.0;
  for (var i = 0; i <= 30; i++) {
    line(80, 0, 0, 0);
    rotate(angle);
  }
  pop();
```

```
  // Head
  noStroke();
  fill(0);
  ellipse(12, neckY, radius, radius);
  fill(255);
  ellipse(24, neckY-6, 14, 14);
  fill(0);
  ellipse(24, neckY-6, 3, 3);

}
```