Core HTML5 Canvas
2.16 Conclusion

This chapter took an in-depth look at drawing in a canvas. We began by discussing the coordinate system and the Canvas drawing model, and then we looked at drawing simple rectangles, specifying colors and transparencies, using gradients and patterns, and applying shadows.

Then we took a look at paths and subpaths, and stroking and filling. We also looked at the nonzero winding rule that Canvas uses when filling intersecting subpaths, and you saw how to put that knowledge to practical use by implementing cutouts.

Then we focused on drawing lines, and you learned how to draw true one-pixel-wide lines and how to draw lines that appear to be less than one pixel wide. You saw how to use lines to draw grids and axes, and you learned how to let users interactively draw lines with rubber bands. You also saw how to draw dashed lines, which are not explicitly supported by the Canvas context, and then you saw how to extend the Canvas context so that dashed lines are explicitly supported. Finally, we wrapped up the section on lines by looking at line caps and joins, which determines how the Canvas context draws line endpoints.

From there we moved on to arcs and circles, and you saw how to let users interactively create circles by dragging the mouse. You also learned how to draw a rounded rectangles with the arcTo() method and how to implement dials and gauges.

From arcs and circles, we moved on to bézier curves, both quadratic and cubic, and you saw how to use those types of curves to implement a checkmark and an arrowhead. Then we looked at drawing polygons, implementing polygon objects and using the Canvas context’s isPointInPath() method to drag polygons. You also saw how to use isPointInPath() to implement an interactive editor that creates bézier curves.

From there we moved on to transformations, where you saw how to translate, rotate, and scale the Canvas coordinate system. You also saw how to create custom transformations, such as shear.

Finally, we looked at compositing, which determines how Canvas draws shapes on top of each other. We wrapped up the chapter by looking at the Canvas’s Swiss Army knife—the clipping region—and you saw how to erase and implement a telescoping animation with that knife.
At this point you know how to draw pretty much anything you can imagine in a canvas. In the chapters that follow we will put that knowledge to good use by exploring images, animation, sprites, physics, collision detection, game development, implementing custom controls, and manipulating video frames, as a video is running, inside the canvas. And we will also explore using Canvas to implement mobile applications that you can run on smart phones or tablet computers.

Canvas provides a powerful drawing API that’s based on other, proven graphics system such as Adobe Illustrator and Apple’s Cocoa. In the pages that follow, we will continue to explore that API.
This page intentionally left blank
Nearly every Canvas-based application deals with text. Some applications merely configure and display text, whereas other applications provide sophisticated text editing support.

The canvas element only minimally supports text; at the time this book was written, it does not offer many of the features that you will find in basic text editors, features such as text selection, copy and paste, and text scrolling. However, it does support basic necessities such as stroking and filling text, placing text within the canvas, and measuring the width, in pixels, of an arbitrary string. The Canvas context provides three methods pertaining to text:

- `strokeText(text, x, y)`
- `fillText(text, x, y)`
- `measureText(text)`

The `measureText()` method returns an object with a `width` property, which represents the width of the text you pass to the method. Three Canvas context properties are related to text:

- `font`
- `textAlign`
- `textBaseline`

The `font` property lets you set the font of text that you subsequently draw, and `textAlign` and `textBaseline` let you position text within the canvas. Let’s take a closer look at those methods and properties.
3.1 Stroking and Filling Text

Figure 3.1 shows an application that strokes and fills text.

![Figure 3.1 Stroking and filling text](image)

This application provides checkboxes that let you control whether the text is stroked, filled, or drawn with shadows.

The HTML for the application shown in Figure 3.1, which is omitted from the book in the interests of brevity, creates the checkboxes and canvas and includes the application’s JavaScript, which is listed in Example 3.1.

This JavaScript obtains references to the three checkboxes and adds an onchange handler to each that draws the background and text.

The application uses `fillText()` and `strokeText()` to fill and stroke the text, respectively. Each of those methods takes three arguments. The first argument is the text, and the remaining two arguments specify the text’s location. Exactly where the text is drawn depends on the `textAlign` and `textBaseline` properties, which we discuss in Section 3.3, “Positioning Text,” on p. 210.
**Example 3.1 Stroking and filling text**

```javascript
var canvas = document.getElementById('canvas'),
    context = canvas.getContext('2d'),
    fillCheckbox = document.getElementById('fillCheckbox'),
    strokeCheckbox = document.getElementById('strokeCheckbox'),
    shadowCheckbox = document.getElementById('shadowCheckbox'),
    text = 'HTML5';

// Functions……………………………………………………………………………….

function draw() {
    context.clearRect(0, 0, canvas.width, canvas.height);
    drawBackground();

    if (shadowCheckbox.checked) turnShadowsOn();
    else turnShadowsOff();

    drawText();
}

function drawBackground() { // Ruled paper
    var STEP_Y = 12,
        TOP_MARGIN = STEP_Y * 4,
        LEFT_MARGIN = STEP_Y * 3,
        i = context.canvas.height;

    // Horizontal lines
    context.strokeStyle = 'lightgray';
    context.lineWidth = 0.5;

    while(i > TOP_MARGIN) {
        context.beginPath();
        context.moveTo(0, i);
        context.lineTo(context.canvas.width, i);
        context.stroke();
        i -= STEP_Y;
    }

    // Vertical line
    context.strokeStyle = 'rgba(100,0,0,0.3)';
    context.lineWidth = 1;
    context.beginPath();
    context.moveTo(LEFT_MARGIN, 0);
    context.lineTo(LEFT_MARGIN, context.canvas.height);
    context.stroke();
}
```

(Continues)