Bringing Out Your Inner Artist

Eric D. Wills
University of Oregon
Graphics!

• Most programming languages have a built-in graphics library which allows manipulation of individual pixels.
  – Useful for drawing primitives (lines, circles, rectangles, triangles, etc).
  – Useful for adding images to a page dynamically.
  – Useful for drawing 3D graphics.
  – Useful for games!
Why HTML5?

• Why not Flash?
  – Could use Flash, but the industry is moving towards HTML5. No other language currently provides the same cross-platform support!

• Why not generate images on a server?
  – Could build a server, but would need to scale up as the numbers of users increases. Instead have the users’ devices to do as much of the work as possible!
Hello Rectangle

• Consider the follow code:
  
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.fillRect(10, 10, 100, 100);

• Assuming that the canvas is 300 x 300 pixels, what does this code do?
<!DOCTYPE html>
<html>
<head>
  <title>Hello Rectangle</title>
  <meta charset="utf-8">
  <style>canvas { border: 1px solid black; }</style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.fillRect(10, 10, 100, 100);
    }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
Hello Rectangle

```html
<!doctype html>
<html>
<head>
  <title>Hello Rectangle</title>
  <meta charset="utf-8">
  <style>canvas { border: 1px solid black; }</style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.fillRect(10, 10, 100, 100);
    }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
```
Hello Rectangle

• `<style>canvas { border: 1px solid black; }</style>`
  – Add some CSS to the page which specifies that all canvas elements have a 1-pixel black border.
  – This allows us to see the bounds of the canvas.
Hello Rectangle

<!doctype html>
<html>
<head>
  <title>Hello Rectangle</title>
  <meta charset="utf-8">
  <style>canvas { border: 1px solid black; }
  </style>
  <script>
      window.onload = function() {
          var canvas = document.getElementById("myCanvas");
          var context = canvas.getContext("2d");
          context.fillRect(10, 10, 100, 100);
      }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
Hello Rectangle

• window.onload = function() {
  ...
}

  – This is equivalent to:
    • function init() {
      ...
    }
    window.onload = init;

  – This is known as an anonymous function because the function isn’t assigned a name.
    • This means that the function can’t later be called explicitly.
Hello Rectangle

<!doctype html>
<html>
<head>
  <title>Hello Rectangle</title>
  <meta charset="utf-8">
  <style>canvas { border: 1px solid black; }</style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.fillRect(10, 10, 100, 100);
    }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
Hello Rectangle

- var canvas = document.getElementById("myCanvas");
- var context = canvas.getContext("2d");
- context.fillRect(10, 10, 100, 100);
  - Get the element with ID “myCanvas” on the page.
  - Get that canvas element’s 2D context.
  - Fill a 100 x 100 pixel rectangle with the top-left corner at pixel location (10, 10).
Hello Rectangle

<!doctype html>
<html>
<head>
  <title>Hello Rectangle</title>
  <meta charset="utf-8">
  <style>canvas { border: 1px solid black; }</style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.fillRect(10, 10, 100, 100);
    }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
Path Methods

• `context.beginPath(); // start drawing a path`
• `context.moveTo(x, y); // move the pen to the x, y coordinates as part of a path`
• `context.lineTo(x, y); // draw a line from the pen’s current x, y coordinates to the specified x, y coordinates (moves the pen) as part of a path`
• `context.closePath(); // stop drawing a path, connecting the last location to the first location`
• `context.stroke(); // draw the path (but don’t fill interior)`
• `context.fill(); // fill the path (including interior)`
Path Properties

- `context.lineWidth = 5; // sets the line width of the stroke to 5.`
  - Can use any valid number.

- `context.strokeStyle = "green"; // sets the stoke color to green.`
  - Can use any colors accepted by HTML, either by name (e.g., “green”, “yellow”) or by code (e.g., “#00FF00”, “#FFFF00”).

- `context.fillStyle = "yellow"; // sets the fill color to yellow.`
  - Same rules as for strokeStyle.
Hello Triangle

<!doctype html>
<html>
<head>
  <title>Hello Triangle</title>
  <meta charset="utf-8">
  <style>canvas {
    border: 1px solid black;
  }</style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.beginPath();
      context.moveTo(50, 50);
      context.lineTo(75, 75);
      context.lineTo(25, 75);
      context.closePath();
      context.strokeStyle = "green";
      context.lineWidth = 5;
      context.stroke();
      context.fillStyle = "yellow";
      context.fill();
    }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
Hello Triangle

window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50, 50);
    context.lineTo(75, 75);
    context.lineTo(25, 75);
    context.closePath();
    context.strokeStyle = "green";
    context.lineWidth = 5;
    context.stroke();
    context.fillStyle = "yellow";
    context.fill();
}
window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50, 50);
    context.lineTo(75, 75);
    context.lineTo(25, 75);
    context.closePath();
    context.strokeStyle = "green";
    context.lineWidth = 5;
    context.stroke();
    context.fillStyle = "yellow";
    context.fill();
}
Hello Triangle

- `context.beginPath();`
  
  `context.moveTo(50, 50);`
  
  `context.lineTo(75, 75);`
  
  `context.lineTo(25, 75);`
  
  `context.closePath();`

  - Begin a path, move the pen to (50, 50), draw a line to (75, 75), then draw a line to (25, 75), then close the path.
window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50, 50);
    context.lineTo(75, 75);
    context.lineTo(25, 75);
    context.closePath();
    context.strokeStyle = "green";
    context.lineWidth = 5;
    context.stroke();
    context.fillStyle = "yellow";
    context.fill();
}
Hello Triangle

• context.strokeStyle = "green";
  context.lineWidth = 5;
  context.stroke();
  context.fillStyle = "yellow";
  context.fill();

  – Set the stroke color to “green” and the line with
to 5 and then draw the path. Then set the fill
color to “yellow” and fill the path.
window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.beginPath();
    context.moveTo(50, 50);
    context.lineTo(75, 75);
    context.lineTo(25, 75);
    context.closePath();
    context.strokeStyle = "green";
    context.lineWidth = 5;
    context.stroke();
    context.fillStyle = "yellow";
    context.fill();
}
Circles

• We can’t draw circles directly, instead we need to add an arc to a path using:
  – context.arc(x, y, r, start, end, direction)
    • x is the x coordinate of the center of the arc.
    • y is the y coordinate of the center of the arc.
    • r is the radius of the arc.
    • start is the starting angle in radians (0 is pointing right).
    • end is the ending angle in radians.
    • direction is true for counter clockwise, false for clockwise.
Hello Circle

<!doctype html>
<html>
<head>
  <title>Hello Rectangle</title>
  <meta charset="utf-8">
  <style>canvas { border: 1px solid black; }
</style>
  <script>
    window.onload = function()
      {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.beginPath();
        context.arc(100, 100, 50, 0, 2*Math.PI, false)
        context.closePath();
        context.strokeStyle = "green";
        context.lineWidth = 5;
        context.stroke();
        context.fillStyle = "yellow";
        context.fill();
      }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
Hello Circle

window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.beginPath();
    context.arc(100, 100, 50, 0, 2*Math.PI, false)
    context.closePath();
    context.strokeStyle = "green";
    context.lineWidth = 5;
    context.stroke();
    context.fillStyle = "yellow";
    context.fill();
}
Hello Circle

window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.beginPath();
    context.arc(100, 100, 50, 0, 2*Math.PI, false)
    context.closePath();
    context.strokeStyle = "green";
    context.lineWidth = 5;
    context.stroke();
    context.fillStyle = "yellow";
    context.fill();
}
Hello Circle

• `context.arc(100, 100, 50, 0, 2*Math.PI, false)`
  
  – Draw an arc centered at (100, 100) with radius 50 pixels. The arc should start at 0 radians (pointing right) and end at 2-\(\pi\) radians (pointing right). This will draw a complete circle, so the direction doesn’t matter, but we’ll pass the Boolean value `false` to specify clockwise (`true` would specify counter clockwise).
Hello Circle

window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.beginPath();
    context.\text{arc}(100, 100, 50, 0, 2*Math.PI, false)
    context.closePath();
    context.strokeStyle = "green";
    context.\text{lineWidth} = 5;
    context.sty\text{roke}();
    context.\text{fillStyle} = "yellow";
    context.\text{fill}();
}
Drawing and Repetition

• We can use loops to draw groups of primitives.
  – For example, if we defined a function `drawCircle(x, y, context)` which draws a circle at the location `(x, y)` to the context, we could draw some circles using:
    • for (var i = 50; i < 250; i = i + 25) {
      drawCircle(i, i, context);
    }
  – Or we could draw circles at random locations:
    • for (var i = 1; i < 10; i++) {
      var x = Math.random()*200+ 50;
      var y = Math.random()*200+ 50;
      drawCircle(x, y, context);
    }
Hello Circles (Part 1)

```javascript
function drawCircle(x, y, context) {
    context.beginPath();
    context.arc(x, y, 25, 0, 2*Math.PI, false)
    context.closePath();
    context.strokeStyle = "green";
    context.lineWidth = 5;
    context.stroke();
    context.fillStyle = "yellow";
    context.fill();
}
```
window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    for (var i = 1; i < 10; i++) {
        var x = Math.random()*200+ 50;
        var y = Math.random()*200+ 50;
        drawCircle(x, y, context);
    }
}
Hello Circles (Part 2)

• var x = Math.random()*200+ 50;
  var y = Math.random()*200+ 50;
  drawCircle(x, y, context);
  – Create two random numbers x and y between 50 and 250 (exclusive) and draw a circle to the context at (x, y).
window.onload = function() {
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  for (var i = 1; i < 10; i++) {
    var x = Math.random()*200+ 50;
    var y = Math.random()*200+ 50;
    drawCircle(x, y, context);
  }
}
Drawing Images

• Can draw images to the canvas, but need to wait for the image to load.
  – Can load images using the built-in Image class (i.e., by creating objects of this type).
  – Use a callback function, similar to a button’s onclick handler.
  – Order is important when drawing to the canvas, so must wait to draw elements which will be on top of the image until after the image has loaded.
<!doctype html>
<html>
<head>
  <title>Hello Image</title>
  <meta charset="utf-8">
  <style>canvas { border: 1px solid black; }
  </style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      var image = new Image();
      image.onload = function() {
        context.drawImage(image, 10, 10, 200, 200);
      }
      image.src = "http://www.cs.uoregon.edu/Classes/13W/cis115/seal.gif"
    }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    var image = new Image();
    image.onload = function() {
        context.drawImage(image, 10, 10, 200, 200);
    }
    image.src = "http://www.cs.uoregon.edu/Classes/13W/cis115/seal.gif "
}
window.onload = function() {
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  var image = new Image();
  image.onload = function() {
    context.drawImage(image, 10, 10, 200, 200);
  }
  image.src = "http://www.cs.uoregon.edu/Classes/13W/cis115/seal.gif "
}
Hello Image

• var image = new Image();
  – Create a new instance of an Image object and store this new instance as the variable “image”.
window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    var image = new Image();
    image.onload = function() {
        context.drawImage(image, 10, 10, 200, 200);
    }
    image.src = "http://www.cs.uoregon.edu/Classes/13W/cis115/seal.gif "
}
Hello Image

- image.onload = function() {
  context.drawImage(image, 10, 10, 200, 200);
}

- Create a new anonymous function which will draw the image to the context with the top-left corner of the image at (10, 10), an image width of 200 and an image height of 200.

- Specify this anonymous function as the image’s onload handler (i.e., to be called when the image finishes loading).
window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    var image = new Image();
    image.onload = function() {
        context.drawImage(image, 10, 10, 200, 200);
    }
    image.src = "http://www.cs.uoregon.edu/Classes/12S/cis115/seal.gif"
}
Hello Image

• 

image.src = "http://www.cs.uoregon.edu/Classes/13W/cis115/seal.gif"

  – Specify that the image will load from the above URL.
  – The src property is just like that of an HTML image element:
    • Can be a path to a local file or an absolute URL.
window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    var image = new Image();
    image.onload = function() {
        context.drawImage(image, 10, 10, 200, 200);
    }
    image.src = "http://www.cs.uoregon.edu/Classes/13W/cis115/seal.gif" 
}
Drawing Text

• Can draw text to the canvas using:
  – context.fillText(text, x, y):
    • text is the text (a string).
    • x is the x coordinate of the upper-left corner of the text.
    • y is the y coordinate of the upper-left corner of the text.

• Can also modify the font using the context’s font property and the color using the context’s fillStyle property.

• Can also stroke text using the strokeText method (same parameters as fillText).
<!doctype html>
<html>
<head>
  <title>Hello Rectangle</title>
  <meta charset="utf-8">
  <style>
    canvas {
      border: 1px solid black;
    }
  </style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.fillStyle = "green";
      context.fillText("GO DUCKS!", 100, 100);
    }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
<!doctype html>
<html>
<head>
  <title>Hello Rectangle</title>
  <meta charset="utf-8">
  <style>
    canvas {
      border: 1px solid black;
    }
  </style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      context.fillStyle = "green";
      context.fillText("GO DUCKS!", 100, 100);
    }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
Hello Text

- context.fillStyle = "green";
  context.fillText("GO DUCKS!", 100, 100);
  - Set the fill style to the color green, then draw the text “GO DUCKS!” with the top-left corner of the text at (100, 100).
<!DOCTYPE html>
<html>
<head>
  <title>Hello Rectangle</title>
  <meta charset="utf-8">
  <style>canvas { border: 1px solid black; }
  </style>
  <script>
  window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.fillStyle = "green";
    context.fillText("GO DUCKS!", 100, 100);
  }
  </script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
Transparency

• Two ways to set transparency:
  – Globally using:
    • `context.globalAlpha = a;`
      – `a` is the alpha-channel value \([0.0, 1.0]\) (0.0 for transparent, 1.0 for opaque).
  – Per color using:
    • `context.fillStyle = “rgba(r, g, b, a)”;
      – `r` is the red-channel value \([0, 255]\).
      – `g` is the green-channel value \([0, 255]\).
      – `b` is the blue-channel value \([0, 255]\).
      – `a` is the alpha-channel value \([0.0, 1.0]\) (0.0 for transparent, 1.0 for opaque).
Canvas Interaction

• We can add user interaction to the canvas by adding handlers for mouse and keyboard events.
  – Mouse events: onclick, onmousedown, onmouseup, onmouseover, onmouseout, onmousemove, etc.
  – Keyboard events: onkeypress, onkeydown, onkeyup.
Paint (HTML)

<!doctype html>
<html>
<head>
  <title>Paint</title>
  <meta charset="utf-8">
  <style>canvas {
    border: 1px solid black;
  }</style>
  <script src="paint.js"></script>
</head>
<body>
  <canvas width="300" height="300" id="myCanvas"></canvas>
</body>
</html>
Paint (JS Part 1)

```javascript
var canvas;
var context;
var mouseDown = false;

window.onload = function() {
    canvas = document.getElementById("myCanvas");
    context = canvas.getContext("2d");
    canvas.onmousedown = handleMouseDown;
    canvas.onmouseup = handleMouseUp;
    canvas.onmousemove = handleMouseMove;
}
```
Paint (JS Part 1)

```javascript
var canvas;
var context;
var mouseDown = false;

window.onload = function() {
    canvas = document.getElementById("myCanvas");
    context = canvas.getContext("2d");
    canvas.onmousedown = handleMouseDown;
    canvas.onmouseup = handleMouseUp;
    canvas.onmousemove = handleMouseMove;
}
```
• var canvas;
  var context;
  var mouseDown = false;

  – Define global canvas and context variables so that they can be accessed by any function on the page (not that this will only work for pages with a single canvas).

  – Also define a global mouseDown variable to track when the user is holding down the mouse button (i.e., when they are painting).
Paint (JS Part 1)

```javascript
var canvas;
var context;
var mouseDown = false;

window.onload = function() {
    canvas = document.getElementById("myCanvas");
    context = canvas.getContext("2d");
    canvas.onmousedown = handleMouseDown;
    canvas.onmouseup = handleMouseUp;
    canvas.onmousemove = handleMouseMove;
}
```
Paint (JS Part 1)

• canvas = document.getElementById("myCanvas");
  context = canvas.getContext("2d");
  canvas.onmousedown = handleMouseDown;
  canvas.onmouseup = handleMouseUp;
  canvas.onmousemove = handleMouseMove;
  
  — Assign the global canvas and context variables based on the
  canvas with ID “myCanvas” and set up onmousedown, onmouseup,
  and onmousemove handler functions (we’ll define
  these next).
Paint (JS Part 1)

```javascript
var canvas;
var context;
var mouseDown = false;

window.onload = function() {
    canvas = document.getElementById("myCanvas");
    context = canvas.getContext("2d");
    canvas.onmousedown = handleMouseDown;
    canvas.onmouseup = handleMouseUp;
    canvas.onmousemove = handleMouseMove;
}
```
Paint (JS Part 2)

```javascript
function handleMouseDown(event) {
    paint(event);

    mouseDown = true;
}

function handleMouseUp(event) {
    mouseDown = false;
}

function handleMouseMove(event) {
    if (mouseDown) {
        paint(event);
    }
}
```
function handleMouseDown(event) {
    paint(event);

    mouseDown = true;
}

function handleMouseUp(event) {
    mouseDown = false;
}

function handleMouseMove(event) {
    if (mouseDown) {
        paint(event);
    }
}

Paint (JS Part 2)
Paint (JS Part 2)

- function handleMouseDown(event) {
  paint(event);

  mouseDown = true;
}

  When the mouse button is pressed, call the paint function (we’ll define this next) to draw at the current mouse location and then assign the mouseDown variable indicating that the mouse button is down.

  Note that the onmousedown handler takes an event parameter which specifies information about the current state of the mouse (e.g., location, key modifiers).
function handleMouseDown(event) {
    paint(event);

    mouseDown = true;
}

function handleMouseUp(event) {
    mouseDown = false;
}

function handleMouseMove(event) {
    if (mouseDown) {
        paint(event);
    }
}
Paint (JS Part 2)

• function handleMouseUp(event) {
    mouseDown = false;
}

    – Assign the mouseDown variable indicating that the mouse button is no longer down.
function handleMouseDown(event) {
    paint(event);

    mouseDown = true;
}

function handleMouseUp(event) {
    mouseDown = false;
}

function handleMouseMove(event) {
    if (mouseDown) {
        paint(event);
    }
}
function handleMouseMove(event) {
  if (mouseDown) {
    paint(event);
  }
}

– Call the paint function to paint at the current mouse location if the mouse button is down.
function handleMouseDown(event) {
    paint(event);

    mouseDown = true;
}

function handleMouseUp(event) {
    mouseDown = false;
}

function handleMouseMove(event) {
    if (mouseDown) {
        paint(event);
    }
}
Paint (JS Part 3)

```javascript
function paint(event) {
    context.fillRect(event.clientX, event.clientY, 6, 6);
}

– Fill a 6x6-pixel rectangle with its top-left corner at the current mouse location.
```
Oops!

• This isn’t quite what we want because the clientX and clientY properties are relative to the entire browser window.
• We’ll need to subtract off the x and y position of the canvas.
function paint(event) {
    context.fillRect(event.clientX - canvas.offsetLeft,
                     event.clientY - canvas.offsetTop, 6, 6);
}

– Fill a 6x6-pixel rectangle with its top-left corner at the current mouse location (relative to the canvas).
Oops!

• This still isn’t quite what we want because we probably want the rectangle centered on the current mouse location.
  – Need to subtract half of the width and height from the rectangle’s position to center it.
Paint (JS Part 3)

function paint(event) {
    context.fillRect(event.clientX - canvas.offsetLeft - 3,
                     event.clientY - canvas.offsetTop - 3, 6, 6);
}

– Fill a 6x6-pixel rectangle centered at the current mouse location (relative to the canvas).