



COMP 4021

Internet Computing

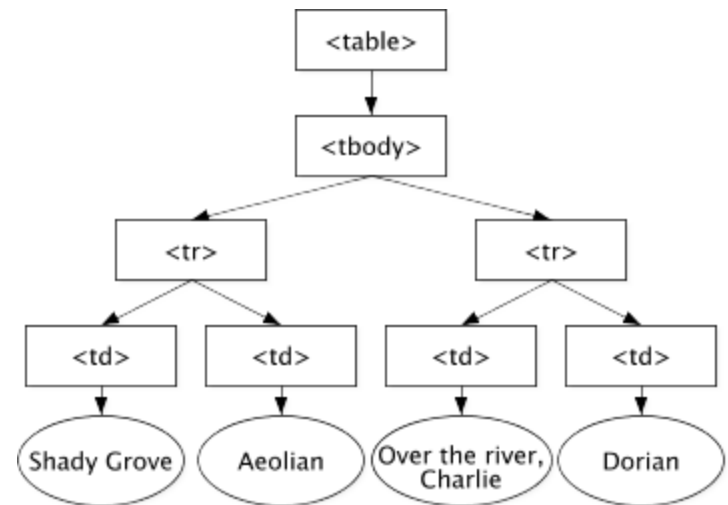
Dynamic SVG Using Javascript

Dr. Kenneth LEUNG

Slides created by Dr. David ROSSITER

Using JavaScript

- JavaScript can access and control any content stored in the browser DOM, at any time
- So it can change HTML, SVG, XML, and anything else that is currently loaded into the browser
- This doesn't apply to applets and Flash which are 'black boxes' and do not use the browser DOM



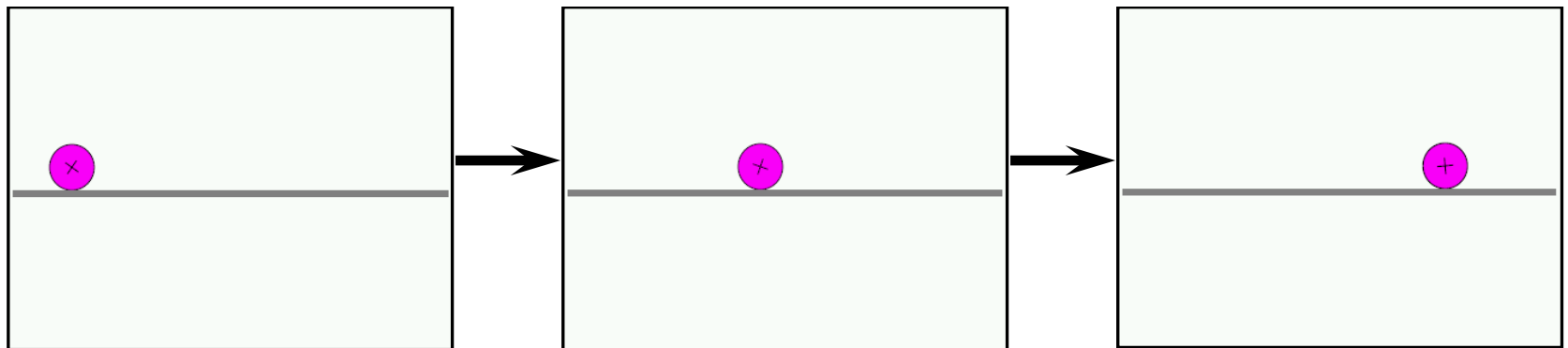
Some DOM Commands

- Many commands are available for accessing and changing the DOM, e.g.,
- `thing = getElementById("name")`
 - search for *name* in the DOM
- `thing.setAttribute("x",100)`
 - change the x attribute of *thing* to 100
- `x_position = thing.getAttribute("x")`
 - return the value of the x attribute and store it in a JavaScript variable

In SVG the x position of an object is stored in the 'x' attribute

Example Animation - Ball

- Object is shown moving and rotating, as if it was real
 - ball's y coordinate does not change
 - ball's x coordinate increments
 - to make the ball “roll” (can be observed from the cross at the ball's center), the ball has to rotate by a fixed degree in each increment



Example Animation - Ball

- This is achieved by using a matrix to handle the translation and rotation of the ball
- A timer is used to call a JavaScript function several times a second
- The matrix() parameters are then updated by the JavaScript code

Example Animation - Ball

- Definition of the ball:

```
<g id="ball">  
  <circle style="fill:magenta;stroke:black;"  
    cx="30" cy="140" r="20" />  
  <path style="fill:none;stroke:black;"  
    d="M25,135 35,145 M35,135 25,145"/>  
</g>
```

The cross at
the center of
the ball

When “ball” is transformed, both the circle
and the cross are changed in the same way

Example Animation - Ball

```
function Animate()
{
  var sp = Math.sin(psi), cp = Math.cos(psi),
      x2 = x1 + r*psi, y2 = y1, // r = ball radius = 20
      matrix = "matrix(" + cp + "," + sp + "," + (-sp) + "," + cp + ","
                + (-x1*cp+y1*sp+x2) + "," + (-x1*sp-y1*cp+y2) + ")";

  ball.setAttribute("transform", matrix);
  psi += Math.PI/45; // increment every 4 degrees

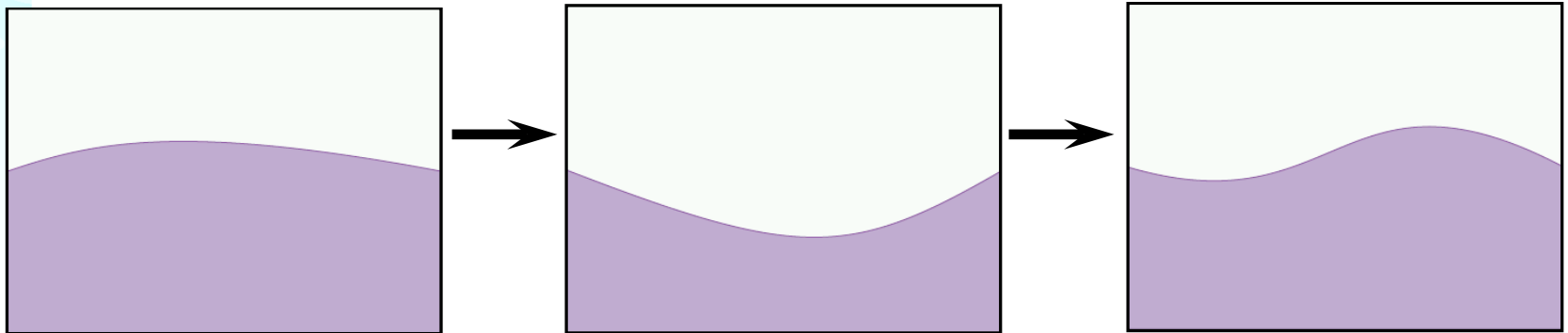
  if (x2 < xend)
    window.setTimeout("window.Animate()", 1);
  return true;
}
```

Rotate 4
degrees

Translate and
rotate the ball
around its center

Example Animation - Wave

- Object is shown behaving as if it was a wave



Anim02_js_path.svg

Wave – Basic Line

- A basic flat line is produced as follows:

```
<svg>
  <rect style="fill:none;stroke:black"
    x="0" y="0" width="399" height="299"/>
  <path id="line"
    style="fill:#905CA8; fill-opacity:0.5; stroke:#905CA8"
    d="M0,150 h400"/>
</svg>
```

Wave – Getting Started

- You can see that the SVG element which is being animated is a path called 'line'
- To do the animation we first find the element:
`var linenode = svgdoc.getElementById('line');`
- The path is stored in the 'd' attribute of that node
- So the path can be updated like this:
`linenode.setAttribute('d', new path data)`
- As long as we update the path data appropriately, we can achieve the animation effect

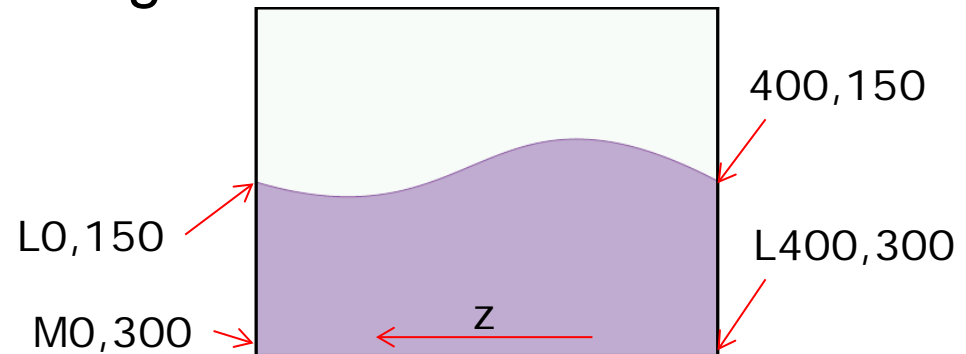
Wave Line Control Points

- In this example, four control points are used

`<path d="...L0,150C100,100,200,200,400,150...">`

The diagram shows the path command `<path d="...L0,150C100,100,200,200,400,150...">` with four points labeled below: **start-point** (at 0,150), **control-point-1** (at 100,100), **control-point-2** (at 200,200), and **end-point** (at 400,150). Brackets connect the labels to their respective coordinates in the command string.

- The animation is performed by changing the two central control points several times per second
- The end points are left unchanged



Animating the Control Points

- After generating new positions, setAttribute is used to put them in the path element like this:

```
linenode.setAttribute('d',  
'M0,300 L0,150 C' + x0 + ',' + y0 + ',' + x1 + ',' + y1 +  
' , 400,150 L400,300 z');
```

1st control point

2nd control point

End point

Create text sequence to draw the path
for the boundary of the violet region in
the animation (see previous slide)

- Recall that: M=move C=cubic Bezier curve
L=draw line to z=finish/go back to the start

Main Animation Function

```
function next_frame () {  
    var linenode = svgdoc.getElementById ('line'); // Get path object being  
    animated  
    if (!linenode) return;  
    if (tx0 < 0 || (tx0 == x0 && ty0 == y0 && tx1 == x1 && ty1 == y1))  
    {  
        tx0 = Math.floor (400*Math.random());  
        ty0 = Math.floor (300*Math.random());  
        tx1 = Math.floor (400*Math.random());  
        ty1 = Math.floor (300*Math.random());  
    }  
    // Change current coordinates by up to +/-10 pixels towards target values.  
    x0 = change_coord (x0, tx0); y0 = change_coord (y0, ty0);  
    x1 = change_coord (x1, tx1); y1 = change_coord (y1, ty1);  
    // Change the path element's "d" attribute to use the new coordinates.  
    linenode.setAttribute ('d',  
        'M0,300L0,150C'+x0+', '+y0+', '+x1+', '+y1+',400,150L400,300z');  
}
```

Generate new x,y of the control points only when the current target is reached

Function shown on next slide

Function for Changing Control Points

- Every call moves the coordinate value (current_value) 10 points closer to the target value to create smooth motion

```
function change_coord (current_value, target_value)    {  
    if (current_value < target_value)    {  
        current_value += 10;  
        if (current_value > target_value)  
            current_value = target_value;    }  
    if (current_value > target_value)    {  
        current_value -= 10;  
        if (current_value < target_value)  
            current_value = target_value;    }  
    return current_value;    }
```