UDC 004.92

SCALABLE VECTOR GRAPHICS

NIKOLAY YURTSEVICH, TATYANA RUDKOVA
Polotsk State University, Belarus

Scalable Vector Graphics (SVG) is a language for describing two-dimensional graphics in XML. This graphics can consist of paths, images, and/or text that may be scaled and resized without losing image quality.

Inline SVG refers to the embedded code written within HTML to generate this graphics in a browser. There are many advantages of using SVG in this way, including having access to all graphics individual parts for interactivity purposes, generating searchable text, DOM access for direct edits and promoting user accessibility.

There are a number of ways to include SVG in your projects: inline, an <img>, a background-image, an <object>, or as Data URI. I will specifically refer to the use of SVG inline which involves writing SVG code within the body of a properly structured HTML document.

SVG details reside within a <svg> element. This element contains several attributes which permit the customization of your graphics "canvas". Although these attributes are not completely necessary for image rendering, their absence may leave more complex graphics vulnerable when performing in browsers and make them susceptible to undesirable rendering. Inline graphics can be written "by hand", or embedded by accessing the XML code generated by vector graphic software. Anyway proper organization and structure are crucial to writing efficient SVG code, primarily because the order of these graphical elements determines their stacking order.

An SVG document fragment is made up of any number of SVG elements contained within the <svg> element. Organization within this document is crucial. Content in the document can be rapidly expanded and proper organization promotes accessibility and efficiency in everything that benefits both the author and users.

The <svg> element is classified as both a container and a structural element, and can be used to nest a standalone SVG fragment inside the document. This fragment establishes its own coordinate system. The attributes used in this element, such as width, height, save Aspect Ratio and viewbox, define the canvas for writing graphics.

The <g> element is a container element for grouping related graphics together. Using this element in combination with the description and title elements provides information about your graphics and aids in organization and accessibility by grouping related graphical components together. Also, by grouping related elements together I can manipulate the group as a whole versus the individual parts. This is especially convenient when animating these elements, for example, as the animation can be applied to the group. Any element that is not contained within a <g> is assumed to be its own group.

The <use> element allows you to reuse elements throughout a document. There are additional attributes that can be included in this element, such as x, y, width, and height, which define the mapping location details of the graphics within the coordinate system. Using the href attribute here enables you to call on the element to be reused. For example, if there was a <g> with an id of “square” containing the image of a square that needed to be reused this image can be referenced by <use>: <use x="50" y="50" href="# square " />. This element can be a significant time saver and helps to minimize the required code.

While <use> allows you to reuse already rendered graphics, graphics in the <defs> element is not displayed on the canvas, but can be specified and then displayed using href. Graphics is defined in <defs> and can then be used throughout the document by referencing the id of that graphics. The contents of the <defs> has no visual output until it is called on by referencing its unique id.

The <symbol> element is similar to <g> as it provides the way to group elements, however, elements within <symbol> have no visual output (like <defs>) until called on with the <use> element. Also unlike the <g> element, <symbol> establishes its own coordinate system separate from the viewport it's rendered in.

The stacking order of SVG cannot be manipulated by z-index in CSS as other elements in HTML. The order in which SVG elements are stacked depends entirely on their placement in the document fragment.

SVG contains the following set of basic shape elements: rectangles, circles, ellipses, straight lines, polylines, and polygons. Each element requires a set of attributes before it renders, like coordinates and size details.

The <rect> element defines a rectangle. The width and height attributes establish the size of the rectangle, while fill sets the interior color for the shape. The numerical values default to pixels and fill would default to black when left unspecified. Other attributes that can be included are x and y coordinates. These values will move the shape along the appropriate axis according to the dimensions set by the <svg> element. It is also pos-

sible to create rounded corners by specifying values in rx and ry attributes. For example, rx="5" ry="10" will produce horizontal sides of corners that have a 5px radius, and vertical sides of corners that have a 10px radius.

The <circle> element is mapped based on a center point and an outer radius. The cx and cy coordinates establish the location of the center of the circle in relation to the workplace dimensions set by the <svg>. The r attribute sets the size of the outer radius.

An <ellipse> element defines an ellipse that is mapped based on a center point and two radii. While the cx and cy values are establishing the center point based on pixel distance into the SVG coordinate space, the rx and ry values define the radius of the sides of the shape.

The <line> element defines a straight line with a start and end point. Together the x1 and y1 values establish the coordinates for the start of the line, while the x2 and y2 values establish the end of the line.

The <polyline> element defines a set of connected straight line segments, generally resulting in an open shape (start and end points that are not connected). The values in points establish the shapes location on the x and y axis throughout the shape and are grouped as x, y throughout the list of values. An odd number of points here is an error.

A <polygon> element defines a closed shape consisting of connected lines. The points of the polygon shape are defined through a series of grouped x, y values. This element can also produce different closed shapes depending on the number of defined points.

SVG <paths> represent the outline of a shape. This shape can be filled, stroked, used to navigate text, and/or used as a clipping path. Depending on the shape this path can get very complex, especially when there are many curves involved. Basic understanding of how they work and the syntax involved, however, will help to make these particular paths much more manageable.

The path data is contained in a d attribute in the <path> element, defining the outline for the shape: <path d="<path data specifics>". This data included in the d attribute spell out the moveto, line, curve, arc and closepath instructions for the path.

The moveto commands (M or m) establish a new point, as lifting a pen and start to draw in a new location on paper. The line of code comprising the path data must begin with a moveto command. Moveto commands that follow the initial one represent the start of a new subpath, creating a compound path. An uppercase M here indicates absolute coordinates will follow, while a lowercase m indicates relative coordinates.

The closepath (Z or z) ends the current subpath and results in a straight line being drawn from that point to the initial point of the path. If the closepath is followed immediately by a moveto, these moveto coordinates represent the start of the next subpath. If this same closepath is followed by anything other than moveto, the next subpath begins at the same point as the current subpath. Both the uppercase and the lowercase z have here identical outcomes.

The line to commands draws straight lines from the current point to a new point.

The L and l commands draw a line from the current point to the next provided point coordinates. This new point then becomes the current point, and so on. The uppercase L signals that absolute positioning will follow, while the lowercase l is relative.

The H and h commands draw a horizontal line from the current point. The uppercase H signals that absolute positioning will follow, while the lowercase h is relative.

The V and v commands draw a vertical line from the current point. The uppercase V signals that absolute positioning will follow, while the lowercase v is relative.

There are three groups of commands that draw curved paths: Cubic Bezier (C, c, S, s), Quadratic Bezier (Q, q, T, t), and Elliptical arc (A, a).

The C and c Cubic Bezier commands draw a curve from the current point using (x1,y1) parameters as a control point at the beginning of the curve and (x2,y2) as the control point at the end, defining the shape details of the curve. The S and s commands also draw the Cubic Bezier curve, but in this instance there is an assumption that the first control point is a reflection of the second control point. Manipulating the first and last sets of values for this curve will impact its start and end location, while manipulating the two center values will impact the shape and positioning of the curve itself at the beginning and end. The S and s commands also draw a Cubic Bézier curve, but in this instance there is an assumption that the first control point is a reflection of the last control point for the previous C command. This reflection is relative to the starting point of the S command. The uppercase C signals that absolute positioning will follow, while the lowercase c is relative. The same logics is applied to S and s.

Quadratic Bézier curves (Q, q, T, t) are similar to Cubic Bézier curves except that they only have one control point.

Manipulating the first and last sets of values impacts the positioning of the beginning and end points of the curve. The center set of values, Q defines the control point for the curve, establishing its shape. Q and q draw the curve from the initial point to the end point using (x1, y1) as the control. T and t draw the curve from the initial point to the end point by assuming that the control point is a reflection of the control on the previously listed command relative to the start point of the new T or t command. The uppercase Q signals that absolute positioning will follow, while the lowercase q is relative. The same logics is applied to T and t.

An Elliptical Arc (A, a) defines a segment of an ellipse. These segments are created through the A or a commands which create the arc by specifying the starting point, end point, x and y radii, rotation, and direction.

The viewport is the visible section of an SVG. While SVG can be as wide or as high as you wish, limiting the viewport means that only a certain section of the image can be visible in a certain period of time. The viewport is set through height and width attributes in the <svg>. If these values are not defined, the dimensions of the viewport are generally determined by other indicators in the SVG, like the width of the outermost SVG element. However, leaving them undefined it makes our artwork susceptible to be cut off.

The viewbox allows for the specification that a given set of graphics stretches to fit a particular container element. These values include four numbers separated by commas or spaces: min-x, min-y, width, and height that should generally be set to the bounds of the viewport. The min values represent at what point within the image the viewBox should start, while the width and height establish the size of the box. If we choose not to define a viewBox the image will not scale to match the bounds set by the viewport.

The min values within the viewBox define the origin of the viewBox within the parent element. In other words, the point in the viewbox in which you want it to be started is to match the viewport.

Thus, all of these elements are container structural elements in SVG that help to reuse elements easier while keeping the code cleaner and more readable. And each of the elements I considered in this article has its own use cases. Now as I know what each one does and how they differ, I can decide which one to use, depending on my needs.

REFERENCES

1. Sara Soueidan. Structuring, Grouping, and Referencing in SVG [Electronic resource] // www.w3schools.com. – Mode of access: https://www.sarasoueidan.com/blog/structuring-grouping-referencing-in-svg/. – Date of access: 20.01.2018.
2. SVG Tutorial [Electronic resource] // www.w3schools.com. – Mode of access: https://www.w3schools.com/graphics/svg_intro.asp. – Date of access: 21.01.2018.
3. Chris Coyier. Using SVG [Electronic resource] // www. css-tricks.com. – Mode of access: https://css-tricks.com/using-svg/. – Date of access: 21.01.2018.
4. About SVG [Electronic resource] // www.cs.sfu.ca. – Mode of access: https://www.cs.sfu.ca/CourseCentral/165/common/study-guide/content/jssvg-svg-about.html. – Date of access: 22.01.2018.
5. MDN web docs SVG [Electronic resource] // www. developer.mozilla.org. – Mode of access: https://developer.mozilla.org/en-US/docs/Web/SVG/Element/svg. – Date of access: 23.01.2018.