SVG

LoftSchool
от мыслителя к создателю

Содержание

1

Общие сведения

Масштабируемая векторная графика (Scalable Vector Graphics — SVG) является языком разметки, расширенным из XML для описания двухмерной векторной графики. SVG, по существу, является графикой, так же, как XHTML — текстом.

SVG по своим возможностям приближается к запатентованной технологии Adobe Flash, но отличается от неё тем, что SVG является рекомендацией W3C (то есть стандартом), и тем, что это формат, основанный на XML, в противовес закрытому двоичному формату Flash. Он явно спроектирован для работы с другими стандартами W3C, такими, как CSS, DOM и SMIL.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Created with Inkscape (http://www.inkscape.org/) -->

<svg
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:cc="http://creativecommons.org/ns#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns="http://www.w3.org/2000/svg"
```

Преимущества SVG:

1. Масштабирование: в отличие от растровой графики, SVG не теряет в качестве при масштабировании, поэтому ее удобно использовать для разработки под retina.
2. Уменьшение HTTP-запросов: при использовании SVG сокращается количество обращений к серверу, соответственно, увеличивается скорость загрузки сайта.
3. Стайлинг и скриптинг: при помощи CSS можно менять параметры графики на сайте, например, фон, прозрачность или границы.
4. Анимация и редактирование: при помощи javascript можно анимировать SVG, а также редактировать в текстовом или графическом редакторе (InkScape или Adobe Illustrator).
5. Малый размер: объекты SVG зачастую весят намного меньше растровых изображений.

2

Способы подключения

1. С помощью открывающего и закрывающего тега:

```
<svg> <!--SVG content--> </svg>
```

2. Как обыкновенный рисунок в HTML:

```
<img  src="logo.svg"  alt="Company Logo“ >
```

3. Подключить через свойство фона в CSS **background:**

```
.element {
background: url(mySVG.svg);
}
```

4. С помощью тега **<object>**, который сообщает браузеру, как загружать и отображать объекты, которые исходно браузер не понимает. Внутри этого тега можно разместить **fallback**, то есть резервное содержимое, которое подгрузится, если изображение с атрибутом **data** не доступно. Например, вставить туда загрузку картинки в другом формате с помощью тега **img**.

```
<object type="image/svg+xml" data="mySVG.svg">
<!--fallback-->
</object>
```

5. Как элемент **<embed>**, который используется для загрузки и отображения объектов, которые исходно браузер не понимает.

```
<embed type="image/svg+xml" src="mySVG.svg" />
```

6. Через тег **<iframe>**, который является контейнером и находится внутри обычного документа, он позволяет загружать в область заданных размеров любые другие независимые документы. Можно указать альтернативный

текст **fallback**, который увидят пользователи, если этот тег браузером не поддерживается.

```
<iframe  src="mySVG.svg">
   <!--fallback-->
</iframe>
```

3

Базовые фигуры в SVG

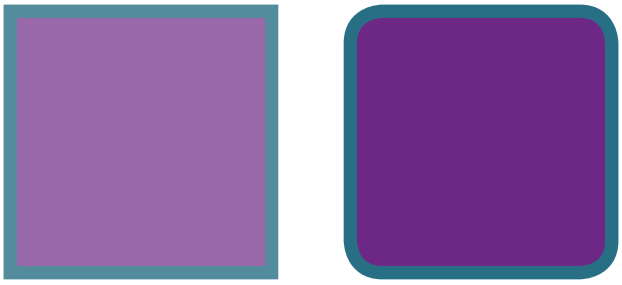
Прямоугольник <rect>

```
<svg width="500" height="200">
   <rect x="50" y="20"  width="150" height="150"
   class="svg-rect" />
   <rect x="250" y="20" rx="40" ry="40" width="150"
   height="150" class="svg-rect" style = "fill-
   opacity: 1; stroke-opacity: 1;"/>
</svg>
```

Строится с помощью тега **rect**. Тег имеет атрибуты x и y, которые указывают расстояние в пикселях от левого верхнего угла svg-элемента. Так же задаем высоту **height** и ширину **width** прямоугольника. Для оформления используем класс **"svg-rect"**.

```
.svg-rect {
    fill: #6D2885;
    stroke: #286F85;
    stroke-width: 5;
    fill-opacity: 0.7;
    stroke-opacity: 0.8;
  }
```

Где свойство **fill** — это цвет заливки прямоугольника, **stroke** — цвет обводки, а stroke-width — это толщина обводки в пикселях. И, наконец, **fill-opacity** и **stroke-opacity** — прозрачность заливки и обводки соответственно.

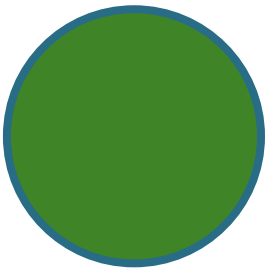


Для второго прямоугольника мы указываем дополнительные **inline-стили**, которые перекрывают правила из таблицы стилей. Так же используем два новых атрибута **rx** и **ry**, которые указывают радиусы округления углов прямоугольника.

Окружность <circle>

```
<svg height="200" width="200">
   <circle cx="100" cy="100" r="50" style="stroke: #286D85;
   stroke-width: 3; fill: #3F8528;"/>
</svg>
```

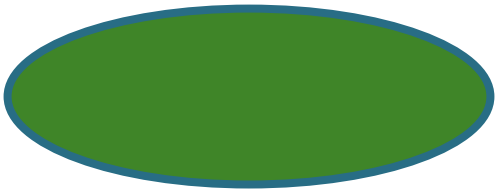
Здесь **cx** и **cy** — это координаты центра окружности, относительно левого верхнего угла svg-элемента. Радиус окружности указываем через атрибут **r**.



Эллипс <ellipse>

```
<svg height="200" width="300" class = "brd">
   <ellipse cx="150" cy="100" rx="100" ry="50" style="stroke:
   #286D85; stroke-width: 3; fill: #3F8528;"/>
</svg>
```

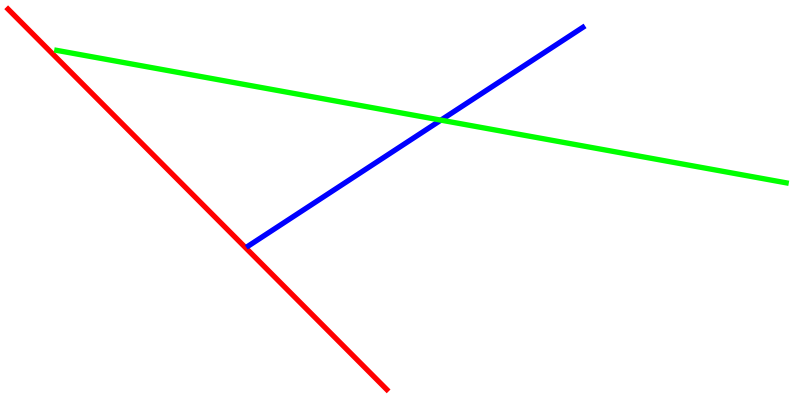
В эллипсе, из-за эксцентриситета, задаются два атрибута радиуса для каждой из осей координат, соответственно, **rx** и **ry**.



Линия <line>

```
<svg height="210" width="500" class = "brd">
  <line x1="10" y1="10" x2="200" y2="200" style="stroke:
rgb(255,0,0);stroke-width:2" />
  <line x1="30" y1="30" x2="400" y2="100" style="stroke:
rgb(0,255,0);stroke-width:2" />
  <line x1="130" y1="130" x2="300" y2="20" style="stroke
:rgb(0,0,255);stroke-width:2" />
</svg>
```

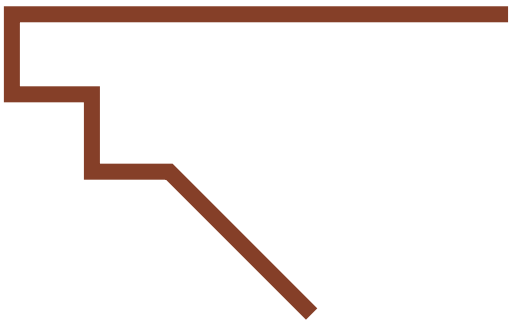
У линий задаётся начальная точка **x1="10" y1="10"** и конечная точка **x2="200" y2="200"** построения. У линий нет свойств заливки, и оформляются они свойствами обводки.



Полилиния <polyline>

```
<svg height="200" width="300">
  <polyline points="290,40 40,40 40,80 80,80 80,120
  120,120 190,190" fill="white" stroke="#853F28" stroke-
  width="6" />
</svg>
```

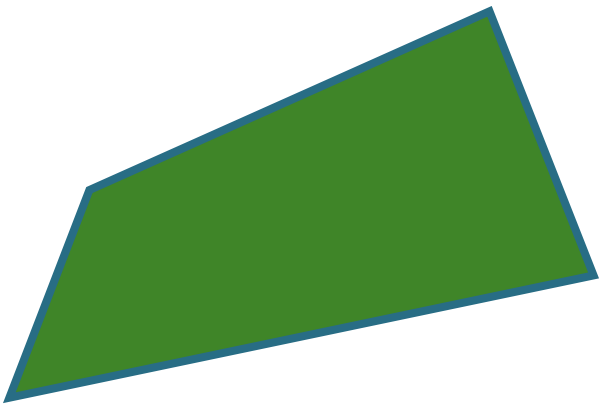
Координаты полилинии задаются в атрибуте **points**, они разделены пробелами, а значения самой точки по осям — запятыми. Все точки соединяются между собой последовательно от первой **290,40** до последней **190,190**.



Полигон <polygon>

```
<svg height="210" width="500" class = "brd">
   <polygon points="300,10 350,140 60,200 100,100" style=
   "fill:#408528;stroke:#286E85;stroke-width:3" />
</svg>
```

Построение полигона происходит по тем же правилам, что и полилинии, но последняя координата всегда соединяется с первой, и внутренняя область закрашивается согласно значению свойства **fill**.



Текст <text>

```
<svg width="500" height="200" class = "brd">
   <text x="20" y="120" fill="#ED6E46" font-size="100"
   font-family="'Arial', cursive">Loftschool</text>
</svg>
```

Координаты x и y указывают в нашем случае нижнюю левую точку буквы **“L”**.

4

Организация документа

Фрагмент SVG-документа состоит из неограниченного количества SVG-элементов, находящихся внутри элемента **<svg>**. Организация внутри этого документа является ключевой.

Элемент svg

Элемент ***<svg>*** является и контейнером, и структурным элементом, и может быть использован для вложения отдельного SVG-фрагмента внутрь документа. Этот фрагмент устанавливает свою собственную систему координат.

Атрибуты, используемые внутри элемента, такие как width, height, preserveAspectRatio и viewBox определяют холст для создаваемой графики.

Элемент g

Элемент ***g*** является контейнером для группировки связанных графических элементов.

```
<g
   inkscape:label="Layer 1"
   inkscape:groupmode="layer"
   id="layer1">
  <path
     sodipodi:type="star"
     style="fill:#00ff00;stroke:#000000;stroke-lir
     opacity:1"
     id="path4136"
     sodipodi:sides="5"
     sodipodi:cx="222.85714"
     sodipodi:cy="232.3622"
     sodipodi:r1="228.57143"
     sodipodi:r2="114.28571"
     sodipodi:arg1="0.92729522"
     sodipodi:arg2="1.5556137"
     inkscape:flatsided="false"
     inkscape:rounded="0"
     inkscape:randomized="0"
     d="M 360,415.21934 224.59224,346.63474 91.329
     114.71367,269.32452 4.4255155,165.03821 154.2
     219.38698,3.817115 288.62114,138.89391 439.14
     07296,266.02419 Z"
     inkscape:transform-center-x="1.0723416"
     inkscape:transform-center-y="-20.804256" />
</g>
```

Элемент <use>

Позволяет повторно использовать элементы в любом месте документа. К этому элементу можно добавить такие атрибуты, как **x, y, width** и **height**, которые определяют подробности положения элемента в системе координат.

```
<svg>
  <g id = “apple”>
    code
  </g>
  <use x="50" y="50" xlink:href="#apple" />
</svg>
```

Атрибут **xlink:href** здесь позволяет обратиться к элементу, чтобы использовать его повторно. Например, если у нас есть элемент **<g>** с идентификатором "apple", содержащий изображение яблока, то на это изображение можно ссылаться с помощью:

```
<use>: <use x="50" y="50" xlink:href="#apple" />.
```

Элемент defs

Графика внутри элемента ***<defs>*** не отображается на холсте, но на неё можно ссылаться и затем отображать её посредством **xlink:href**.

Элемент symbol

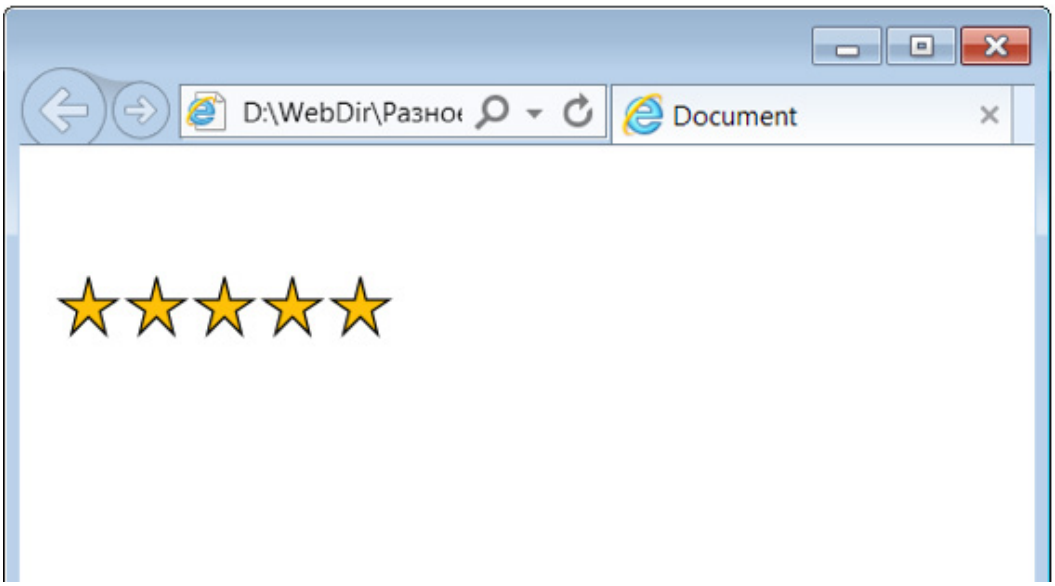
Элемент ***<symbols>*** похож на **<g>**, так как предоставляет возможность группировать элементы, однако элементы внутри **<symbols>** не отображаются визуально (как и в **<defs>**) до тех пор, пока не будут вызваны с помощью элемента **<use>**.

Также, в отличие от элемента **<g>, <symbols>** устанавливает свою собственную систему координат, отдельно от области просмотра, в которой он отображается.

Пример

Для лучшего понимания рассмотрим следующий пример:

```
<svg xmlns="http://www.w3.org/2000/svg">
   <defs>
        <polygon id="elem" points="25,0 30,20 50,20 35,30 40,
        50 25,35 10,50 15,30 0,20 20,20" style="stroke:black;
        stroke-width:2" transform="scale(0.5)" />
   </defs>
   <use x="10" y="50" class="star" xlink:href="#elem" />
   <use x="70" y="50" class="star" xlink:href="#elem" />
   <use x="100" y="50" class="star" xlink:href="#elem" />
   <use x="130" y="50" class="star" xlink:href="#elem" />
</svg>
```



Класс **star** имеет следующий вид:

```
.star {
   fill: #FFC107;
   stroke: #E65100;
}
```

Порядок наложения

Порядок наложения SVG не может управляться через **z-index** в CSS, как другие элементы внутри HTML. Порядок, в котором раскладываются SVG-элементы, полностью зависит от их размещения внутри фрагмента документа.

5

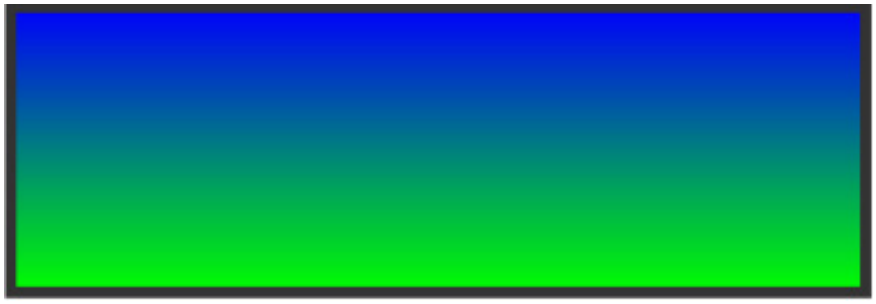
Градиент

Существует два типа SVG-градиентов: ***линейные*** и ***радиальные***. У линейных градиентов переход между цветами происходит вдоль прямой линии, а у радиальных — в круге.

Линейные градиенты изменяют цвет равномерно вдоль прямой линии и каждая ключевая точка, которая определена на этой линии, будет представлять соответствующий цвет в пределах элемента **<linearGradient>**. В каждой точке цвет является на 100% чистым, в промежуточных точках — смесь в разном соотношении, а область между ними отображает переход от одного цвета к другому.

Элементам **<stop>** также можно задавать прозрачность при помощи **stop-opacity="<значение>" offset** передаёт градиенту, в какой точке установить соответствующий **stop-color**.

```
<svg width="400" height="200" class = "brd">
   <defs>
        <linearGradient id="Gradient1" x1="0" y1="0"
        x2="0" y2="100%">
             <stop offset="0%" stop-color="#00F" />
             <stop offset="100%" stop-color="#0F0" />
        </linearGradient>
   </defs>
   <rect x="50" y="50" width="300" height="100" fill=
   "url(#Gradient1)" stroke="#333333" stroke-width="4px" />
</svg>
```



Если в нашем примере (см. рис.) атрибуту **x2** задать значение **«100%»**, а атрибуту **y2** — **«0»**, то мы получим горизонтальный градиент, а если наоборот — вертикальный. Установив оба значения в **«100%»** (или в любое значение, отличное от 0), мы получим наклонный градиент.

```
<svg width="400" height="200" class = "brd">
   <defs>
        <linearGradient id="Gradient1" x1="0" y1="100%"
   x2="100%" y2="0">
             <stop offset="0%" stop-color="#00F" />
             <stop offset="100%" stop-color="#0F0" />
        </linearGradient>
   </defs>
   <rect x="50" y="50" width="300" height="100" fill=
    "url(#Gradient1)" stroke="#333333" stroke-width="4px" />
</svg>
```

6

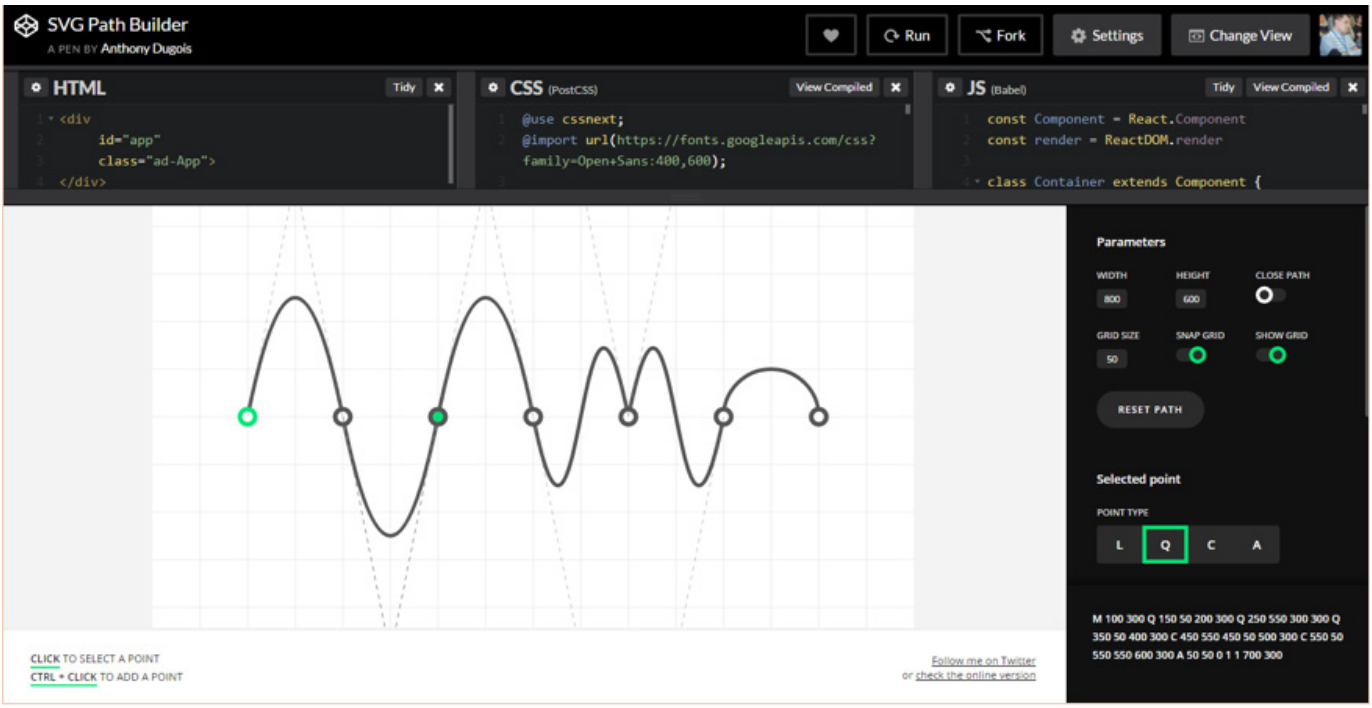
Элемент path

Сложная фигура или контур. Данные **path** содержатся в атрибуте **d** внутри элемента **<path>**, определяя форму фигуры:

```
<path d="<конкретные данные path>" />
```

Данные, включённые в атрибут **d**, описывают команды для **path: moveto, line, curve, arc** и **closepath**.

Для лучшего понимания есть замечательный пример. Разберитесь в нём. Через этот элемент работают векторные графические редакторы, такие как Adobe Illustrator и Inkscape.



Более подробно об элементе path можно почитать в данной лекции на Intuit.ru

7

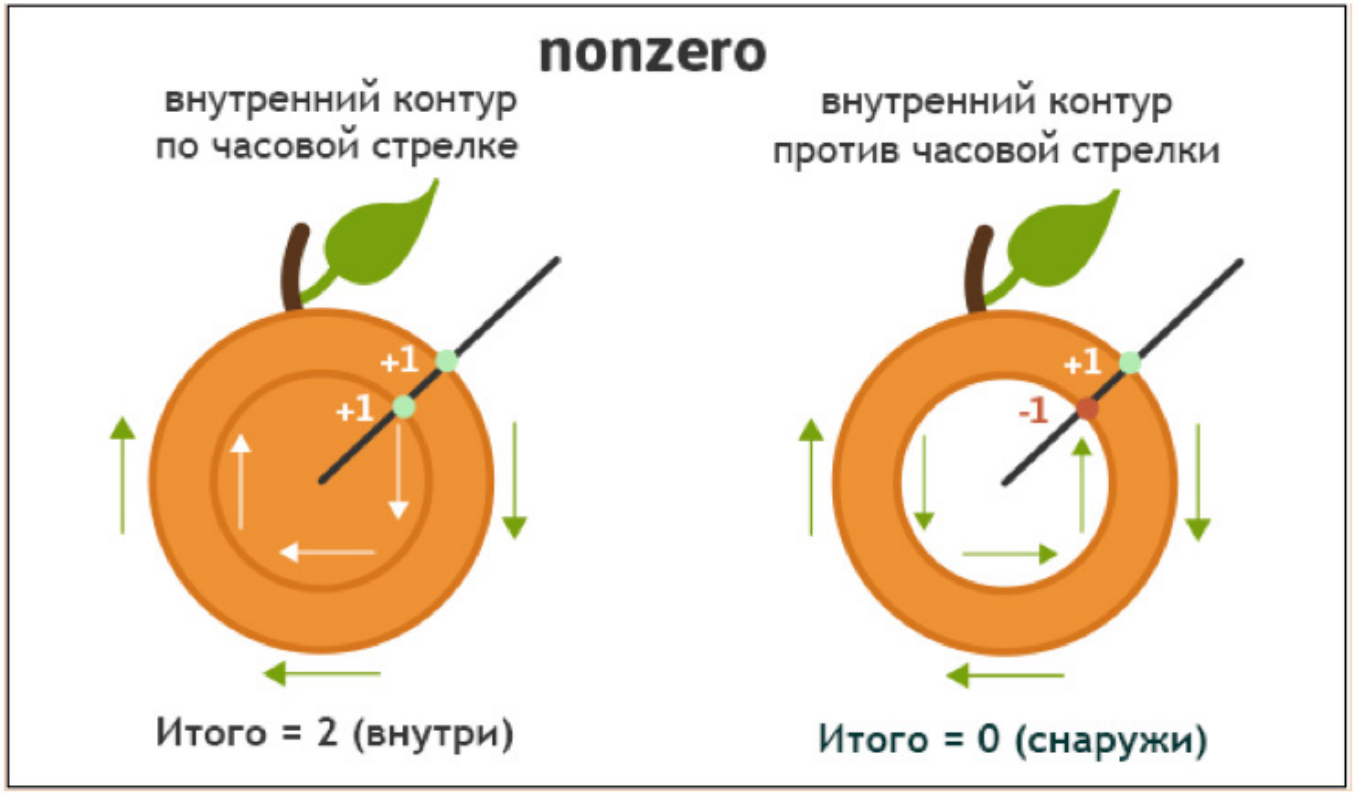
Заливки и обводки

Атрибуты ***fill*** и ***stroke*** позволяют раскрашивать внутреннюю часть и границу SVG.

Под «раскрашиванием» понимаются операции добавления цвета, градиентов или паттернов для графики при помощи **fill** и/или **stroke**. Атрибут **fill** раскрашивает внутреннюю часть определённого графического элемента. Это заливка может состоять из сплошного цвета, градиента или паттерна.

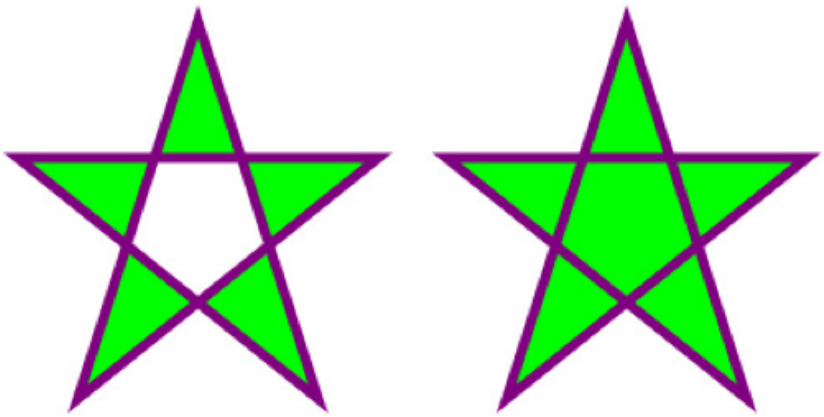
Внутренняя часть фигуры определяется путём анализа всех подконтуров и параметров, описанных в **fill-rule**.

Допустимые значения **fill-rule: nonzero, evenodd, inherit**.



```
<svg height="210" width="210">
   <polygon points="100,10 40,198 190,78 10,78 160,198"
   style="fill:lime;stroke:purple;stroke-width:5;fill-
   rule:evenodd;" />
</svg>

<svg height="210" width="210">
   <polygon points="100,10 40,198 190,78 10,78 160,198"
   style="fill:lime;stroke:purple;stroke-width:5;
   fill-rule:nonzero;" />
</svg>
```



Обводка

Атрибут **stroke** определяет закрашивание «границы» конкретных фигур и контуров.

У следующего изображения будет сиреневая обводка: **stroke = "#765373"**. Определяет форму, которая будет на концах линий атрибут:

stroke-linecap

А за форму соединения линий отвечает атрибут:

stroke-linejoin

8

Рабочая область

Рабочая область определяется размерами области просмотра и атрибутами **viewBox**.

Область просмотра является видимой частью SVG. Хотя SVG может быть какой угодно ширины или высоты, ограничение области просмотра будет означать, что в любой момент времени может быть видна только часть изображения.

Область просмотра устанавливается атрибутами height и width в элементе **<svg>**.

Если эти значения не заданы, размеры рабочей области обычно будут определены по другим показателям в SVG, например, по ширине самого внешнего элемента SVG. Однако без указания этих значений есть риск, что графический объект обрежется.

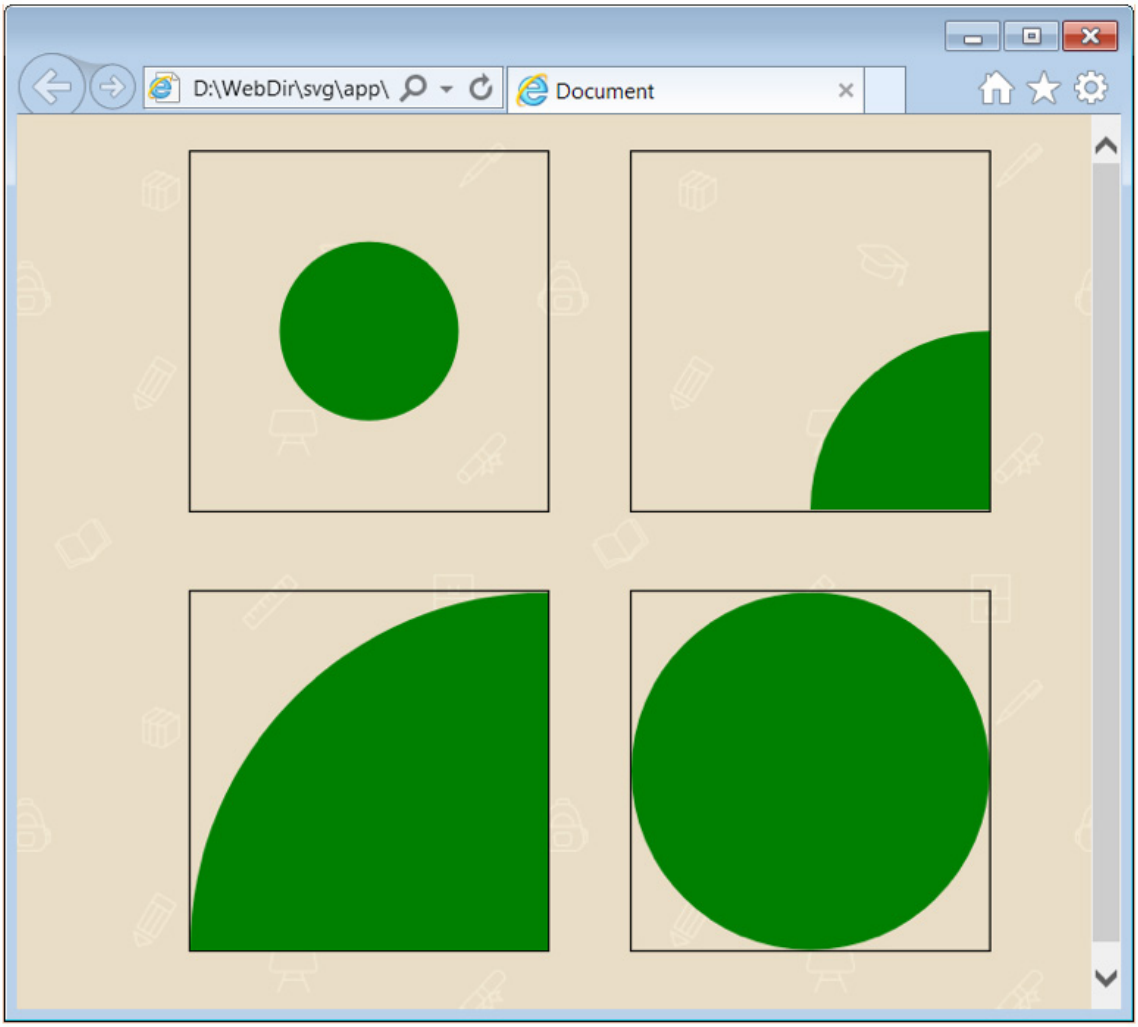
ViewBox даёт возможность указать, что данный набор графических элементов растягивается, чтобы уместиться в определенный элемент-контейнер. Эти значения включают четыре числа, разделённые запятыми или пробелами: **min-x, min-y, width** и **height,** которым чаще всего следует задать значения границ области просмотра.

Значения min определяют, в какой точке внутри изображения должен начинаться viewBox, в то время как **width** и **height** устанавливают размер блока.

Если мы решим не определять **viewBox**, тогда изображение не будет масштабироваться, чтобы совпадать с границами, установленными областью просмотра.

Фактически мы задаем окошко, через которое смотрим на **svg** и изображение при этом растягивается на всю область просмотра. Лучше это показать на примерах:

```
<div class="row">
   <svg class = "brd" width ="200" height ="200"
   viewBox = "0 0 200 200">
       <circle cx="100" cy="100" r="50"
       style ="fill:green;" />
   </svg>
   <svg class = "brd" width ="200" height ="200"
   viewBox = "0 0 100 100">
       <circle cx="100" cy="100" r="50"
       style ="fill:green;" />
   </svg>
</div>
<div class="row">
   <svg class = "brd" width ="200" height ="200"
   viewBox = "50 50 50 50">
       <circle cx="100" cy="100" r="50"
       style ="fill:green;" />
   </svg>
   <svg class = "brd" width ="200" height ="200"
   viewBox = "50 50 100 100">
       <circle cx="100" cy="100" r="50"
       style ="fill:green;" />
   </svg>
</div>
```



В первом случае **viewBox** совпадает с размерами svg-элемента, и зелёный круг мы видим четко по центру. Во втором квадрате мы уменьшили **viewBox** наполовину и, само собой, мы увидим четверть исходного изображения, ещё и растянутого на всю область просмотра. В третьем случае мы изменили **min-x, min-y** и подошли к краю зелёного круга своей областью просмотра, а затем задали ширину и высоту **viewBox** по **50**, равной радиусу нашего зелёного круга. Если же задать для ширины и высоты **viewBox** диаметр окружности **100**, то зелёный круг займет всю область просмотра (четвертый вариант).

9

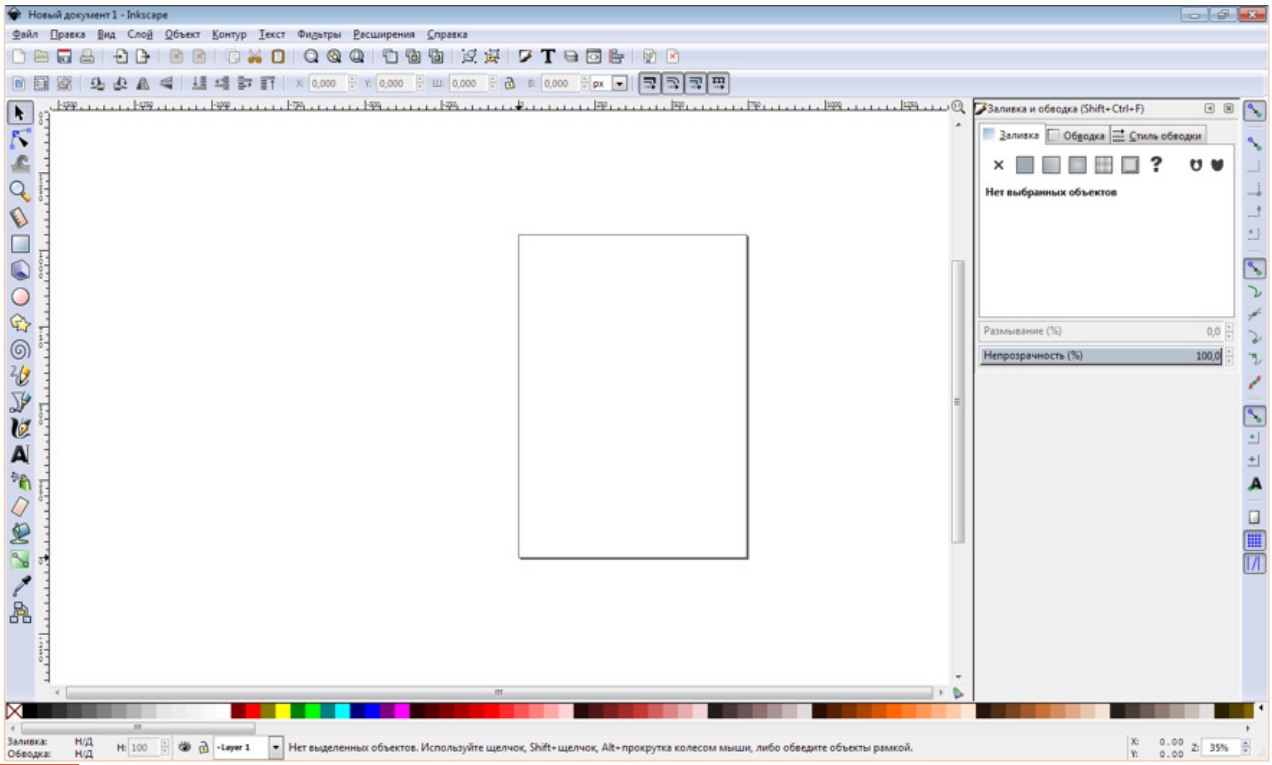
Графические приложения, позволяющие создавать SVG-графику

- Adobe Illustrator
- CorelDRAW Graphics Suite
- Mayura Draw
- Sketsa SVG Editor
- Inkscape

Бесплатным является **Inkscape** — мощный бесплатный инструмент для дизайна.

Основные возможности:

- Гибкие инструменты для рисования;
- совместимость со многими форматами файлов;
- мощный инструмент для работы с текстом;
- кривые Безье и Корню.



10

Основные библиотеки для работы с SVG

- Vivus – http://maxwellito.github.io/vivus/
- Bonsai – https://bonsaijs.org/
- Velocity – http://julian.com/research/velocity/
- Raphaël – http://dmitrybaranovskiy.github.io/raphael/
- Snap – http://snapsvg.io/
- Lazy Line Painter – http://lazylinepainter.info/
- SVG.js – http://svgjs.com/
- Walkway – https://github.com/ConnorAtherton/walkway
- Highcharts JS – HIGHCHARTS JS

Пример использования библиотеки на основе **Highcharts JS**:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>SVG</title>
</head>
<body>
   <div id="container" style="min-width: 310px;
   height: 400px; margin: 0 auto"></div>
   <script type="text/javascript" src="js/jquery.js"></script>
   <script src = "js/highcharts.js"></script>
   <script>
        $('#container').highcharts({
             chart: {
                  renderTo: 'charts', defaultSeriesType: 'bar'
             },
             title: {
```

```
                text: 'Количество выигранных турниров
                игроками в покер'
            },
            xAxis: {
                categories: ['Январь', 'Февраль', 'Март']
            },
            yAxis: {
                title: { text: 'Количество турниров' }
            },
            series: [{ name: 'Женя', data: [1, 3, 4] },
            { name: 'Саша', data: [5, 7, 3]}]
        });
    </script>
</body>
</html>
```

11

Список ссылок на литературу и источники

Примеры и учебники:

- Примеры svg-графики с применением фильтров обработки изображений
- Учебник на русском
- Учебник на английском
- Учебник Jacoba Jenkova
- Примеры CSS анимации
- Формат SVG: от иконок до живых картин
- SVG элементы
- Быстрые и простые диаграммы на svg
- Эффекты blur на svg

Подключение иконок и полезности:

- Как мы используем SVG-спрайты(новый способ)
- SVG-иконки – много и со стилем
- Animated SVG Icon
- Хранилище иконок
- Caching SVG Sprite in localStorage
- CREATING SVG SPRITES USING GULP AND SASS
- Хранилище популярных лого на SVG
- Конвертор файлов формата PNG в SVG
- Онлайн генератор svg спрайтов

Доклады:

- Приручаем SVG – Лев Солнцев
- Dmitry Baranovskiy - You Don't Know SVG
- Introduction to SVG and RaphaelJS