SVG

LoftSchool
от мыслителя к создателю

Оглавление

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"
   xmlns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd"
   xmlns:inkscape="http://www.inkscape.org/namespaces/inkscape"
   width="160.29124mm"
   height="155.20148mm"
   viewBox="0 0 567.9611 549.92649"
   id="svg2"
   version="1.1"
   inkscape:version="0.91 r13725"
   sodipodi:docname="drawing.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.
6. В SVG, при необходимости, можно вставлять ***растровые изображения***.

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 данного тега можно разместить fallback, то есть резервное содержимое, которое подгрузится если изображение с атрибута 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>
```

Базовые фигуры в SVG

LINE. Рисуем линию

Линию легко нарисовать при помощи <line> тега и всего двух параметров, – точки начала и точки окончания линии.

```
<svg>

  <line x1="5" y1="5" x2="100" y2="100" stroke-width="8" stroke="violet"/>

</svg>
```

С помощью параметра `stroke-width` можно задать ширину линии, а с помощью параметра `stoke` – цвет. У линий нет свойств заливки, и оформляются они свойствами обводки.

POLYLINE. Рисуем ломаную линию

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.

```
<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>
```

RECT. Рисуем прямоугольник

```
<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.

CIRCLE. Рисуем окружность

Параметры cy и cx представляют собой центральную точку, а r – радиус.

```
<svg>
  <circle cx="50" cy="50" r="50" fill=«yellowgreen"/>
</svg>
```

Без указания координат центра окружности, центр определяется автоматически, в верхнем левом углу полотна svg.

```
<svg>
  <circle r="50" fill="yellowgreen" />
</svg>
```

ELLIPSE. Рисуем эллипс

Эллипс ожидает параметры центральной точки и двух радиусов.

```
<svg>

  <ellipse cx="50%" cy="50%" rx="100" ry="40%"
fill="red"/>

</svg>
```

Без указания центральной точки, аналогично окружности, центр определится автоматически в верхнем левом углу полотна svg.

```
<svg>

  <ellipse rx="100" ry="40%" fill="red"/>

</svg>
```

POLYGON. Рисуем многоугольник

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` .В параметр `points` мы передаем точки, по которым будет рисоваться наша замкнутая фигура, состоящая из связанных линий.

```
<svg>

  <polygon points="50,5 100,5 125,30 75,80 25, 30"
fill=«lightblue"/>

</svg>
```

```
<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”.

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"
   transform="translate(-105.19796,-173.79311)">
  <path
     sodipodi:type="star"
     style="fill:#80d086;fill-opacity:1;stroke:#000000;stroke-linejoin:round;stroke-opacity:1"
     id="path8068"
     sodipodi:sides="5"
     sodipodi:cx="394.28571"
     sodipodi:cy="472.36221"
     sodipodi:r1="298.52701"
     sodipodi:r2="149.2635"
     sodipodi:arg1="0.88708702"
     sodipodi:arg2="1.5154055"
     inkscape:flatsided="false"
     inkscape:rounded="0"
     inkscape:randomized="0"
     d="M 582.85714,703.79079 402.54931,621.39679 232.45582,723.21967 255.09899,526.27558 105.6979,395.97207 l
     194.30208,-39.32414 77.75854,-182.35488 97.44212,172.64043 197.4585,17.60201 -134.07953,146.02179 z"
     inkscape:transform-center-x="5.1071823"
     inkscape:transform-center-y="-23.605852" />
</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 width="100" height="100" viewBox="0 0 100 100">
    <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="10" xlink:href="#elem" />
</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 нее можно ссылаться и затем отображать ее 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:

```
<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="40" 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.

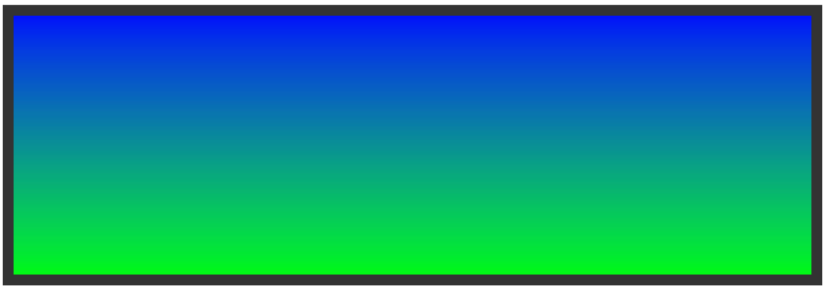
Градиент

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.

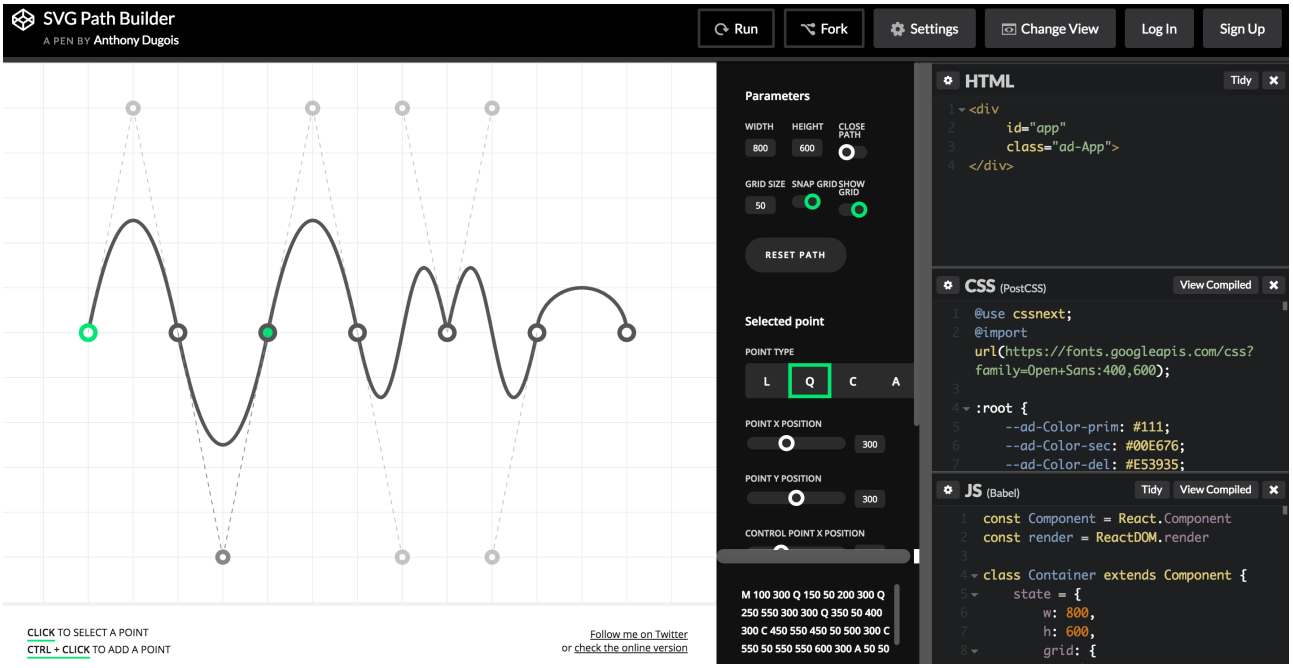
```
<svg width="400" height="200" class = "brd">

  <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>
```

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 нем.

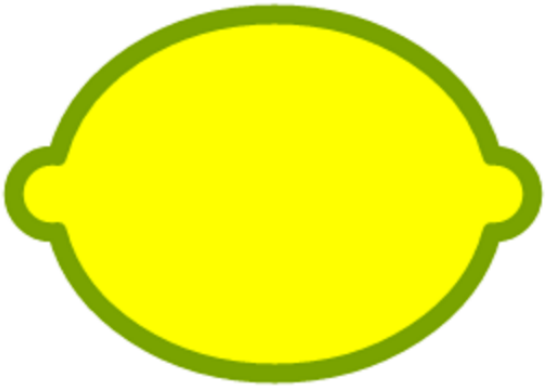
Через этот элемент работают векторные графические редакторы такие как Adobe Illustrator и Inkscape.



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

Давайте нарисуем произвольную фигуру.
В параметр d мы передадим команды, определяющие контур фигуры. Например: moveto, lineto, curve, arc, closepath и т.д.

```
<svg width="258px" height=«184px">

  <path fill="yellow" stroke="#7AA20D" stroke-width="9"
stroke-linejoin="round"
d="M248.761,92c0,9.801-7.93,17.731-17.71,17.731c-0.319,0-
0.617,0-0.935-0.021c-10.035,37.291-51.174,65.206-100.414,
65.206
c-49.261,0-90.443-27.979-100.435-65.334c-0.765,0.106-1.53
1,0.149-2.317,0.149c-9.78,0-17.71-7.93-17.71-17.731
c0-9.78,7.93-17.71,17.71-17.71c0.787,0,1.552,0.042,2.317,
0.149C39.238,37.084,80.419,9.083,129.702,9.083
c49.24,0,90.379,27.937,100.414,65.228h0.021c0.298-0.021,0
.617-0.021,0.914-0.021C240.831,74.29,248.761,82.22,248.76
1,92z" />

</svg>
```



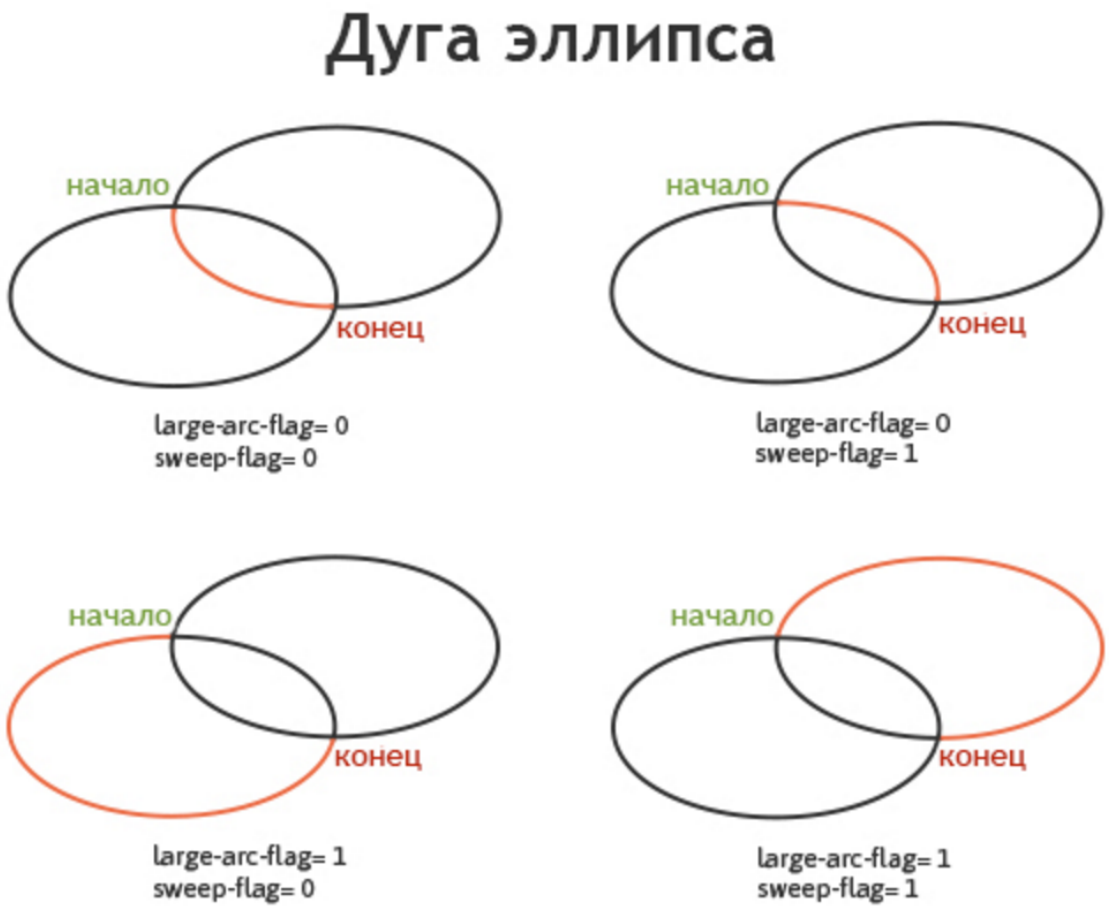
moveto(M/m) – указывает куда будет смотреть курсор и точку, с которой следующим шагом начнется процесс рисования.

lineto – рисует линию.
L/l – линия от текущей точки до следующих координат точки;
H/h – горизонтальная линия от текущей точки.
V/v – вертикальная линия от текущей точки.

C/c, S/s – кубическая кривая Безье

Q/q, T/t – квадратичная кривая Безье

A/a – дуга эллипса



Заливки и обводки

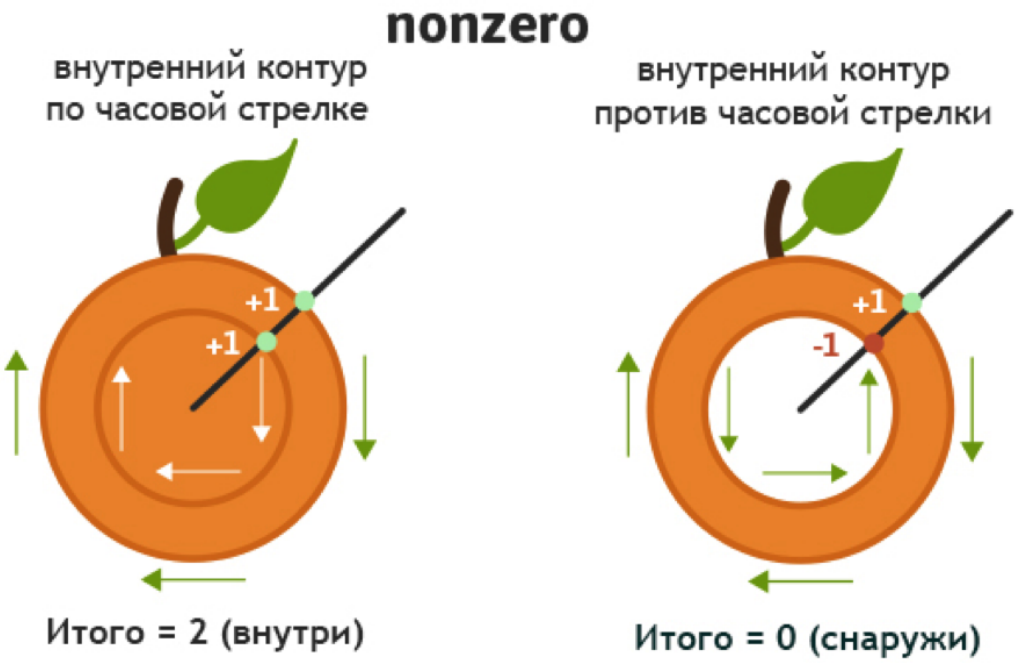
Атрибуты `fill` и `stroke` позволяют раскрашивать внутреннюю часть и границу SVG.

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.

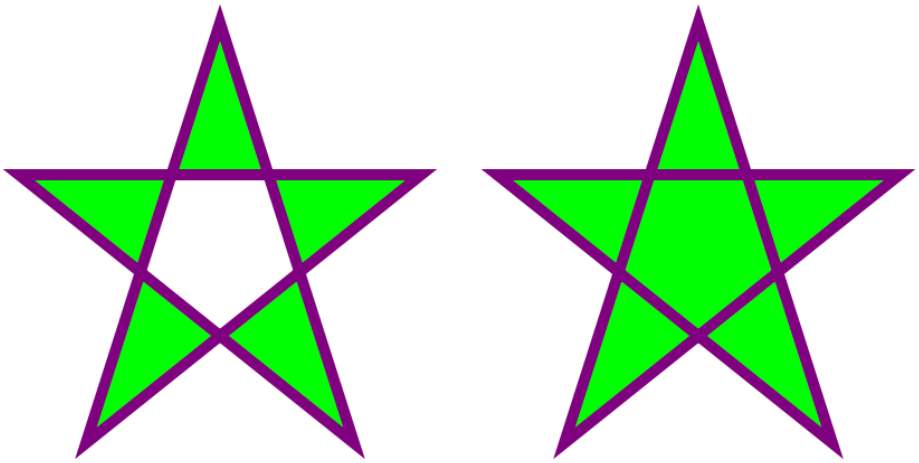
Внутренняя часть фигуры определяется путём анализа всех подконтуров и параметров, описанных в fill-rule.

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>
```

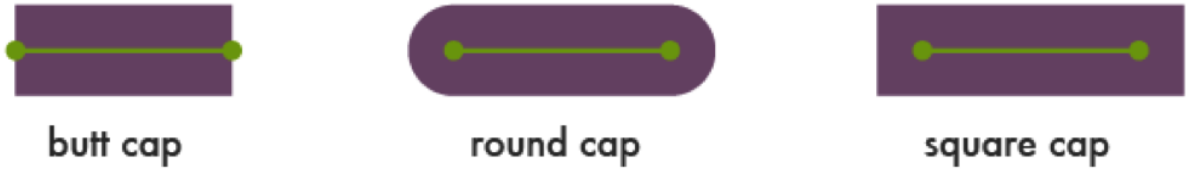


Атрибут `stroke` определяет закрашивание «границы» конкретных фигур и контуров.

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

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 дает 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.

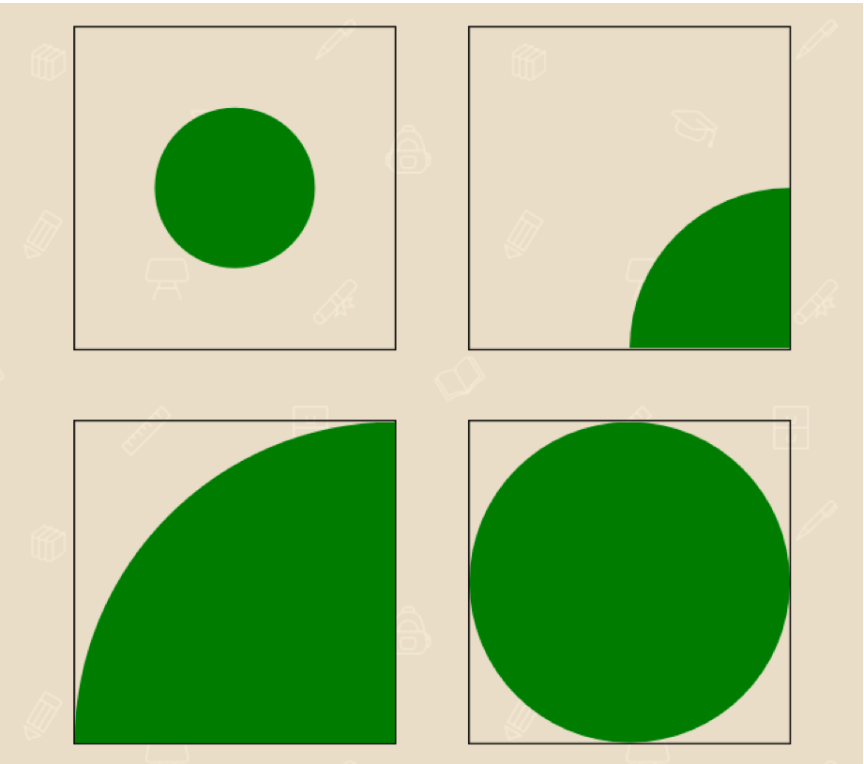
Значения min определяют, в какой точке внутри изображения должен начинаться viewBox, в то время как width и height устанавливают размер блока.

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 показать это 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>
  </svg>
</div>
```



В первом случае viewBox совпадает с размерами svg-элемента, и зеленый круг мы видим четко по центру.

Во втором квадрате, мы наш viewBox уменьшили на половину и само собой мы увидим четверть исходного изображения, еще и растянутого на всю область просмотра.

В третьем случае, мы изменили min-x, min-y и подошли к краю зеленого круга своей областью просмотра, а затем задали ширину и высоту viewBox по 50, равной радиусу нашего зеленого круга.

Если же задать для ширины и высоты viewBox диаметр окружности 100, то зеленый круг займет всю область просмотра, ***четвертый вариант***.

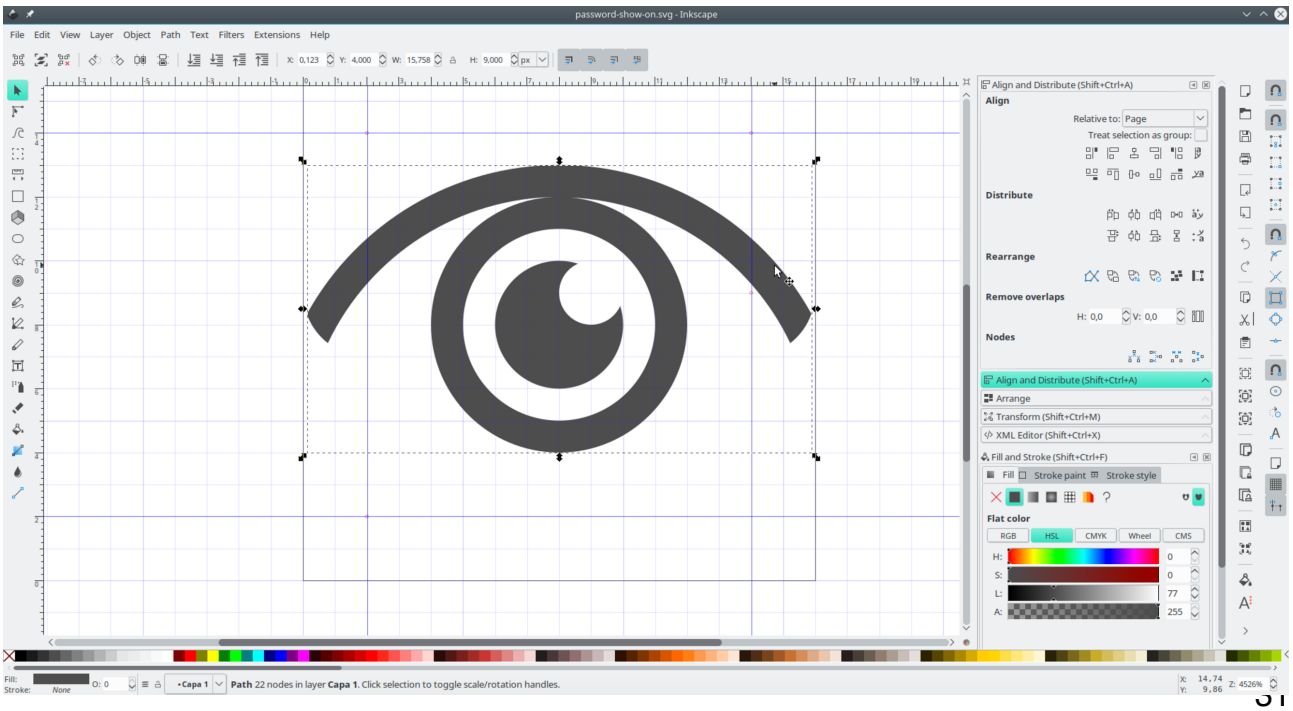
Графические приложения, позволяющие создавать SVG-графику

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

Бесплатным является **Inkscape** – мощный, бесплатный инструмент для дизайна.

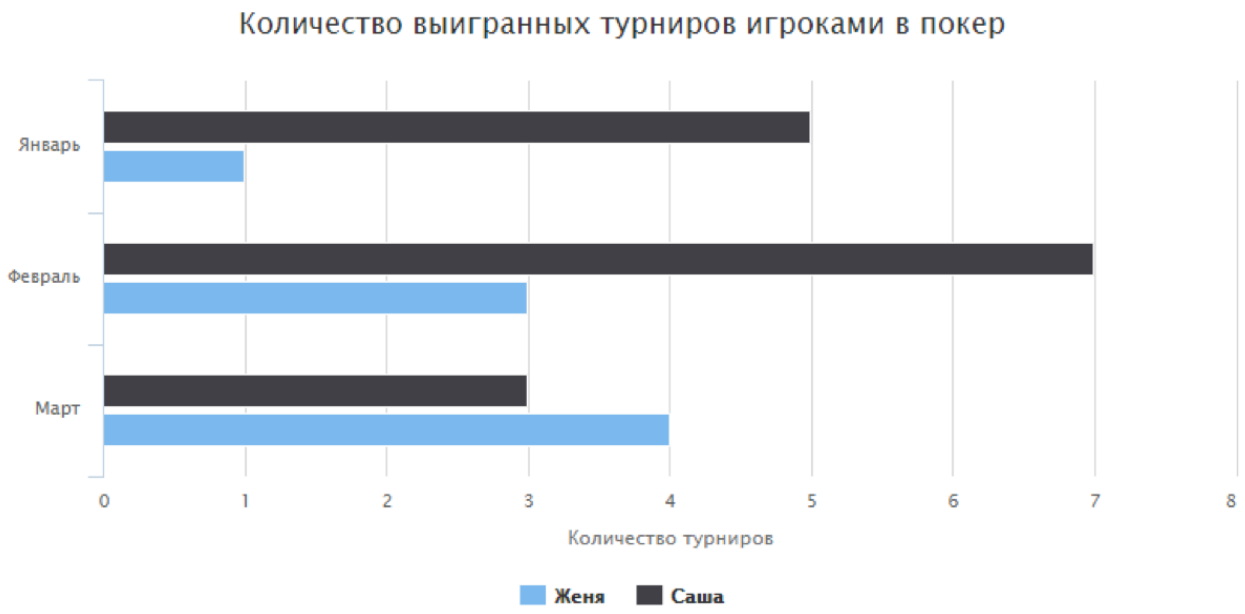
Основные возможности:

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



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



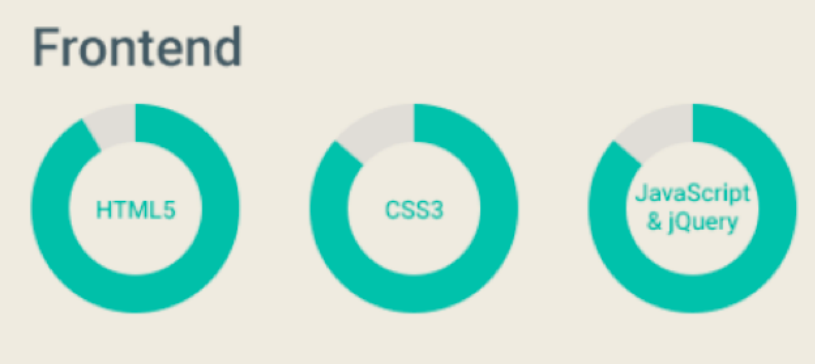
Пример использования библиотеки на основе Highcharts JS:

```
<!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>
```

Построение круговых диаграмм из макета

В нашем макете есть круговые диаграммы навыков следующего вида:



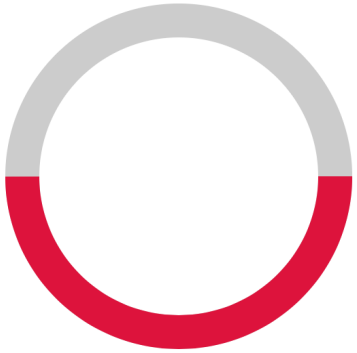
Как их сделать? Можно использовать готовый плагин, но мы попробуем сделать сами. Есть два атрибута (свойства) построения границы, которые нам помогут.

Первый атрибут stroke-dasharray управляет видом пунктирной обводки. Можно задавать в единицах длины или в процентах. Если задано одно значение, второе значение считается равным первому. Например, stroke-dasharray="1" нарисует линию из отрезков длиной одну единицу разделенных пробелами такой же ширины.

Второе свойство stroke-dashoffset позволяет задать смещение пунктирной обводки относительно первоначального положения. Значение задается в единицах длины или в процентах, значения могут быть отрицательными. Значение по умолчанию: 0.

```
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style>
    .g-circles {
      fill: none;
      stroke-width: 25;
    }

    .sector {
      stroke-dasharray: 722;
      stroke-dashoffset: 361;
      transition: all 1s;
    }

  </style>
</head>

<body>
<svg width="270" height="270" class="g-circles">
  <circle r="115" cx="135" cy="135" stroke="#ccc"
fill="none" />
  <circle r="115" cx="135" cy="135" stroke="crimson"
class="sector" />
</svg>
</body>

</html>
```

Без поворота



С поворотом

Что здесь происходит?

Построено две окружности радиусом 115.
Первая окружность с серым цветом и толщиной границы 25, без заливки, вторая окружность с малиновым цветом границы. Вторая окружность занимает нижнюю часть общего построения. Ее свойства:

```
stroke-dasharray: 722;
stroke-dashoffset: 361;
```

Откуда взялось число 722?
Это длина нашей окружности с радиусом 115. Высчитывается по формуле длины 2*Радиус*(На число Пи) или численном виде 2*115*3.1415 = 722. Число 361 это половина 772 или наши 50% навыков.

Но ее надо еще повернуть на 90 градусов против часовой стрелки, чтобы ее начало совпадало с макетом, для этого построение второго круга изменим следующим образом:

```
<circle r="115" cx="135" cy="135" stroke="crimson"
transform="rotate(-90 135 135)" class="sector" />
```

А можно было обойтись только одним свойством `stroke-dasharray`. Как? Очень просто, достаточно класс .sector сделать таким:

```
.sector {
  stroke-dasharray: 361 722;
  transition: all 1s;
}
```

Очень много полезных ссылок

Примеры и учебники:

1. Примеры svg-графики с применением фильтров обработки изображений
2. Учебник на русском
3. Учебник на английском
4. Учебник Jacoba Jenkova
5. Примеры CSS анимации
6. Формат SVG: от иконок до живых картин
7. SVG элементы
8. Быстрые и простые диаграммы на svg
9. Эффекты blur на svg

Подключение иконок и полезности:

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

Доклады:

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