

# Карманное руководство по написанию SVG.

Перевод книги [\*Pocket Guide to Writing SVG\*](#), с разрешения автора — [\*Джони Трайтел\*](#).



- **Введение**
- [\*Глава 1. Организация документа\*](#)
- [\*Глава 2. Базовые фигуры и контуры\*](#)
- [\*Глава 3. Рабочая область\*](#)
- [\*Глава 4. Заливки и обводки\*](#)
- [\*Глава 5. Элемент `<text>`\*](#)
- [\*Глава 6. Продвинутое функции: градиенты, паттерны, контуры обрезки\*](#)
- [\*Заключение\*](#)

## Спасибо вам!

Этот раздел я хотела бы целиком посвятить тому, чтобы сказать «Огромное спасибо!»:

- [CSS-Tricks](#)
- [Lincoln Loop](#)
- [Designmodo](#)
- [Tahoe Partners](#)



Ваша поддержка этой книги очень ценна для меня, и я очень надеюсь, что никого из вас не обидела, если нечаянно забыла включить ваш любимый фрукт.

# Введение

Масштабируемая векторная графика (**SVG**) – язык для описания двумерной графики в формате **XML**. Эта графика может состоять из путей, изображений и/или текста, который способен масштабироваться и изменять размеры без потери качества изображения.

Под встроенным **SVG** понимают код, написанный прямо внутри **HTML**, чтобы сгенерировать в браузере эту графику, о которой эта книга.

У такого использования **SVG** есть много преимуществ, включая доступ ко всем отдельным частям графики для интерактивности, генерации текста с возможностью поиска, доступ к DOM для прямого редактирования, и улучшение доступности для пользователя.

Начав с базовой структуры и простых фигур, мы перейдем к описанию системы **SVG**-координат или «холста», рисованию заливки и/или обводки фигур, трансформациям, а так же использованию и обработке графического текста. Ну и завершим, коснувшись более продвинутых функций, таких как градиенты и паттерны.

Цель этого руководства — быстрое, но основательное введение в построение встроенного **SVG**, и, хотя оно ни в коей мере не покрывает всех его возможностей, оно должно помочь вам начать. Это руководство предназначено для дизайнеров и разработчиков, желающих добавить **SVG** в свою работу самым доступным способом.

От мелких подробностей обводки до введения в ручное создание паттернов, это руководство задумано как некий "универсальный справочник" по написанию **SVG**.

## Прежде чем начать

Хотя это «Карманное руководство» предназначено для тех, кто уже немного разбирается в **HTML** и **CSS**, есть несколько дополнительных вещей, о которых полезно знать, прежде чем браться за код **SVG** в вашем любимом браузере. В их числе: какая информация в **SVG**-фрагменте нужна для правильного отображения, как сделать вашу графику максимально доступной, и знание, как и когда пользоваться векторными графическими редакторами.

## Использование SVG

Есть [несколько способов включить SVG](#) в ваши проекты: как встроенный, как `<img>`, фоновое изображение, `<object>`, или в виде **Data URI**. Мы же с вами будем использовать именно встроенный **SVG**, который предполагает написание **SVG**-кода внутри тела правильно структурированного **HTML**-документа.

Несмотря на то, что в этой книге мы будем использовать встроенный **SVG**, бывают случаи, когда другие методы могут быть наиболее подходящими. Например, если вам не нужны в возможностях редактирования самой графики или доступа к её отдельным частям, то использование её в виде `<img>` может лучше подойти для вашего проекта.

## Программы для векторной графики

Векторные графические редакторы могут быть оправданным выбором, когда нужно создавать более сложную графику, которую нерационально писать вручную. Такие программы, как [Adobe Illustrator](#), [Inkscape](#), [Sketch](#), [iDraw](#) или [WebCode](#) могут оказаться полезными инструментами, которые стоит добавить в вашу **SVG**-копилку.

Преимуществом этих инструментов является то, что вы можете экспортировать их **SVG**-код и встроить его прямо в ваш **HTML**. Мы коснёмся этого немного позже.

## Встроенный SVG в вебе

Для краткости на всём протяжении этой книги **SVG DOCTYPE**, номер версии `xmlns`, и `xml:space` был исключён из всех примеров кода.

Эти атрибуты определяют [используемую версию SVG](#) и [пространство имён](#) документа. Главное, что здесь надо запомнить — вам, как правило, не придется указывать эти атрибуты, чтобы ваша графика успешно отобразилась в браузере.

Давайте теперь взглянем на эти атрибуты в примере **SVG**-кода, сгенерированном в Иллюстраторе, чтобы гарантировать, что для вас это не будет неожиданностью, когда вы начнёте работать с **SVG**.

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
xml:space="preserve"></svg>
```

В большинстве случаев в **DOCTYPE** и этих атрибутах внутри элемента `<svg>` нет необходимости и от них можно избавиться, существенно «почистив» ваш код.

## Доступность SVG для пользователей

Использование [средств доступности SVG](#) — отличная привычка, которую стоит выработать, но опять же, для краткости, описания и заголовки не будут включены в код на всём протяжении книги.

Как только вы станете более опытными в написании **SVG**, включение этих элементов сделают вашу графику более доступной для пользователей. Например, содержимое внутри элемента `<desc>` позволит вам обеспечить подробное описание графики для пользователей скринридеров.

С точки зрения доступности текст в **SVG** также обеспечивает огромное преимущество перед традиционными растровыми изображениями, поскольку **SVG**-текст может быть обнаружен и прочтен, и легко меняет размер, подстраиваясь под определённые настройки чтения.

## Общие замечания

Еще пара общих замечаний, прежде чем перейти к главному: шрифты, используемые в примерах во всей книге, доступны на [Google Fonts](#). Это будет видно по значениям `font-family`, но соответствующий код для `link` или `@import`, получаемый с **Google Font**, здесь не приводится, и вам придется вставить его в документ самостоятельно.

В примерах книги используются исключительно пиксели и проценты в качестве единиц измерения. **SVG** поддерживает следующие единицы длины: `em`, `ex`, `px`, `pt`, `pc`, `cm`, `mm`, `in`, и проценты.

**SVG**-код из этой книги можно добавить в любой текстовый редактор, а затем просмотреть в любом браузере [с поддержкой встроенного SVG](#). Хотя в целом браузерная поддержка **SVG** очень хороша, для более продвинутых функций, например, градиентов, между браузерами могут возникнуть разночтения. [Can I Use](#) — прекрасное место для проверки поддержки таких особенностей, но в конечном счёте ничто

не сравнится с тем, чему вы научитесь методом проб и ошибок.

Помимо сказанного, вы также можете скопировать код из примеров, вставить его в окошко "HTML" на [CodePen](#) и мгновенно увидеть вашу графику на экране. Мне не хватает лестных слов для этого инструмента, так как по сути именно он в первую очередь заставил меня полюбить SVG. Это мой любимый способ учиться: играя, экспериментируя, иногда даже совершая досадные ошибки.

Наконец, в некоторых примерах отдельные части кода графики будут закомментированы, для уменьшения размера самого блока кода, когда эта конкретная часть не будет относиться к обсуждаемой теме.

```
<svg>

  <!--<путь d=<этот путь закомментирован> />-->

</svg>
```

Публикуется под лицензией [Attribution-NonCommercial-ShareAlike 4.0](#)

## Глава 1. Организация документа

Все детали SVG находятся в элементе `<svg>`. Этот элемент содержит несколько атрибутов, которые позволяют настроить ваш графический «холст». Хотя отрисовать изображение можно и без этих атрибутов, но если их не указывать, то более сложная графика может остаться уязвимой к особенностям браузеров и риск, что она в них не отобразится как надо, будет выше.

Как уже упоминалось, встроенная графика может быть написана «вручную», или копированием XML-кода, сгенерированного программой для векторной графики. В любом случае, правильная организация и структура являются ключевыми для написания эффективного SVG-кода, и прежде всего потому, что эти графические элементы определяют их порядок наложения.

### Организация и семантика

Фрагмент SVG-документа состоит из неограниченного количества SVG-элементов, находящихся внутри элемента `<svg>`. Организация внутри этого документа является ключевой. Содержимое внутри документа может быстро расширяться, а правильная организация способствует доступности и эффективности во всех отношениях, от чего выигрывают как автор, так и пользователи.

Этот раздел будет знакомством с основами написания SVG – элементом `<svg>` и обзором некоторых общих атрибутов, которые помогают в начальной установке документа.

### 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.

При получении SVG-кода из определённых графических программ, внутри элемента `<svg>`

[добавляется много информации](#), такой как номер SVG-версии (указывает используемую версию языка SVG) и DOCTYPE. Как я упоминала, эта информация не будет включена в примеры на всём протяжении этого руководства, и их исключение не мешает вашей графике отобразиться на экране.

## Элемент g

Элемент `g` является контейнером для группировки связанных графических элементов. Его использование в связке с элементами заголовка и описания предоставляет информацию о вашей графике и помогает в организации и доступности объединяя связанные графические компоненты в группу.

Также благодаря группировке связанных элементов вы можете управлять группой как единым целым, а не отдельными частями. Это особенно удобно при анимации этих элементов, так как можно анимировать всю группу.

Любой элемент, который не содержится внутри `g`, рассматривается как группа из него одного.

## Элемент use

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.

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" />`.

Этот элемент может существенно сэкономить время и помочь минимизировать требуемый код.

## Элемент defs

Если `<use>` позволяет повторно использовать отображённую графику, графика внутри элемента `<defs>` не отображается на холсте, но на нее можно ссылаться и затем отображать ее посредством `xlink:href`.

Графика определяется внутри `<defs>` и затем может быть использована во всём документе с помощью привязки по `id`.

Например, следующий код рисует очень простой градиент внутри прямоугольника:

```
<svg>

  <defs>

    <linearGradient id="Gradient-1">

      <stop offset="0%" stop-color="#bbc42a" />

      <stop offset="100%" stop-color="#765373" />

    </linearGradient>

  </defs>

  <rect x="0" y="0" width="100%" height="100%" style="fill: url(#Gradient-1);"/>

</svg>
```

```
</linearGradient>

</defs>

<rect x="10" y="10" width="200" height="100" fill= "url(#Gradient-1)" stroke="#333333" stroke-
width="3px" />

</svg>
```

Содержимое `<defs>` визуально никак не отображается, пока на него не сошлется по уникальному `id`, в данном случае с помощью атрибута `fill` прямоугольника

## Элемент `symbol`

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

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

Область просмотра SVG и `viewBox`, которые устанавливают систему координат для графики, преобразуемой для вывода, будут рассмотрены далее в другом разделе.

## 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.

Виноград и арбуз ниже находятся внутри одного и того же элемента `<svg>`. Арбуз располагается впереди винограда, так как группа, содержащая пути, из которых в итоге получается арбуз, следует в документе после винограда.

```
<svg>

  <g class="grapes">

    <!--<path <путь для черешка> />-->

    <!--<path <путь для винограда> />-->

    <!--<path <путь для листочка> />-->

  </g>

  <g class="watermelon">
```

```

    <!--<path <путь для корки> />-->

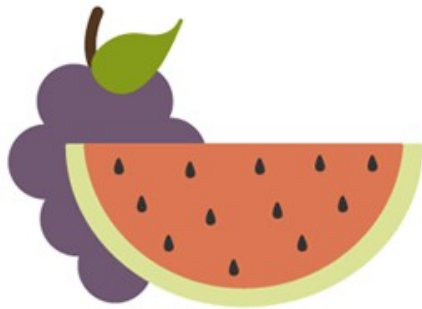
    <!--<path <путь для мякоти> />-->

    <!--<path <путь для семечек> />-->

</g>

</svg>

```



Если бы группа, содержащая виноград, была перемещена в конец документа, тогда она оказалась бы перед арбузом.

```

<svg>

  <g class="watermelon">

    <!--<path <путь для корки> />-->

    <!--<path <путь для мякоти> />-->

    <!--<path <путь для семечек> />-->

  </g>

  <g class="grapes">

    <!--<path <путь для черешка> />-->

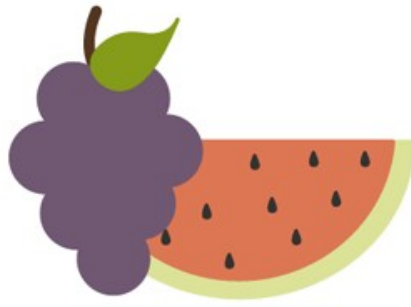
    <!--<path <путь для винограда> />-->

    <!--<path <путь для листочка> />-->

  </g>

</svg>

```



Этот метод, определяющий порядок наложения, также применяется к отдельным элементам внутри группы. Например, перемещение пути для черешка на картинке с виноградом в конец его группы приведет к тому, что он окажется поверх всего.



Публикуется под лицензией [Attribution-NonCommercial-ShareAlike 4.0](#)

## Глава 2. Базовые фигуры и контуры

Базовые фигуры SVG можно вписывать в HTML вручную, но со временем вы можете столкнуться с необходимостью вписать более сложную графику. Такую графику можно создавать в векторных редакторах, а пока давайте рассмотрим основы, код для которых легко писать вручную.

### Базовые фигуры

SVG содержит следующий набор основных фигур: прямоугольник, круг, эллипс, прямые линии, ломаные линии и многоугольники. Каждому элементу требуется набор атрибутов, чтобы он мог отобразиться, напр. координаты и параметры размера.

### Прямоугольник

Элемент `<rect>` определяет прямоугольник

```
<svg>

  <rect width="200" height="100" fill="#BBC42A" />

</svg>
```





[Увидеть демо можно здесь.](#)

Атрибуты `width` и `height` устанавливают размер прямоугольника, в то время как `fill` — внутренний цвет фигуры. Если не указаны единицы измерения, то по умолчанию это будут пиксели, а если не указать `fill`, по умолчанию цвет будет чёрным.

Другие атрибуты, которые могут быть добавлены – координаты `x` и `y`. Эти значения передвинут фигуру вдоль соответствующей оси согласно размерам, установленным элементом `<svg>`.

Также есть возможность создать закруглённые углы, указав значения в атрибутах `rx` и `ry`. К примеру, `rx="5" ry="10"` сгенерирует горизонтальные стороны углов с радиусом `5px`, а вертикальные в `10px`.

## Круг

Элемент `<circle>` строится, основываясь на центральной точке и внешнем радиусе.

```
<svg>

  <circle cx="75" cy="75" r="75" fill="#ED6E46" />

</svg>
```



[Увидеть демо можно здесь.](#)

Координаты `cx` и `cy` определяют положение центра круга относительно размеров рабочей области, заданных элементом `<svg>`.

Атрибут `r` устанавливает размер внешнего радиуса.

## Эллипс

Элемент `<ellipse>` описывает эллипс, который строится по центральной точке и двум радиусам.

```
<svg>

  <ellipse cx="100" cy="100" rx="100" ry="50" fill="#7AA20D" />

</svg>
```



[Увидеть демо можно здесь.](#)

В то время как значения атрибутов `cx` и `cy` помещают центральную точку на указанном расстоянии в пикселях от начала SVG-координат, значения `rx` и `ry` определяют радиус сторон фигуры.

## Линия

Данный элемент определяет прямую линию с начальной и конечной точкой.

```
<svg>

  <line x1="5" y1="5" x2="100" y2="100" stroke="#765373" stroke-width="8"/>

</svg>
```



[Увидеть демо можно здесь.](#)

Значения атрибутов `x1` и `y1` устанавливают координаты начала линии, а значения `x2` и `y2` — определяют конец линии.

## Ломаная линия

`<polyline>` определяет набор соединённых отрезков прямой линии, что в результате даёт, как правило, незамкнутую фигуру (начало и конец точек которой не связаны).

```
<svg>
```

```
  <polyline points="0,40 40,40 40,80 80,80 80,120 120,120 120,160" fill="white" stroke="#BBC42A" stroke-width="6" />
```

```
</svg>
```



[Увидеть демо можно здесь.](#)

Значения в `points` определяют положение фигуры по осям  $x$  и  $y$  от начала до конца фигуры и разбиты по парам  $x, y$  во всём списке значений.

Нечётное число точек является ошибкой.

## Многоугольник

Элемент `<polygon>` определяет замкнутую фигуру состоящую из связанных линий.

```
<svg>
```

```
  <polygon points="50,5 100,5 125,30 125,80 100,105 50,105 25,80 25,30" fill="#ED6E46" />
```

```
</svg>
```



[Увидеть демо можно здесь.](#)

Вершины многоугольника заданы последовательностью из восьми пар значений  $x, y$ .

Также, в зависимости от числа определяемых точек, этот элемент может создавать и другие замкнутые фигуры.

## Элемент path

SVG-элемент `path` представляет собой контур фигуры. Эта фигура может быть заполнена, обведена, использована как направляющая для текста и/или как контур обрезки.

В зависимости от фигуры, эти контуры могут быть очень сложными, особенно когда в них содержится множество кривых. Но если разобраться в их работе и соответствующем синтаксисе, то и такие контуры станут гораздо более управляемыми.

## Данные path

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*.

Детали `<path>` ниже определяют особенности контура для рисунка лайма:

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

  <path fill="#7AA20D" 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-0.765,0.106-1.531,0.149-
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-
0.021C240.831,74.29,248.761,82.22,248.761,92z" />

</svg>
```



### moveto

Команды `moveto` (`M` или `m`) устанавливают новые точки, как будто мы поднимаем ручку и начинаем рисовать в новом месте на листе бумаги. Строка кода, составляющего данные `path`, должна начинаться с команды `moveto`, как показано выше в примере с лаймом.

Команды `moveto`, которые следуют за исходной, представляют собой начало нового фрагмента контура, создавая составной контур. Заглавная `M` указывает, что после нее идут абсолютные координаты, тогда как строчная `m` указывает на относительные координаты.

## closepath

Closepath (Z и z) заканчивает текущий фрагмент контура приводит к рисованию прямой линии от текущей точки до начальной.

Если команда moveto следует непосредственно за closepath, то координаты moveto представляют собой начало следующего фрагмента контура. Если за closepath следует любая команда кроме moveto, то следующий фрагмент контура начинается в той же самой точке, где и текущий фрагмент контура.

И заглавная и строчная буква z здесь имеют одинаковые результаты.

## lineto

Команды lineto рисуют прямые линии от текущей точки до новой.

## L, l

Команды L и l рисуют линию от текущей точки до следующих предоставленных координат точки. Эта новая точка затем становится текущей точкой и так далее.

Заглавная L означает, что после неё идёт абсолютное позиционирование, в то время как после l — относительное.

## H, h

Команды H и h рисуют горизонтальную линию от текущей точки.

Заглавная H означает, что после неё идёт абсолютное позиционирование, в то время как после h — относительное.

## V, v

Команды V и v рисуют вертикальную линию от текущей точки.

Заглавная V означает, что после неё идёт абсолютное позиционирование, в то время как после v — относительное.

## Команды для создания кривых

Для рисования кривых есть три группы команд: кубическая кривая Безье (C, c, S, s), квадратичная кривая Безье (Q, q, T, t), и дуга эллипса (A, a)..

Следующие разделы о кривых вводят основные понятия каждой команды для кривых, рассматривают подробности построения и затем приводят диаграмму для дальнейшего понимания.

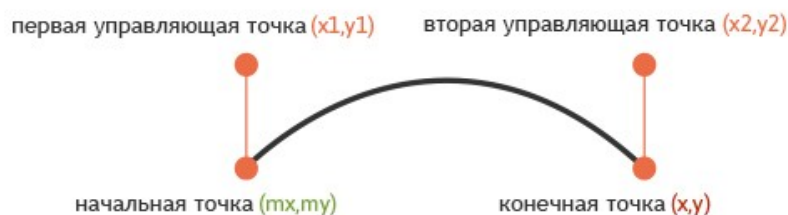
### Кубическая кривая Безье

Команды C и c рисуют кубическую кривую Безье от текущей точки, используя параметры (x1, y1) в качестве управляющей точки в начале кривой и (x2, y2) в качестве управляющей точки в конце, определяющих особенности формы кривой.

Команды S и s также рисуют кубическую кривую Безье, но в данном случае предполагается, что первая

управляющая точка является *отражением* второй (имеется в виду вторая управляющая точка предыдущей команды, см. более подробное описание ниже — прим. перев.).

## Кубическая кривая Безье



Следующий код рисует базовую кубическую кривую Безье:

```
<svg>

  <path fill="none" stroke="#333333" stroke-width="3" d="M10,55 C15,5 100,5 100,55" />

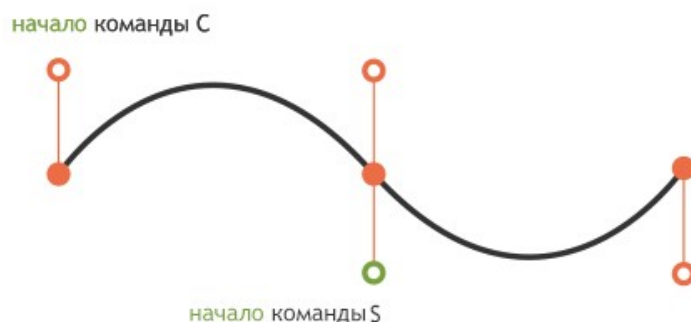
</svg>
```

[Увидеть демо можно здесь.](#)

Управление первыми и последними наборами значения для этой кривой повлияет на положение ее начала и конца, в то время как управление двумя центральными значениями повлияет на форму и ориентацию самой кривой в начале и конце.

Команды S и s также рисуют кубическую кривую Безье, но в данном случае предполагается, что первая управляющая точка является *отражением* последней для предшествующей команды C. Отражение производится относительно начальной точки команды S.

## Отражение команды S

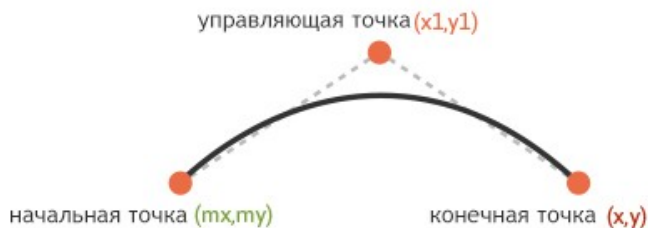


Заглавная C сигнализирует, что за ней следует абсолютное позиционирование, в то время как после строчной c — относительное. Та же самая логика применяется к S и s.

## Квадратичная кривая Безье

Квадратичные кривые Безье (Q, q, T, t) похожи на кубические, но имеют всего одну управляющую точку.

## Квадратичная кривая Безье



Следующий код рисует базовую квадратичную кривую Безье.

```
<svg>

  <path fill="none" stroke="#333333" stroke-width="3" d="M20,50 Q40,5 100,50" />

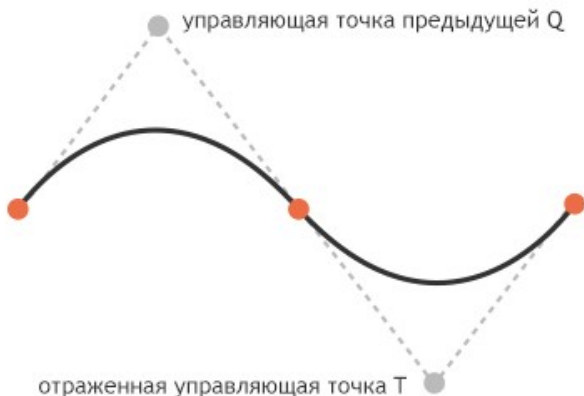
</svg>
```

[Увидеть демо можно здесь.](#)

Управление первыми и последними наборами значений,  $M_{20,50}$  и  $100,50$  будет влиять на позиционирование начала и конца точек кривой. Центральный набор значений  $Q_{40,5}$  задаёт управляющую точку для кривой, определяя ее форму.

Q и q рисуют кривую от начальной точки до конечной, используя  $(x1,y1)$  в качестве управляющей точки. T и t рисуют кривую от начальной точки до конечной, предполагая, что управляющая точка является отражением управляющей точки *предыдущей* команды относительно начальной точки новой команды T или t.

## Управляющая точка команды T



Заглавная Q сигнализирует, что за ней следует абсолютное позиционирование, в то время как после строчной q относительное. Та же самая логика применяется к T и t.

## Дуга эллипса

Дуга эллипса (A, a) определяет часть эллипса. Эти части создаются с помощью команд A или a, которые создают дугу путем указания начальной и конечной точки, радиусов  $x$  и  $y$ , вращения и направление.

Взгляните на код для базовой дуги эллипса:

```
<svg>
```

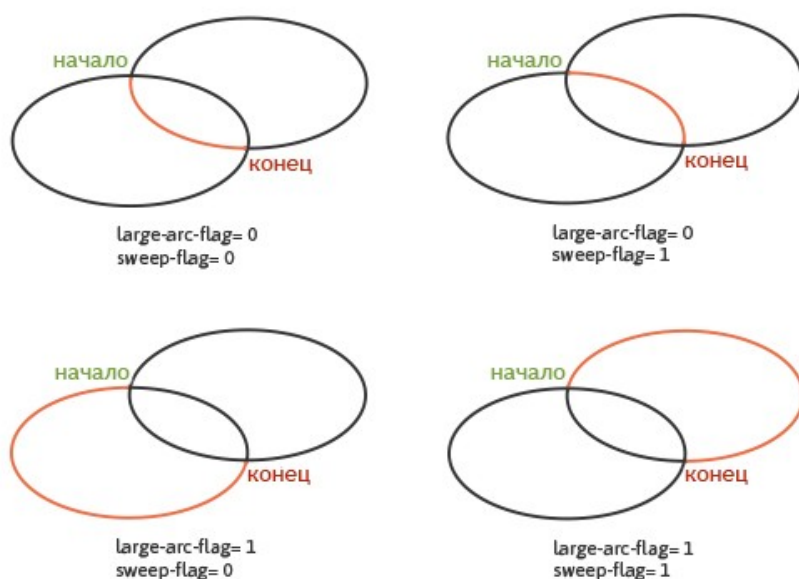
```
  <path fill="none" stroke="#333333" stroke-width="3" d="M65,10 a50,25 0 1,0 50,25" />
```

```
</svg>
```

Первые и последние наборы значений внутри этого контура,  $M65, 10$  и  $50, 25$  представляют начальные и конечные координаты, а вторые наборы значений определяют два радиуса. Значения  $1, 0$  (`large-arc-flag` и `sweep-flag`) определяют то, как будет отрисована дуга, поскольку для этого есть четыре различных возможных варианта.

Следующая диаграмма показывает четыре варианта выбора дуги и то, как влияют значения `large-arc-flag` и `sweep-flag` на конечное отображение отрезка дуги.

## Дуга эллипса



[Увидеть демо можно здесь.](#)

## Копирование из векторных редакторов

Программы для векторной графики позволяют генерировать более сложные фигуры и контуры, в то же время производя SVG-код, который может быть взят, использован и обработан где-то ещё.

После того, как графика готова, сгенерированный XML-код, который может быть достаточно длинным в зависимости от сложности, может быть скопирован и встроен в HTML. Разбивка каждого раздела SVG и наличие правильных организационных элементов может значительно помочь в навигации и понимании этих, казалось бы, сложных и многословных документов.

Здесь представлен SVG-код для изображения нескольких вишенек с добавлением классов для расширенной навигации:

```
<svg width="215px" height="274px" viewBox="0 0 215 274">
```

```
<g>
```



```

    <path class="stems" fill="none" stroke="#7AA20D" stroke-width="8" stroke-linecap="round"
stroke-linejoin="round" d="M54.817,169.848c0,0,77.943-73.477,82.528-104.043c4.585-
30.566,46.364,91.186,27.512,121.498" />

    <path class="leaf" fill="#7AA20D" stroke="#7AA20D" stroke-width="4" stroke-linecap="round"
stroke-linejoin="round" d="M134.747,62.926c-1.342-6.078,0.404-12.924,5.762-19.681c11.179-
14.098,23.582-17.539,40.795-17.846 c0.007,0,22.115-0.396,26.714-20.031c-2.859,12.205-
5.58,24.168-9.774,36.045c-6.817,19.301-22.399,48.527-47.631,38.028
C141.823,75.784,136.277,69.855,134.747,62.926z" />

</g>

<g>

    <path class="r-cherry" fill="#ED6E46" stroke="#ED6E46" stroke-width="12" d="M164.836,193.136
c1.754-0.12,3.609-0.485,5.649-1.148c8.512-2.768,21.185-
6.985,28.181,3.152c15.076,21.845,5.764,55.876-18.387,66.523 c-27.61,12.172-58.962-16.947-56.383-
45.005c1.266-13.779,8.163-35.95,26.136-27.478 C155.46,191.738,159.715,193.485,164.836,193.136z"
/>

    <path class="l-cherry" fill="#ED6E46" stroke="#ED6E46" stroke-width="12" d="M55.99,176.859
c1.736,0.273,3.626,0.328,5.763,0.135c8.914-0.809,22.207-2.108,26.778,9.329c9.851,24.647-
6.784,55.761-32.696,60.78 c-29.623,5.739-53.728-29.614-44.985-56.399c4.294-13.154,15.94-
33.241,31.584-20.99c47.158,173.415,50.919,176.062,55.99,176.859z" />

</g>

</svg>

```



Атрибуты в элементе `svg` определяют рабочую область или «холст» для графики. Листочек и черешки находятся в одном элементе `<g>` (группе), а вишенки в другом. Строка числовых значений определяет контур графики, а атрибуты `fill` и `stroke` устанавливают цвет для фона и границ.

Прежде чем поместить этот код в HTML, его можно скопировать и пропустить через SVG-оптимизатор, который в свою очередь поможет устранить лишний код, пробелы и значительно сократить размер файла. [SVG-оптимизатор Питера Коллингриджа](#) или [SVGO](#) — очень полезные в этом плане инструменты.

Публикуется под лицензией [Attribution-NonCommercial-ShareAlike 4.0](#)

## Глава 3. Рабочая область

Пожалуй, самым важным аспектом SVG, после понимания его главной структуры и умения создавать основные фигуры, является получение представления об используемой рабочей области, или другими словами, системы координат, в которой будет отображаться графика.

Понимание рабочей области SVG полезно в правильной отрисовке вашего графического объекта, но становится решающим, как только вы столкнётесь с более продвинутыми возможностями SVG. Например, отображение градиентов и паттернов в значительной степени зависит от установленной системы координат. Рабочая область определяется размерами области просмотра и атрибутами `viewBox`.

У этой груши, к счастью, область просмотра и `viewBox` совпадают:

```
<svg width="115" height="190" viewBox="0 0 115 190">

  <!--<path <контур изображения груши> />-->

</svg>
```



[Посмотреть демо можно здесь](#)

Вся груша видна в браузере и будет масштабироваться соответственно при изменении размеров области просмотра.

### Область просмотра

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`

`viewBox` дает возможность указать, что данный набор графических элементов растягивается, чтобы уместиться в определенном 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.

Если отнять по 50px от `width`— и `height`-составляющих `viewBox` изображения груши, видимая часть груши уменьшится, но затем эта оставшаяся видимая часть отмасштабируется, чтобы вписаться в границы области просмотра.

```
<svg width="115px" height="190px" viewBox="0 0 65 140">  
  
  <!--<path <контур изображения груши> />-->  
  
</svg>
```



[Посмотреть демо можно здесь](#)

Значения `min` во `viewBox` определяют начало `viewBox` в пределах родительского элемента. Другими словами, точку во `viewBox`, в которой вы хотите, чтобы начиналась область просмотра. Чуть выше в изображении груши значения `min` установлены в 0, 0 (верхний левый угол). Давайте изменим их на 50, 30: `viewBox="50 30 115 190"`.

```
<svg width="115" height="190" viewBox="50 30 115 190">  
  
  <!--<path <контур изображения груши> />-->  
  
</svg>
```



[Посмотреть демо можно здесь](#)

Теперь `viewBox` начинается с 50px по оси `x` и с 30px по оси `y`. В ходе редактирования этих значений та часть груши, которая находится в фокусе, изменилась.

## preserveAspectRatio

Если пропорции ширины и высоты области просмотра и `viewBox` не совпадают, то атрибут `preserveAspectRatio` указывает браузеру, как отображать рисунок.

`preserveAspectRatio` принимает два параметра, `<align>` и `<meetOrSlice>`. Первый состоит из двух частей и задаёт выравнивание `viewBox` в области просмотра. Второй является необязательным и указывает, как пропорции должны быть сохранены.

```
preserveAspectRatio="xMaxYMax meet"
```

Эти значения выравнивают нижний правый угол `viewBox` по нижнему правому углу области просмотра. `meet` сохраняет пропорции, масштабируя `viewBox`, чтобы уместиться в область просмотра настолько это возможно.

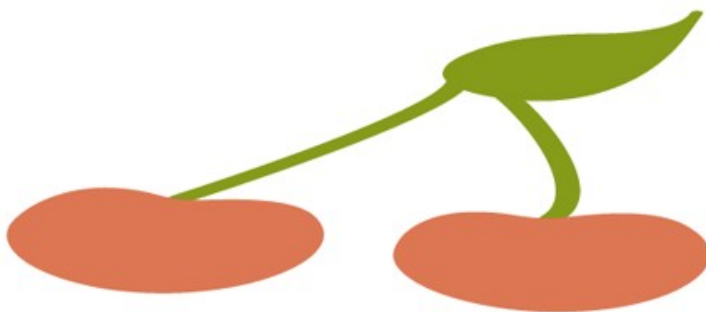
У атрибута `<meetOrSlice>` может быть одно из трех значений: `meet` (по умолчанию), `slice`, и `none`. В то время как `meet` гарантирует полную видимость графики (насколько это возможно), `slice` старается заполнить область просмотра с `viewBox` и затем обрезать все части изображения, которые не поместились в область просмотра после этого масштабирования. `none` не сохраняет пропорции и скорее всего исказит изображение.

Возможно самое незамысловатое значение здесь – это «none», которое показывает, что масштабирование не должно быть равномерным. Если мы увеличим пиксельные значения области просмотра на следующем изображении вишенки, то оно будет неравномерно растянуто и выглядеть искажённым.

```
<svg width="500" height="400" viewBox="0 0 250 600" preserveAspectRatio="none">

  <!--<path <контур изображения вишенки> />-->

</svg>
```



[Посмотреть демо можно здесь](#)

`preserveAspectRatio` для изображения ниже установлен в `xMinYMax meet` и выравнивает нижний левый угол `viewBox` по нижнему левому углу области просмотра (которая теперь обведена). `meet` гарантирует, что изображение отмасштабируется таким образом, чтобы вписаться в область просмотра насколько это возможно.

```
<svg width="350" height="150" viewBox="0 0 300 300" preserveAspectRatio="xMinYMax meet" style="border: 1px solid #333333;">
```

```
<!--<path <контур изображения вишенки> />-->
```

```
</svg>
```



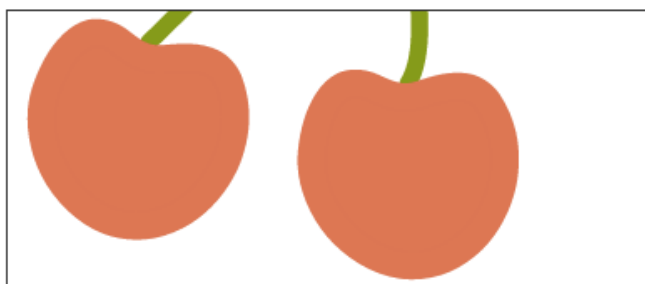
[Посмотреть демо можно здесь](#)

Здесь те же самые вишенки, но `meet` изменён на `slice`:

```
<svg width="350" height="150" viewBox="0 0 300 300" preserveAspectRatio="xMinYMax slice" style="border: 1px solid #333333;">
```

```
<!--<path <контур изображения вишенки> />-->
```

```
</svg>
```



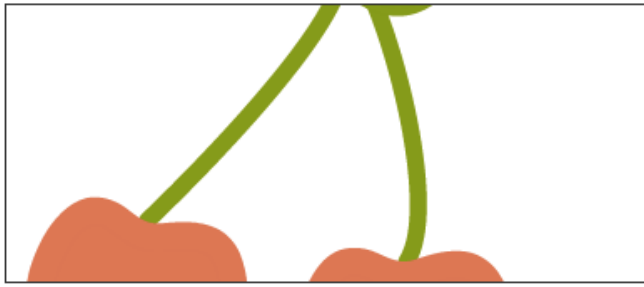
[Посмотреть демо можно здесь](#)

Заметьте, что значения выравнивания не зависят друг от друга.

```
<svg width="350" height="150" viewBox="0 0 300 300" preserveAspectRatio="xMinYMid slice" style="border: 1px solid #333333;">
```

```
<!--<path <контур изображения вишенки> />-->
```

```
</svg>
```



[Посмотреть демо можно здесь](#)

В примере выше значения `preserveAspectRatio` установлены в `xMinYMid slice`; теперь вишенки выравнены по середине оси `y` области просмотра.

## Преобразования системы координат

SVG включает дополнительные возможности изменения графики, такие как вращение, масштабирование, перемещение и наклон при помощи трансформации. SVG-автор может добавить трансформации к определённым элементам или к целой группе элементов.

Эти функции включены в управляемый элемент и находятся в атрибуте `<transform>`. Можно использовать и множественные трансформации, подключив несколько функций внутри этого атрибута. Например: `transform="translate(<tx>,<ty>) rotate(<угол поворота>)" />`.

При трансформации SVG важно учитывать то, что она влияет на вашу систему координат, т.е. на рабочую область. Это происходит потому, что [трансформация создаёт новую действующую систему координат SVG-элемента](#), фактически копируя оригинал, а затем накладывая трансформацию на новую систему координат.

Следующая картинка показывает трансформацию системы координат, происходящую при наложении сдвига на (100,100) на группу с графикой:



Сама система координат переместилась, а изображение лайма и лимона сохранило своё исходное местоположение в системе. Новая действующая система координат начинается в точке (100,100) в исходной системе координат.

Из-за этой связи с системой координат, многие из этих функций будут передвигать графику даже без явного задания сдвига для нее. Например, при попытке увеличить размер изображения в три раза, путём установки `scale` в значение «3», на «3» умножатся и координаты `x` и `y`, а изображение

отмасштабируется вместе с ними, сместившись по экрану при этом.

В случае вложенных трансформаций эффекты накапливаются, поэтому окончательная трансформация на дочернем элементе будет основываться на суммарном эффекте трансформаций его предков.

## translate

Функция `translate` задаёт подробности сдвига фигуры, а два числовых значения, используемые здесь, управляют сдвигом самой фигуры вдоль *x*— и *y*-осей:

`transform="translate(<tx>, <ty>)"`. Эти значения могут быть разделены пробелом или запятой.

Значение *y* является необязательным, и если оно опущено, то предполагается, что оно равно «0».

## rotate

Значение в `rotate` указывает поворот фигуры вокруг ее начальной точки (в градусах), которой в SVG является 0, 0 (левый верхний угол): `transform="rotate(<угол поворота>)"`.

Также здесь есть возможность добавить значения *x* и *y*: `transform=rotate(<угол поворота> [<cx>, <cy>]) </cy></cx>`. Если эти значения выставлены, то они определяют новые координаты центра поворота, отличные от тех, которые выставлены по умолчанию (т.е. 0, 0).

На рисунке ниже изображено два состояния яблока до и после применения к нему поворота в 20 градусов: `transform="rotate(20)"`. Заметьте, что это изображение не отражает изменение координат, которое происходит вследствие этой трансформации.



## scale

Масштабирование позволяет изменять размеры SVG-элемента при помощи функции `scale`. Эта функция принимает одно или два значения, которые указывают горизонтальный и вертикальный масштаб вдоль соответствующей оси: `transform="scale(<sx> [<sy>]) </sy>`.

Значение *sy* является необязательным, и если оно не указано, то предполагается, что оно равно *sx*, что гарантирует пропорциональное изменение размеров.

Значение масштаба ".5" отобразит графику в половину от ее исходного размера, тогда как значение "3" увеличит исходный размер втрое. Значение "4,2" отмасштабирует графику вчетверо по ширине и вдвое по высоте относительно исходных размеров.

## skew

SVG-элемент может быть наклонён или скошен набок при помощи функций `skewX` и `skewY`. Значение, включённое в эти функции представляет трансформацию наклона в градусах вдоль соответствующей оси.

На рисунке ниже изображено два состояния яблока до и после добавления ему `skewX` в значении «20»: `transform="skewX(20)"`. Заметьте, что это изображение не отражает изменение координат, которое происходит вследствие этой трансформации.



Публикуется под лицензией [Attribution-NonCommercial-ShareAlike 4.0](#)

## Глава 4. Заливки и обводки

`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`.

### Свойства заливки

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`.

При заливке фигуры или контура `fill` будет заливать незамкнутые контуры, как если бы последняя точка контура была соединена с первой, даже несмотря на то, что цвет `stroke` в этой части контура не будет отображаться.

### fill-rule

Свойство `fill-rule` указывает алгоритм, которым будет определяться, какие части холста внутри фигуры нужно заливать. Это не всегда очевидно при работе с более сложными пересекающимися и замкнутыми контурами.

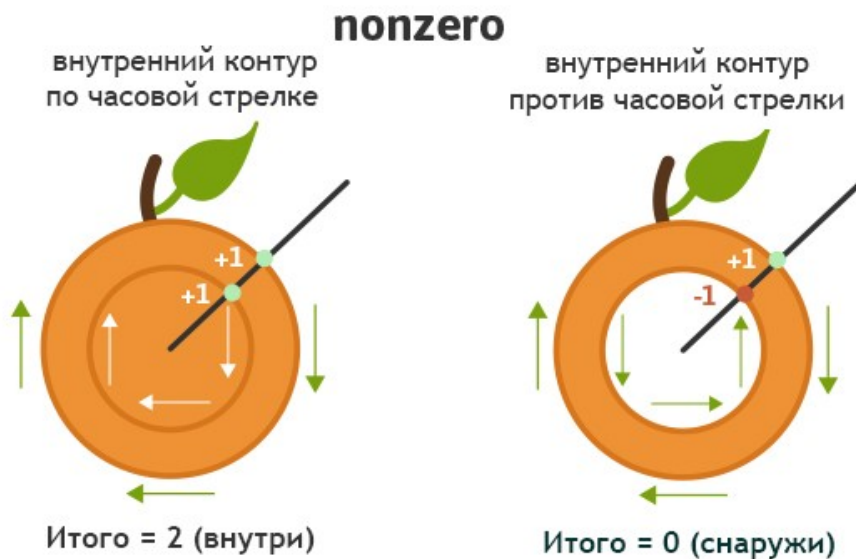
Допустимые значения здесь — `nonzero`, `evenodd`, `inherit`.



## nonzero

Значение `nonzero` определяет, является ли точка на холсте внутренней, проводя прямую линию из рассматриваемой области сквозь всю фигуру в любом направлении, а затем рассматривая места пересечений сегментов фигуры с этой линией. Начальным значением берется 0, затем прибавляется единица каждый раз, когда сегмент контура пересекает линию слева направо, и вычитается единица, когда он пересекает ее справа налево

Если после вычисления и подсчёта этих пересечений результат равен нулю, тогда точка оказывается с внешней стороны контура, иначе – внутри.



По сути, если внутренний контур нарисован по часовой стрелке, будет считаться, что он «внутри», если же контур нарисован против часовой стрелки, то будет считаться, что он «снаружи», и поэтому будет исключён из закрашивания.

## evenodd

Значение `evenodd` определяет внутреннюю область на холсте, проводя линию из этой области через всю фигуру в любом направлении, подсчитывая сегменты контура, пересекаемых линией. Если при этом получится нечетное число, то точка находится внутри, если четное — то снаружи.



Учитывая особенности алгоритма правила `evenodd`, направление рисования внутренней части рассматриваемой фигуры не имеет значения, в отличие от `nonzero`, поскольку мы просто подсчитываем контуры, когда они пересекают линию.

Хотя это свойство обычно не требуется, тем не менее оно позволит значительно лучше контролировать заливку (`fill`) более сложной графики, как уже упоминалось.

## inherit

Значение `inherit` заставит элемент взять значение свойства `fill-rule`, установленное у его родительского элемента.

## fill-opacity

Значение `fill-opacity` относится к уровню прозрачности внутренней части раскрашенной заливки. Значение «0» приводит к полной прозрачности, значение «1» задает полную непрозрачность, а значения между ними представляют степень непрозрачности.

## Атрибуты обводки

В SVG есть целый ряд атрибутов, связанных с обводкой, которые позволяют управлять и манипулировать её деталями. Возможности этих атрибутов обеспечивают большой контроль над SVG, который кодировался вручную, но также окажутся удобным при необходимости сделать правки в имеющейся внедрённой графике.

В следующих примерах используются встроенное SVG-изображение винограда. Используемые атрибуты располагаются прямо в соответствующем элементе фигуры.

### stroke

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

У следующего изображения винограда — сиреневая обводка: `stroke="#765373"`.



[Посмотреть демо можно здесь](#)

### stroke-width

Значение `stroke-width` устанавливает ширину обводки винограда, которая на изображении с ним равна 6px.

Значением по умолчанию для этого атрибута является 1. Если использовано процентное значение, то оно высчитывается из размеров области просмотра.

### stroke-linecap

`stroke-linecap` определяет, какую форму примет конец незамкнутого контура, и существует четыре значения: `butt`, `round`, `square`, `inherit`.



butt cap



round cap



square cap

Значение `inherit` прикажет элементу принять значение атрибута `stroke-linecap`, установленное у его родительского элемента.

У черешка на следующем изображении значение `stroke-linecap` равно `square`:

```
<svg>

  <!--<path <контур для винограда> />-->

  <!--<path stroke-linecap="square" <контур для черешка> />-->

  <!--<path <контур для листочка> />-->

</svg>
```



[Посмотреть демо можно здесь](#)

## stroke-linejoin

`stroke-linejoin` определяет, как будут выглядеть углы обводки контуров и базовых фигур.



miter join



round join



bevel join

Вот так выглядит изображение винограда, у которого свойство `stroke-linejoin` установлено в значении «`bevel`»:

```
<svg>

  <!--<path stroke-linejoin="bevel" <контур для винограда> />-->

  <!--<контур для черешка> />-->

  <!--<path <контур для листочка> />-->

</svg>
```



[Посмотреть демо можно здесь](#)

#### stroke-miterlimit

Когда две линии соприкасаются под острым углом и для них задано `stroke-linejoin="miter"`, атрибут `stroke-miterlimit` учитывает указание того, насколько продолжается это соединение/угол

Длина этого соединения названа длиной среза, и измеряется от внутреннего угла линии соединения до внешнего кончика соединения.

Это значение является ограничением на отношение длины среза к `stroke-width`.

1.0 – наименьшее возможное значение для этого атрибута.

Для первого изображения винограда задано `stroke-miterlimit="1.0"`, что создаёт эффект скоса. `stroke-miterlimit` для второго изображения установлено в 4.0.



[Посмотреть демо можно здесь](#)

#### stroke-dasharray

Атрибут `stroke-dasharray` меняет контуры на штриховые линии вместо сплошных.

В этом атрибуте вы можете указать длину штрихов и расстояние между ними, разделив значения запятыми или пробелами.

Если установить нечётное число значений, то их список удвоится, чтобы в результате получить чётное число. Например, 8, 6, 4 преобразуется в 8, 6, 4, 8, 6, 4, как показано ниже на втором изображении винограда.

Выставление в качестве значения всего одного числа приведёт к тому, что расстояние между штрихами будет равно длине штриха.



[Посмотреть демо можно здесь](#)

На первом изображении здесь показан эффект, который четное число значений в списке дает для контура винограда: `stroke-dasharray="20,15,10,8"`.

### **stroke-dashoffset**

`stroke-dashoffset` определяет смещение начала первого штриха в штриховом паттерне.

```
<svg>

  <!--<path stroke-dasharray="40,10" stroke-dashoffset="35" <контур для винограда> />-->

  <!--<path <контур для черешка> />-->

  <!--<path <контур для листочка> />-->

</svg>
```



[Посмотреть демо можно здесь](#)

В примере выше, длина штриха установлена в 40px, а `dashoffset` — в 35px. В начальной точке контура первый 40-пиксельный штрих будет виден только начиная с отметки 35px, вот почему первый штрих

выглядит значительно короче.

## stroke-opacity

Атрибут `stroke-opacity` позволяет установить уровень прозрачности для обводки.

Значение этого атрибута представляет десятичную дробь между 0 и 1, где 0 будет означать полную прозрачность.

Публикуется под лицензией [Attribution-NonCommercial-ShareAlike 4.0](#)

## Глава 5. Элемент `<text>`

Элемент `<text>` определяет графику, состоящую из текста. Для управления текстом есть специальные атрибуты, а также к нему можно применять градиенты, паттерны, контуры обрезки, маски или фильтры.

Написание и редактирование элемента `<text>` в SVG предоставляют очень мощные возможности для создания масштабируемого текста в качестве графики, которую можно легко изменять и редактировать в коде SVG.

Помните, что надо быть внимательными к размерам области просмотра при работе с примерами в этом разделе. Область просмотра, как уже упоминалось, определяет видимую часть SVG, и может понадобиться изменить область просмотра в зависимости от конкретных операций.

### Базовые атрибуты

Текстовые атрибуты SVG располагаются в элементе `<text>`, который в свою очередь находится в элементе `<svg>`. При помощи этих атрибутов мы можем управлять некоторым базовым оформлением для нашего текста, а также описывать тонкости его отображения на холсте, дающее возможность полного контроля для его размещения на экране.

### x, y, dx, dy

Первый символ в элементе `<text>` отображается в соответствии с установленными значениями `x` и `y`. Значение `x` определяет начало текста по оси `x`, а `y` — горизонтальное положение нижнего края текста.

`x` и `y` устанавливают координаты в абсолютных единицах, а `dx` и `dy` — относительные координаты. Это особенно удобно при использовании в связке с элементом `<tspan>`, который будет рассмотрен в следующем разделе.

```
<svg width="620" height="100">

  <text x="30" y="90" fill="#ED6E46" font-size="100" font-family="'Leckerli One',
  cursive">Watermelon</text>

</svg>
```

# Watermelon

[Посмотреть демо можно здесь](#)

На изображении выше текст начинается с 30px от левого края области просмотра SVG, а нижний край текста установлен в 90px от верхней границы этой области просмотра: `x="30" y="90"`.

## rotate

Поворот может быть установлен для определённой буквы/символа и/или для элемента в целом.

Если в атрибуте `rotate` установлено только одно значение, это приведёт к повороту каждого символа на это значение. Также можно использовать строку значений, которая выбирает и устанавливает разное значение поворота каждому символу. Если число значений меньше количества символов, в этом случае последнее значение установит поворот для остальных символов.

У текста на изображении ниже для всей графики задан поворот элементом `transform`, но каждому символу тоже установлено своё значение: `rotate="20,0,5,30,10,50,5,10,65,5"`.

```
<svg width="600" height="250">

  <text x="30" y="80" fill="#ED6E46" font-size="100" rotate="20,0,5,30,10,50,5,10,65,5"
  transform="rotate(8)" font-family="'Leckerli One', cursive">Watermelon</text>

</svg>
```



[Посмотреть демо можно здесь](#)

## textLength & lengthAdjust

Атрибут `textLength` указывает длину текста. Изменяя интервал между символами, длина текста регулируется таким образом, чтобы в итоге соответствовать длине, указанной в этом атрибуте.

`textLength` следующего примера установлен в 900px. Заметьте, как интервалы между символами

увеличились, чтобы заполнить это пространство.

```
<svg width="950" height="100">

  <text x="30" y="90" fill="#ED6E46" font-size="100" textLength="900" font-family="'Leckerli
One', cursive">Watermelon</text>

</svg>
```

Watermelon

[Посмотреть демо можно здесь](#)

При использовании атрибутов `textLength` и `lengthAdjust` вместе, межсимвольный интервал и размер символа будут отрегулированы таким образом, чтобы соответствовать этим новым значениям длины.

Значение «`spacing`» приведёт к виду, который напоминает пример выше, где интервалы между символами увеличены, чтобы заполнить пространство: «`lengthAdjust="spacing"`».

Значение «`spacingAndGlyphs`» приказывает интервалу и размеру символа отрегуливаться соответственно: `lengthAdjust="spacingAndGlyphs"`.

Watermelon

[Посмотреть демо можно здесь](#)

## Элемент `tspan`

Элемент `<tspan>` очень важен, потому что в настоящее время SVG не поддерживает автоматические разрывы строк или перенос слов. `<tspan>` позволяет рисовать множественные линии текста путём выделения определённых слов или символов, чтобы затем манипулировать ими независимо друг от друга.

Вместо определения новой системы координат для дополнительных линий, элемент `<tspan>` позиционирует эти новые линии текста относительно предыдущей.



Элемент `<tspan>` сам по себе не имеет визуального вывода, но указывая больше подробностей в этих элементах, мы можем выделить конкретный текст и лучше управлять его дизайном и позиционированием.

В примере ниже слова «are» и «delicious» находятся в отдельных элементах `<tspan>` в элементе `<text>`. Используя атрибут `dy` в каждом из этих `<tspan>`, мы позиционируем слово по оси `y` относительно слова перед ним.

«are» спозиционированно в -30px от «Watermelons», а «delicious» в 50px от «are».

```
<svg width="775" height="500">

  <text x="15" y="90" fill="#ED6E46" font-size="60" font-family="'Leckerli One', cursive">
Watermelons

    <tspan dy="-30" fill="#bbc42a" font-size="80">are</tspan>

    <tspan dy="50">delicious</tspan>

  </text>

</svg>
```



Watermelons are delicious

[Посмотреть демо можно здесь](#)

Вы также можете передвигать каждый символ отдельно при помощи списка значений, как показано в примере ниже. Буква/символ затем передвинутся в соответствии с позицией буквы/символа перед ними, и «delicious» теперь отпозиционирован в соответствии с «е» в слове «are».



Watermelons are delicious

[Посмотреть демо можно здесь](#)

`tspan`, содержащий «are», имеет следующий список значений атрибута `dy`: `dy="-30 30 30"`.

## Свойства интервалов

Существует ряд свойств, доступных при использовании элемента `<text>` во встроенном SVG, которые управляют интервалами слов и букв, подобно возможностям программ для встроенной векторной графики.

Понимание, как правильно использовать эти свойства, поможет гарантировать, что графика отобразится именно так, как задумывалось.

### kerning & letter-spacing

Кернинг относится к процессу регулирования интервалов между символами. Свойство `kerning` позволяет регулировать этот интервал, основанный на таблицах кернинга, которые включены в используемый шрифт или устанавливать уникальную длину.

Значение `auto` указывает, что межсимвольный интервал должен быть основан на таблице кернинга, которая включена в используемый шрифт.

В примере ниже свойство `kerning` установлено в значение `auto`, что в данном случае не имеет визуального эффекта, поскольку это значение по умолчанию.

```
<svg width="420" height="200">

  <text x="2" y="50%" fill="#ef9235" font-size="100" font-family="'Raleway', sans-serif" font-
weight="bold" kerning="auto">Oranges</text>

</svg>
```



Отрегулировать расстояние между символами можно путём простого включения числовых значений: `kerning="30"`.



Также допускается и значение `inherit`.

letter-spacing позволяет выбрать одно из трёх значений: normal, <length> или inherit. Числовые значения здесь имеют точно такой же эффект с интервалом, как и в случае с kerning. Свойство letter-spacing предназначено служить добавкой к интервалам, получившимся благодаря свойству kerning.

## word-spacing

Свойство word-spacing определяет интервал между словами.

```
<svg width="750" height="200">

  <text x="2" y="50%" fill="#ef9235" font-size="70" font-family="'Raleway', sans-serif" word-
spacing="30">Oranges are Orange</text>

</svg>
```

Oranges are Orange

Другие допустимые значения здесь — normal (по умолчанию) и inherit.

## text-decoration

Свойство text-decoration позволяет использовать underline, overline и line-through в тексте SVG.

В то время как порядок рисования не всегда оказывает эффект на визуальный вывод, тем не менее порядок имеет значение в случае с text-decoration. Все значения для декорирования текста, кроме line-through, должны быть применены до того, как текст будет закрашен и/или обведён; таким образом текст отобразится поверх декорирования.

line-through должно быть применено после того, как текст будет закрашен и/или обведён, тем самым декорирование отобразится поверх текста.

Вот так выглядят результаты применения text-decoration="underline" и text-decoration="line-through".



## Текст по контуру

Как уже упоминалось, встроенный SVG предоставляет на выбор продвинутые настройки, аналогичные возможностям программ для встроенной векторной графики. В самом SVG-коде мы можем позиционировать текст точно так, как мы хотим отобразить его на экране.

Продвигаясь в этой манипуляции еще дальше, `<text>` в SVG можно настроить так, чтобы он следовал за элементом `<path>`.

## Элемент `textPath`

Элемент `textPath` – вот где находится вся магия этой функции. Хотя обычно SVG-текст размещается в элементе `<text>`, в данном случае он будет располагаться в элементе `<textPath>`, который как раз уже находится в элементе `<text>`.

Затем `<textPath>` обратится по `id` к выбранному контуру, который болтается в элементе `<defs>` и готов к использованию.

Базовый синтаксис:

```
<svg>

  <defs>

    <path id="testPath" d="<...>"/>

  </defs>

  <text>

    <textPath xlink:href="#testPath">Place text here</textPath>

  </text>

</svg>
```

Вот так выглядит векторный контур, используемый в коде ниже.



После создания этого контура в программе для векторной графики сам SVG-код элемента `<path>` (который не будет включать в себя цвет, как показано выше) можно скопировать и вставить в элемент `<defs>` в `<svg>`, который также размещается в коде выше.

There are over 8,000 grape varieties worldwide.

```
<svg width="620" height="200">

  <defs>

    <path id="testPath" d="M3.858,58.607 c16.784-5.985,33.921-10.518,51.695-12.99c50.522-
7.028,101.982,0.51,151.892,8.283c17.83,2.777,35.632,5.711,53.437,8.628
c51.69,8.469,103.241,11.438,155.3,3.794c53.714-7.887,106.383-20.968,159.374-32.228c11.166-
2.373,27.644-7.155,39.231-4.449" />

  </defs>

  <text x="2" y="40%" fill="#765373" font-size="30" font-family="'Lato', sans-serif">

    <textPath xlink:href="#testPath">There are over 8,000 grape varieties worldwide.</textPath>

  </text>

</svg>
```

#### **xlink:href**

Атрибут `xlink:href` в `<textPath>` позволяет ссылаться на контур, по которому будет отображаться текст.

#### **startOffset**

Атрибут `startOffset` представляет величину смещения текста от начала контура. Значение «0%»

указывает начальную точку контура, а значение «100%» — конечную.

Значение `startOffset` в примере ниже установлено в «20%», что заставляет текст начаться с 20% вдоль контура. Размер шрифта был специально уменьшен, чтобы предотвратить частичное выпадение текста из области просмотра при перемещении.

Если добавить цвет к обводке контура при помощи элемента `<use>`, станет понятнее, что именно здесь происходит.

```
<svg width="620" height="200">

  <defs>

    <path id="testPath" d="M3.858,58.607 c16.784-5.985,33.921-10.518,51.695-12.99c50.522-
7.028,101.982,0.51,151.892,8.283c17.83,2.777,35.632,5.711,53.437,8.628
c51.69,8.469,103.241,11.438,155.3,3.794c53.714-7.887,106.383-20.968,159.374-32.228c11.166-
2.373,27.644-7.155,39.231-4.449" />

  </defs>

  <use xlink:href="#testPath" fill="none" stroke="#7aa20d" stroke-width="2"/>

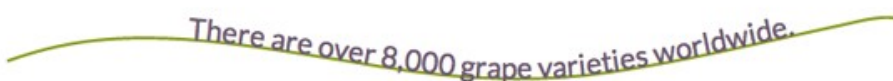
  <text x="2" y="40%" fill="#765373" font-size="20" font-family="'Lato', sans-serif">

    <textPath xlink:href="#testPath" startOffset="20%">There are over 8,000 grape varieties
worldwide.

  </textPath>

</text>

</svg>
```



[Посмотреть демо можно здесь](#)

Публикуется под лицензией [Attribution-NonCommercial-ShareAlike 4.0](#)

## Глава 6. Продвинутые функции: градиенты, паттерны, контуры обреза

### Градиенты

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

Очень простой линейный градиент имеет вот такую структуру:

```
<svg>

  <defs>

    <linearGradient id="gradientName">

      <stop offset="<%>" stop-color="<color>" />

      <stop offset="<%>" stop-color="<color>" />

    </linearGradient>

  </defs>

</svg>
```

`<svg>` содержит элемент `<defs>`, позволяющий создавать повторно используемые определения, к которым можно обратиться позже. Эти определения не отображаются (визуально) до тех пор, пока они не будут вызваны по их уникальному `id` в атрибутах обводки и/или заливке для SVG-фигур или `<text>`. Эти фигуры и/или текст также находятся в элементе `<svg>`, но снаружи `<defs>`.

Как только градиент создан и ему задано ID, к нему можно обратиться через атрибуты `fill` и/или `stroke` в SVG. Например: `fill= "url(#gradientName)"`.

### Линейные градиенты

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.

### Элементы stop

Элементам `<stop>` также можно задавать прозрачность при помощи `stop-opacity="<значение>"`

Ниже представлен код для простого линейного градиента с двумя ключевыми точками, которые применены к прямоугольнику:

```
<svg width="405" height="105">

  <defs>

    <linearGradient id="Gradient1" x1="0" y1="0" x2="100%" y2="0">

      <stop offset="0%" stop-color="#BBC42A" />

      <stop offset="100%" stop-color="#ED6E46" />

    </linearGradient>

  </defs>

  <rect x="2" y="2" width="400" height="100" fill= "url(#Gradient1)" stroke="#333333" stroke-
width="4px" />

</svg>
```



[Посмотреть демо можно здесь](#)

`offset` передаёт градиенту, в какой точке установить соответствующий `stop-color`.

### **x1, y1, x2, y2**

Значения атрибутов `x1`, `y1`, `x2` и `y2` определяют начальные и конечные точки, к которым привязаны ключевые точки градиента. Эти процентные значения размещают градиенты как надо вдоль нужных осей.

Если атрибуту `x2` задать значение «100%», а атрибуту `y2` — «0», то мы получим горизонтальный градиент (при `x1` и `y1`, равных нулю — прим. перев.), а если наоборот — вертикальный. Установив оба значения в «100%» (или в любое значение отличное от 0), мы получим наклонный градиент.

### **gradientUnits**

Атрибут `gradientUnits` определяет систему координат для значений `x1`, `x2`, `y1` и `y2`. Для него возможны два значения: «`userSpaceOnUse`» или «`objectBoundingBox`». `userSpaceOnUse` устанавливает систему координат градиента в абсолютных единицах, а `objectBoundingBox` (значение по умолчанию) создаёт эту систему в границах самой фигуры SVG, целевого объекта.



## spreadMethod

Значение атрибута `spreadMethod` указывает, как распространится градиент по фигуре, если он начинается или заканчивается внутри границ объекта. Если градиенту указано не закрашивать фигуру целиком, то `spreadMethod` определяет, как должен распространиться градиент, покрывая эту пустую область. Для этого атрибута возможны три значения: «pad», «repeat» или «reflect».

Значение `pad` (по умолчанию) указывает первому и последнему цвету градиента распространиться по всей оставшейся непокрытой областью объекта. `repeat` указывает градиенту повторять паттерн, постоянно начиная заново. Значение `reflect` отражает паттерн градиента, повторяясь поочередно то от начала к концу, то от конца к началу.

Начальной и конечной точкой для градиента ниже является: `x1=«20%» y1=«30%» x2=«40%» y2=«80%»`.



pad

repeat

reflect

## gradientTransform

Атрибут `gradientTransform` необязателен и предусматривает дальнейшую трансформацию градиента, прежде чем тот отобразится, например, добавление поворота.

## xlink:href

Атрибут `xlink:href` позволяет обращаться к ID другого градиента, чтобы унаследовать его значения, но также можно включить различные значения.

```
<linearGradient id="repeat" xlink:href="#Gradient-1" spreadMethod="repeat" />
```

Этот градиент унаследовал значения первого градиента, который находится в начале этого раздела, но для него задано другое значение «`spreadMethod`».

## Радиальные градиенты

Большинство атрибутов для `<radialGradient>` совпадают с атрибутами `<linearGradient>` кроме того, что для работы с ними существует другой набор координат.

### cx, cy, r

Атрибуты `cx`, `cy`, и `r` определяют внешний край круга, а значение `100% stop-color` градиента будет привязано к границе этого круга. `cx` и `cy` определяют координаты центра, а `r` устанавливает радиус градиента.

**fx, fy**

Атрибуты  $fx$ ,  $fy$  представляют координаты фокальной точки градиента или внутреннего круга. Фактически, центр градиента не обязан также быть его фокальной точкой, которая может быть изменена при помощи этих значений.

Несмотря на то, что по умолчанию фокальная точка радиального градиента будет в центре, атрибуты для фокальной точки могут изменить это поведение. Значениями центральной точки на изображении ниже являются  $fx="95\%"$   $fy="70\%"$ .

```
<svg width="850px" height="300px">

  <defs>

    <radialGradient id="Gradient2" cy="60%" fx="95%" fy="70%" r="2">

      <stop offset="0%" stop-color="#ED6E46" />

      <stop offset="10%" stop-color="#b4c63b" />

      <stop offset="20%" stop-color="#ef5b2b" />

      <stop offset="30%" stop-color="#503969" />

      <stop offset="40%" stop-color="#ab6294" />

      <stop offset="50%" stop-color="#1cb98f" />

      <stop offset="60%" stop-color="#48afc1" />

      <stop offset="70%" stop-color="#b4c63b" />

      <stop offset="80%" stop-color="#ef5b2b" />

      <stop offset="90%" stop-color="#503969" />

      <stop offset="100%" stop-color="#ab6294" />

    </radialGradient>

  </defs>

  <text x="20%" y="75%" fill= "url(#Gradient2)" font-family= "'Signika', sans-serif" font-
size="200">Cherry</text>

</svg>
```



[Посмотреть демо можно здесь](#)

В этом примере фокальная точка смещена к правому нижнему краю изображения.

## Паттерны

Как правило, паттерны считаются одним из самых сложных вариантов закрашивания, доступных для цвета заливок и обводок SVG. Понимание основы и базового синтаксиса могут сделать эти, казалось бы, более сложные паттерны гораздо более доступными.

Ниже приведён синтаксис для базового паттерна, который применён к прямоугольнику:

```
<svg width="220" height="220">

  <defs>

    <pattern id="basicPattern" x="10" y="10" width="40" height="40"
    patternUnits="userSpaceOnUse">

      <circle cx="20" cy="20" r="20" fill= "#BBC42A" />

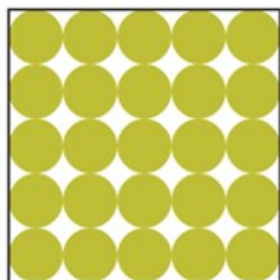
    </pattern>

  </defs>

  <rect x="10" y="10" width="200" height="200"

    stroke="#333333" stroke-width="2px" fill="url(#basicPattern)" />

</svg>
```



[Посмотреть демо можно здесь](#)

## Базовые атрибуты

Атрибуты и значения для паттернов определяют «холст», дизайн и позиционирование в целом. Паттерны состоят из контуров и/или фигур, также могут закрашивать текст и даже быть вложенными в другие паттерны.

### x, y, width, height

Атрибуты x и y в элементе `<pattern>` определяют в какой точке начнётся паттерн внутри фигуры. Ширина и высота, используемые в элементе `<pattern>`, определяют фактическую ширину и высоту, отведённые для области паттерна.

Упомянутый выше «basicPattern» содержит следующие значения: `x="10" y="10" width="40" height="40"`. В этом случае паттерн начнётся с 10px от начала оси x, 10px от начала оси y и создаст «холст» 40px ширины и такой же высоты.

### patternUnits

Атрибут `patternUnits` определяет в каких координатах указываются значения x, y, ширина и высота. Здесь можно выбрать два значения: `userSpaceOnUse` и `objectBoundingBox` (по умолчанию).

При `userSpaceOnUse` система координат паттерна определяется системой координат элемента, ссылающегося на `<pattern>`, а `objectBoundingBox` берет в качестве системы координат для паттернов прямоугольник, описанный вокруг элемента, к которому паттерн применяется.

### patternContentUnits

Значения атрибута `patternContentUnits` совпадают со значениями для `patternUnits`, за исключением того, что теперь система координат определяется для содержимого самого паттерна.

Для атрибута `patternContentUnits`, в отличие от `patternUnits`, значением по умолчанию является `userSpaceOnUse`, которое означает, что если не указаны один или оба из этих атрибутов, фигуры, отрисованные в `<pattern>`, рисуются не в той системе координат, которую использует `<pattern>`.

Определение `patternUnits="userSpaceOnUse"` в элементе `<pattern>` упрощает этот процесс и обеспечивает единообразную рабочую область.

## Вложенные паттерны

Паттерны могут быть вложенными, чтобы создать еще более неповторимый и подробный дизайн.

Вот так выглядит структура базового вложенного паттерна:

```
<svg width="204" height="204">

  <defs>

    <pattern id="circlePattern"

      x="4" y="4" width="75" height="75"
```

```

    patternUnits="userSpaceOnUse">

    <circle cx="12" cy="12" r="8"

    stroke="#ed6e46" stroke-width="3" fill="#765373" />

</pattern>

<pattern id="rectPattern"

    x="10" y="10" width="50" height="50"

    patternUnits="userSpaceOnUse">

    <rect x="2" y="2" width="30" height="30"

    stroke="#bbc42a" stroke-width="3" fill="url(#circlePattern)" />

</pattern>

</defs>

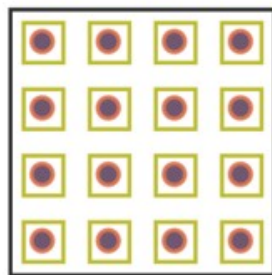
<rect x="2" y="2" width="200" height="200"

    stroke="#333333" stroke-width="3" fill="url(#rectPattern)" />

</svg>

```

Элемент `<defs>` содержит оба паттерна. В `<defs>` паттерн для прямоугольника обращается к паттерну круга при помощи `fill`, а главный прямоугольник затем также обращается к паттерну прямоугольника через `fill`, закрашивая внутреннюю часть главной фигуры вложенным паттерном.



[Посмотреть демо можно здесь](#)

## Контур обрезки

Контур обрезки ограничивает область, к которой будет применено закрашивание в SVG. Любая отрисованная область за пределами границ, установленными контуром обрезки, не будет отображаться.

Для демонстрации возможностей этой функции, давайте используем контур обрезки в виде текста «Apples», наложенный на прямоугольник цвета томата и зелёный круг.

К следующим фигурам не применён контур обрезки, но они настроены так, чтобы растягиваться за пределы области просмотра.



А теперь давайте взглянем на код, позволяющий применить текст «Apples» к этому «холсту».

```
<svg width="400px" height="200px">

  <clipPath id="clip-text">

    <text x="0" y="50%" fill="#f27678" font-size="120px" font-family=" 'Signika', sans-serif">Apples</text>

  </clipPath>

  <rect x="0" y="0" width="200" height="200" fill="#ed6e46" clip-path="url(#clip-text)" />

  <circle cx="310" cy="100" r="135" fill="#bbc42a" clip-path="url(#clip-text)" />

</svg>
```

The image displays the word "Apples" in a large, bold, sans-serif font. The letter 'A' is orange, and the two 'p's are also orange. The remaining letters 'l', 'e', and 's' are green. The text is centered horizontally and vertically within the frame.

[Посмотреть демо можно здесь](#)

Контур обрезки определяется в элементе `<clipPath>`, а затем вызывается обеими фигурами путём обращения к их уникальному `id`.

## Заключение

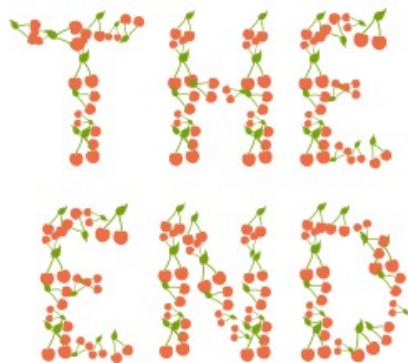
Написание встроенного SVG дает очень полезные возможности редактирования и позволяет нам, как авторам, иметь полный доступ ко всем графическим элементам в отдельности. В этом коде мы генерируем графику, которая масштабируется без потери качества изображения, видна поисковикам и

повышает доступность.

Скорее всего, вам придётся повозиться некоторое время, чтобы освоиться с написанием SVG, но как только вы этого достигнете, я бы порекомендовала поработать над тем, чтобы сделать ваш код настолько коротким и эффективным, насколько это возможно, исследовать [SMIL-анимацию](#) и поэкспериментировать со [стилизованными SVG-элементами при помощи CSS](#).

Надеюсь, это руководство стало как ценным справочником, так и вдохновением в плане понимания мощного потенциала построения и манипулирования встроенным SVG.

За новостями и обновлениями, пожалуйста, заходите на [сайт книги](#). Если у вас есть какие-то вопросы или комментарии насчет книги, вы можете связаться со мной [в твиттере](#) или по почте [info@jonibologna.com](mailto:info@jonibologna.com).



Публикуется под лицензией [Attribution-NonCommercial-ShareAlike 4.0](#)