ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
им. проф. М. А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

Ф. В. Филиппов

ОБРАБОТКА ГРАФИЧЕСКОЙ ИНФОРМАЦИИ В ФОРМАТЕ SVG

УЧЕБНОЕ ПОСОБИЕ

Часть 1

САНКТ-ПЕТЕРБУРГ
2017

УДК 004.92(075.8)
ББК 32.973-018.2я73
Ф 53

Рецензенты:
кандидат технических наук, доцент кафедры робототехники
и автоматизации производственных систем СПбГЭТУ «ЛЭТИ»
А. В. Шевченко,
кандидат технических наук, доцент кафедры конструирования
и производства радиоэлектронных средств СПбГУТ
Т. В. Матюхина

Утверждено редакционно-издательским советом СПбГУТ
в качестве учебного пособия

Филиппов, Ф. В.
Ф 53 Обработка графической информации в формате SVG : учебное пособие : часть 1 / Ф. В. Филиппов ; СПбГУТ. – СПб., 2017. – 84 с.

Рассматриваются практические аспекты использования формата SVG при разработке структурированных графических компонентов информационных управляющих систем на базе web-технологий.

Предназначено для студентов, обучающихся по направлению 09.03.02 «Информационные системы и технологии», и будет полезно при изучении дисциплин «Технология программирования», «Технологии обработки информации» и «Технологии проектирования программного обеспечения информационных систем».

УДК 004.92(075.8)
ББК 32.973-018.2я73

Содержание

ВВЕДЕНИЕ

Технология масштабируемой векторной графики (*Scalable Vector Graphics – SVG*) позволяет объединить в одном формате текст, графику, анимацию и интерактивные компоненты и базируется на трех типах графических изображений: векторных формах, рисунках и тексте. Объекты, как это принято в векторной графике, представлены либо прямолинейными и криволинейными контурами, либо графическими примитивами (прямоугольниками, эллипсами и др.), а рисунки представляют собой импортированные растровые изображения. Помимо этого формат *SVG* поддерживает различные виды анимационных (напоминающих *GIF* и *flash*-анимацию) и интерактивных объектов, таких как гиперссылки, реакции на внешние события и прочие элементы навигации. Важно и то, что поскольку данный стандарт основан на языке *XML*, то *SVG*-файл наряду с элементами, предназначенными для визуального отображения, может содержать также различные метаданные.

Указанные особенности выдвигают *SVG* на лидирующие места в области проектирования и разработки веб-ресурсов. Цель настоящего учебного пособия состоит в знакомстве студентов с основными характеристиками и приемами практического использования формата *SVG* для построения информационных управляющих систем различного назначения.

1. ГРАФИЧЕСКИЕ ОБЪЕКТЫ SVG

Масштабируемая векторная графика является приложением *XML*, что позволяет представлять графическую информацию в компактной текстовой форме [1–6]. Интерес к *SVG* быстро растет, и уже доступно большое число инструментов для создания *SVG*-файлов. Главное достоинство этих файлов заключается в возможности размещения в них семантических данных, доступных для поисковых систем.

Формат *SVG* является частью семейства стандартов векторной графики, которая отличается от растровой более эффективным способом хранения информации о цвете пикселов в изображении. Наиболее распространенными растровыми форматами, используемыми в сети Интернет в настоящее время, являются *JPEG, GIF* и *PNG*, каждый из которых имеет свои достоинства и недостатки.

Формат *SVG* обладает рядом достоинств перед любым растровым форматом. Преимущества *SVG*-формата следующие:

- файлы *SVG*, как правило, гораздо меньше, чем растровые изображения;
- *SVG*-графику можно масштабировать без потери качества изображения;
- конечный пользователь может взаимодействовать с *SVG*-графикой сервера без необходимости использования сложных и дорогостоящих клиент-серверных коммуникаций;
- обеспечивается встроенная поддержка анимации и интерактивности;
- формат *SVG* легко встраивается в HTML-документы, и наоборот, внутри SVG-файла легко разместить код HTML;
- формат *SVG* прямо взаимодействует со скриптами *JavaScript*, имеет один и тот же язык сценариев, используемый в *HTML*-среде.

Исходный файл изображения в формате SVG представлен в текстовом виде, поэтому он является доступным и дружественным как для ручной, так и для автоматизированной обработки.

1.1. Координаты объектов

Изображения в формате *SVG* могут находиться в произвольной точке координатной системы с бесконечной осью *X*, расположенной слева направо, и осью *Y*, расположенной сверху вниз. Однако на экране монитора мы можем видеть только то, что расположено в пределах системной области (или окна) просмотра *viewport*.

Размер окна просмотра *viewport* определяется с помощью атрибута ширины *width* и высоты *height* в теге <*svg*>. Значения атрибутов указываются как число, за которым следует один из идентификаторов единицы измерения: *in* – дюйм, *mm* – миллиметр, *sm* – сантиметр, *pc* – 1/6 дюйма, *pt* –

1/72 дюйма, *px* – пиксел, *ex* – высота буквы *x*, *em* – размер шрифта, % – процент от размера экрана дисплея. По умолчанию, без указания единицы измерения, значения этих атрибутов задают размер в пикселах:

```
<svg width="800" height="600">
   <!-- SVG контент ->
</svg>
```

После того как ширина и высота самого дальнего от центра координат *SVG*-элемента установлены, браузер устанавливает начальное окно просмотра с началом в верхнем левом углу в точке (0, 0).

Если в теге <*svg*> значения атрибутов *width* и *height* не заданы, то по умолчанию пределами системной области или окна просмотра является весь экран монитора. В подавляющем большинстве примеров мы не будем задавать эти значения.

Наряду с системной можно использовать пользовательскую область (или видовое окно) просмотра *viewBox*, размеры и положение которой относительно начала системных координат можно задавать произвольно (рис. 1.1).

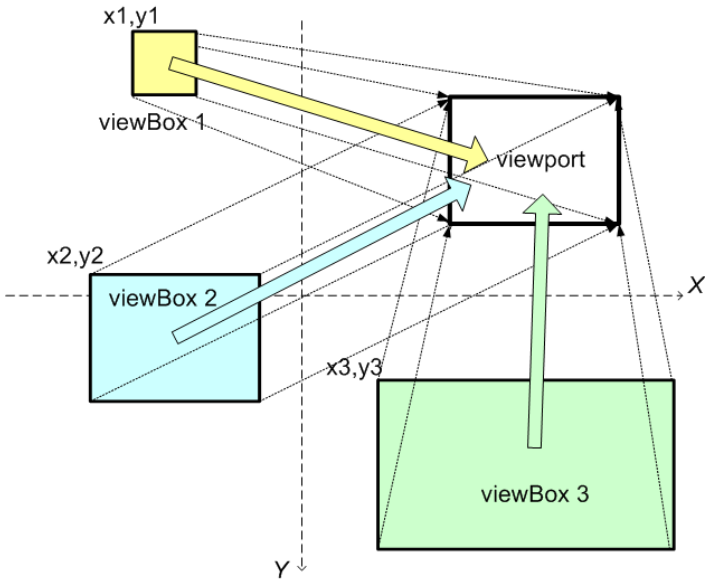


Рис. 1.1. Передача изображения
из видового окна в окно просмотра

Видовое окно *viewBox* можно переместить в любое место документа *SVG*, при этом изображение из видового окна будет отображаться в окне просмотра *viewport*, занимая всю его площадь. Очевидно, что изображение из *viewBox 1* будет увеличено в окне просмотра, а из *viewBox 3* – наоборот, уменьшено.

Использование видового окна *viewBox* позволяет получать очень интересные эффекты, такие как плавное масштабирование, вертикальный и горизонтальный параллакс и другие, которые будут рассмотрены в разд. 3.4.

1.2. Базовые объекты

Все графические фигуры определяются некоторой комбинацией прямых и кривых линий. При работе с графикой в формате SVG в качестве базовых объектов для получения необходимых графических изображений консорциум *World Wide Web* предлагает использовать следующий набор тегов:

- *line* – для изображения отрезков прямой линии;
- *rect* – для изображения прямоугольников;
- *circle* – для изображения кругов и окружностей;
- *ellipse* – для изображения эллипсов;
- *polyline* – для изображения ломаных прямых линий;
- *polygon* – для изображения многоугольников;
- *path* – для изображения произвольных траекторий и фигур;
- *text* – для вставки текста;
- *image* – для вставки изображений.

Использование каждого тега предполагает указание координат (*x*, *y*) места расположения соответствующего объекта и некоторых его свойств. Как принято в *XML* нотациях, для указания свойств объекта используются атрибуты. Существует огромное количество атрибутов. Некоторые атрибуты являются практически универсальными и используются для многих объектов, тогда как другие являются весьма специфическими и редко используются. Примером универсального атрибута может служить *stroke* (штрих, мазок), определяющий цвет линий. Значение атрибута цвета может задаваться по-разному:

- названием цвета, как и в *HTML* – *black, white* и т. д.;
- 6-разрядным 16-ричным *RGB* значением – #ff0b7b;
- 3-разрядным 16-ричным *RGB* значением – #fa5 = #ffaa55;
- тремя десятичными числами (в диапазоне 0–255) – *rgb*(12,540,255);
- процентным отношением – *rgb*(90%,50%,20%).

Значение любого атрибута должно быть обязательно задано в кавычках. Например, зеленый цвет может быть задан как *stroke="green", "*#0f0*", "*#00ff00*", "rgb*(0,255,0)*"* или *"rgb*(0%,100%,0%)*"*.

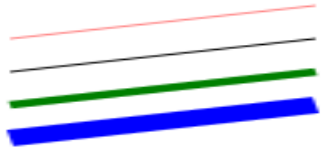
1.2.1. *Линия <line>*

Прямая линия представляет собой простейший графический элемент. Тег <*line*> «рисует» отрезок прямой линии между двумя заданными точками: (*x1, y1*) *и* (*x2, y2*). Если записать тег <*line*> в виде <*line x1="5" y1="10" x2="100" y2="30"/*>, то этот код в большинстве браузеров изобразит невидимую линию. Для того чтобы линия была видимой, необходимо добавить атрибут *stroke*, который определит цвет линии. Таким образом,

своего рода «минимальный» код линии выглядит как: *<line x1="5" y1="10" x2="100" y2="30" stroke="black"/>*. Другим важным атрибутом является *stroke-width*, который определяет толщину линии, и по умолчанию его значение равно 1. Влияние значений атрибутов *stroke* и *stroke-width* отражено в табл. 1.1.

Таблица 1.1

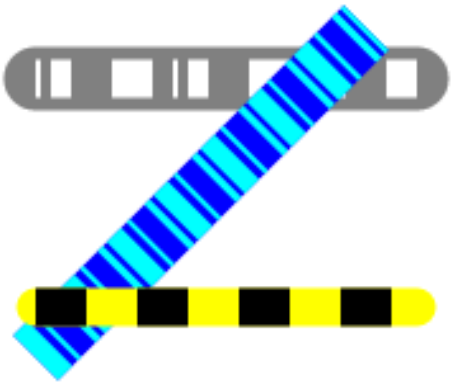
Влияние атрибутов *stroke* и *stroke-width*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <line x1="5" y1="30" x2="100" y2="10" stroke-width=".5" stroke="red"/> <line x1="5" y1="50" x2="100" y2="30" stroke-width="1" stroke="black"/> <line x1="5" y1="70" x2="100" y2="50" stroke-width="4.5" stroke="green"/> <line x1="5" y1="90" x2="100" y2="70" stroke-width="10" stroke="blue"/> </svg>	

Для линий существует еще целый ряд атрибутов, например два из них *stroke-dasharray* и *stroke-linecap*, которые часто используются. Атрибут *stroke-dasharray* предоставляет удобный способ оформления различных штрих-пунктирных линий, а атрибут *stroke-linecap* позволяет определить форму концов отрезка. Прежде чем приводить пример по использованию этих атрибутов, следует подчеркнуть, что изображения, инициированные тегами, появляются в порядке следования тегов. Поэтому, когда координаты разных изображений совпадают, один объект накладывается на другой. В табл. 1.2 приведен пример трех пар линий. Каждая пара состоит из нижней «фоновой» линии и верхней штрих-пунктирной линии.

Таблица 1.2

Влияние атрибутов *stroke-linecap* и *stroke-dasharray*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <line x1="30" y1="50" x2="180" y2="50" stroke-width="25" stroke="grey" stroke-linecap="round"/> <line x1="30" y1="50" x2="180" y2="50" stroke-width="15" stroke="white" stroke-dasharray="2,4,8,16,16,8"/> <line x1="30" y1="160" x2="160" y2="30" stroke-width="25" stroke="blue"/> <line x1="30" y1="160" x2="160" y2="30" stroke-width="25" stroke="aqua" stroke-dasharray="8,3,2"/> <line x1="30" y1="140" x2="180" y2="140" stroke-width="15" stroke="yellow" stroke-linecap="round"/> <line x1="30" y1="140" x2="180" y2="140" stroke-width="15" stroke="black" stroke-dasharray="20,20"/> </svg>	

Первая пара: серая линия (*stroke="gray"*) с закругленными концами (*stroke-linecap="round"*), поверх которой нарисована белая линия

(*stroke="white"*) в форме штрихов (*stroke-dasharray="2,4,8,16,16,8"*). Массив значений атрибута *stroke-dasharray* определяет последовательность длины штрихов и пробелов между ними. Так, в нашем случае: штрих 2 пиксела, пробел 4 пиксела, штрих 8 пикселов, пробел 16 пикселов, пробел 16 пикселов, штрих 8 пикселов. Далее указанное чередование повторяется вдоль всей линии. Вторая пара: синяя «фоновая» линия и сверху бирюзовая штрих-пунктирная. Поскольку атрибут *stroke-linecap* не задан, по умолчанию концы линии прямые. Третья пара: желто-черная комбинация с равными длинами штрихов и пробелов.

Для атрибута *stroke-linecap* можно указать три значения *"square"*, *"round"* и *"butt"*, которые определяют концы (*cap*) линий, представленные на рис. 1.2.

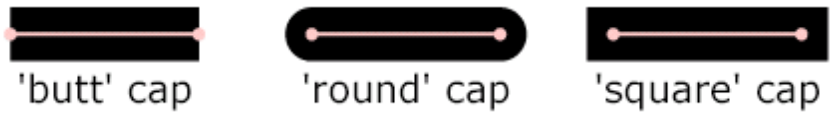


Рис. 1.2. Формы концов линий
для различных значений атрибута *stroke-linecap*

Наконец, полезным бывает использование атрибута *stroke-opacity*, который позволяет легко изменять прозрачность линий. В табл. 1.3 приведен пример использования этого атрибута.

Таблица 1.3

Влияние атрибута прозрачности *stroke-opacity*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <line x1="80" y1="80" x2="80" y2="80" stroke-width="100" stroke="blue" stroke-linecap="round"/> <line x1="10" y1="40" x2="150" y2="40" stroke-width="15" stroke="yellow" stroke-opacity="0.4"/> <line x1="10" y1="70" x2="150" y2="70" stroke-width="15" stroke="yellow" stroke-opacity="0.6"/> <line x1="10" y1="100" x2="150" y2="100" stroke-width="15" stroke="yellow" stroke-opacity="0.8"/> <line x1="10" y1="130" x2="150" y2="130" stroke-width="15" stroke="yellow" stroke-opacity="1.0"/> </svg>	

Отметим, что синий круг на рисунке (табл. 1.3) на самом деле сформирован с помощью тега *<line x1="80" y1="80" x2="80" y2="80" stroke-width="100" stroke="blue" stroke-linecap="round"/>*, который определяет отрезок прямой нулевой длины, поскольку (*x1*,*y1*) = (*x2*,*y2*). Левый и правый полукруг образованы атрибутом *stroke-linecap*, для которого указано значение *"round"*.

Еще один полезный аспект для линий добавляет тег <marker> используемый для формирования стрелок или иных фигур, которые затем могут быть «прикреплены» в начале, середине или на конце линии с помощью атрибутов marker-start, marker-mid и marker-end соответственно. Хороший пример использования маркеров для линий можно найти в [10].

1.2.2. *Прямоугольник <rect>*

Тег <*rect*> отвечает за вывод самой простой из основных фигур – прямоугольника. Его основными атрибутами являются координаты левого верхнего угла *x,y*, ширина *width* и высота *height*. Простейший код для изображения прямоугольника выглядит следующим образом: *<rect x="62" y="25" height="110" width="16"/>* – он представлен первым на рисунке в табл. 1.4 в виде черной вертикальной полосы. Цвет заполнения прямоугольника задается атрибутом *fill*, который по умолчанию имеет значение «*black*» – черный.

Если заполнение цветом не требуется, то нужно атрибуту *fill* задать значение *fill="none"*, как это сделано для прямоугольника с зеленой рамкой. Атрибуты *stroke* и *stroke-width* относятся к рамке прямоугольника.

Для закругления углов прямоугольника используются два атрибута: *rx* – радиус закругления по оси абсцисс и *ry* – радиус закругления по оси ординат. Использование *rx*, равного половине стороны равностороннего прямоугольника, показано в табл. 1.4 для белого прямоугольника с красной рамкой, что привело к его изображению в виде круга.

Таблица 1.4

Влияние атрибутов на вид тега <*rect*> прямоугольников

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <rect x="62" y="25" height="110" width="16"/> <rect x="35" y="35" height="30" width="50" fill="yellow" stroke="black" stroke-width="2" rx="15" /> <rect x="5" y="60" height="30" width="50" fill="red" stroke="black" stroke-width="2"/> <rect x="25" y="70" height="30" width="50" fill="cyan" stroke="black" stroke-width="2" rx="20" ry="5" /> <rect x="65" y="60" height="30" width="50" fill="none" stroke="green" stroke-width="2"/> <rect x="65" y="80" height="30" width="30" fill="white" stroke="red" stroke-width="5" rx="15" /> </svg>	

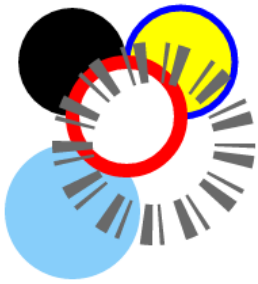
1.2.3. *Окружность <circle>*

Тег <*circle*> служит для вывода базовой фигуры – окружности. С помощью этого тега можно нарисовать круг не задавая координаты его центра: <*circle r*="40"/>- попробуйте и убедитесь. Этот минимальный код

эквивалентен коду *<circle cx="0" cy="0" r="40" fill="black"/>*. Основными атрибутами тега являются координаты центра окружности *cx*, *cy* и радиус *r*. Однако если использовать только их, то будет изображен черный круг, как на рисунке в табл. 1.4. Чтобы изобразить окружность, нужно обязательно определить цвет с помощью атрибута *stroke* и удалить заполнение цветом *fill="none"*. Различные варианты значений атрибутов для тега *<circle>* приведены в табл. 1.5.

Таблица 1.5

Влияние атрибутов тега *<circle>* на вид окружностей

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <circle cx="80" cy="50" r="40"/> <circle cx="160" cy="50" r="40" fill="yellow" stroke="blue" stroke-width="5"/> <circle cx="80" cy="160" r="50" fill="lightskyblue"/> <circle cx="120" cy="90" r="40" fill="white" stroke="red" stroke-width="10" /> <circle cx="140" cy="110" r="60" fill="none" stroke="dimgray" stroke-width="30" stroke-dasharray="3,5,8,13"/> </svg>	

Итак, мы рассмотрели особенности использования трех первых базовых тегов *<line>*, *<rect>*, *<circle>* и их атрибутов. Создадим на их основе изображение робота логотипа ОС Android, используя всего 4 линии, 3 круга и 6 прямоугольников (табл. 1.6).

Таблица 1.6

Робот Android из *<line>*, *<rect>* и *<circle>*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <line x1="145" y1="75" x2="185" y2="145" stroke-width="15" stroke="yellowgreen" stroke-linecap="round"/> <line x1="375" y1="75" x2="330" y2="145" stroke-width="15" stroke="yellowgreen" stroke-linecap="round"/> <line x1="260" y1="200" x2="260" y2="250" stroke-width="15" stroke="white" stroke-linecap="round"/> <line x1="200" y1="270" x2="325" y2="270" stroke-width="15" stroke="white" stroke-linecap="round"/> <circle cx="260" cy="270" r="150" fill=" yellowgreen"/> <circle cx="185" cy="200" r="15" fill="white"/> <circle cx="330" cy="200" r="15" fill="white"/> <rect x="110" y="270" height="260" width="300" fill="yellowgreen" rx="25"/> <rect x="110" y="270" height="10" width="300" fill="white"/> <rect x="30" y="280" height="180" width="70" fill="yellowgreen" rx="35"/> <rect x="418" y="280" height="180" width="70" fill="yellowgreen" rx="35"/> <rect x="150" y="500" height="180" width="70" fill="yellowgreen" rx="35"/> <rect x="295" y="500" height="180" width="70" fill="yellowgreen" rx="35"/> </svg>	

1.2.4. *Эллипс <ellipse>*

Тег <ellipse> служит для вывода базовой фигуры – эллипса. Атрибуты *cx* и *cy* задают центр фигуры, а *rx* и *ry* определяют длины горизонтальной и вертикальной осей соответственно. По умолчанию фигура не имеет контура и заливается черным цветом. В табл. 1.7 приводятся различные примеры элемента ellipse.

Таблица 1.7

Влияние атрибутов тега <*ellipse*> на вид эллипсов

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <ellipse cx="80" cy="110" rx="75" ry="105" fill="blue"/> <ellipse cx="80" cy="110" rx="60" ry="40" fill="black" stroke="red" stroke-width="25"/> <ellipse cx="80" cy="110" rx="35" ry="20" fill="green" stroke="yellow" stroke-width="25"/> <ellipse cx="80" cy="50" rx="40" ry="30" fill="red" stroke="black" stroke-width="25"/> <ellipse cx="80" cy="50" rx="30" ry="20" fill="orange" stroke="red" stroke-width="10" stroke-dasharray="5,5"/> /> <ellipse cx="80" cy="170" rx="40" ry="30" fill="yellow" stroke="orange" stroke-width="25" stroke-dasharray="2,2"/> <ellipse cx="80" cy="170" rx="30" ry="20" fill="red" stroke="black" stroke-width="10"/> </svg>	

Очевидно, что все атрибуты эллипса схожи с атрибутами окружности.

1.2.5. *Ломаная линия <polyline>*

Для рисования фигуры, состоящей из нескольких соединенных отрезков, можно воспользоваться тегом <*line*>, задавая для каждого отрезка координаты пары точек – начальной и конечной. Однако более удобным в данном случае, является применение тега <*polyline*>, где конечная точка предыдущего отрезка служит начальной точкой для последующего.

Атрибут *points* позволяет задавать пары координатных точек, разделенных пробелами. Для удобства восприятия кода координаты внутри пары *x*,*y* можно записывать через запятую, а можно этого и не делать, т. е. все значения разделять пробелами:

```
<polyline points="80 70 30 70 70 20 70 110" />
```

По умолчанию фигура заполняется заливкой черного цвета. В табл. 1.8 приводятся различные примеры использования тега <*polyline*>.

Как можно догадаться, ломаная линия изображает цифру 4, в первом случае помешала черная заливка по умолчанию, во втором – хотелось пока-

зать, как сделать линию красной *stroke="red"* и потолще *stroke-width="7"*. И наконец, убрав заливку *fill="none"*, *можно* получить красную четверку.

Таблица 1.8

Влияние атрибутов тега <*polyline*> на заливку

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <polyline points="80,70 30,70 70,20 70,110"/> <polyline points="80,70 30,70 70,20 70,110" stroke="red" stroke-width="7" transform="translate(90,0)"/> <polyline points="80,70 30,70 70,20 70,110" stroke="red" stroke-width="7" fill="none" transform="translate(180,0)"/> </svg>	

Во втором и третьем тегах <*polyline*> добавлен атрибут *transform* со значениями *"translate(90,0)"* и *"translate(180,0)"*. Это сделано только для того, чтобы сдвинуть второе и третье изображения на 90 и 180 пикселов вправо соответственно. Атрибут transform будет подробно рассмотрен в разд. 1.4.

1.2.6. *Многоугольник <polygon>*

Тег <*polygon*> предназначен для рисования фигур, заполненных заливкой. Конечно, с помощью атрибутов этого элемента можно рисовать контуры без заполнения цветом, но поскольку начальная и конечная точки будут соединяться автоматически, контур будет всегда замкнутым.

Атрибут *points* так же, как и в теге <*polyline*>, позволяет задавать пары координатных точек. В табл. 1.9 приводятся различные примеры использования тега <*polylgon*>. Как и в предыдущем примере, во втором и третьем тегах добавлен атрибут *transform* только для того, чтобы сместить второе и третье изображения на этот раз по вертикали.

Таблица 1.9

Влияние атрибутов тега <*polygon*>

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <polygon points="100,10 40,198 190,78 10,78 160,198" fill="yellow" stroke="black" stroke-width="5" fill-rule="nonzero"/> <polygon points="100,10 40,198 190,78 10,78 160,198" fill="yellow" stroke="black" stroke-width="5" fill-rule="evenodd" transform="translate(0,180)"/> <polygon points="100,10 40,198 190,78 10,78 160,198" fill="none" stroke="black" stroke-width="5" transform="translate(0,360)"/> </svg>	

Как видно из приведенного примера, возможны различные способы заливки, которые определяются значением атрибута *fill-rule.* Если атрибуту *fill* присвоить значение *fill="none",* то заполнения цветом не будет.

1.2.7. *Траектория <path>*

Тег <*path*> позволяет создавать произвольные траектории с использованием набора специальных функций, приведенных в табл. 1.10. С помощью тега <*path*> можно создать любую геометрическую фигуру из всех предыдущих примеров, и значительно более сложную.

Таблица 1.10

Функции рисования, поддерживаемые тегом <*path*>

Функция	Описание
M-m	Переместиться в точку
L-l	Провести линию до точки
H-h	Провести горизонтальную линию до точки
V-v	Провести вертикальную линию до точки
C-c	Нарисовать кубическую кривую Безье
S-s	Нарисовать гладкую кривую
Q-q	Нарисовать квадратичную кривую Безье
T-t	Нарисовать гладкую квадратичную кривую Безье
A-a	Нарисовать эллиптическую дугу до точки
Z-z	Замкнуть траекторию в точке

Эти функции можно использовать как в верхнем, так и в нижнем регистрах. Если функция задана в верхнем регистре, применяется абсолютное позиционирование. Если функция используется в нижнем регистре, применяется относительное позиционирование. В табл. 1.11 приведен пример использования функций рисования эллиптической дуги и кубических кривых Безье.

Кривые Безье весьма часто используются в SVG-графике для формирования путей и замысловатых фигур. На втором рисунке табл. 1.11 изображена одна из таких фигур и в дополнение приведены координаты точек сопряжения кривых отдельных кривых. Используя эти данные, а также генератор online из [19], попробуйте изменять значения атрибутов и понять их влияние на характер кривых.

Еще один пример использования кривых Безье для формирования контура кузова автомобиля *id="jeep"* приведен в табл. 1.12. Отметим, что колеса *id="w1"* и *id="w2"* сформированы отдельно с помощью тега <*circle*>.

Таблица 1.11

Пример использования функций рисования

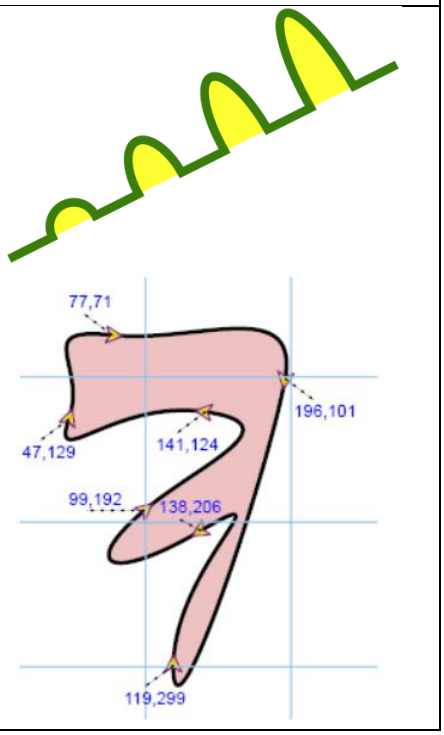
Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <path d="M20,300 l 50,-25 a25,25 -30 0,1 50,-25 l 50,-25 a25,50 -30 0,1 50,-25 l 50,-25 a25,75 -30 0,1 50,-25 l 50,-25 a25,100 -30 0,1 50,-25 l 50,-25" fill="yellow" stroke="green" stroke-width="10"/> </svg> <svg xmlns="http://www.w3.org/2000/svg"> <path stroke="black" stroke-width="3" fill="#eec1c2" d="M 99 192 C 137 160 204 133 141 124 C 78 115 34 167 47 129 C 60 91 20 65 77 71 C 134 77 206 43 196 101 C 186 159 118 368 119 299 C 120 230 201 169 138 206 C 75 243 53 231 99 192" /> </svg>	

Таблица 1.12

Пример использования кривых Безье

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <g id="jeep"> <g transform="translate(0,283) scale(0.1,-0.1)" fill="red" stroke="none"> <path d="M1289 2454 c-8 -9 -37 -83 -64 -163 l-51 -146 -134 -6 c-74 -4 -207 -7 -296 -8 -141 -1 -163 -3 -177 -19 -13 -14 -17 -45 -19 -152 l-3 -135 -32 -3 -33 -3 0 -105 0 -104 110 0 110 0 0 55 c0 130 95 230 231 242 49 5 66 1 122 -24 95 -45 148 -125 149 -224 l0 -49 284 0 284 0 0 33 c0 95 60 196 144 240 57 30 165 30 221 0 83 -44 132 -117 142 -213 l6 -60 98 0 99 0 0 59 0 60 -52 3 -53 3 0 40 0 39 47 -3 c41 -2 51 1 73 25 25 27 25 28 25 206 0 178 0 179 -25 204 -21 21 -33 25 -73 23 l-47 -3 -3 89 c-2 49 -8 95 -14 102 -8 10 -116 13 -533 13 -472 0 -524 -2 -536 -16z m451 -184 l0 -140 -215 0 c-253 0 -231 -19 -184 162 l32 118 183 0 184 0 0 -140z m545 0 l0 -135 -242 -3 -243 -2 0 140 0 140 243 -2 242 -3 0 -135z"></path> </g> <circle id="w1" cx="95" cy="118" r="20" fill="#E0E0E0" stroke="black" stroke-width="8" stroke-dasharray="2,2"> </circle> <circle id="w2" cx="203" cy="118" r="20" fill="#E0E0E0" stroke="black" stroke-width="8" stroke-dasharray="2,2"> </circle> </g> </svg>	

Пока можно не обращать внимания на незнакомые теги и атрибуты, использованные в этом примере. Они будут изучены в дальнейшем. Полученный рисунок автомобиля нам потребуется в других примерах.

1.2.8. *Вставка текста <text>*

Тег <*text*> в *SVG* позволяет делать гораздо больше, чем в обычном HTML. Рассмотрим обычный синтаксис, представленный в табл. 1.13. Координаты *x,y* определяют точку начала текста, а атрибуты *font-size* и *fill* – размер шрифта 20 и цвет букв *blue*. Если не задать указанные атрибуты, то текст будет стандартного размера, черного цвета, написан шрифтом *Times New Roman* (это параметры текста по умолчанию).

Таблица 1.13

Пример использования тега <*text*>

Код SVG	Текст в SVG
<svg xmlns="http://www.w3.org/2000/svg"> <text x="10" y="30" font-size="20" fill="blue"> Текст в SVG! </text> </svg>	Текст в SVG!

Как и в HTML, текст может быть стилизован с помощью атрибутов приведенных в табл. 1.14, как и обычный HTML. Если необходимо применить стиль к отдельному элементу, то необходимо использовать дополнительно тег <*tspan*>.

Таблица 1.14

Атрибуты тега <*text*>

Атрибут	Описание
font-family	Определяет шрифт, например *Arial*
font-size	Задает размер шрифта
kerning	Задает промежуток между буквами
letter-spacing	То же, что и *kerning*
word-spacing	Задает промежуток между словами
text-decoration	Может принимать значения *underline, overline* или *line-through*
stroke	Задает цвет обводки букв
stroke-width	Задает толщину обводки букв
fill	Задает цвет букв

Пример использования возможностей представленных атрибутов приведен в табл. 1.15. Наряду с этими атрибутами, для тега <*tspan*> используется атрибут *dy*, который смещает очередной фрагмент текста на указанное число пикселов по оси *y* относительно текущего значения. Аналогично, атрибут *dx* осуществляет смещение по оси *x*. Можно использовать *dx* и *dy* совместно, задавая положительные и отрицательные значения.

Таблица 1.15

Пример использования атрибутов

Код *SVG*
<svg xmlns="http://www.w3.org/2000/svg"> <text x="10" y="20" font-size="20" > Пример: <tspan font-weight="bold">жирный шрифт</tspan>, <tspan font-style="italic" fill="red">наклонный шрифт</tspan> и <tspan text-decoration="underline">подчеркнутый текст</tspan>. </text> <text x="10" y="50" fill="green" font-size="20" font-family="Bookman Old Style"> E = m c <tspan dy="-10" font-size="15">2</tspan> </text> <text x="10" y="85" font-size="40" kerning="3" font-family="American Typewriter" fill="yellow" stroke="blue" stroke-width="1">Эйнштейн </text> </svg>
Текст в *SVG*
Пример: **жирный шрифт**, *наклонный шрифт* и подчеркнутый текст. E = m c 2 Эйнштейн

Наконец, приведем еще один атрибут тега <*text*>, который часто используется при оформлении интернет-ресурсов, это тег <*textPath*>, который позволяет расположить текст вдоль произвольной траектории. Пример его использования приведен в табл. 1.16. Описание траектории или пути (*path*) выполняется в теге <*defs*> (разд. 1.3.3), причем тег <*path*> должен быть снабжен идентификатором *id* для обеспечения возможности ссылки.

Далее при вводе текста используется тег <*textPath*>, где сначала задается ссылка на описание траектории, а затем записывается текст.

Таблица 1.16

Пример использования тега <*textPath*>

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <defs> <path id="path1" d="M75,20 a1,1 0 0,0 100,20"/> </defs> <text x="10" y="100" fill="green"> <textPath xlink:href="#path1">Текст по пути path1 !</textPath> </text> </svg>	Текст по пути path1 !

Примечание. Поскольку в теге <*textPath*> используется атрибут *xlink*, который не определен в пространстве имен *http://www.w3.org/2000/svg*, необходимо в заголовок *SVG*-документа добавить ссылку, определяющую новое пространство имен *xmlns:xlink= "http://www.w3.org/1999/xlink"*.

Существует еще много возможностей стилизации и декорирования текста в *SVG*, но самые впечатляющие из них можно осуществлять, используя градиент и анимацию. Особенности этих технологий будут рассмотрены в соответствующих разделах пособия.

1.2.9. *Вставка изображений <image>*

Тег <*image*> в *SVG* очень похож на тег <*img*> в *HTML* – он предоставляет возможность размещения контента в виде изображения формата *PNG*, *JPEG* или *SVG* в прямоугольник на страницу. Фактически атрибуты этого тега определяют координаты (*x*,*y*) левого верхнего угла прямоугольника и его размеры – ширину *width* и высоту *height*, а также имя файла или его *URL*, из которого загружается изображение.

Заметим, что:

- изображения могут перекрывать друг друга;
- прозрачность, если она существовала в исходном в файле, сохраняется;
- по умолчанию изображение растягивается, чтобы заполнить прямоугольник;
- можно сохранить соотношение сторон изображения.

Важно подчеркнуть, что для поддержки браузерами обработки тега <*image*> необходимо в заголовок *SVG*-документа добавлять ссылку *xmlns:xlink= http://www.w3.org/1999/xlink* (см. примечание к табл. 1.16), определяющую новое пространство имен.

Таблица 1.17

Вставка изображений различных форматов

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" **xmlns:xlink= "http://www.w3.org/1999/xlink"**> <image xlink:href="photo.png" height="200" width="100" x="10" y="10"/> <image xlink:href="photo.png" height="100" width="100" x="100" y="10"/> <image xlink:href="photo.png" height="30" width="30" x="150" y="150"/> </svg>	

В табл. 1.17 приведен пример вставки изображения формата *PNG* в разные точки плоскости координат с разными размерами предоставляемого пространства.

1.3. Группировка объектов

SVG предоставляет возможности для структурирования документа посредством специальных элементов, которые позволяют определять и группировать объекты, а также ссылаться на них в дальнейшем. Они упрощают повторное использование кода и делают его удобно читаемым. В данном разделе будут рассмотрены эти элементы, их различия и предпочтения для использования.

1.3.1. *Группировка <g>*

Тег <*g*> используется для логической группировки набора связанных графических элементов. Эту процедуру можно сравнить с группировкой объектов в графических редакторах. Тег <*g*> играет роль контейнера, в котором объединяет в группу все свое содержимое. В этой группе присваивается идентификатор, по которому производится обращение к ней в дальнейшем. Любые свойства, которыми наделяется элемент группы <*g*>, будут также применены ко всем его потомкам. Это позволяет задавать различные стили и преобразования, а также добавлять интерактивность и анимацию сразу целой группе объектов.

Например, чтобы не задавать стиль каждому отдельному кругу, можно использовать тег <*g*>, в котором задан цвет заполнения, цвет и толщина контура, как показано в примере табл. 1.18.

Таблица 1.18

Пример использования тега группировки <*g*>

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg"> <g id="gr1" stroke="red" fill="white" stroke-width="5"> <circle cx="25" cy="25" r="15"/> <circle cx="40" cy="25" r="15"/> <circle cx="55" cy="25" r="15"/> <circle cx="70" cy="25" r="15" stroke="green" fill="yellow" stroke-width="3"/> </g> </svg>	

Из примера табл. 1.18 также видно, что изменение стиля последнего круга превалировало над групповым стилем.

Особую пользу группировка объектов может принести, если необходимо добавить к *SVG*-графике интерактивности или задать какие-то преобразования. Сгруппировав элементы, можно перемещать их, масштабировать или поворачивать все вместе, сохраняя их положение друг относительно друга. Эти возможности будут продемонстрированы в соответствующих разделах.

Группировка объектов очень полезна для структурирования документа.

Тег <*g*> имеет еще одну важную и интересную особенность: он может содержать теги <*title*> и <*desc*>, которые добавляют метаданные, позволяющие упростить классификацию и индексацию графического контента в вебе.

Наконец, в случае необходимости непосредственно не отображать объект, сформированный в теге <*g*>, а использовать его в другом месте достаточно добавить атрибут *display="none"*.

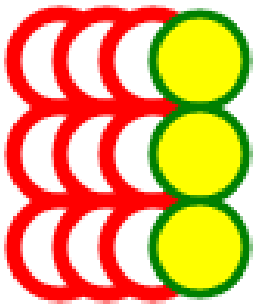
1.3.2. *Повторное использование <use>*

При работе с графикой часто можно встретить ситуацию, когда применяются повторяющиеся объекты. В графических редакторах в этом случае обычно используется метод копировать-вставить, что удобнее создания объекта с нуля. В *SVG* подобную функциональность реализует тег <*use*>. Он может применяться для повторного использования как отдельных объектов, так и групп объектов.

Тег <*use*> использует в качестве атрибутов координаты *x, y* для размещения объекта и ссылку *xlink:href* на объект. В качестве ссылки выступает идентификатор объекта или группы. В табл. 1.19 приведен пример использования группы *gr1* из предыдущего примера для ее клонирования с помощью тега <*use*>. Обратите внимание на то, что тег <*g*> помещен внутрь тега <*defs*>. Попробуйте убрать тег <*defs*> и посмотрите на реакцию браузера. Особенности использования тега <*defs*> будут рассмотрены в следующем разделе.

Таблица 1.19

Пример использования тега <*use*>

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <defs> <g id="gr1" stroke="red" fill="white" stroke-width="5"> <circle cx="25" cy="25" r="15"/> <circle cx="40" cy="25" r="15"/> <circle cx="55" cy="25" r="15"/> <circle cx="70" cy="25" r="15" stroke="green" fill="yellow" stroke-width="3"/> </g> </defs> <use x="10" y="10" xlink:href="#gr1" /> <use x="10" y="40" xlink:href="#gr1" /> <use x="10" y="70" xlink:href="#gr1" /> </svg>	

Обратите внимание, что в атрибуте *xlink:href* можно ссылаться на любой *SVG*-объект, даже находящийся во внешнем файле. Это позволяет, например использовать для работы внешнее хранилище с повторно используемыми компонентами.

Следует обратить внимание на то, что координаты, задаваемые элементу <*use*> отсчитываются не от начала координат всего *SVG*-изображения. На самом деле это сокращенная форма записи атрибута transform. Следующие две строчки являются эквивалентными:

```
<use x="10" y="10" xlink:href="#gr" />
<use xlink:href="#gr1" transform="translate(10, 10)" />
```

Проще говоря, координаты элемента <*use*> задаются относительно исходного элемента. Такое поведение не всегда оптимально и может быть недостатком.

Другим недостатком <*use*> является то, что копии будут использовать те же стили, что и исходный элемент. При применении стилей или преобразований к группе эти стили и преобразования будут распространяться на все ее копии.

Однако все-таки можно применить независимое преобразование к элементу <*use*>. Например, следующая строка кода позволяет повторно использовать объект *gr1*, размеры которого будут составлять лишь половину от размеров исходного объекта:

```
<use x="10" y="10" xlink:href="#gr1" transform="scale(0.5)" />
```

Попробуйте добавить это преобразование к одному из тегов <*use*> в примере из табл. 1.19. При этом принцип работы системы координат может показаться несколько неожиданным. Она также масштабируется. Если исходный элемент был расположен в 10 пикселах от края изображения, то такая его копия будет расположена в 5 пикселах от края. На заданные значения *x* и *y* это тоже распространяется. Впрочем, особенности использования преобразований типа *transform="scale(0.5)"* будут подробно рассмотрены в разд. 1.4.5.

В отличие от преобразований, переопределить стили копии нельзя. Таким образом, если будет необходимо создать множество клонов объекта разного цвета, то использовать для этого <*use*> не получится (если только исходный элемент не определен внутри <*defs*> без своих стилей, но об этом в разд. 1.3.3).

Элемент <*use*> позволяет повторно использовать объект, который уже отображается на *SVG*-изображении. Если нужно просто определить объект, не отображая его, а затем отрисовать в нужном месте, когда это потребуется, на помощь придет элемент <*defs*>.

1.3.3. *Определение <defs>*

Тег <*defs*> может использоваться для хранения содержимого, которое не будет отображаться при определении. Содержимое в нем хранится в скрытом виде и ждет, когда оно будет использовано и отображено другими элементами *SVG*. Храниться в <*defs*> может что угодно, начиная с группы элементов и заканчивая маской, градиентом или фильтром. Это просто контейнер для дальнейшего использования, сам по себе он никогда не отображается, отображаются только использующие его сущности.

В табл. 1.20 приведен пример, в котором градиент сначала определяется, а затем используется для заливки круга.

Таблица 1.20

Пример использования тега <*defs*>

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <defs> <radialGradient id="gradient"> <stop offset="0%" stop-color="#d1d4e5" /> <stop offset="100%" stop-color="#8ca0ff" /> </radialGradient> </defs> <circle cx="50" cy="50" r="40" fill="url(#gradient)"/> </svg>	

Определение радиального градиента внутри <*defs*> гарантирует, что он не будет внезапно отображен сам по себе, только при использовании где-либо. В частности, использование градиента при окрашивании круга сделало его похожим на воздушный шарик. Подробно эффекты градиента будут рассмотрены в разделе 2.1.

В предыдущем разделе было упомянуто два недостатка тега <*use*>:

- положение нового объекта задается относительно исходного;
- стили исходного объекта не могут быть переопределены в копиях.

Кроме того, исходный объект отображается сам по себе. Всех этих недостатков тег <*defs*> лишен. Оригинальный объект не отображается и при использовании объекта, определенного внутри <*defs*>, положение каждого экземпляра задается относительно начала системы координат.

В качестве еще одного примера приведем код табл. 1.21. В нем определена группа *gr1*, объединяющая круг и прямую линию. Результирующий объект напоминает шарик на веревочке.

С помощью тега <*use*> сделано три копии этого шарика. Как можно видеть, каждый расположен в соответствии с указанными координатами (правда с учетом масштабирования) и имеет индивидуальный цвет.

Таблица 1.21

Пример использования тега <*use*>

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <defs> <g id="gr1"> <circle cx="25" cy="25" r="15"/> <line x1="25" y1="40" x2="25" y2="100" stroke-width="1" stroke="gray"/> </g> </defs> <use x="10" y="10" xlink:href="#gr1" fill="palegreen"/> <use x="50" y="25" xlink:href="#gr1" fill="salmon" transform="scale(0.8)"/> <use x="40" y="10" xlink:href="#gr1" fill="khaki" transform="scale(1.4)"/> </svg>	

Таким образом, <*defs*> прекрасно подходит для того, чтобы определить самую основу, которая будет использована и раскрашена при необходимости. Используя только <*use*> без <*defs*>, добиться такой гибкости было бы невозможно. Обратите внимание, что элементы внутри <*defs*> не отображаются, т. е. ведут себя точно так же, как элемент <g> с установленным свойством *display="none"*.

1.3.4. *Группировка <symbol>*

Тег <*symbol*> по своим возможностям похож на тег <*g*> – он также предоставляет возможность группировать элементы. Можно выделить два основных отличия:

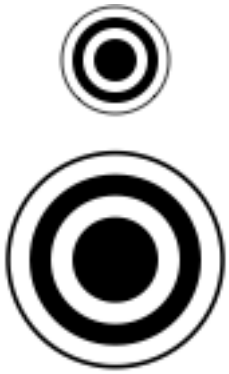
- тег <*symbol*> не отображается сам по себе. Этим он похож на <*defs*>;
- тег <*symbol*> может иметь собственные атрибуты *viewBox* и *preserveAspectRatio*. Это позволяет ему уместиться в области просмотра (*viewport*) так, как необходимо, а не как это определено по умолчанию.

В большинстве случаев <*symbol*> подходит для определения повторно используемых объектов (символов). Он все так же служит шаблоном для <*use*>, а имея собственные атрибуты *viewBox* и *preserveAspectRatio*, он может растягиваться на прямоугольную область просмотра, задаваемую в ссылающемся на него элементе <*use*>. Следует учитывать, что элементы <*symbol*> определяют новую область просмотра каждый раз, когда вызываются элементом <*use*>. Это прекрасная особенность тега <*symbol*>. Она позволяет определять объекты, не зависящие от области просмотра, в которую они попадут. Они будут всегда отображаться заданным образом.

В качестве примера использования тега <*symbol*> рассмотим код табл. 1.22.

Таблица 1.22

Пример использования тега <*symbol*> и <*use*>

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="256" height="256" viewBox="0 0 48 48" > <symbol id="target" viewBox="0 0 200 200" preserveAspectRatio="xMidYMid meet"> <circle cx="120" cy="120" r="50" fill="white" stroke="black" stroke-width="2"/> <circle cx="120" cy="120" r="40" fill="black"/> <circle cx="120" cy="120" r="30" fill="white"/> <circle cx="120" cy="120" r="20" fill="black"/> </symbol> <use x="9" y="0" width="15" height="15" xlink:href="#target" /> <use x="0" y="5" width="30" height="30" xlink:href="#target" /> </svg>	

Для понимания предоставляемых возможностей необходимо знать, как работают атрибуты *viewBox* и *preserveAspectRatio.* Эти вопросы будут рассматриваться в разд. 3.2. Дополнительную информацию по тегу <*symbol*> можно получить в [18].

В заключение раздела отметим, что все рассмотренные теги являются структурными контейнерами *SVG*, позволяющими облегчить повторное использование объектов, делая при этом код более понятным. Каждый из этих тегов имеет свою сферу применения. Теперь, зная, что каждый из них делает и чем они отличаются друг от друга, будет несложно решить, какой из них использовать в зависимости от ситуации.

1.4. Трансформации объектов

Трансформации служат для изменения формы или места расположения *SVG*-объектов. В качестве объектов могут выступать как отдельные *SVG*-элементы, так и группы элементов. Все преобразования осуществляются при помощи атрибута *transform*, который позволяет реализовать следующие трансформации: *translate* – перемещение объекта, *rotate* – поворот на заданный угол, *scale* – масштабирование, *skewX/skewY* – скос объекта вдоль горизонтальной и/или вертикальной оси и *matrix(a, b, c, d, e, f)* – задание параметров трансформации в виде матрицы.

К объектам SVG можно последовательно применять любое число различных преобразований, перечисленных выше.

1.4.1. *Перемещение translate*

Функция *translate* представляет собой перенос начала системы отсчета в новую точку с заданными координатами. В табл. 1.23 приведен пример использования функции перемещения объекта. Исходный и перемещенный объекты имеют идентичные исходные координаты левого верхнего угла *x = 25 y = 25*.

Таблица 1.23

Пример использования функции перемещения *translate*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <rect x="25" y="25" width="50" height="25" fill="red"/> <rect x="25" y="25" width="50" height="25" fill="blue" transform="translate(50,25)"/> </svg>	

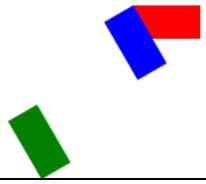
В процессе реализации функции исходные координаты изменяются на величины, указанные в *translate(50,25)*. В результате исходный объект перемещается в точку *x = 75* и *y = 50*.

1.4.2. *Поворот rotate*

Функция *rotate* поворачивает систему координат относительно точки отсчета на задаваемый угол. Если в функцийе *rotate* задан только угол поворота, то поворот объекта осуществляется относительно точки начала координат (0,0). Для поворота фигуры вокруг точки принадлежащей самой фигуре в функцийе *rotate* следует указать координаты этой точки. В табл. 1.24 приведен пример использования функции поворота.

Таблица 1.24

Пример использования функции поворота *rotate*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <rect x="100" y="25" width="50" height="25" fill="red"/> <rect x="100" y="25" width="50" height="25" fill="green" transform="rotate(60)"/> <rect x="100" y="25" width="50" height="25" fill="blue" transform="rotate(60, 100, 25)"/> </svg>	

Исходный объект – красный прямоугольник. Первая операция поворота *rotate(60)* в результате дала зеленый прямоугольник, который получился при повороте красного треугольника на 60° по часовой стрелке относительно центра координат *(0,0)*.

Вторая операция поворота *rotate(60, 100, 25)* дала в результате синий прямоугольник, который получился при повороте красного треугольника на 60° по часовой стрелке относительно координаты *(100,25)* левого верхнего угла.

1.4.3. *Масштабирование scale*

Функция *scale* позволяет масштабировать объект с указанием кратности изменения размеров вдоль горизонтальной и вертикальной осей. Если в функции указывается один параметр, то он принимается для обеих осей.

В табл. 1.25 приведен пример использования функции масштабирования. Обратите внимание, что все прямоугольники имеют идентичные размеры и координаты верхнего левого угла, а расположены в разных местах и разновелики.

Таблица 1.25

Пример использования функции масштабирования *scale*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <rect x="10" y="25" width="50" height="25" fill="red"/> <rect x="10" y="25" width="50" height="25" fill="green" transform="scale(2)"/> <rect x="10" y="25" width="50" height="25" fill="blue" transform="scale(2,3)"/> </svg>	

Это происходит потому, что масштабированию, т. е. умножению на указанные в функции scale коэффициенты, подлежат все атрибуты, в том числе *x* и *y*. Таким образом, координаты верхнего левого угла зеленого прямоугольника стали равными (20,50), а синего – (20,75).

Как показано в примере табл. 1.26, подобные метаморфозы происходят и с текстом. Наряду с тем что пропорционально с заданными коэффициентами масштабирования изменяются размеры букв, происходит и сдвиг текста.

Таблица 1.26

Пример использования функции масштабирования *scale* для текста

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <text x="50" y="50" font-size="20" fill="red"> ИСиТ</text> <text x="50" y="50" font-size="20" fill="blue" transform="scale(3,1)"> ИСиТ</text> <text x="50" y="50" font-size="20" fill="green" transform="scale(1,3)"> ИСиТ</text> <text x="50" y="50" font-size="20" fill="black" transform="scale(3)"> ИСиТ</text> </svg>	

Наряду с тем что пропорционально с заданными коэффициентами масштабирования изменяются размеры букв, происходит и сдвиг текста.

Функция *scale* может иметь полезное применение, когда необходимо получить зеркальное отображение объекта. Если объект нужно отобразить зеркально относительно оси *X*, то используется функция *scale(–1,1)*, а относительно *Y*, то *scale(1,–1)*. Единственное, о чем нельзя забывать, так это о том, что объект надо предварительно сдвинуть либо по оси *X*, либо по оси *Y*, иначе он не будет виден, так как перейдет в область отрицательных значений соответствующей ординаты.

В табл. 1.27 приведен пример зеркального отражения трапеции относительно оси *X*.

Таблица 1.27

Использование функции *scale* для получения зеркального отображения

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <path d="M20,20 l20,20 l0,20 l-20,20 Z" fill="red"/> <path d="M20,20 l20,20 l0,20 l-20,20 Z" fill="blue" transform="translate(100, 0) scale(-1, 1)" /> </svg>	

Исходная трапеция красного цвета. Прежде чем получить ее зеркальное отображение, т. е. выполнить функцию *scale(–1,1)*, она была окрашена в синий цвет и был выполнен ее сдвиг на 100 пикселов вправо.

1.4.4. *Скос skewX и skewY*

Функции *skewX* и *skewY* предназначены для искажения формы фигуры в направлении горизонтальной и вертикальной осей.

Функция *skewX*(*n*) вычисляет новые значения ординат по формуле *Y**tg(*n*), а значения абсцисс оставляет без изменения (*n* – значение угла в градусах). Это создает эффект покосившегося объекта.

В свою очередь, применение функции *skewY* оставляет прежними ординаты, изменяя абсциссы по формуле *X**tg(*n*). При совместном использовании этих функций эффект будет суммарным. В табл. 1.28 приводятся примеры использования этих функций.

Таблица 1.28

Пример использования функций скоса *skewX* и *skewY*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <rect x="20" y="30" width="50" height="25" fill="red"/> <rect x="20" y="30" width="50" height="25" fill="green" transform="skewX(45)"/> <rect x="20" y="30" width="50" height="25" fill="blue" transform="skewY(45)"/> <rect x="20" y="30" width="50" height="25" fill="yellow" transform="skewX(45) skewY(45)"/> </svg>	

1.4.5. *Трансформация matrix*

Преобразования на плоскости в общем виде могут быть выражены в виде матрицы третьего порядка:

a c e
b d f
0 0 1,

причем третья строка имеет постоянные значения. Поэтому произвольное преобразование координат можно описывать шестью значениями *a,b,c,d,e,f*. Таким образом, можно использовать функцию *matrix(a,b,c,d,e,f)* для более компактной записи целой серии преобразований на основе атрибута *transform*. Важно понимать, что эта функция представляет применение одной или нескольких функций типа *translation, rotate, scale, skewX* и *skewY*.

В табл. 1.29 приведены примеры записи функций элементарных трансформаций с помощью функции *matrix(a,b,c,d,e,f)*.

Таблица 1.29

Запись элементарных трансформаций в виде матрицы

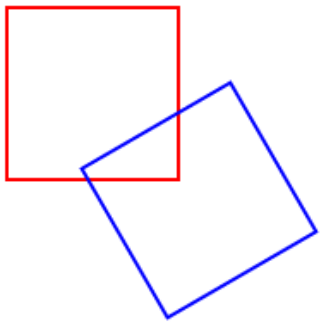
Преобразование	Матричная запись
translate (x, y)	matrix(1,0,0,1,x,y)
rotate (a)	matrix(cos(a),sin(a),-sin(a),cos(a),0,0)
scale (kx, ky)	matrix(kx,0,0,ky,0,0)
skewX (a)	matrix(1,0,tg(a),1,0,0)
skewY (a)	matrix(1,tg(a),0,1,0,0)
Тождественное преобразование	matrix(1,0,0,1,0,0)

Если над объектом выполняется несколько элементарных преобразований, то результирующая матрица представляет собой произведение отдельных матриц.

Предположим, выполняется два преобразования: поворот *rotate(30)* и перемещение *translate(40,100)*. Перемножим матрицы соответствующих функций *matrix(cos(30),sin(30),–sin(30),cos(30),0,0)* и *matrix(1,0,0,1,40,100)*. В результате получится функция *matrix(0.866,–0.5,0.5,0.866,40,100)*. В табл. 1.30 приводится пример использования полученной функции для преобразования квадрата.

Таблица 1.30

Пример использования функции *matrix(0.866 –0.5 0.5 0.866 40 100)*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <polygon points="0,0 0,100 100,100 100,0" fill="none" stroke="red" stroke-width="2"/> <polygon points="0,0 0,100 100,100 100,0" fill="none" stroke="blue" stroke-width="2" transform="matrix(.866 -.5 .5 .866 40 100)"/> </svg>	

Исходный квадрат красного цвета. Синий результирующий квадрат получен поворотом исходного на 30° и перемещением по оси *X* на 40 пикселов и по оси *Y* на 100 пикселов.

2. СТАТИЧЕСКИЕ ВИЗУАЛЬНЫЕ ЭФФЕКТЫ

2.1. Градиент

Понятие градиента тесно связано с процессом окрашивания объектов – различных фигур и текста. Процесс окрашивания часто называют заливкой.

Градиентом называют плавный переход от одного цвета к другому, причем самих цветов и переходов между ними может быть несколько. С помощью градиентов создаются самые причудливые эффекты веб-дизайна, например блики, фон, объемность и др.

При указании цвета значение соответствующего атрибута должно быть обязательно задано в кавычках. Причем допускается задание цвета одним из пяти возможных способов, например зеленый цвет может быть задан как: *"green"*, *"#0f0"*, *"#00ff00"*, *"rgb*(0,255,0)*"* или *"rgb*(0%,100%,0%)*"*. В табл. 2.1 приведено краткое описание атрибутов заливки.

Таблица 2.1

Атрибуты заливки

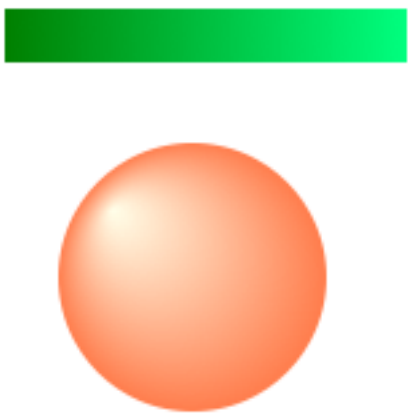
Атрибут	Описание	Диапазон значений	Значение по умолчанию
fill	Заливка цветом	Название или код цвета	black
fill-opacity	Прозрачность	0–1	1 – полная непрозрачность
fill-rule	Режим заливки	nonzero или evenodd	nonzero – полная заливка

Различают линейную градиентную заливку, предназначенную для получения плавного цветового перехода в направлении заданной прямой, и радиальную градиентную заливку, которая позволяет получать плавное изменение цвета в направлении от центра или к центру окружности. В первом случае используют тег <linearGradient>, а во втором – <radialGradient>.

Пример двухцветной линейной и радиальной градиентной заливки приведен в табл. 2.2. Первая использована для прямоугольника, а вторая – для круга.

Таблица 2.2

Пример линейного и радиального градиента

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <linearGradient id="Grad_1" x1="0%" y1="0%" x2="100%" y2="0%"> <stop offset="0" stop-color="green"/> <stop offset="100%" stop-color="springgreen"/> </linearGradient> <radialGradient id="Grad_2" fx="20%" fy="25%" > <stop offset="0" stop-color="lightyellow"/> <stop offset="100%" stop-color="orangered"/> </radialGradient> <rect x="20" y="20" width="150" height="20" fill="url(#Grad_1)" /> <circle cx="90" cy="120" r="50" fill="url(#Grad_2)" fill-opacity="0.7"/> </svg>	

Обязательный атрибут любого градиента – идентификатор *id*.

Градиент не отображается на странице, пока не будет применен к объекту. Так же как и обычная заливка или обводка, градиент задается с помощью атрибута *fill* или *stroke* объекта, подлежащего заливке, где и используется идентификатор.

Для получения желаемого эффекта от применения градиентной заливки необходимо изучить вспомогательные теги и атрибуты, которые при этом используются. Настоящий раздел и посвящен их изучению.

2.1.1. *Линейный градиент <linearGradient>*

Линейная градиентная заливка предназначена для получения плавного цветового перехода в направлении, заданном прямой. Она реализуется с помощью тега <*linearGradient*> и его вспомогательного тега <*stop*>. Вспомогательный тег использует, в свою очередь, три атрибута: *offset* – для определения границы цвета, *stop-color* – для указания цвета градиента и *stop-opacity* – для указания прозрачности. Атрибут *offset* определяет области покрытия исходных цветов и формируемого ими промежуточного градиента. Так, если задать значения атрибута *offset* следующим образом:

```
<linearGradient id="Grad1">
<stop offset="30%" stop-color="green"/>
<stop offset="70%" stop-color="yellow"/>
</linearGradient>,
```

то при такой градиентной заливке объекта 30 % (слева) будет залито чистозеленым цветом, а область, расположенная за 70 % (справа), – чистожелтым. Всю остальную область, между 30 и 70 % будет занимать градиент – постепенный переход от зеленого к желтому.

Ясно, что если задать значения 50 и 50 %, то никакого градиента не будет – левая половина объекта будет зеленой, правая – желтой. Второй предельный случай – 0 и 100 % – сплошной градиент от зеленого к желтому.

В табл. 2.3 приведены примеры для рассмотренного выше случая задания градиентов.

В случаях когда распространение линейного градиента должно быть не горизонтальным, тег <*linearGradient*> позволяет использовать две пары атрибутов *x1, y1* и *x2, y2*, которые позволят определить любое направление. Значения атрибутов *x1, y1* и *x2, y2* задаются в процентах, и их подбор можно понять из рис 2.1.

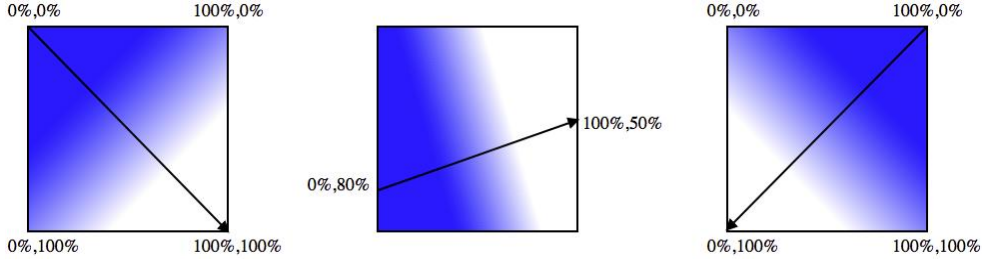


Рис. 2.1. Примеры задания направления градиента

Таблица 2.3

Пример использования атрибута *offset*

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <linearGradient id="Grad1"> <stop offset="30%" stop-color="green"/> <stop offset="70%" stop-color="yellow"/> </linearGradient> <linearGradient id="Grad2"> <stop offset="50%" stop-color="green"/> <stop offset="50%" stop-color="yellow"/> </linearGradient> <linearGradient id="Grad3"> <stop offset="0%" stop-color="green"/> <stop offset="100%" stop-color="yellow"/> </linearGradient> <rect x="20" y="20" width="50" height="50" fill="url(#Grad1)" /> <rect x="20" y="90" width="50" height="50" fill="url(#Grad2)" /> <rect x="20" y="160" width="50" height="50" fill="url(#Grad3)" /> </svg>	

В качестве примера задания направления градиента для нескольких цветов приведем пример кода из табл. 2.4. Полученный результат совпадает с цветами радуги.

Таблица 2.4

Градиент нескольких цветов

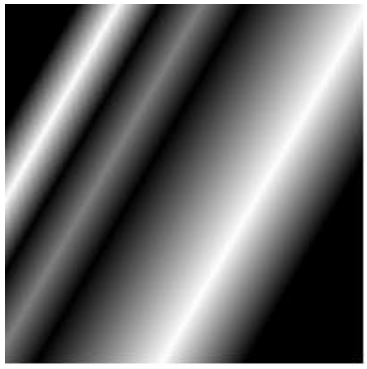
Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <linearGradient id="Grad" x1="0%" y1="80%" x2="100%" y2="20%" > <stop offset="0%" stop-color="lightskyblue"/> <stop offset="20%" stop-color="red"/> <stop offset="30%" stop-color="orange"/> <stop offset="40%" stop-color="yellow"/> <stop offset="50%" stop-color="green"/> <stop offset="60%" stop-color="deepskyblue"/> <stop offset="70%" stop-color="blue"/> <stop offset="80%" stop-color="purple"/> <stop offset="100%" stop-color="lightskyblue"/> </linearGradient> <rect x="30" y="60" width="150" height="150" fill="url(#Grad)"/> <text x="35" y="250" font-family="Apple Chancery" font-size="29" stroke="black" stroke-width="1" fill="url(#Grad)"> RAINBOW </text> </svg>	

Наряду с заливкой квадрата, в коде табл. 2.4 приведен пример заливки градиентом текста.

Отметим, что значение атрибута может записываться не только в процентах, но и с помощью дробных вещественных чисел в диапазоне от 0,0 до 1,0. Код табл. 2.5 показывает пример такой записи для формирования линейного множественного градиента трех цветов. Кроме того, здесь использован другой способ определения направления градиента с помощью атрибута *gradientTransform="rotate(30 .5 .5)"*. Вспомните, что определяют значения атрибута *rotate*, описанного в разд. 1.4.2.

Таблица 2.5

Множественный линейный градиент трех цветов

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <linearGradient id="l" gradientTransform="rotate(30 .5 .5)"> <stop offset=".0" stop-color="black"/> <stop offset=".1" stop-color="white"/> <stop offset=".2" stop-color="black"/> <stop offset=".3" stop-color="grey"/> <stop offset=".4" stop-color="black"/> <stop offset=".7" stop-color="white"/> <stop offset=".9" stop-color="black"/> </linearGradient> <rect fill="url(#l)" width="200" height="200"/> </svg>	

Наконец, атрибут *stop-opacity* служит для управления прозрачностью цвета, и его значение может записываться как в процентах, так и с помощью дробных вещественных чисел в диапазоне от 0,0 до 1,0. Попробуйте добавить этот атрибут в определение какого-нибудь градиента из предыдущих примеров и оцените эффект его влияния.

Эффект от линейного градиента можно дополнить с помощью вспомогательного атрибута *spreadMethod*. Этот атрибут позволяет использовать три значения:

pad – значение по умолчанию, распространяет начальный и конечный цвет до краев объекта;

repeat – повторяет зону градиента до краев объекта;

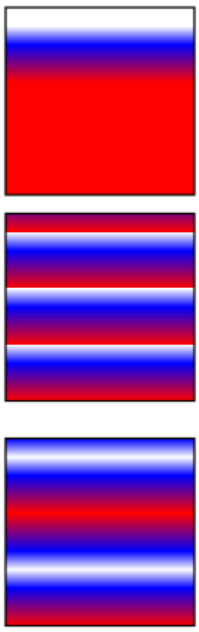
reflect – повторяет «перевернутую», отраженную зону градиента до краев объекта.

Использование атрибута *spreadMethod* продемонстрировано в SVG-коде (табл. 2.6).

Из примера видно, что значение атрибута *pad* не производит дополнительного эффекта от использования *spreadMethod*.

Таблица 2.6

Эффекты от использования атрибута *spreadMethod*

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <defs> <linearGradient id="lG" x1="0" y1=".1" x2="0" y2=".4"> <stop offset="0%" stop-color="white"/> <stop offset="33%" stop-color="blue"/> <stop offset="100%" stop-color="red"/> </linearGradient> <linearGradient id="padded" xlink:href="#lG" spreadMethod="pad"/> <linearGradient id="repeated" xlink:href="#lG" spreadMethod="repeat"/> <linearGradient id="reflected" xlink:href="#lG" spreadMethod="reflect"/> </defs> <rect x="40" y="20" width="100" height="100" fill="url(#padded)" stroke="black"/> <rect x="40" y="130" width="100" height="100" fill="url(#repeated)" stroke="black"/> <rect x="40" y="250" width="100" height="100" fill="url(#reflected)" stroke="black"/> </svg>	

2.1.2. *Радиальный градиент <radialGradient>*

Радиальная градиентная заливка позволяет получать плавное изменение цвета в направлении от центра или к центру окружности. Она реализуется с помощью тега <*radialGradient*>. Вспомогательный тег <*stop*> и три атрибута: *offset, stop-color и stop-opacity* работают так же, как и в линейном градиенте. Атрибут *offset* в данном случае отсчитывается в направлении радиуса окружности.

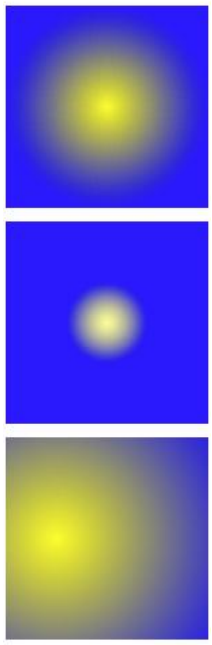
Для элемента *radialGradient* можно задать область распространения непосредственно в его определении, при помощи атрибутов *cx, cy* и *r*. Очевидно, что когда координаты *cx, cy* будут соответствовать центру объекта, то градиент будет распространяться симметрично во все стороны. Если координаты не указывать, то по умолчанию эпицентр градиента располагается в центре объекта, к которому он применяется. Если задать координаты со смещением, то эпицентр градиента сместится.

Атрибут *r* влияет на площадь распространения градиента, или, точнее, определяет радиус от эпицентра в процентах от размера объекта. По умолчанию *r* ="*50%*" (или, что то же, *r* ="*0.5*").

В табл. 2.7 приведены примеры, поясняющие сказанное. В градиенте *G1* значения атрибутов *cx, cy* и *r* браузер устанавливает по умолчанию. В градиенте *G2* радиус распространения ограничен до 20 %. Наконец, в градиенте *G3* эпицентр градиента смещен: *cx="25%" cy="50%"*, при этом градиент занимает площадь *r="100%"* всего объекта, в данном случае квадрата.

Таблица 2.7

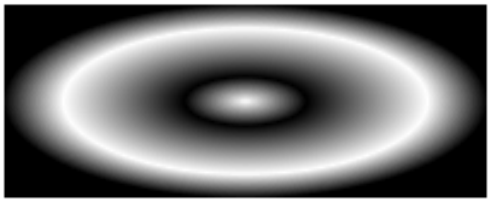
Радиальный градиент

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <radialGradient id="G1"> <stop offset="0%" stop-color="yellow"/> <stop offset="100%" stop-color="blue"/> </radialGradient> <rect height="150" width="150" x="0" y="0" fill="url(#G1)"/> <radialGradient id="G2" cx="50%" cy="50%" r="20%"> <stop offset="0%" stop-color="yellow" stop-opacity=".4"/> <stop offset="100%" stop-color="blue"/> </radialGradient> <rect height="150" width="150" x="0" y="160" fill="url(#G2)"/> <radialGradient id="G3" cx="25%" cy="50%" r="100%"> <stop offset="0%" stop-color="yellow"/> <stop offset="100%" stop-color="blue"/> </radialGradient> <rect height="150" width="150" x="0" y="320" fill="url(#G3)"/> </svg>	

В случае когда для заливки используется объект с отличающимися размерами по горизонтали и вертикали, радиальный градиент «подстраивается» под них. Пример показан в табл. 2.8.

Таблица 2.8

Радиальный градиент на прямоугольном объекте

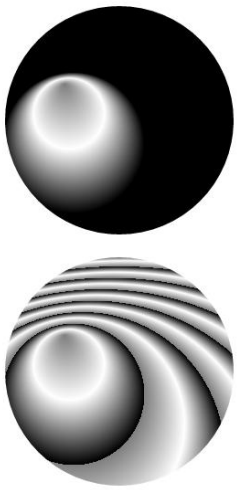
Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <radialGradient id="Rad1"> <stop offset="0" stop-color="white"/> <stop offset=".25" stop-color="black"/> <stop offset=".75" stop-color="white"/> <stop offset="1" stop-color="black"/> </radialGradient> <rect x="30" y="60" width="200" height="60" fill="url(#Rad1)"/> </svg>	

В арсенале средств адаптации градиента под геометрические размеры фигуры есть также атрибуты *fx* и *fy*. В частности, они позволяют получить эффект светового блика. Пример использования атрибутов *fx ="20%"* и *fy ="25%"* приведен в самом начале раздела, в табл. 2.1. Для радиального

градиента также можно использовать атрибут *SpreadMethod*. Пример эффекта от использования этого атрибута в сочетании с атрибутами *fx* и *fy* приведен в табл. 2.9.

Таблица 2.9

Эффект от применения атрибута *SpreadMethod="repeat"*

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <radialGradient id="gr1" cx="30%" cy="60%" r="31%" fx="26%" fy="34%"> <stop offset="0" stop-color="grey"/> <stop offset="0.5" stop-color="white"/> <stop offset="1" stop-color="black"/> </radialGradient> <circle cx="200" cy="200" r="100" fill="url(#gr1)"/> <radialGradient id="gr2" cx="30%" cy="60%" r="31%" fx="26%" fy="34%" spreadMethod="repeat"> <stop offset="0" stop-color="grey"/> <stop offset="0.5" stop-color="white"/> <stop offset="1" stop-color="black"/> </radialGradient> <circle cx="200" cy="420" r="100" fill="url(#gr2)"/> </svg>	

Попробуйте изменять значения различных атрибутов, в частности *fx* и *fy*, наблюдая и объясняя производимые этими изменениями эффекты.

В заключение напомним, что как и в случае линейного градиента, для радиального градиента дополнительно допускается управление прозрачностью при помощи атрибута *stop-opacity*.

2.1.3. *Узор <pattern>*

Тег *<pattern>* предназначен для создания узора в виде графической текстуры, которая может применяться в качестве заливки любого объекта или его контура. Определение тега включает в себя графическое содержимое, которое будет представлять собой узор, и использует следующие атрибуты:

x, y – координаты верхнего левого угла узора;

width, height – ширина и высота площади, которую занимает узор;

patternUnits – координатная система для атрибутов *x, y, width* и *height.*

Для *patternUnits* возможны два значения:

- *userSpaceOnUse* (атрибуты *x, y, width* и *height* определяются в системе координат самого узора);
- *objectBoundingBox* (атрибуты *x, y, width* и *height* определяются в системе координат того объекта, для которого будет применяться узор).

Если параметр *patternUnits* не указан, то по умолчанию принимается значение *objectBoundingBox.*

Узор, созданный при помощи тега <pattern>, не может быть отображен в документе непосредственно. Для его включения обязательно требуется объект с установленными свойствами *stroke* или *fill.*

В табл. 2.10 приведен пример формирования простейшего узора в виде зеленого треугольника. При формировании самого узора ему был присвоен идентификатор *id="fir"*. Объектом для включения узора выбран круг с указанием в качестве значения атрибута *fill* идентификатора узора *fill="url(#fir)"*.

Таблица 2.10

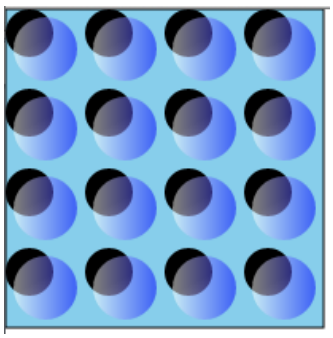
Использование тега *pattern*

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <pattern id="fir" width="10" height="10" patternUnits="userSpaceOnUse"> <polygon points="5,0 10,10 0,10" fill="green"/> </pattern> <circle cx="60" cy="60" r="50" fill="url(#fir)"/> </svg>	

Рассмотрим еще один пример узора, приведенный в табл. 2.11. Здесь использованы дополнительно градиент *Gr1* и атрибут *fill-opacity.*

Таблица 2.11

Использование тега *pattern*

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <linearGradient id="Gr1"> <stop offset="5%" stop-color="white"/> <stop offset="95%" stop-color="blue"/> </linearGradient> <pattern id="Pattern" x="0" y="0" width=".25" height=".25"> <rect x="0" y="0" width="50" height="50" fill="skyblue"/> <circle cx="15" cy="15" r="15" fill="black"/> <circle cx="25" cy="25" r="20" fill="url(#Gr1)" fill-opacity="0.5"/> </pattern> <rect fill="url(#Pattern)" stroke="black" x="0" y="0" width="200" height="200"/> </svg>	

Поскольку здесь атрибут *patternUnits* не указан, то он принимается равным по умолчанию: *patternUnits="objectBoundingBox*". В этом случае атрибуты *x, y, width* и *height* определяются в системе координат того объ-

екта, для которого будет применяться узор. Объект имеет размеры 200×200, узор 50×50, поэтому, для того чтобы узор умещался 4×4 раза, следует установить атрибуты *width="0.25"* и *height="0.25"*. Для лучшего понимания влияния атрибута *patternUnits* установите его значение *patternUnits= "userSpaceOnUse"* и подберите значения *width* и *height* так, чтобы рисунок *SVG* не изменился.

2.1.4. *Атрибуты gradientTransform и patternTransform*

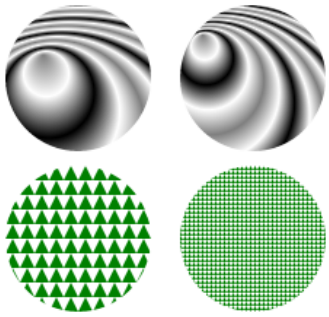
Атрибуты *gradientTransform* и *patternTransform* позволяют изменять свойства, заданные для градиента и узора и получать новые композиции. В примерах табл. 2.12 показано использование обоих атрибутов.

При использовании атрибута *gradientTransform* создается новый градиент, которому присваивается идентификатор gr2, и трансформации подвергается масштаб градиента gr1.

При использовании атрибута *patternTransform* создается новый идентификатор fir2, и трансформации подвергается масштаб градиента fir1.

Таблица 2.12

Использование атрибутов *gradientTransform* и *patternTransform*

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <radialGradient id="gr1" cx="30%" cy="60%" r="31%" fx="26%" fy="34%" spreadMethod="reflect"> <stop offset="0" stop-color="grey"/> <stop offset="0.5" stop-color="white"/> <stop offset="1" stop-color="black"/> </radialGradient> <radialGradient id="gr2" gradientTransform="scale(.6)" xlink:href="#gr1"/> <circle cx="60" cy="60" r="50" fill="url(#gr1)"/> <circle cx="180" cy="60" r="50" fill="url(#gr2)"/> <pattern id="fir1" x="0" y="0" width="10" height="10" patternUnits="userSpaceOnUse"> <polygon points="5,0 10,10 0,10" fill="green"/> </pattern> <pattern id="fir2" patternTransform="scale(0.3)" xlink:href="#fir1"/> <circle cx="60" cy="170" r="50" fill="url(#fir1)"/> <circle cx="180" cy="170" r="50" fill="url(#fir2)"/> </svg>	

Для демонстрации эффекта от произведенной трансформации градиента и узора в табл. 2.12 приведены результаты применения исходных и трансформированных изображений.

2.2. Вырезание и маскирование

Иногда требуется выделить только часть изображения, скрывая остальные детали. Либо продемонстрировать объект, показывая его через полупрозрачный занавес. SVG позволяет осуществлять такие эффекты с помощью тегов <*clipPath*> и <*mask*>.

2.2.1. *Вырезание <clipPath>*

Тег <*clipPath*> предназначен для отсекания заданной областью части изображения и используется для определения фигуры, которая будет ограничивать отсекаемую часть изображения. Возьмем в качестве исходного объекта изображение рис. 2.2, а в качестве фигуры, ограничивающей отсекаемую часть, – круг (табл. 2.13). Верхний рисунок в табл. 2.13 показывает полученный результат.

Рис. 2.2. Исходное изображение bird.png

Таблица 2.13

Примеры использования тега <*clipPath*>

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <clipPath id="c1"> <circle cx="275" cy="110" r="80"> </circle> </clipPath> <image x="10" y="10" width="550" height="400" clip-path="url(#c1)" xlink:href="bird.png"/> <clipPath id="c2"> <circle cx="220" cy="110" r="80"> </circle> <circle cx="320" cy="110" r="80"> </circle> </clipPath> <g transform="translate(10,200)"> <image x="10" y="10" width="550" height="400" clip-path="url(#c2)" xlink:href="bird.png"/> </g> </svg>	 

Выберем в качестве фигуры, ограничивающей отсекаемую часть – два круга (табл. 2.13). Нижний рисунок в табл. 2.13 показывает полученный результат.

2.2.2. *Маскирование <mask>*

При работе с тегом <*clipPath*> игнорируется внутреннее оформление фигуры, используемой для вырезания. Тег <*mask*> является инструментом, который поддерживает все свойства элемента *clipPath* с учетом графической структуры заполнителя.

Таблица 2.14

Примеры использования тега <*clipPath*>

Код *SVG*	Рисунок *SVG*
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <radialGradient id="Grad"> <stop offset="80%" stop-color="white" stop-opacity="1"/> <stop offset="100%" stop-color="white" stop-opacity="0"/> </radialGradient> <mask id="Circ"> <circle cx="275" cy="110" r="80" fill="url(#Grad)"/> </mask> <image x="10" y="10" width="550" height="400" mask="url(#Circ)" xlink:href="bird.png"/> </svg>	

В табл. 2.14 приведен пример применения в качестве заполнителя маски радиальной градиентной заливки *<radialGradient id="Grad">*. Уменьшением размера градиентной области *<stop offset="80%" stop-color="white" stop-opacity="1"/>* достигается увеличение площади открытой поверхности. Попробуйте заменить *stop offset="80%"* на *stop offset="0%"* и посмотрите на получившийся эффект.

2.3. Фильтры

SVG-фильтры – очень мощный и вместе с тем достаточно сложный инструмент для получения различных визуальных эффектов.

Для описания любого фильтра используется тег <*filter*>, причем каждому фильтру необходимо задать уникальный идентификатор *id*. Без применения к конкретному объекту *определение* фильтра не отображается в документе.

После того как фильтр определен, его можно применять к SVG-объектам, ссылаясь на его идентификатор.

В спецификации консорциума *W3C* [2] определено 16 базовых типов фильтрации с большим количеством возможных атрибутов для настройки их параметров, а также 3 типа фильтров с эффектом источника света. В табл. 2.15 приведены теги фильтрации и дано краткое описание их назначения.

Таблица 2.15

Типы *SVG*-фильтров и источников освещения

Тег (тип фильтра)	Назначение
feBlend	Смешивание изображений
feColorMatrix	Коррекция цвета и яркости изображения
feComponentTransfer	Коррекция контраста и цветового баланса
feComposite	Арифметические и логические операции над слоями
feConvolveMatrix	Сверточные операции
feDiffuseLighting	Освещение изображения с использованием альфа-канала
feDisplacementMap	Смещение пикселов в изображении
feFlood	Тонирование изображений
feGaussainBlur	Операция размытия изображения
feImage	Позиционирование изображения
feMerge	Совмещение нескольких фильтров
feMorphology	Художественная обработка изображений
feOffset	Смещение исходного изображения
feSpecularLighting	Добавление источника света к изображению
feTile	Формирование текстуры из изображения
feTurbulence	Синтез текстур с использованием шума Перлина
Фильтры – источники света	
fePointLight	Точечное освещение изображения
feDistantLight	Дистанционное освещение изображения
feSpotLight	Спот-освещение изображения

Наименования всех тегов фильтрации имеет приставку *fe* – от *filter effect*. Далее будут рассмотрены примеры использования некоторых фильтров.

2.3.1. *Фильтр feGaussianBlur*

Этот фильтр выполняет гауссово размытие входного изображения. Ядро размытия является приближением нормированной свертки: $G(x,y) = H(x)I(y)$, где $H(x) = exp(–x^2/ (2s^2)) / sqrt(2* pi*s^2)$ и $I(y) = exp(–y^2/ (2t^2)) / sqrt(2* pi*t^2)$. Переменные *s* и *t* – стандартные отклонения по оси *X* и *Y* соответственно, задаются аттрибутом *stdDeviation*: чем больше отклонение, тем изображение более размытое.

Таблица 2.16

Результат фильтрации <*feGaussianBlur*>

Код SVG

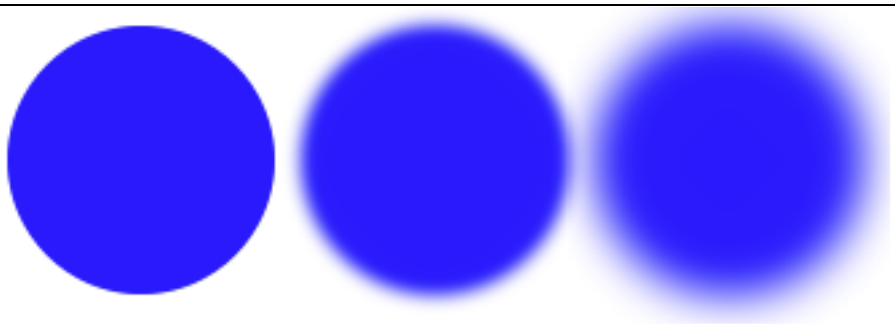
```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">

 <circle cx="60"  cy="60" r="50" fill="blue" />

 <filter id="fGB1">
   <feGaussianBlur in="SourceGraphic" stdDeviation="3" />
 </filter>
 <circle cx="170" cy="60" r="50" fill="blue" filter="url(#fGB1)" />

 <filter id="fGB2">
   <feGaussianBlur in="SourceGraphic" stdDeviation="9" />
 </filter>
 <circle cx="280" cy="60" r="50" fill="blue" filter="url(#fGB2)" />
</svg>
```

Рисунок SVG



В примере *SVG*-кода, приведенном в табл. 2.16, использованы два фильтра <*feGaussianBlur*> *fGB1* и *fGB2* со значениями *stdDeviation=3* и *stdDeviation=9* для размытия изображения круга.

2.3.2. *Фильтр feImage*

Фильтр <*feImage*> используется для позиционирования изображения, загружаемого из внешнего файла. В табл. 2.17 приведен пример использования тега <*feImage*>. В фильтре с *id="Default"*> не использованы атрибуты для его настройки. В фильтре с *id="Fitted"* задано значение атрибута *preserveAspectRatio="none"* для формирования пропорциональных размеров изображения в пределах заданного пространства. В фильтре *id="Shifted"* в качестве атрибутов заданы параметры смещения загружаемого изображения.

Если во втором фильтре с *id="Fitted"* исключить атрибут *preserveAspectRatio="none",* то изображение полностью займет предоставляемое пространство, но исказится в пропорциях.

Таблица 2.17

Результаты использования фильтра <*feImage*>

Код SVG

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">

 <defs>
  <filter id="Default">
   <feImage xlink:href="smily.jpg"/>
  </filter>
  <filter id="Fitted" primitiveUnits="objectBoundingBox" >
   <feImage xlink:href="smily.jpg" x="0" y="0" width="100%" height="100%" preserveAspectRatio="none">/>
  </filter>
  <filter id="Shifted">
   <feImage xlink:href="smily.jpg" x="500" y="5"/>
  </filter>
 </defs>

 <g>
  <rect x="50"  y="25" width="100" height="200" filter="url(#Default)"/>
  <rect x="50"  y="25" width="100" height="200" fill="none" stroke="green"/>
  <rect x="250" y="25" width="100" height="200" filter="url(#Fitted)"/>
  <rect x="250" y="25" width="100" height="200" fill="none" stroke="green"/>
  <rect x="450" y="25" width="100" height="200" filter="url(#Shifted)"/>
  <rect x="450" y="25" width="100" height="200" fill="none" stroke="green"/>
 </g>
</svg>
```

Рисунок SVG

2.3.3. *Фильтр feConvoleMatrix*

Фильтр <*feConvolveMatrix*> вычисляет новое значение пиксела, используя значения соседних пикселов. Представьте себе некоторый пиксел *P*, окруженный девятью соседними пикселами:

A B C
D ***P*** *E*
F G H

Новое значение каждого пиксела отфильтрованного изображения будет формироваться с использованием значений элементов матрицы, задаваемой атрибутом *kernelMatrix*. Допустим этот атрибут определяет следующую матрицу:

0 1 2
3 4 5
6 7 8

Тогда определим новое значение пиксела *P* как:

*P' = ((0*A) + (1*B) + (2*C) + (3*D) + (4*P) + (5*E) + (6*F) +*
*+ (7*G) + (8*H)) / (0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8).*

В качестве примера использования фильтра <*feConvolveMatrix*> с разными атрибутами *kernelMatrix* рассмотрим *SVG*-код из табл. 2.18, с помощью которого осуществлялась обработка оригинального изображения рис. 2.3, заимствованного из [9].

Рис. 2.3. Оригинальное изображение
girl.png

Первые три фильтра используют матрицу *kernelMatrix* третьего порядка (в примере они записаны одной строкой), а четвертый фильтр-матрицу 5×5. Как видно из примера, фильтр *#fCM1* несколько заостряет исходное изображение, а *#fCM2* делает его еще более заостренным. Фильтр *#fCM3* добавляет обводку, а фильтр *#fCM4* делает изображение полностью состоящим из штрихов с обводкой.

Таблица 2.18

Использование фильтра <*feConvolveMatrix*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <filter id="fCM1"> <feConvolveMatrix order="3" kernelMatrix="1 -1 1 -1 -1 -1 1 -1 1"/> </filter> <image filter="url(#fCM1)" xlink:href="girl.png" width="30%" height="30% />	
<filter id="fCM2"> <feConvolveMatrix order="3" kernelMatrix="1 -1 1 -1 -.1 -1 1 -1 1 "/> </filter> <image filter="url(#fCM2)" xlink:href="girl.png" width="30%" height="30%" />	
<filter id="fCM3"> <feConvolveMatrix order="3" kernelMatrix="-1 -1 -1 -1 7 -1 -1 -1 -1"/> </filter> <image filter="url(#fCM3)" xlink:href="girl.png" width="30%" height="30%" />	
<filter id="fCM4"> <feConvolveMatrix order="5" kernelMatrix="1 1 1 1 1 1 -2 -2 -2 1 1 -2 .01 -2 1 1 -2 -2 -2 1 1 1 1 1 1"/> </filter> <image filter="url(#fCM4)" xlink:href="girl.png" width="30%" height="30%" /> </svg>	

2.3.4. *Фильтр feDiffuseLighting*

Фильтр <*feDiffuseLighting*> используется для освещения изображения с помощью альфа-канала в качестве рельефной карты. Получаемое RGBA-изображение зависит от цвета и расположения источника света и рельефной карты. Карта света, формируемая одним подобным фильтром, может объединяться с другими фильтрами, в частности с фильтром <*feComposite*>.

Для примера рассмотрим использование этого элемента в совокупности со всеми тремя источниками освещения: точечным <*fePointLight*>, дистанционным <*feDistantLight*> и в виде спота <*feSpotLight*>. В табл. 2.19 приведены эффекты фильтрации для исходного изображения квадрата золотистого цвета.

В табл. 2.19 определены три фильтра с идентификаторами *PointLight*, *DistanceLight* и *SpotLight*. Отметим, что точечный свет расположен точно в левом верхнем углу квадрата на «высоте» z = "20", определенном фильтром

PointLight. Дистанционный свет, определенный фильтром *DistanceLight*, расположен по азимуту (*azimuth*) 65° с углом возвышения (*elevation*) 95°. Угол конуса светового луча (*limitingConeAngle*) от спота в фильтре *SpotLight* ограничен 10°.

Таблица 2.19

Эффекты фильтрации с различными источниками освещения

Код SVG

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">

 <rect x="60" y="80" width="100" height="100" fill="gold" />

 <filter id="PointLight">
  <feDiffuseLighting in="SourceGraphic" result="light" lighting-color="white">
   <fePointLight x="200" y="120" z="20" />
  </feDiffuseLighting>
  <feComposite in="SourceGraphic" in2="light" operator="arithmetic" k1="1" k2="0" k3="0" k4="0"/>
 </filter>
 <rect x="170" y="80" width="100" height="100"  fill="gold" filter="url(#PointLight)" />

 <filter id="DistantLight">
  <feDiffuseLighting in="SourceGraphic" result="light" lighting-color="white">
   <feDistantLight azimuth="65" elevation="95"/>
  </feDiffuseLighting>
  <feComposite in="SourceGraphic" in2="light" operator="arithmetic" k1="1" k2="0" k3="0" k4="0"/>
 </filter>
 <rect x="280" y="80" width="100" height="100"  fill="gold" filter="url(#DistantLight)" />


 <filter id="SpotLight">
  <feDiffuseLighting in="SourceGraphic" result="light" lighting-color="white">
   <feSpotLight x="400" y="25" z="30" limitingConeAngle="10"
            pointsAtX="440" pointsAtY="130" pointsAtZ="0"/>
  </feDiffuseLighting>
  <feComposite in="SourceGraphic" in2="light" operator="arithmetic" k1="1" k2="0" k3="0" k4="0"/>
 </filter>
 <rect x="390" y="80" width="100" height="100"  fill="gold" filter="url(#SpotLight)" />
</svg>
```

Рисунок SVG

2.3.5. *Фильтр feDisplacementMap*

Фильтр <*feDisplacementMap*> использует значения пикселов из изображения *in2* для частичной замены значений пикселов в изображении *in*. Следующая формула [2] описывает подобное преобразование:

$$P'(x,y) \leftarrow P(\ x + scale * (XC(x,y) - 0.5),\ y + scale * (YC(x,y) - 0.5)),$$

где $P(x,y)$ – исходное изображение *in*; $P'(x,y)$ – результирующее изображение; $XC(x, y)$ и $YC(x, y)$ – значения компонентов канала из *RGBA*, назначенного атрибутами *xChannelSelector* и *yChannelSelector*. Коэффициент смещения задается атрибутом *scale*.

Таблица 2.20

Результат фильтрации <*feDisplacementMap*> и <*feTurbulence*>

Код SVG	Рисунок SVG
<svg xmlns=http://www.w3.org/2000/svg xmlns:xlink="http://www.w3.org/1999/xlink">> <ellipse cx="80" cy="50" rx="40" ry="30" fill="grey" /> <filter id="DisplacementMap"> <feTurbulence type="turbulence" baseFrequency="0.1" numOctaves="5" result="turbulence"/> <feDisplacementMap in2="turbulence" in="SourceGraphic" scale="50" xChannelSelector="R" yChannelSelector="B"/> </filter> <ellipse cx="80" cy="130" rx="40" ry="30" fill="grey" filter="url(#DisplacementMap)"/> </svg>	

В табл. 2.20 приведен пример SVG-кода, в котором использованы два фильтра. Фильтр <*feTurbulence*> использован для формирования изображения in2, в то время как в качестве изображения in используется эллипс. Фильтр <*feTurbulence*> создает изображение с использованием функции Перлина. Это позволяет синтезировать такие искусственные текстуры, как облака, дым, огонь, мрамор и т. п.

2.3.6. *Фильтр feColorMatrix*

Фильтр <*feColorMatrix*> позволяет осуществлять коррекцию цвета и яркости изображения. Эта операция выполняется в виде матричного преобразования, в котором каждый пиксел исходного изображения умножается на матрицу *M* размерности 5×5. Пиксел *P* исходного изображения представляется в виде вектора, образованного из значений $P = R, G, B, A, 1$ (единица дополняет вектор до размерности 5). Результирующий пиксел P' получается в виде $P' = M*P$.

Фильтр <*feColorMatrix*> обычно применяется к растровым изображениям, импортированным в SVG-документ. Поэтому для приводимых ниже примеров особенностей применения этого фильтра будем использовать изображение рис. 2.4 в формате *jpg*.

Рис. 2.4. Оригинальное изображение *b1.jpg*,
подлежащее фильтрации

В табл. 2.21 приведен пример двух фильтров, первый из которых создает эффект под названием «сепия», а второй, корректируя цвета, повышает яркость оригинального изображения.

Таблица 2.21

Результаты фильтрации <*feColorMatrx*>

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <filter id="fCM1"> <feColorMatrix type="matrix" values="0.343 0.669 0.119 0 0 0.249 0.626 0.130 0 0 0.172 0.334 0.111 0 0 0 0 0 1 0" > </feColorMatrix> </filter> <image filter="url(#fCM1)" xlink:href="b1.jpg" width="30%" height="30%" x="10" y="0" /> <filter id="fCM2"> <feColorMatrix type="matrix" values="0.788 -0.262 0.474 0 0 0.1 1.032 -0.132 -0.1 0 -1 2.1 0.912 -2.8 0 0 0 0 1 0" > </feColorMatrix> </filter> <image filter="url(#fCM2)" xlink:href="b1.jpg" width="30%" height="30%" x="10" y="300" /> </svg>	 

В примерах табл. 2.22 используется стандартная матрица типа *type="saturate"*, с помощью которой оригинальная цветность увеличена в 5 раз за счет атрибута *values="5"*, и изображение полностью обесцвечено за счет атрибута *values="0"*.

Таблица 2.22

Результаты фильтрации <*feColorMatrx*>

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <filter id="fCM1"> <feColorMatrix type="saturate" values="5" /> </filter> <image filter="url(#fCM1)" xlink:href="b1.jpg" width="30%" height="30%" x="10" y="0" /> <filter id="fCM2"> <feColorMatrix type="saturate" values="0" /> </filter> <image filter="url(#fCM2)" xlink:href="b1.jpg" width="30%" height="30%" x="10" y="300" /> </svg>	

В примере табл. 2.23 используется также стандартная матрица типа *type="hueRotate"*, которая осуществляет поворот цветов оригинального изображения по цветовому кругу на число градусов, определяемое атрибутом *values="40"* для первого фильтра и *values="120"* для второго.

Таблица 2.23

Результаты фильтрации <*feColorMatrx*>

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <filter id="fCM1"> <feColorMatrix type="hueRotate" values="40"/> </filter> <image filter="url(#fCM1)" xlink:href="b1.jpg" width="30%" height="30%" x="10" y="0" /> <filter id="fCM2"> <feColorMatrix type="hueRotate" values="120"/> </filter> <image filter="url(#fCM2)" xlink:href="b1.jpg" width="30%" height="30%" x="10" y="300" /> </svg>	

2.3.7. *Комплексная фильтрация*

В предыдущих разделах рассматривались в основном эффекты от применения отдельных тегов, определяющих тип фильтрации. Одним неотъемлемым достоинством SVG-фильтров является возможность сочетания нескольких типов фильтров. То есть результат, полученный после применения одного фильтра, может являться источником изображения для другого фильтра и т. д. На практике используются два отличающихся друг от друга подхода к комплексной фильтрации.

Первый подход служит для определения последовательной фильтрации изображений. В этом случае можно использовать одну из эквивалентных схем построения последовательного мульти-фильтра представленных в табл. 2.24.

Второй подход предполагает использование фильтра *feMerge*. Фильтр *feMerge* позволяет использовать комбинацию фильтров одновременно, а не последовательно. Вместо того чтобы каждый фильтр применялся к выходному результату предыдущего фильтра, *feMerge* дает нам возможность временного хранения выхода каждого фильтра. После того как несколько

слоев созданы и сохранены как результаты различных фильтров, они могут быть совмещены на холсте, в порядке от нижнего к верхнему. Более верхние слои должны иметь некоторую прозрачность в области заполнения, с тем чтобы слои, размещенные под ними, были видны. Более подробно вопросы комплексной фильтрации рассмотрены в [8].

Таблица 2.24

Схемы построения последовательного мультифильтра

Схема 1	Схема 2
<filter id="Fs"> <FP1/> <FP2/> <FP3/> <FP4/> <FP5/> </filter>	<filter id="Fs"> <FP1 in="SourceGraphic" result="A"/> <FP2 in="A" result ="B" /> <FP3 in="B" result ="C" /> <FP4 in="C" result ="D" /> <FP5 in="D" /> </filter>
FPx обозначает один из типов фильтров (*feGaussianBlur, feColorMatrix* и т. д.)	

3. ДИНАМИЧЕСКИЕ ВИЗУАЛЬНЫЕ ЭФФЕКТЫ

Анимация – важный аспект *SVG*-графики. Она может быть реализована как с помощью специальных анимационных тегов, так и с использованием *JavaScript*. Вопросы взаимодействия *SVG*-графики и сценариев *JavaScript* будут рассмотрены в разд. 4, а здесь сосредоточимся на знакомстве со встроенными средствами анимации.

В зависимости от цели анимации применяется один из четырех тегов:

1) <*animate*> – позволяет анимировать скалярные атрибуты в течение определенного периода времени;

2) <*set*> – является удобной заменой <*animate*> для задания нечисловых атрибутов наподобие свойства *visibility*;

3) <*animateTransform*> – позволяет анимировать один из атрибутов трансформации SVG, например атрибут transform;

4) <*animateMotion*> – позволяет передвигать элемент по заданной траектории.

К этому следует добавить тег <*mpath*>, который используется в сочетании с тегом <*animateMotion*> для указания траектории движения. Элемент *mpath* ставится внутрь элемента *animateMotion*, перед закрывающим тегом.

Кроме того, для тега <*animateMotion*> важными атрибутами являются:

- *path* – позволяет использовать все возможности синтаксиса для SVG-контуров;
- *keypoints* – используется для точного управления скоростью движения;
- *rotate* – используется для управления автоматическим поворотом объекта так, чтобы его ось указывала в ту же (или противоположную) сторону, что и касательный вектор направления траектории движения.

Наряду с отмеченными выше основными тегами и атрибутами для анимации могут быть использованы свойства взаимодействия системной и пользовательской областей просмотра, viewport и wiewBox. На их базе можно реализовать вертикальную и горизонтальную прокрутку изображения, так называемый параллакс, а также пропорционально увеличивать и уменьшать отдельные фрагменты изображения.

Перед тем как перейти к деталям использования встроенных средств анимации, подчеркнем их важную особенность. Анимации, реализованные на основе перечисленных выше возможностей, работают даже тогда, когда *SVG*-графика встроена в *HTML* страницу в виде *img* или использована как *background-image* в *CSS* – это существенное преимущество перед сценариями *JavaScript*.

Для того чтобы использовать любой из четырех анимационных тегов, прежде всего нужно указать целевой объект, для которого этот тег задает анимацию. Чтобы указать этот объект, можно использовать атрибут

xlink:href. Этому атрибуту необходимо задать значение в виде *URI*-ссылки на элемент, который будет подвергаться анимации:

```
<circle id="circ1" . . . />
<animation xlink:href="#circ1". . . />.
```

Можно также не задавать атрибут *xlink:href,* а определить анимацию непосредственно в целевом объекте:

```
<circle id="circ1" . . . />
<animation . . . />
</circle>.
```

У всех анимационных тегов может быть еще один общий атрибут: *attributeName* для указания имени атрибута, подлежащего анимации. Например, если необходимо анимировать положение центра <*circle*> на оси *X*, можно задать параметр *cx* в качестве значения атрибута *attributeName*.

Атрибут *attributeName* может принимать лишь одно значение, а не список значений, таким образом, можно анимировать только по одному атрибуту за один раз. Если необходимо анимировать более одного атрибута, то нужно определить более одной анимации для одного целевого объекта.

Указывая имя атрибута, можно добавить *XMLNS*-префикс, чтобы обозначить пространство имен для атрибута. Пространство имен можно указать с помощью атрибута *attributeType*. Например, некоторые атрибуты входят в пространство имен *CSS*, а другие есть только в *XML*.

Атрибут *attributeType* может принимать три значения:

- *CSS* – анимация свойств, относящихся к спецификации *CSS* (шрифт, цвет шрифта и др.);
- *XML* – анимация свойства, относящегося к *SVG*-графике (перенос, вращение, искажение и др.);
- *auto* – значение по умолчанию, включающее в себя значения свойств *CSS* и *XML*.

Если значение *attributeType* не задано явно или равно *auto*, браузер должен сначала поискать соответствующее имя свойства в списке *CSS*-свойств и, если ничего не найдено, затем поискать в пространстве имен *XML* по умолчанию для данного элемента.

Например, следующий код анимирует «прозрачность» круга. Поскольку атрибут *opacity* доступен как *CSS*-свойство, значение *attributeType* указывает на пространство имен *CSS*:

```
<circle>
<animate attributeType="CSS" attributeName="opacity"
   from="1" to="0" dur="5s" repeatCount="indefinite" />
</circle>.
```

Перейдем к другим анимационным атрибутам более подробно для каждого типа тега и различных целей анимации.

3.1. Анимация атрибутов <*animate*>

Тег <*animate*> – позволяет анимировать скалярные атрибуты в течение определенного периода времени. Для создания анимации следует определить свойства объекта, которые будут изменяться, их результирующий вид, а также начало и продолжительность всех изменений.

В табл. 3.1 дано краткое описание атрибутов анимации – это общие атрибуты для всех элементов анимации.

Таблица 3.1

Атрибуты анимации

Атрибут	Описание
begin	Задание времени старта анимации
dur	Продолжительность анимации
end	Задание времени окончания
restart	Возможность повторного воспроизведения
repeatCount	Число повторений анимации
repeatDur	Задание времени для возможности повторного воспроизведения
fill	Определение поведения анимируемого элемента по окончании анимации
min	Минимальный период анимации
max	Максимальный период анимации
from	Начальное значение анимируемого свойства (цвет, координата, размер)
to	Конечное значение анимируемого свойства
by	Указание относительного сдвига анимации
values	Промежуточные значения анимируемого свойства
keyTimes	Указание ключевых кадров
calcMode	Задание темпа анимации
keySplines	Задание темпа анимации каждого ключевого кадра
additive	Задание способа отсчета значений атрибутов from и to
accumulate	Задание накопительной анимации

Описание анимации представляет собой последовательную *запись* пар «*атрибут – значение* атрибута». Для создания заданной анимации, например для перемещения объекта, следует указать необходимый набор, без которого *анимация* просто не будет воспроизводиться. Однако кроме необходимого набора можно указывать дополнительные пары атрибутов, описывающих анимацию детальным образом или придавая ей ряд новых свойств.

Перейдем к последовательному описанию атрибутов анимации и их возможных значений.

3.1.1. *Атрибуты from, to, by, dur, begin, fill* и *restart*

Приведем пример перемещения круга из одного положения в другое. Будем делать это, меняя значение его атрибута *cx* (который указывает координату *X* его центра).

Чтобы изменить значение с одного на другое за период времени, используются атрибуты *from, by, to* и *dur*. В дополнение к ним можно, например, указать, когда анимация должна начаться с помощью атрибута *begin*:

Листинг 3.1. Перемещение круга вдоль оси X:

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
<circle id="circ1" r="30" cx="50" cy="50" fill="gold"/>
<animate        xlink:href="#circ1"
            attributeName="cx"
            from="50"
            to="450"
            dur="2s"
            begin="click"
            fill="freeze" />
</svg>
```

В листинге 3.1 определен круг и затем описана анимация для этого круга. Центр круга перемещается с начальной позиции в 50 единиц до 450 единиц вдоль оси *X*. Атрибуту *begin* задано значение *click*. Это означает, что круг начнет двигаться только при наведении на него курсора мыши и нажатии клавиши. Можно также задать ему некоторое значение времени, например: *begin="2s"* запустит анимацию через две секунды после загрузки.

Интересная особенность *begin* заключается в том, что можно задавать значение типа *begin="click + 5s"*, чтобы запустить анимацию через 5 секунд после нажатия клавиши мыши.

Более того, можно использовать другие значения, позволяющие синхронизировать анимации без необходимости рассчитывать их длительности и задержки. Атрибут *begin* может принимать в качестве значения идентификатор *id* другой анимации, за которым следует ссылка на событие. Если есть две (и более) анимаций (причем неважно, применяются ли они к одному и тому же свойству объекта или нет) и необходимо синхронизировать их так, чтобы одна из них запускалась в зависимости от другой, то можно сделать это, даже не зная длительности второй анимации.

Например, в листинге 3.2 квадрат начинает двигаться через 2 секунды после начала анимации круга. Это достигается присвоением каждой анимации своего идентификатора с последующим использованием этого идентификатора *id="circ-anim"* с событием начала движения *begin ="circ-anim.begin + 2s"*.

Листинг 3.2. Задержка анимации квадрата на 2 секунды:

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
<circle id="circ1" r="30" cx="50" cy="50" fill="red" />
<rect id="rect1" width="50" height="50" x="25" y="200" fill="blue"></rect>

 <animate
  xlink:href="#circ1"
  attributeName="cx"
  from="50"
  to="450"
  dur="5s"
  begin="click"
  fill="freeze"
  id="circ-anim" />

 <animate
  xlink:href="#rect1"
  attributeName="x"
  from="50"
  to="425"
  dur="5s"
  begin="circ-anim.begin + 2s"
  fill="freeze"
  id="rect-anim" />
</svg>
```

Отметим, что идентификатор *id="rect-anim"* может быть использован для следующей синхронизации.

Можно также начать анимацию квадрата после полного завершения движения круга, используя событие end: *begin="circ-anim.end"* или запустить ее в определенный момент до конца движения: *begin="circ-anim.end – 2s"*.

Атрибут *fill* (который неудачно получил одинаковое название с атрибутом *fill*, определяющим цвет заливки элемента) определяет, должен ли объект вернуться в начальное состояние после завершения анимации или нет. Этому атрибуту можно присвоить одно из значений:

- *freeze* – результат анимации определен как «застывший» в конечном значении активного периода анимации. Результат анимации остается «застывшим», пока документ открыт (или до перезапуска анимации);
- *remove* – результат анимации убирается (больше не применяется), когда активный период анимации завершился. Как только достигнут конец анимации, анимация больше не влияет на целевой объект (если ее не перезапустить).

Атрибут *by* служит для указания относительного сдвига анимации. С его помощью можно указать величину, на которую анимация должна продвинуться. Например, можно посмотреть эффект от использования *by* совместно с атрибутом *from*.

Бывает также полезно не дать анимации перезапуститься, пока она активна. Для этого служит атрибут *restart*. Ему можно задать одно из трех возможных значений:

- *always* – анимация может быть перезапущена в любой момент. Значение по умолчанию;
- *whenNotActive* – анимация может быть перезапущена, только когда неактивна (т.е. после завершения активности). Попытки перезапустить анимацию во время активного промежутка игнорируются;
- *never* – элемент не может быть перезапущен весь остаток простой длительности его родительского контейнера времени (в случае *SVG*, поскольку родительским контейнером времени является фрагмент *SVG* документа, анимация не может быть перезапущена, пока открыт документ).

В заключение раздела приведем один простой, но эффектный вариант использования <*animate*> атрибута *stroke-dasharray* (листинг 3.3).

Листинг 3.3. Анимация атрибута stroke-dasharray:

```
<svg xmlns="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink">
 <path class="path" fill="#FFFFFF" stroke="#000000"
                stroke-width="4" stroke-dasharray="0,1000"
          d="M 99 192
                C 137 160 204 133 141 124
                C 78 115 34 167 47 129
                C 60 91 20 65 77 71
                C 134 77 206 43 196 101
                C 186 159 118 368 119 299
                C 120 230 201 169 138 206
                C 75 243 53 231 99 192" >
     <animate attributeName="stroke-dasharray" from="0,1000" to="1000,0"
       begin="0"  dur="10s" repeatCount="indefinite" restart="whenNotActive"
fill="freeze"/>
   </path>
</svg>
```

В качестве анимируемого контура взят второй путь из табл. 1.1, к которому добавлен атрибут *stroke-dasharray="0,1000"*. Указанные значения атрибута говорят о том, что первоначально штрих-пунктирный контур будет иметь видимый штрих длиной 0, и разрыв длиной 1000 пикселов практически будет невидим. Постепенно, в течение *dur="10s"*, значения атрибута будут изменяться *from="0,1000" to="1000,0"*, будет расти длина

видимого штриха и уменьшаться разрыв. Все это будет происходить в пределах 1000 пикселов. Значение 1000 выбрано не случайно – это длина пути. Запустите пример листинга 3.3. Попробуйте изменять значения атрибута *stroke-dasharray* и объясните получаемый эффект.

Используйте *SVG*-коды листингов 3.1–3.3 для изучения описанных выше особенностей применения атрибутов и их возможных значений.

3.1.2. *Атрибуты repeatCount и repeatDur*

Если необходимо выполнить анимацию больше одного раза, можно использовать атрибут *repeatCount*. Он позволяет определить нужное количество повторений, либо использовать ключевое слово *indefinite*, чтобы анимация повторялась бесконечно. Так что, чтобы повторить анимацию круга дважды, в код листинга 3.2 достаточно добавить атрибут *repeatCount="2"*. Если для круга в листинге 3.2 добавить атрибут *repeatCount="2"*, а для квадрата – *repeatCount="indefinite"*, то круг остановится после двух повторов, а квадрат будет делать неограниченное число повторов анимации.

Бесконечное повторение анимации может раздражать или отталкивать пользователей, если анимация снова и снова возобновляется долгое время. Так что может иметь смысл ограничить возможность повторения анимации конечным периодом времени, и прекратить повторять ее через какое-то время после начала отображения документа. Это время известно как *время представления*, и оно указывается с помощью атрибута *repeatDur*. Синтаксис атрибута похож на часовое значение, но отсчитывается не относительно события другой анимации или действия пользователя, а от начала отображения документа.

Например, код листинга 3.3а остановит повторение анимации через 1 мин 10 с после появления документа.

Листинг 3.3а. Ограничение времени повторения анимации:

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
<circle id="circ1" r="30" cx="50" cy="50" fill="red" />
<animate
   xlink:href="#circ1"
   attributeName="cx"
   from="50"
   to="450"
   dur="2s"
   begin="0s"
   repeatCount="indefinite"
   repeatDur="01:10"
   fill="freeze"
   id="circ-anim" />
</svg>
```

Теперь рассмотрим вопрос синхронизации между двумя анимациями, зависящей от количества повторов. В *SVG* можно синхронизировать анимации так, что одна анимация запускается в зависимости от количества повторений другой. Например, можно запустить анимацию после любого по счету повторения другой анимации плюс (или минус) произвольное количество времени.

Листинг 3.4. Синхронизация анимации по числу повторений и времени:

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
 <circle cx="60" cy="60" r="15" fill="red">
  <animate
   id="circ-anim"
   attributeName="cx"
   attributeType="XML"
           begin="0s"
           dur="5s"
           repeatCount="3"
           from="60"
           to="260"
           fill="freeze"/>
 </circle>

 <rect x="230" y="80" width="60" height="30" fill="blue">
  <animate
   attributeName="x"
   attributeType="XML"
           begin="circ-anim.repeat(1)+2.5s"
           dur="5s"
           from="230"
           to="30"
           fill="freeze"/>
 </rect>
</svg>
```

Количество повторений задается с помощью параметра *repeat,* добавляемого к идентификатору повторяющегося процесса. Код листинга 3.4 содержит *begin="circ-anim.repeat(1)+2.5s"* , что запускает анимацию прямоугольника после одного повторения анимации круга через 2,5 секунды.

3.1.3. *Атрибуты keyTimes и values*

В случае когда темп анимации должен быть непостоянным, можно разбить весь временной интервал на отдельные, так называемые *ключевые кадры*. Для этой цели используется атрибут *keyTimes*, который формируется как список относительных интервалов времени, выделенного для каждого ключевого кадра.

Каждое значение в *keyTimes* указывается в виде вещественного числа в интервале от 0 до 1 включительно. Например, *keyTimes="0; 0.3; 0.5; 0.8; 1"* разобьет все время анимации на 4 ключевых кадра, для которых выделено 30, 20, 30 и 20 % общего времени анимации соответственно.

Для указания промежуточных значений анимируемого свойства служит атрибут *values*, который формируется также в виде списка. Для приведенного выше *keyTimes* атрибут *values* должен содержать пять значений, например *values="50, 400, 70, 300, 200"*.

Листинг 3.5. Разбиение времени анимации на ключевые кадры:

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
<circle id="circ1" r="30" cx="50" cy="50" fill="red" />
 <animate
   xlink:href="#circ1"
   attributeName="cx"
   dur="6s"
   begin="click"
   keyTimes="0; 0.3; 0.5; 0.8; 1"
   values="50; 400; 70; 300; 200 "
   fill="freeze"/>
</svg>
```

Следует отметить, что если задан список values, то любые значения атрибутов *from* и *to* игнорируются. В листинге 3.5 приведен пример изменения положения круга за промежуток времени 6 секунд, который делится на четыре ключевых кадра. Здесь, как и раньше, в качестве анимируемого свойства выступает атрибут *attributeName="cx"*. Поменяйте его на *attributeName="fill"*, а значения *values* на *values="blue; yellow; green; red; brown"* и посмотрите результат в браузере.

Листинг 3.6. Разбиение времени анимации на ключевые кадры:

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
<circle id="circ1" r="30" cx="50" cy="50" fill="red" />
<animate
   xlink:href="#circ1"
   attributeName="cy"
   dur="3s"
   begin="click"
   values="50; 250; 120;250; 170; 250; 210; 250"
   keyTimes="0; 0.15; 0.3; 0.45; 0.6; 0.75; 0.9; 1"
   fill="freeze"/>
</svg>
```

В листинге 3.6 приведен еще один пример *SVG*-кода, создающий эффект «прыгающего» шарика. Анимация начинается по нажатию клавиши мыши и останавливается, как только достигнет конечного значения. При повторном нажатии анимация возобновляется. Этот пример будет дополнен в следующем разделе при изучении атрибута *keySplines.*

3.1.4. *Атрибуты calcMode и keySplines*

Управление темпом анимации можно осуществлять с помощью произвольной функции плавности с помощью атрибута *calcMode*. По умолчанию темп линейный (*linear*) у всех анимационных элементов, кроме *animateMotion* (разд. 3.2). Помимо значения *linear*, можно задавать еще три значения:

discrete – указывает, что функция анимации перескакивает с одного значения на следующее без какой-либо интерполяции;

paced – похоже на *linear*, за исключением того, что оно игнорирует временные метки, заданные с помощью *keyTimes*. Оно рассчитывает расстояния между последовательными значениями и соответственно делит их на время. Если все значения находятся в линейном порядке, то разницы не будет заметно. Но если они смещаются взад-вперед, или если это цвета, то промежуточные значения будут заметны;

spline – строит интерполяцию от одного значения в списке *values* до следующего в соответствии с функцией от времени, заданной кубическим сплайном Безье. Временные границы для сплайна задаются в атрибуте *keyTimes*, а управляющие точки для каждого ключевого кадра задаются в атрибуте *keySplines*.

Для лучшего представления отличительных особенностей трех первых значений атрибута *calcMode*: *linear, discrete* и *paced* – можно воспользоваться SVG-кодом листинга 3.7.

Листинг 3.7. Управление темпом анимации linear, discrete и paced:

```
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <circle cx="50" cy="100" r="100" fill="red">
   <animate attributeName="cx" attributeType="XML"
        begin="0;click"
        values="50;500;300;400;50"
        dur="20s"
        calcMode="linear" />
   <animate attributeName="fill" attributeType="XML"
        begin="0;click"
        dur="20s"
        values="gray;darkred;red;cyan;lightcyan;green;purple;gray"
        calcMode="linear" />
```

```
  </circle>

  <circle cx="50" cy="300" r="100" fill="yellow">
   <animate attributeName="cx" attributeType="XML"
         begin="0;click"
         values="50;500;300;400;50"
         dur="20s"
         calcMode="paced"/>
   <animate attributeName="fill" attributeType="XML"
         begin="0;click"
         dur="20s"
         values="gray;darkred;red;cyan;lightcyan;green;purple;gray"
         calcMode="paced" />
  </circle>

  <circle cx="50" cy="500" r="100" fill="green">
   <animate attributeName="cx" attributeType="XML"
         begin="0;click"
         values="50;500;300;400;50"
         dur="20s"
         calcMode="discrete" />
   <animate attributeName="fill" attributeType="XML"
         begin="0;click"
         dur="20s"
         values="gray;darkred;red;cyan;lightcyan;green;purple;gray"
         calcMode="discrete" />
  </circle>
</svg>
```

В этом коде сформированы три круга с идентичными атрибутами анимации, за исключением одного атрибута *calcMode*. Для первого, верхнего круга *calcMode="linear"*, для второго – *calcMode="paced"*, для третьего, нижнего, – *calcMode="discrete"*. Анимация запускается одновременно для всех трех кругов при загрузке документа. Причем анимируются одновременно два атрибута – расположение центра на оси *X* и цвет. Это позволяет понять отличительные особенности трех указанных значений атрибута *calcMode*.

По завершении анимации круги будут окрашены в разные цвета – *red*, *yellow* и *green* – в соответствии со значением fill, заданным при их формировании. Кроме того, тег *begin="0;click"*, что позволяет повторно запускать анимацию каждого круга наведением курсора и нажатием клавиши мыши.

Теперь рассмотрим назначение последнего значения для атрибута *calcMode* – это *spline*. Оно позволяет указывать темп анимации внутри каждого ключевого кадра, вместо того чтобы задавать темп всей анимации. Когда мы устанавливаем значение *calcMode="spline"*, чтобы указать темп для каждого ключевого кадра, следует добавить атрибут *keySplines*.

Атрибут *keySplines* задает набор *управляющих точек* Безье, которые ставятся в соответствие списку *keyTimes*, определяя тем самым кубическую кривую Безье для управления темпом анимации интервала. Каждое описание управляющих точек представляет собой набор из четырех значений x_1, y_1, x_2, y_2, описывающий управляющие точки Безье для одного временного отрезка. Все значения должны быть в диапазоне от 0 до 1, и если атрибуту *calcMode* не задано значение *spline*, то атрибут *keySplines* игнорируется.

Вместо того чтобы задавать кубические функции Безье в качестве значений, в *keySplines* задаются координаты двух управляющих точек, используемых для построения кривой. Спецификация *SVG* позволяет разделять эти значения либо запятыми с необязательным пробелом после, либо только пробелами.

Для лучшего представления анимационного эффекта от использования режима *spline* и атрибута *keySplines* дополним ранее приводимый листинг 3.6 значениями координат управляющих точек, используемых для построения кривой. В листинге 3.8 заданы режим *calcMode="spline"* и координаты управляющих точек в атрибуте *keySplines*.

Листинг 3.8. Изменение темпа анимации внутри ключевых кадров:

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
<circle id="circ1" r="30" cx="50" cy="50" fill="red" />
<animate
   xlink:href="#circ1"
   attributeName="cy"
   dur="3s"
   begin="click"
  calcMode="spline"
   values="50; 250; 120;250; 170; 250; 210; 250"
   keyTimes="0; 0.15; 0.3; 0.45; 0.6; 0.75; 0.9; 1"
   keySplines=".42 0 1 1;
                    0 0 .59 1;
                    .42 0 1 1;
                    0 0 .59 1;
                    .42 0 1 1;
                    0 0 .59 1;
                    .42 0 1 1;"
   fill="freeze"/>
</svg>
```

Отметим, что значения атрибута *keyTimes* задают границы интервалов отдельных кадров анимации и являются опорными точками кривых Безье, а значения атрибута *keySplines* задаются для самих интервалов, поэтому в первом наборе будет на один элемент больше, чем во втором.

Для более детального ознакомления с кривыми Безье можно обратиться к ресурсу [15], где представлен онлайн генератор кривых третьего порядка, которые используются в атрибуте *keySplines*.

В заключение этого раздела подчеркнем, что если необходимо задать единую функцию плавности для всей анимации без каких-либо промежуточных значений, то все равно нужно задавать ключевые кадры с помощью атрибута *keyTimes*, но понадобятся только начальный и конечный кадры, т. е. 0 и 1, без промежуточных значений атрибута *values*.

3.1.5. *Атрибуты additive и accumulate*

Иногда бывает полезно определить анимацию, начинающуюся с того места, на котором предыдущая анимация закончилась, или такую, которая использует суммарный накопленный результат предыдущих анимаций как стартовое значение. Для этого в *SVG* есть два атрибута: *additive* и *accumulate*. Каждому из них можно присвоить одно из двух значений:

- для *additive – replace* или *sum*;
- для *accumulate – none* или *sum*.

Первые указанные значения являются значениями по умолчанию.

Предположим, есть объект, который должен перемещаться с места на место поэтапно. Как для любой другой анимации, зададим значения *from* и *to*. Попробуем использовать атрибут *additive="sum"*. Возьмем *круг <circle id="circ1" r="30" cx="50" cy="50" fill="red" />* и запустим *SVG*-код листинга 3.9.

Листинг 3.9. Иллюстрация эффекта анимации от атрибута additive="sum":

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
 <circle id="circ1" r="30" cx="50" cy="50" fill="red" />

  <animate
        xlink:href="#circ1"
        attributeName="cx"
        from="0"
        to="300"
        additive="sum"
        repeatCount="5"
        calcMode="spline"
        keyTimes="0;1"
        keySplines=".42 0 1 1"
        dur="3s"
        begin="click"
        fill="freeze" />
</svg>
```

В результате видим, что круг при каждом повторе стартует не с позиции *X = 0*, как указано в атрибуте *from ="0"*, а с позиции *X = 50*. Это означает, что значение его позиции по оси *X* отсчитывается относительно исходного значения анимируемого атрибута.

У круга исходная позиция *cx* равна 50. Если задано *from="0" to="300"*, то ноль на самом деле означает исходные 50, а 300 на самом деле означает 50 + 300; другими словами, практически это как бы задает *from="50" to="150"*. Это все, что делает атрибут *additive*. Он просто указывает, должны ли значения *from* и *to* отсчитываться относительно текущего значения или нет. Атрибут принимает лишь одно из двух значений: *sum* и *replace*. Если не заменить его на *sum*, то это может привести к неприятному скачку перед началом анимации (попробуйте заменить *sum* на *replace* в листинге 3.9, чтобы видеть это).

Для того чтобы каждое следующее повторение анимации начиналось с конечного значения предыдущей анимации, используется атрибут *accumulate*.

Атрибут *accumulate* отвечает за то, является ли анимация накопительной. По умолчанию его значение *none*, это означает, что при повторении анимация стартует сначала. Но если задать ему значение *sum*, то каждое следующее повторение анимации будет основываться на конечном значении предыдущей анимации.

Так, если вернуться к *SVG*-коду листинга 3.9 и добавить значение *accumulate="sum"*, то получится желаемый результат (листинг 3.10), которого мы и добивались.

Листинг 3.10. Иллюстрация эффекта анимации от атрибута accumulate="sum":

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
<circle id="circ1" r="30" cx="50" cy="50" fill="red" />

 <animate
       xlink:href="#circ1"
       attributeName="cx"
       from="0"
       to="300"
       additive="sum"
       accumulate="sum"
       repeatCount="5"
       calcMode="spline"
       keyTimes="0;1"
       keySplines=".42 0 1 1"
       dur="3s"
       begin="click"
       fill="freeze" />
</svg>
```

Следует обратить внимание, что атрибут *accumulate* игнорируется, если целевой атрибут не поддерживает сложения значений, или если анимация не повторяется. Он также будет игнорироваться, если функция анимации задана только с атрибутом *to*.

3.1.6. *Атрибуты begin и end*

С помощью атрибутов *begin* и *end* можно указать, когда анимация начнется и когда закончится. Например, можно установить анимацию на бесконечное повторение, а затем остановить ее, когда начнется анимация другого элемента. Атрибут *end* принимает значения, похожие на те, что принимает атрибут *begin*. Можно задавать абсолютные или относительные значения/смещения времени, количество повторений, значения событий и т. д. Например, в примере листинга 3.11 красный круг медленно движется справа налево. Зеленый круг тоже анимируется, но только после наведения на него курсора и нажатия клавиши мыши.

Листинг 3.11. Синхронизация анимации двух объектов

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
<circle id="circ1" r="30" cx="50" cy="50" fill="red" />
<circle id="circ2" r="30" cx="50" cy="150" fill="green" />

 <animate
       xlink:href="#circ1"
       attributeName="cx"
       from="50"
       to="450"
       dur="30s"
       begin="0s"
       end="anim2.begin"
       fill="freeze"/>
 <animate
       xlink:href="#circ2"
       attributeName="cx"
       from="50"
       to="450"
       dur="1s"
       begin="click"
       fill="freeze"
       id="anim2"/>
</svg>
```

Движение красного круга прекращается, когда начинается анимация зеленого. Запустите код листинга 3.11 в браузере и посмотрите, как он работает. Дополните код так, чтобы обеспечить возможность перезапуска анимации красного шара.

Естественно, что такая же синхронизация возможна и для анимаций, применяемых к одному и тому же элементу. Например, зададим анимацию цвета круга, бесконечно меняющую его с одного значения на другое. Затем, при нажатии клавиши наведенной мыши остановим анимацию цвета и запустим анимацию движения. Эти действия реализованы в коде листинга 3.12.

Листинг 3.12. Синхронизация двух анимаций одного объекта:

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
 <circle id="circ1" r="30" cx="50" cy="50" fill="#red" />

 <animate
       xlink:href="#circ1"
       attributeName="cx"
       from="50"
       to="450"
       dur="1s"
       begin="click"
       fill="freeze"
       id="move"/>
 <animate
       xlink:href="#circ1"
       attributeName="fill"
       from="red"
       to="blue"
       dur="30s"
       repeatCount="indefinite"
       begin="0s"
       end="move.begin"
       fill="freeze"/>
</svg>
```

Цвет круга непрерывно меняется от красного к синему в течение 30 с. При нажатии клавиши наведенной на круг мыши изменение цвета прекращается и начинается движение *end="move.begin"*.

Использование нескольких значений атрибутов *begin* и *end* позволяет задавать интервалы анимации. Для этого оба атрибута принимают список значений, разделенных точками с запятой. Каждое значение атрибута *begin* соответствует одному значению атрибута *end*, что формирует активные и неактивные интервалы анимации. Пример использования этого способа приведен в листинге 3.13.

Листинг 3.13. Задание интервалов анимации:

```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
 <circle id="circ1" r="30" cx="50" cy="50" fill="gray"
          stroke="black" stroke-width="15" stroke-dasharray="10,1"/>
```

```
 <animateTransform
        xlink:href="#circ1"
        attributeName="transform"
        attributeType="XML"
        type="rotate"
        from="0 50 50"
        to="360 50 50"
        dur="5s"
        begin="0s; 10s; 20s;"
        end="5s; 15s; 25s;"
        fill="freeze"
        restart="whenNotActive"/>
</svg>
```

Если запустить этот SVG-код в браузере, то результат можно представить как что-то вроде движущейся машины, где у колес машины бывают периоды «активности» и «неактивности», в зависимости от того, движется машина или нет. На основе кода листинга 3.13 можно сделать анимированный эффект движущейся машины, применив две анимации: первую, которая перемещает машину по координатной оси или некой траектории, причем эта анимация является относительной и накопительной, и вторую, которая вращает колеса машины в периоды, синхронизированные с ее движением.

В коде листинга 3.13 использован тег <*animateTransform*> для вращения объекта вокруг центра. Этот тег будет рассмотрен подробнее в разд. 3.3.

3.1.7. *Атрибуты min и max*

Точно так же, как вы можете ограничить время повторения анимации, вы можете ограничить и время активной длительности анимации. Атрибуты *min* и *max* указывают минимальную и максимальную активную длительность анимационного объекта соответственно. Они дают возможность задавать активной длительности объекта верхний и нижний пределы. Значениями обоих атрибутов могут быть часовые значения.

Для *min* это указывает минимальную величину активной длительности, измеряемую во время активности объекта. Значение должно быть больше либо равно 0, нуль является значением по умолчанию.

Для *max* часовое значение указывает максимальную величину активной длительности, измеряемую во время активности объекта. Значение также должно быть больше нуля. По умолчанию *max* имеет значение *indefinite*. Оно вообще не ограничивает активную длительность.

Если заданы оба атрибута *min* и *max*, то значение *max* должно быть больше или равно значению *min*. Если это требование не выполняется, оба атрибута игнорируются.

Поскольку атрибутов, которые так или иначе ограничивают длительность анимации, несколько (*dur*, *repeatCount*, *repeatDur*, *end*, *min* и *max*), необходимо знать об их приоритетах.

Порядок такой: сначала браузер вычисляет активную длительность, исходя из значений *dur*, *repeatCount*, *repeatDur* и *end*. Затем он сравнивает результат вычисления с заданными значениями *min* и *max*. Если результат находится в этих пределах, вычисленное на первом шаге значение считается верным и не меняется. Иначе, возможны две ситуации:

1) если первоначально вычисленная длительность превышает значение *max*, активная длительность объекта становится равной значению *max*;

2) если первоначально вычисленная длительность меньше значения *min*, активная длительность объекта становится равной значению *min*, и объект ведет себя следующим образом: если длительность повторения (либо простая длительность, если анимация не повторяется) объекта больше, чем *min*, то активность объекта ограничена *min*. Иначе, объект активен в течение его длительности повторения (либо простой длительности), а затем «застывает» или не отображается в зависимости от значения атрибута *fill*.

В спецификации [13] приведена исчерпывающая таблица с различными комбинациями атрибутов *dur*, *repeatCount*, *repeatDur* и *end*, которая показывает, чему будет равна активная длительность объекта анимации в случае каждой комбинации (табл. 3.2).

Таблица 3.2

Определение времени активности объекта анимации

dur	repeatCount	repeatDur	end	Время активности
d				d
d	defined			repeatCount*d
d		defined		repeatDur
d			defined	MIN(d, end-begin)
d	defined	defined		MIN(repeatCount*d, repeatDur)
d	defined		defined	MIN(repeatCount*d, (end-begin))
d		defined	defined	MIN(repeatDur, (end-begin))
d	defined	defined	defined	MIN(repeatCount*d, repeatDur, (end-begin))
indefinite	defined			indefinite
indefinite	defined	defined		repeatDur
indefinite	defined		defined	end-begin
indefinite	defined	defined	defined	MIN(repeatDur, (end-begin))
indefinite	indefinite			indefinite
indefinite		indefinite		indefinite
indefinite	indefinite	indefinite		indefinite
indefinite	indefinite		defined	end-begin
indefinite		indefinite	defined	end-begin
indefinite	indefinite	indefinite	defined	end-begin

3.1.8. *Использование тега <set>*

Тег <*set*> предоставляет простое средство для задания значения атрибуту на определенный отрезок времени. Он поддерживает все типы атрибутов, включая те, которые по своей логике не могут быть интерполированы, например строки и булевы значения. Тег <*set*> не поддерживает относительных/накопительных анимаций: относительные и накопительные атрибуты недопустимы и игнорируются, даже если указаны.

Поскольку <*set*> используется для выставления элемента в определенное значение в определенный момент и отрезок времени, для него доступны не все атрибуты, упомянутые для предыдущих анимационных элементов. Например, у него нет атрибутов *from* или *by*, потому что изменяемое значение не меняется со временем постепенно.

Для тега <*set*> можно указать целевой элемент, имя и тип атрибута, значение *to* и время анимации, которым можно управлять с помощью следующих атрибутов: *begin, dur, end, min, max, restart, repeatCount, repeatDur* и *fill*.

3.2. Анимация движения <*animateMotion*>

Тег <*animateMotion*> позволяет анимировать движение объекта вдоль заданной траектории. Траектория движения объекта может быть задана двумя способами: с помощью атрибута *path* или с использованием тега внешнего пути <*mpath*>.

Для тега <*animateMotion*> можно применять те же атрибуты, которые были рассмотрены в разд. 3.1. Кроме того, здесь добавляется еще три весьма полезных атрибута, отмеченные выше *path, rotate* и *keyPoints*. Важным отличием у изученного в разд. 3.1.4 атрибута *calcMode* является то, что по умолчанию для тега <*animateMotion*> он имеет значение *paced,* а не *linear*.

Часто тег <*animateMotion*> используется для анимации такого *SVG*-объекта, как текст. Поэтому вопросы, связанные с анимацией текста, также рассматриваются в этом разделе.

Перейдем к последовательному описанию атрибутов анимации и способов их использования.

3.2.1. *Атрибут path*

Атрибут *path* задает траекторию движения. Он задается в том же формате и интерпретируется тем же образом, что и атрибут *d* тега <*path*> (разд. 1.2.7). Эффект анимации движения по траектории заключается в наложении дополнительной матрицы трансформации поверх текущей матрицы

трансформации объекта, вызывающей сдвиг по осям *X* и *Y* в текущей системе координат пользователя на значения *X* и *Y*, рассчитываемые для каждого момента времени. Другими словами, заданная траектория отсчитывается относительно текущего положения элемента, используя данные траектории для перемещения объекта на позицию на этой траектории.

Рассмотрим пример перемещения круга по некоторой заданной траектории:

```
path="M0,0 c3.2-3.4,18.4-0.6,23.4-0.6 c5.7,0.1,10.8,0.9,16.3,2.3 c13.5,3.5,26.1,9.6,38.5,16.2
c12.3,6.5,21.3,16.8,31.9,25.4 c10.8,8.7,21,18.3,31.7,26.9 c9.3,7.4,20.9,11.5,31.4,16.7
c13.7,6.8,26.8,9.7,41.8,9 c21.4-1,40.8-3.7,61.3-10.4
c10.9-3.5,18.9-11.3,28.5-17.8 c5.4-3.7,10.4-6.7,14.8-11.5 c1.9-2.1,3.7-5.5,6.5-6.5"
```

Следует особо обратить внимание на координаты в данных пути. Путь начинается с перемещения (*M*) в точку с координатами (*0, 0*), прежде чем начать рисовать кривую (*c*) в другую точку. Важно отметить, что точка (*0, 0*) в данном случае – это позиция нашего круга (неважно, где он находится), а не верхний левый угол системы координат, поскольку координаты в атрибуте *path* отсчитываются относительно текущего положения объекта.

Перемещение круга по траектории, приведенной выше, реализовано в коде листинга 3.14. Обратите внимание, что круг начинает двигаться, только если навести на него курсор мыши и щелкнуть левой кнопкой: *begin="click"*.

Листинг 3.14. Движение круга по заданной траектории path:

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <circle id="circ1" r="20" cx="100" cy="100" fill="red" />
  <animateMotion
        xlink:href="#circ1"
        dur="3s"
        begin="click"
        fill="freeze"
        path="M0,0 c3.2-3.4,18.4-0.6,23.4-0.6 c5.7,0.1,10.8,0.9,16.3,2.3
    c13.5,3.5,26.1,9.6,38.5,16.2  c12.3,6.5,21.3,16.8,31.9,25.4 c10.8,8.7,21,18.3,31.7,26.9
    c9.3,7.4,20.9,11.5,31.4,16.7  c13.7,6.8,26.8,9.7,41.8,9     c21.4-1,40.8-3.7,61.3-10.4
    c10.9-3.5,18.9-11.3,28.5-17.8 c5.4-3.7,10.4-6.7,14.8-11.5   c1.9-2.1,3.7-5.5,6.5-6.5"/>
</svg>
```

Если указать путь, начинающийся с другой точки, а не с (*0, 0*), то круг в начале движения рывком перескочит на расстояние, заданное координатами начальной точки. Этот перескок можно понаблюдать, повторно запуская круг, так как он останавливается в конечной точке.

Атрибут *path* с траекторией движения может быть указан в любом месте документа; он даже может быть лишь объявлен внутри тега <defs> и вообще не отображаться на холсте. В примере листинга 3.14 путь также не отображается на холсте, так как ему не задан атрибут *stoke*.

3.2.2. *Тег внешнего пути <mpath>*

Другой способ задания траектории движения состоит в использовании ссылки на внешний путь с помощью тега <*mpath*>. Тег <*mpath*>, дочерний по отношению к тегу <*animateMotion*>, будет указывать на внешний путь с помощью атрибута *xlink:href*.

Пример использования тега <*mpath*> приведен в коде *SVG* табл. 3.3. Отметим, что координаты начала пути отличны от (*0,0*) и равны *M99,192*. Для того чтобы избежать начального рывка круга, отмеченного в разд. 3.2.1, здесь применен другой способ – обнулены координаты центра круга *cx="0"*, *cy="0"* перед началом анимации движения по траектории.

Таблица 3.3

Использование тега <mpath> для задания траектории

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <path id="motionPath" fill="none" stroke="gainsboro" stroke-width="5" d="M 99 192 C 137 160 204 133 141 124 C 78 115 34 167 47 129 C 60 91 20 65 77 71 C 134 77 206 43 196 101 C 186 159 118 368 119 299 C 120 230 201 169 138 206 C 75 243 53 231 99 192" /> <circle id="circle" r="10" cx="0" cy="0" fill="red" /> <animateMotion xlink:href="#circle" dur="5s" begin="0s" fill="freeze" repeatCount="indefinite"> <mpath xlink:href="#motionPath" /> </animateMotion> </svg>	

3.2.3. *Атрибут rotate*

В примере листинга 3.15 объект, который подлежал анимации, был кругом. Если нужно анимировать объект, для которого важна ориентация, например машину, реализованную в SVG-коде табл. 1.12, то потребуются некоторые усилия, чтобы получить удовлетворительный эффект. Посмотрим, что нужно изменить в коде SVG из табл. 3.3, чтобы заменить круг на машину.

Прежде всего, нужно заменить описание круга описанием машины из табл. 1.12. Кроме того, идентификатор круга *xlink:href="#circle"* нужно заменить на идентификатор машины *xlink:href="#jeep"*. Теперь можно по-

пробовать, что получилось. Проделайте указанные замены и увидите результат. Траектория движения существует сама по себе, машина ездит сама по себе. Кроме того, размеры машины слишком велики.

Чтобы устранить эти недостатки, сделаем изменения в теге *<use>*:

- для уменьшения размеров машины осуществим масштабирование *scale(0.1)*;
- для привязки машины к траектории передвижения произведем смещение *translate(–80,–180).*

В результате измененному *<use>* присвоим новый идентификатор *id="little"* и запишем его в следующем виде:

```
<use id="little" x="10" y="10" xlink:href="#jeep"
  transform="scale(0.1) translate(-80,-180)"/>.
```

Пусть *little* ассоциируется с машиной величиной 0,1 от исходной. Теперь поменяем ссылку с *xlink:href="#jeep"* на *xlink:href="#little"* в теге *<animateMotion>*.

Теперь машина начинает двигаться по траектории, но не естественно.

Ориентация машины фиксирована и не соответствует направлению движения. Чтобы это изменить, будем использовать атрибут *rotate.*

У атрибута rotate может быть одно из трех значений:

- *auto* – указывает, что объект будет поворачиваться с течением времени на угол направления (т. е. по касательному вектору направления) траектории движения;
- *auto-reverse* – указывает, что объект будет поворачиваться с течением времени на угол направления (т. е. по касательному вектору направления) траектории движения плюс 180°;
- *число* – указывает, что к целевому элементу применена постоянная трансформация вращения, указанное число задает угол поворота в градусах.

Для исправления ориентации машины в нашем случае следует добавить значение *rotate="auto-reverse".*

И наконец, добавим управление началом движения и остановкой. Для этого введем две кнопки в виде кружков:

```
<circle id="start" cx="60" cy="300" r="5" fill="green" />
<circle id="stop" cx="80" cy="300" r="5" fill="red"/>
```

и заменим *begin="0"* на *begin="start.click" end="stop.click".*

Таким образом, заменив в листинге 3.15 описание круга на описание машины из листинга 3.16 (часть *defs*), добавив видоизмененный тег *use* и представив *animateMotion* в виде

```
<animateMotion
```

```
        xlink:href="#little"
        dur="5s"
                begin="start.click" end="stop.click"
        fill="freeze"
        rotate="auto-reverse"
        repeatCount="indefinite">
    <mpath xlink:href="#motionPath" />
  </animateMotion>,
```

получим удовлетворительный результат (рис. 3.1).

Рис. 3.1. Использование атрибута *rotate*
для правильной ориентации автомобиля

Проделайте описанные выше преобразования и изучите все промежуточные результаты. В окончательном варианте изучите эффекты, получаемые при различных допустимых значениях атрибута *rotate*.

3.2.4. *Атрибут keyPoints*

Атрибут *keyPoints* дает возможность определять продвижение по траектории для каждого значения, указанного в *keyTimes*. Когда задан атрибут, *keyPoints* заставляет *keyTimes* применяться к значениям из *keyPoints*, а не к точкам, перечисленным в атрибуте values, или к точкам в атрибуте *path*.

Атрибут *keyPoints* задается как разделенный знаками «точка с запятой» список значений с плавающей точкой от 0 до 1 и указывает, как далеко по траектории должен продвинуться объект в момент времени, заданный в соответствующем значении *keyTimes*. Расчет расстояния определяется алгоритмами браузера. Каждое значение продвижения в списке соответст-

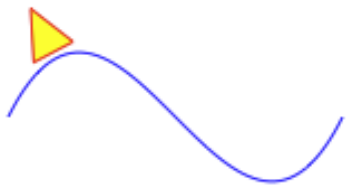
вует одному значению из списка в атрибуте *keyTimes*. Если задан список *keyPoints*, в нем должно быть ровно столько значений, сколько в списке *keyTimes*.

Важный момент, который здесь надо отметить, – необходимость задать значения *linear* для *calcMode*, чтобы *keyPoints* заработал.

Поясним, что путь *M50,125 C 100,25 150,225, 200,125* описан в коде табл. 3.4 дважды для того, чтобы он был видимым (попробуйте убрать его первое определение). С помощью атрибута *keyPoints* весь путь разбит на неравные отрезки: 20, 40 и 20 %, а атрибут *keyTimes* распределяет между ними отведенное время *dur="6s"* поровну, по 2 секунды на каждый. Тем самым при движении треугольника создается впечатление быстрых скатываний с горки и медленных подъемов.

Таблица 3.4

Использование атрибутов *keyPoints* и key*Times*

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <path d="M50,125 C 100,25 150,225, 200, 125" fill="none" stroke="blue"/> <path d="M-10,-3 L10,-3 L0,-25z" fill="yellow" stroke="red"> <animateMotion path="M50,125 C 100,25 150,225, 200, 125" rotate="auto" keyPoints="0;0.2;0.8;1" keyTimes="0;0.33;0.66;1" calcMode="linear" dur="6s" fill="freeze" repeatCount="indefinite"/> </path> </svg>	

3.2.5. *Анимация текста*

Движение текста по произвольной траектории отличается от движения других *SVG*-объектов по траекториям. Для анимации текста необходимо использовать тег <*animate*>, а не <*animateMotion*>.

Начнем с позиционирования текста вдоль траектории. Это можно сделать, вложив тег <*textPath*> внутрь тега <*text*>.

Затем <*textPath*> должен сослаться на фактический путь, который желательно для этого использовать. Путь, на который ссылаемся, может либо отображаться на холсте, либо определяться внутри <*defs*>.

Чтобы анимировать текст вдоль траектории, используем тег <*animate*> для анимации атрибута *startOffset.*

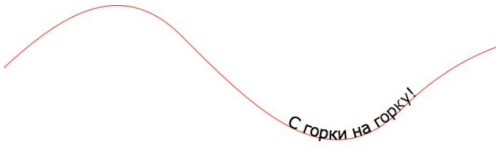
Атрибут *startOffset* определяет отступ текста относительно точки начала пути: 0 % – начало пути, 100 % – конец пути. Так что если, например,

задать отступ в 50 %, текст будет начинаться посередине траектории. Анимируя атрибут *startOffset*, создадим эффект текста, движущегося по траектории.

В табл. 3.5 приведен пример анимации текста вдоль заданной траектории. Параметры атрибутов *calcMode и keySplines* рассматривались в разд. 3.1.4.

Таблица 3.5

Анимация текста вдоль заданной траектории

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <path id="path" fill="none" stroke="red" d="M 100 200 C 200 100 300 0 400 100 C 500 200 600 300 700 200 C 800 100 900 100 900 100" /> <text font-family="Verdana" font-size="22"> <textPath xlink:href="#path"> С горки на горку! <animate attributeName="startOffset" from="0%" to ="100%" begin="0s" dur="5s" repeatCount="indefinite" keyTimes="0;1" calcMode="spline" keySplines="0.1 0.2 0.22 1.0"/> </textPath> </text> </svg>	

3.2.6. *Морфинг*

Морфингом называется процесс плавного преобразования одного изображения в другое небольшими постепенными шагами.

Одним из атрибутов, который можно анимировать в SVG, является атрибут *d* тега <*path*>. Атрибут *d* содержит данные, определяющие контур изображаемого объекта. Данные контура содержат набор команд и координат, которые сообщают браузеру, где и как рисовать точки, кривые и отрезки, образующие итоговый контур.

Анимация этого атрибута делает возможными морфинг *SVG*-контуров и создание эффектов плавного «перетекания» одной формы в другую. Замечательный пример морфинга приведен в [19].

Для того чтобы изменять форму пути было возможно, начальная, конечная и любая из промежуточных форм должны иметь строго одинаковое количество вершин/ключевых точек и порядок их должен оставаться тем же. Если число вершин не совпадает, анимация не заработает. Причина этого в том, что изменение формы на самом деле происходит путем перемещения вершин и интерполяции их положения, так что если одна вершина отсутствует или не совпадает, интерполировать оказывается нечего.

Чтобы анимировать *SVG*-контур, нужно задать для *attributeName* значение *d*, а затем указать для *from* и *to* значения, соответствующие начальной и конечной формам, также можно использовать атрибут *values* для задания любых промежуточных форм, через которые форма должна пройти в ходе изменения. Примеры морфинга можно посмотреть, например, в [18].

Имеется несколько библиотек *JavaScript*, в которых предлагаются достаточно эффективные методы для реализации морфинга. Одна из таких библиотек изучается в разд. 4.5.5.

3.3. Анимация трансформации <*animateTransform*>

Тег <*animateTransform*> анимирует атрибуты трансформации целевого объекта, тем самым позволяя анимации управлять сдвигом, масштабом, вращением и наклоном. Он использует те же атрибуты, что и тег <animate>, плюс один добавочный атрибут: *type*.

Атрибут *type* служит для указания типа трансформации, которую необходимо анимировать. У него может быть одно из пяти значений: *translate, scale, rotate, skewX* и *skewY*.

Атрибуты *from, by* и *to* принимают значения в том же синтаксисе, который подходит для данного типа трансформации:

- для type="translate" каждое отдельное значение выражается как сдвиг по каждой оси;
- для type="scale" каждое отдельное значение выражается как масштаб по каждой оси;
- для type="rotate" каждое отдельное значение выражается как угол поворота и координаты центра вращения;
- для type="skewX" and type="skewY" каждое отдельное значение выражается как угол наклона.

Рассмотрим простейший пример вращения зеленого прямоугольника *id="greenRect"* с помощью элемента <*animateTransform*>. Код для вращения следующий:

```
<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
<rect id="greenRect" width="50" height="50" x="50" y="50" fill="green" />
 <animateTransform
    xlink:href="#greenRect"
    attributeName="transform"
    attributeType="XML"
    type="rotate"
    from="0 75 75"
    to="360 75 75"
    dur="2s"
```

```
        begin="0s"
        repeatCount="indefinite"
        fill="freeze"
        />
    </svg>
```

Атрибуты *from* и *to* указывают угол поворота (начальный и конечный) и центр вращения. Конечно, в обоих случаях центр вращения остается тем же самым. Если не указать центр вращения, то им станет точка с координатами 0,0.

Вот еще один интересный пример, взятый из проекта [16]:

```
<svg xmlns="http://www.w3.org/2000/svg">
<rect width="100" height="100">
  <animateTransform attributeName="transform"
                type="scale" from="2" to="12" repeatCount="10" dur="12s"
                fill="freeze"/>
</rect>
</svg>
```

Здесь анимируются размеры прямоугольника с помощью изменения масштаба *type="scale"* его представления. За время 12 секунд черный прямоугольник увеличивается в размерах в 12 раз и заполняет все окно браузера.

Анимация одной отдельной трансформации очень проста, однако все может стать куда интереснее, когда дело дойдет до множественных трансформаций, особенно с учетом того, что один *animateTransform* может перекрывать другой. Здесь есть широкий простор для экспериментов. В качестве примера множественной трансформации рассмотрим код табл. 3.6.

Таблица 3.6

Множественная трансформация

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <circle r="45" cx="125" cy="125" fill="none" stroke="tomato" stroke-width="20" stroke-dasharray="4"> <animate attributeName="stroke-opacity" values="1;0.1;1" dur="3s" repeatCount="indefinite" /> <animate attributeName="stroke-width" values="100;20;100" dur="3s" repeatCount="indefinite" /> <animateTransform attributeName="transform" dur="3s" type="rotate" from="0 125 125" to="360 125 125" repeatCount="indefinite" /> </circle> <circle r="45" cx="125" cy="125" fill="yellow"> <animate attributeName="fill" values="yellow;tomato;red" dur="6s" repeatCount="indefinite" /> </circle> </svg>	

В приведенном примере динамически изменяются значения прозрачности, ширины и угла наклона обводки, а также цветовая гамма круга.

3.4. Анимация с использованием *wiewBox*

В разд. 1.1 отмечалось, что наряду с системной областью просмотра – окном просмотра *viewport* – можно использовать видовое окно *viewBox*. Изображение из видового окна автоматически транслируется в окно просмотра, когда определены оба окна, например:

```
<svg width="400" height="300"
        viewBox="-800 0 400 300">
```

Что означает приведенное выше определение? Размер окна просмотра и видового окна одинаков: 400×300. Верхний левый угол видового окна расположен в точке (–800,0), и именно оттуда изображение будет транслироваться на дисплей.

В случае когда размеры видового окна и окна просмотра совпадают, изображение из видового окна будет отображаться в окне просмотра «один в один» без искажений. Если заставить видовое окно перемещаться, наподобие перемещения видеокамеры, изображение в окне просмотра также будет перемещаться. На базе этого свойства можно создать анимацию, смещая видовое окно вдоль оси *X* (горизонтальный параллакс) и вдоль оси *Y* (вертикальный параллакс). Естественно, можно перемещать видовое окно по произвольной траектории, получая различные анимационные эффекты.

В случае когда размеры окна просмотра *viewport* и видового окна *viewBox* не совпадают, изображение, передаваемое из видового окна, будет отображаться в окне просмотра в искаженном виде, поскольку передаваемое изображение займет всю площадь полностью. Очевидно, что искажение будет находиться в зависимости от соотношения площадей: $S_{viewBox}$ – площадь видового окна и $S_{viewport}$ – площадь окна просмотра.

Если площадь $S_{viewBox} < S_{viewport}$, то изображение пропорционально увеличится, а при $S_{viewBox} > S_{viewport}$ оно пропорционально уменьшится в размерах. При этом естественные пропорции изображения сохранятся только в случае, когда отношения ширины и высоты для обоих окон равны. На базе этого свойства можно создавать анимации по изменению масштаба изображения как с сохранением естественных пропорций, так и с введением различных искажений.

Параллакс. Поскольку изображение из видового окна автоматически транслируется в окно просмотра, то по мере перемещения видового окна будет перемещаться изображение на экране. Для того чтобы заставить видовое окно перемещаться, достаточно анимировать атрибут *viewBox*.

Предположим, перед нами огромная карта звездного неба и мы хотим рассмотреть ее с помощью видового окна. Пусть карта *sky.jpg* имеет размеры 2400×1600 пикселов. Разместим ее на холсте в области положительных значений координат: *<image xlink:href="sky.jpg" x="0" y="0" width="2400" height="1600"/>*.

Возьмем видовое окно размером 600×500 пикселов *viewBox="0 0 600 500"*, совпадающее по размерам с окном просмотра *svg width="600" height="500"*.

Для того чтобы анимировать атрибут *viewBox*, нужно спланировать стратегию перемещения видового окна. На рис. 3.2 для наглядности площадь размещения карты разбита на прямоугольники размером с видовое окно.

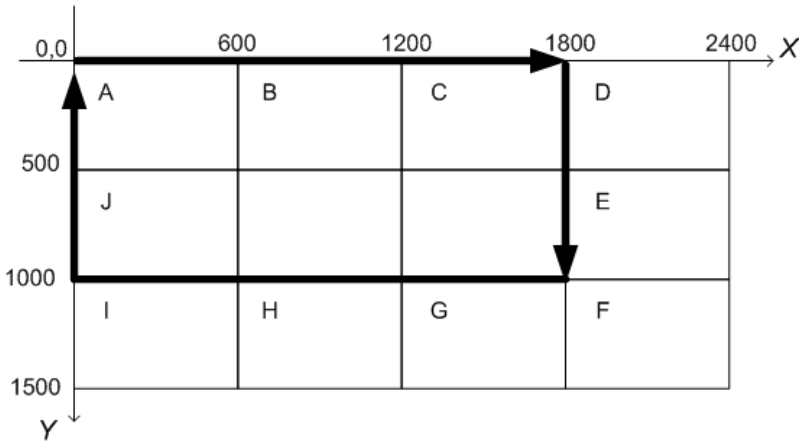


Рис. 3.2. Схема перемещения видового окна *viewBox*

Будем перемещать видовое окно, как указано на рис. 3.2, последовательно охватывая зоны прямоугольников *A, B, C, D, E, F, G, H, I, J* и *A*. Таким образом, стратегия перемещения состоит из четырех этапов:

- горизонтальное перемещение из *A (0,0)* в D *(1800,0)*;
- вертикальное перемещение из *D* в *F (1800,1000)*;
- горизонтальное перемещение из *F* в *I (0,1000)*;
- вертикальное перемещение из *I* в *A (0,0)*.

В соответствии с этой стратегией формируются значения координат левого верхнего угла для *viewBox*. Результирующий SVG-код представлен в листинге 3.15.

Листинг 3.15. Параллакс с помощью анимации атрибута viewBox:

```
<svg width="600" height="500"
        viewBox="0 0 600 500"
        xmlns="http://www.w3.org/2000/svg"
        xmlns:xlink="http://www.w3.org/1999/xlink">
  <image xlink:href="sky.jpg" x="0" y="0" width="2400" height="1600"/>
                                          1800      0 600 500;
                                          1800 1000 600 500;
                                             0 1000 600 500;
                                             0      0 600 500"
  begin="0"  dur="45s" repeatCount="1" restart="whenNotActive" fill="freeze"/>
</svg>
```

В процессе интерпретации полученного кода в браузере можно наблюдать два горизонтальных и вертикальных разнонаправленных параллакса.

При желании увеличить размер изображения достаточно увеличить размеры окна просмотра, скажем, в два раза: *svg width="1200" height="1000"*.

3.5. Анимация удаления <*discard*>

Тег «отбрасывания» <*discard*> позволяет авторам *SVG*-документов указать время, при котором отдельные объекты могут быть удалены, тем самым высвобождаются ресурсы, необходимые для размещения документов. Это особенно полезно для экономии памяти при отображении «долгоиграющего» *SVG*-контента. В качестве иллюстрации использования тега приведем пример из самого последнего проекта консорциума *W3C* [16], посвященного анимации:

```
<svg xmlns="http://www.w3.org/2000/svg" width="352" height="240">
 <ellipse cx="98.5" cy="17.5" rx="20.5" ry="17.5" fill="blue" stroke="black"
       transform="translate(9 252) translate(3 -296)">
  <animateTransform attributeName="transform" begin="0s" dur="2s" fill="remove"
              calcMode="linear" type="translate" additive="sum"
              from="0 0" to="-18 305"/>
  <discard begin="2s"/>
 </ellipse>

 <rect x="182" y="-39" width="39" height="30" fill="red" stroke="black"
     transform="translate(30 301)">
  <animateTransform attributeName="transform" begin="1s" dur="2s" fill="remove"
              calcMode="linear" type="translate" additive="sum"
              from="0 0" to="-26 -304"/>
  <discard begin="3s"/>
 </rect>

 <polygon points="-66,83.5814 -43,123.419 -89,123.419" fill="green" stroke="black"
       transform="matrix(1 0 0 1.1798 0 -18.6096)">
  <animateTransform attributeName="transform" begin="2s" dur="2s"
              fill="remove" calcMode="linear" type="translate" additive="sum"
              from="0 0" to="460 63.5699"/>
  <discard begin="4s"/>
 </polygon>
</svg>
```

Не все SVG-атрибуты можно анимировать, и не все из тех, которые можно анимировать, можно анимировать всеми анимационными тегами. Полный список анимируемых атрибутов и таблицу, показывающую, какой из этих атрибутов можно анимировать и с помощью какого тега, можно найти в разд. 2.18 из [16].

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. *Scalable Vector Graphics (SVG) 1.0 Specification, W3C Recommendation 04 September 2001. – http://www.w3.org/TR/SVG10/.*

2. *Scalable Vector Graphics (SVG) 1.1 (Second Edition), W3C Recommendation 16 August 2011. – http://www.w3.org/TR/SVG/.*

3. *Scalable Vector Graphics (SVG), W3C SVG Working Group. – http://www.w3.org/Graphics/SVG/.*

4. *Scalable Vector Graphics (SVG) 2, W3C Editor's Draft 19 October 2016. – https://svgwg.org/svg2-draft/.*

5. *SVG Working Group specifications. – https://github.com/w3c/svgwg.*

6. *SVG Accessibility API Mappings, W3C Editor's Draft 16 August 2016. – http://w3c.github.io/aria/svg-aam/svg-aam.html.*

7. *https://xmlgraphics.apache.org/batik/Apache™ Batik SVG Toolkit.*

8. *An SVG Primer for Today's Browsers, W3C Working Draft — September 2010. – https://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html.*

9. *http://srufaculty.sru.edu/david.dailey/svg/newstuff/simpleshapes.svg.*

10. *Structuring, Grouping, and Referencing in SVG – The <g>, <use>, <defs> and <symbol> Elements – Jule 2014 – перевод: https://habrahabr.ru/post/230443/. – https://sarasoueidan.com/blog/structuring-grouping-referencing-in-svg/.*

11. *http://www.intuit.ru/studies/courses/1063/210/lecture/5424.*

12. *SMIL Animation, W3C Recommendation 04-September-2001. – https://www.w3.org/TR/2001/REC-smil-animation-20010904/.*

13. *A Guide to SVG Animations, December 2015. – https://css-tricks.com/guide-svg-animations-smil/.*

14. *https://svg-art.ru.*

15. *SVG Animations Level 2, W3C Editor's Draft 10 November 2016. – https://svgwg.org/specs/animations/.*

16. *https://developer.mozilla.org/en-US/docs/Web/SVG/Element/defs.*

17. *http://codepen.io/noahblon/post/an-intro-to-svg-animation-with-smil.*

18. *https://www.w3.org/Consortium/Offices/Presentations/SVG/113.svg.*

19. *Coyer, C. SVG symbol a Good Choice for Icons, June 3, 2014. – https://css-tricks.com/svg-symbol-good-choice-icons/.*

20. *SVG Cubic Bézier Curve Example – постоение on-line. – http://blogs.sitepointstatic.com/examples/tech/svg-curves/cubic-curve.html.*

21. *Accessible SVGs, Heather Miglority, July 6, 2016. – https://css-tricks.com/accessible-svgs/.*

22. *https://developer.mozilla.org/ru/docs/Web/SVG.*

23. *http://ruseller.com/lessons.php?rub=28&id=.*

24. Библиотека *svg.js. – http://svgjs.com/.*

25. Библиотека *vivus.js. – https://github.com/maxwellito/vivus#vivusjs.*

26. *http://editor.method.ac он лайн svg редактор Method Draw.*

27. *https://github.com/duopixel/Method-Draw/.*

28. Редактор *Inkscape. – http://www.inkscape.paint-net.ru/.*

29. *Boxy SVG: A Fast, Simple, Insanely Useful, FREE SVG Editor By Alex Walker April 21, 2016. – https://www.sitepoint.com/boxy-svg-a-fast-simple-insanely-useful-svg-editor/.*

30. Библиотека *d3.js. – https://github.com/d3/d3/releases.*

31. Примеры разработок на *d3.js. – http://bl.ocks.org/mbostock, https:// github.com/mbostock/d3/wiki/Gallery.*

32. Описание функций *d3.js. – https://github.com/d3/d3/blob/master/API.md.*

33. Библиотека *vivus.js. – https://github.com/maxwellito/vivus.*

34. Онлайн ресурс. *– https://maxwellito.github.io/vivus-instant/.*

35. Описание библиотеки *svg.js. – http://svgjs.com/.*

36. Синтаксис библиотеки *svg.js. – http://documentup.com/wout/svg.js#syntax-sugar.*

37. Примеры на *svg.js. – https://www.npmjs.com/package/svgjs.*

38. Программная анимация на JavaScript. *– http://javascript.ru/blog/andrej-paranichev/osnovy-programmnoj-animacii-javascript.*

39. Как манипулировать и анимировать *SVG* через *Snap.svg. – http:// htmlbook.ru/blog/kak-manipulirovat-i-animirovat-svg-cherez-snapsvg.*

40. Библиотека *Snap.svg. – https://github.com/adobewebplatform/Snap.svg/ archive/v0.3.0.zip.*

41. Примеры использования библиотеки *Snap.svg. – http://codepen.io/ collection/edpyJ/.*

42. *SVG Tutorial – W3Schools. – https://github.com/svgdotjs/svg.js.*

43. *https://www.youtube.com/watch?v=ehwK-KM4uYQ&feature=youtu.be.*

44. *https://cdnjs.com/.*

45. *http://editor.method.ac.*

46. *http://archer.graphics/home.html.*

47. *http://codepen.io/archer-graphics/.*

Филиппов Феликс Васильевич

ОБРАБОТКА ГРАФИЧЕСКОЙ ИНФОРМАЦИИ В ФОРМАТЕ SVG

Учебное пособие

Часть 1

Редактор *И. И. Щенсняк*

Компьютерная верстка *Н. А. Ефремовой*

План издания 2017 г., п. 88 а

Подписано к печати 21.06.2017
Объем 5,25 усл.-печ. л. Тираж 26 экз. Заказ 788

Редакционно-издательский отдел СПбГУТ
191186 СПб., наб. р. Мойки, 61

Отпечатано в СПбГУТ