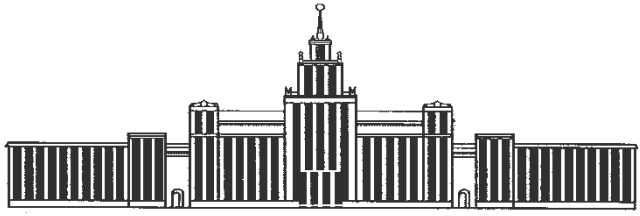
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ



ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Телегин А.И., Тимофеев Д.Н., Читалов Д.И., Пудовкина С.Г.

SVG-РАЗМЕТКА ДВУХМЕРНОЙ ГРАФИКИ

ОПЫТ ИСПОЛЬЗОВАНИЯ SVG

В СОЗДАНИИ ДВУХМЕРНОЙ ГРАФИКИ

МИАСС

2015

Электротехнический факультет

004.92
S 96

Телегин А.И., Тимофеев Д.Н., Читалов Д.И., Пудовкина С.Г.

SVG-РАЗМЕТКА ДВУХМЕРНОЙ ГРАФИКИ

ОПЫТ ИСПОЛЬЗОВАНИЯ SVG

В СОЗДАНИИ ДВУХМЕРНОЙ ГРАФИКИ

МИАСС

2015

004.92:004.43
S 96

SVG-разметка двухмерной графики : Опыт использования SVG в создании двухмерной графики / А. И. Телегин, Д. Н. Тимофеев, Д. И. Читалов, С. Г. Пудовкина ; Юж.-Урал. Гос. Ун-т, ЭТФ. – Миасс : ЭТФ ЮУрГУ, 2015. – 73 с.

Рассматриваются варианты использования языка декларативного программирования масштабируемой векторной графики. Особое внимание уделено основным элементам языка и их атрибутам. На конкретных примерах рассмотрено применение графических узлов, атрибутов трансформации, подключения дополнительных внешних файлов и особенно анимации. Описаны способы реализации изменения поведения отдельных элементов изображения по действиям пользователя.

Коды листингов подробно анализируются, что позволяет постепенно осваивать методологию разметки векторной графики. В качестве примеров рассматриваются как простые фигуры, так и составные изображения. Приведенные примеры демонстрируют эффективность применения рассматриваемой технологии для создания двухмерных моделей механических систем и чертежей объектов различного уровня сложности.

Рецензия: д. т. н. В. И. Пегов

Введение

Язык разметки масштабируемой векторной графики обязан своим появлением консорциуму W3C и разрабатывается с 1999 года. Он предназначен для описания графических элементов на веб-страницах. Термином SVG именуется не только технология программирования, но и формат получаемого изображения.

Возможности языка находят свое применение при разработке веб-сайтов и проектов с большим количеством двухмерной графики. Многие графические пакеты позволяют создавать чертежи и экспортировать их с расширением «.svg». Получаемые изображения имеют высокое качество визуализации как на стандартных персональных компьютерах, так и на мобильных устройствах. Благодаря развитию технологии SVG появилась возможность создавать изображения, обладающие малым размером и не теряющие в качестве при масштабировании.

Векторная графика получает все большее распространение, ее преимущества высоко ценятся веб-разработчиками. Это открытый стандарт, не требующий приобретения лицензии и совместимый с HTML, XHTML, CSS. SVG поддерживается почти всеми современными браузерами, включая Mozilla, Opera, Chrome.

В разделе 1 рассматривается общая структура SVG-документа и правила его составления, базовые фигуры, их атрибуты и способы группировки объектов и стилизации. Второй раздел посвящен особенностям использования путей для рисования объектов различной сложности – как стандартных фигур, так и квадратичных, кубичных кривых. Варианты трансформаций элементов изображения изложены в разделе 3. К ним относятся перемещения, вращения, масштабирование, скосы. Четвертый раздел

содержит информацию об использовании текстового узла для добавления надписей и описательных сообщений к изображениям.

В разделах 5-7 представлено подробное описание принципов создания анимируемых изображений, приведены фрагменты кода простой и сложной анимации, отвечающие за решение конкретных задач, демонстрирующие использование всех видов элементов анимации. Восьмой раздел посвящен способам визуального изменения содержимого SVG-документа по действиям пользователя, каждое из которых описывается определенным событием. В завершающем, девятом разделе, приведены листинги примеров для каждого из теоретических разделов. Коды листингов могут быть воспроизведены пользователем в браузере для проверки примеров на практике.

1. SVG. Быстрое начало

Консорциум W3C разработал спецификации XML-ориентированного графического векторного формата SVG (Scalable Vector Graphics), в элементах и атрибутах которого хранится информация о целых объектах (круге, прямой линии, прямоугольнике и т.д.). Благодаря своей XML-ориентации SVG имеет следующие возможности: создание и изменение изображения с помощью простого текстового редактора; изменение размера изображения без снижения его качества; возможность использования любых шрифтов и стилей, в том числе авторских; поддержка CSS; извлечение текста из изображения; управления с помощью сценариев; применение к графике и тексту сложных эффектов; смешивание SVG-разметки с любой XML-разметкой; анимация; интерактивность и т.д.

SVG-графика отображается в прямоугольной области, имеющей конечные размеры. Эту область называют областью просмотра (ОП). Размеры ОП (ширина и высота) определяются в процессе согласования между фрагментом SVG-документа и его предком, например, html-документом. Как только этот процесс будет завершен, пользовательский агент (ПА) SVG будет обеспечен значениями (обычно целочисленными) ширины и высоты ОП в em, ex, px, pt, pc, cm, mm, in или процентах (px, т.е. пикселы – единицы измерения по умолчанию). Используя эту информацию, ПА определяет ОП так, что начальная система координат (СК) ОП и пользовательская СК являются идентичными, т.е. их начала и оси совпадают. СК «корневой» ОП расположена следующим образом. Во-первых, начало СК ОП совпадает с верхним левым углом экрана монитора. Во-вторых, положительная ось X указывает направо. В-третьих, положительная ось Y указывает вниз, т.е. СК является левой. Текст в ОП отображается слева направо. Единицами измерения в начальной СК по умолчанию являются «пикселы». СК ОП также называют пространством ОП, а пользовательскую СК – пространством пользователя.

В любой точке рисунка можно установить новую ОП, в которой будет содержаться графика включенного элемента внутри SVG-содержимого. С установлением новой ОП неявно устанавливается новая СК ОП, т.е. новая пользовательская СК. Границы новой ОП определяются атрибутами x, y, шириной и высотой элемента, устанавли-

вающего новую ОП. И новая СК ОП и новая пользовательская СК имеют начало координат (x, y), где x и y представляют значение соответствующих атрибутов элемента, устанавливающего ОП. Ориентация новой СК ОП и новой пользовательской СК соответствует ориентации текущей пользовательской СК элемента, устанавливающего ОП. Единица измерения в новой СК ОП и новой пользовательской СК – того же размера как единица измерения в текущей пользовательской СК элемента, устанавливающего ОП. Например:

```
<?xml version="1.0" standalone="no"?>
<svg width="4in" height="3in" version="1.1" xmlns="http://www.w3.org/2000/svg">
  <!--следующая декларация устанавливает новую ОП -->
  <svg x="25%" y="25%" width="50%" height="50%">
    <!-- разметка графики -- >
  </svg>
</svg>
```

Файл с расширением *svg* называют SVG-документом. Корневым элементом SVG-документа является элемент <svg>. Этот элемент и всё его содержимое называется svg-фрагментом графической разметки. Элемент <svg> имеет следующие атрибуты: width – ширина svg-документа; height – высота svg-документа; x – горизонтальная координата левого верхнего угла; y – вертикальная координата этого угла.

Новую ОП устанавливают, например, следующие svg-элементы: svg; symbol – определяет новые ОП всякий раз, когда они приведены элементом use; image – ссылается на SVG файл, приводит к установлению временной новой ОП; foreignObject – создает новую ОП, чтобы рисовать содержание, которое находится в элементе.

Все координаты и длины в SVG могут быть определены с или без единиц измерения. Если координата или значение длины даны без единиц измерения (например, "25"), то данная координата или длина задана в пользовательских единицах (т. е. значение в текущей пользовательской СК). Например:

```
<text style="font-size: 50">Текст с размером в 50 ед.</text>
```

Координата или значение длины могут быть выражены как число со следующей за ним единицей измерения (например, "25cm" или "15em"). Список единиц измерения в

SVG соответствует списку единиц измерения в CSS: em, ex, px, pt, pc, cm, mm, в процентах.

В SVG используются следующие шесть простейших базовых фигур:

<line x1=”47” y1=”31” x2=”260” y2=”31”/> – линия, задаваемая координатами начала и конца;

<circle cx=”129.5” cy=”235.5” r=”50”/> – круг, задаваемый координатами центра и радиусом;

<rect x=”39” y=”94” width=”196” height=”144”/> – прямоугольник, задаваемый координатами левого верхнего угла, а также шириной и высотой;

<ellipse cx=”165” cy=”106.5” rx=”83” ry=”40.5”/> – эллипс, задаваемый координатами центра и радиусами по осям X и Y;

<polyline points=”72, 275, 231, 274, 264, 196”/> – ломаная линия, задаваемая координатами вершин;

<polygon points=”144, 36, 180, 50, 107, 50”/> – многоугольники, описывающие закрытые фигуры, задаются координатами вершин, где последняя и первая вершины связаны прямой линией.

Более сложные SVG-фигуры можно строить с помощью путей (траекторий), которые рассматриваются в следующем разделе.

Для собственной стилизации SVG-элемента, т.е. для применения к конкретной фигуре SVG-стилей (цвет линии, цвет заливки, контрастность и т.д.) можно использовать атрибут style. В SVG также как и в HTML можно использовать локальную и глобальную стилизации. Поскольку SVG – графический формат, он, в отличие от текстов, имеет дополнительные стили визуализации фигур. Например, стили линии фигур (пунктирная, штрихпунктирная, двойная линии и т.д.), стили заливки, стили эффектов фильтрации, световые эффекты, падающие тени и др.

Графические объекты можно группировать при помощи элемента <g>, свойства которого распространяются на все вложенные графические элементы. Например, в следующем фрагменте создается два красных и два синих прямоугольника:

```
<g id=”группа1” style=”fill:red”>
  <rect x=”1cm” y=”1cm” width=”1cm” height=”1cm”/>
  <rect x=”3cm” y=”1cm” width=”1cm” height=”1cm”/>
```

```
</g>
<g id=”группа2” style=”fill:blue”>
  <rect x=”1cm” y=”3cm” width=”1cm” height=”1cm”/>
  <rect x=”3cm” y=”3cm” width=”1cm” height=”1cm”/>
</g>
```

Часто возникает необходимость в добавлении в элемент <svg> растрового графического изображения. Для этого используется элемент <image>. Он определяет прямоугольник, внутри которого будет визуализировано содержимое графического файла. В следующем фрагменте размечается импорт графического изображения в прямоугольник размером 100x100 пикселей, расположенный относительно начала текущей СК на удалении 200 px вдоль оси X и Y.

```
<image x=”200” y=”200” width=”100px” height=”100px” xlink:href=”имя.jpg”/>
```

Для связи с внешними ресурсами в SVG, как и в HTML, кроме элемента <image> используется элемент <a>. Его называют анкерным элементом (гиперссылкой, элементом сетевой связи). В XML-технологиях ссылки размечаются на языке XLink, словарь которого (в первой версии) состоит из имён десяти атрибутов (элементов у XLink 1.0 нет). Графический объект, разметка которого является содержимым элемента <a>, является местом ссылки. То есть, указав на него мышкой, во-первых, стрелка сменит свой вид, во-вторых, после щелчка левой кнопкой, можно перейти на связанный внешний объект: SVG-файл, HTML-файл и т.д. Адрес внешнего документа задается как значение атрибута href, перед именем которого должен стоять префикс пространства имен языка XLink. Общепринято в качестве префикса использовать слово xlink. Таким образом, следующий фрагмент

```
<a xlink:href=”имя.svg”> <circle cx=”20” cy=”40” r=”10”/> </a>
```

превращает круг в ссылку на SVG-документ в файле имя.svg. В корневом элементе SVG-документа или в самом внешнем элементе <svg>, содержащем анкер <a>, необходимо для выбранного префикса в значении атрибута xmlns:xlink указать пространство имен:

```
<svg xmlns:xlink=”http://www.w3.org/1999/XLink”>
  Содержимое с анкерным элементом
</svg>
```

Кроме href в SVG могут использоваться следующие Xlink-атрибуты: role, arcrole, title, show, actuate.

2. Пути

Путь представляет собой контур фигуры. При построении пути используется понятие текущей точки (ТТ). Проводя аналогию с рисованием на бумаге, под ТТ можно понимать кончик карандаша. Перемещая позицию карандаша, можно нарисовать контур фигуры (разомкнутой или замкнутой) по прямой линии или по кривой.

Путь описывается элементом path. Его атрибут d содержит данные пути (ДП), в которых после буквы (через пробельный символ или запятую) перечисляются числа. Буквы являются кодами следующих инструкций (команд): moveto (M, m – установка новой ТТ), line (L, l, H, h, V, v – рисование прямых), curve (S, s, C, c, Q, q, T, t – рисование кривых Безье), arc (A, a – рисование эллиптических кривых) и closepath (Z, z – замыкание пути). Здесь за буквой в верхнем регистре следуют абсолютные координаты точек пути, в нижнем – их относительные координаты. Команды line и curve имеют в дополнение к общим (L, l и S, s) частные формы записи (H, h, V, v). Синтаксис ДП очень краток, что уменьшает размер svg-документа и увеличивает скорость его загрузки. Лишнее пустое пространство и запятые в ДП могут быть удалены, например, строку "M 100 100 L 200 200" можно записать в виде "M100 100L200 200". В следующих абзацах описываются команды ДП.

M (m) x y … – устанавливает новую ТТ, т.е. переносит «карандаш» в новое место с координатами x, y. Новый сегмент (подпуть) ДП должен начинаться с M (m).

Z (z) – заканчивает текущий подпуть и автоматически проводит прямую линию из ТТ в начальную точку текущего подпути. Если Z (z) следует непосредственно после M (m), то M (m) обозначает начальную точку следующего подпути. Если Z (z) следует непосредственно за любой другой командой, то следующий подпуть начинается в той же самой точке, что и текущий подпуть.

L (l) x y … – рисует линию из ТТ в точку с координатами (x,y), которая становится новой ТТ. Для рисования полилинии достаточно записать несколько координатных пар. В конце команды новая ТТ устанавливается в последнюю пару координат.

H (h) x – рисует горизонтальную линию из ТТ (cpx, cpy) в точку (x, cpy). Новой ТТ становится (x, cpy).

V (v) y – рисует вертикальную линию из ТТ (cpx, cpy) в (cpx, y). Новой ТТ становится (cpx, y).

C (c) x1 y1 x2 y2 x y … – рисует кубичную кривую Безье из ТТ в точку (x,y), используя (x1,y1) как контрольную точку (КТ) в начале кривой и (x2,y2) как КТ в конце кривой. Для рисования полилинии Безье необходим набор координат. В конце команды новой ТТ становится последняя пара координат (x,y).

S (s) x2 y2 x y … – рисует кубичную кривую Безье из ТТ в точку (x,y). Первая КТ принимается как отражение второй КТ из предыдущей команды относительно ТТ. Если предыдущих команд нет или предыдущая команда была не C (c), S (s), то считается, что первая КТ совпадает с ТТ. (x2, y2) – вторая КТ, т.е. КТ в конце кривой. Для рисования полилинии Безье достаточен набор координат. В конце команды новой ТТ становится последняя пара координат (x,y).

Q (q) x1 y1 x y … – рисует квадратичную кривую Безье из ТТ в точку (x,y), используя (x1,y1) как КТ. Может быть указано множество наборов координат Для рисования полилинии Безье необходимо множество наборов координат. В конце команды новой ТТ становится последняя пара координат (x,y).

T (t) x y … – рисует квадратичную кривую Безье из ТТ в точку (x,y). За КТ принимается зеркальное отражение КТ из предыдущей команды относительно ТТ. Если предыдущей команды нет или предыдущая команда не Q (q), T (t), то за КТ принимается точка, совпадающая с ТТ. В конце команды новой ТТ становится последняя пара координат (x,y).

A (a) rx, ry, alfa, флаг-большой-дуги, флаг-изгиба, x, y … – рисует эллиптическую дугу из ТТ в точку (x, y). Размер и ориентация дуги определяются двумя радиусами (rx, ry) и углом alfa, который указывает, как эллипс в целом повёрнут относительно текущей СК. Центр эллипса (cx, cy) вычисляется автоматически для удовлетворения ограничений, наложенных флагами. На изображаемую дугу, накладываются следующие ограничения. Во-первых, дуга начинается в ТТ. Во-вторых, дуга оканчивается в точке (x, y). В-третьих, у эллипса существует два радиуса (rx, ry). В-пятых, ось X эллипса вращается на угол alfa относительно оси X текущей СК. Как правило, существу-

ет четыре различные дуги (два разных эллипса, каждый с двумя различными изгибами дуги), которые удовлетворяют перечисленным ограничениям. Флаг-большой-дуги и флаг-изгиба указывают, какая из четырех типов дуг будет нарисована. Из четырех кандидатов дуги два представляют большую дугу (не менее 180 градусов), и два представляют малую дугу (не более 180 градусов). Если флаг-большой-дуги равен '1', то будет выбрана одна из больших дуг, иначе (флаг-большой-дуги равен '0') будет выбрана одна из меньших дуг. Если флаг-изгиба равен '1', то дуга будет нарисована в направлении "положительного угла", т.е. формула эллипса x=cx+rx*cos(theta) и y=cy+ry*sin(theta) соответствует тому, что theta начинается с угла, соответствующему TT, и увеличивается в положительном направлении до достижения дугой координат (x,y). Значение, равное '0', заставляет рисовать дугу в направлении "отрицательного угла", т.е. theta начинается с угла, соответствующего TT, и уменьшается до достижения дугой координат (x,y).

3. Трансформации СК

Новое пространство пользователя (ПП) и новая текущая СК могут быть установлены в любом месте в пределах SVG-фрагмента, используя атрибуты преобразования СК. Трансформации (преобразования) СК являются фундаментальными операциями 2-х мерной графики и представляют собой стандартные методы управления размерами, расположением, углом поворота и углом скоса графических объектов. Математически все svg-трансформации могут быть представлены в виде матрицы 3x3:

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Так как используются только 6 значений этой матрицы, то общее преобразование может быть представлено вектором: [a b c d e f]. Атрибуты преобразования записываются в виде списка вещественных (действительных) чисел, которые следуют в определенном порядке и разделены пробелом и/или запятой. Доступны следующие атрибуты преобразования:

matrix(a b c d e f) – общее матричное преобразование [a b c d e f].

translate(tx [ty]) – перемещение на tx вдоль оси X и на ty вдоль оси Y. Если ty не задано, то его значение равно нулю.

scale(sx [sy]) – масштабирование вдоль оси X в sx раз и вдоль оси Y в sy раз. Если sy не задан, приравнивается к sx.

rotate(alfa [cx cy]) – вращение на угол alfa вокруг точки (cx, cy). Если cx и cy не заданы, то поворот осуществляется относительно начала СК. Если cx и cy заданы, поворот осуществляется вокруг точки (cx, cy), что эквивалентно следующей последовательности преобразований: translate(cx, cy) rotate(alfa) translate(-cx, -cy).

skewX(alfa) – скос вдоль оси X на угол alfa.

skewY(alfa) – скос вдоль оси Y на угол alfa.

Допускается последовательность преобразований, которая эквивалентна вложенным преобразованиям с учётом их порядка следования. Например, следующее последовательное преобразование

```
<g transform="translate(-10,-20) scale(2) rotate(45) translate(5,10)">
 <!-- графические элементы -->
</g>
```

функционально эквивалентно следующим вложенным группам преобразований:

```
<g transform="translate(-10,-20)">
  <g transform="scale(2)">
    <g transform="rotate(45)">
      <g transform="translate(5,10)"> <!-- графические элементы --> </g>
    </g>
  </g>
</g>
```

Преобразование применяется к элементу до применения других его атрибутов. Например:

```
<rect x="10" y="10" width="20" height="20" transform="scale(2)"/>
```

Здесь значения x, y, width и height применяются после того как будет применено масштабирование, т.е. атрибуты x, y, width и height (а так же другие атрибуты) будут учитываться уже в новой (преобразованной) СК. Таким образом, выше приведенный элемент 'rect' функционально эквивалентен следующему:

<g transform="scale(2)"> <rect x="10" y="10" width="20" height="20"/> </g>

Новое ПП (новая текущая СК) может быть установлено преобразованием, которое задаётся атрибутом transform svg-элемента. Это преобразование распространяется на все потомки этого элемента. Преобразование может быть вложенным в каждом случае, дополняя предыдущее преобразование.

4. Разметка текста

С помощью элемента <text> можно вставить любой текст в разметку графики. Каждый элемент <text> описывает одну строку текста. Элемент <text> является родительским элементом для всех текстовых объектов в SVG. Элемент <text> позволяет контролировать положение строки текста, выравнивание текста и другие свойства. Следующий код отображает текст “SVG Текст”.

```
<text x="200" y="80" style="text-anchor:middle; font-size:60; font-weight:800;
      font-family:Verdana; font-style:italic">
  SVG Текст
</text>
```

Атрибуты x и y элемента <text> управляют положением "текстового якоря" (text-anchor). Якорь является точкой отсчета, относительно которой размещается текст. Атрибут style содержит CSS-свойства, определяющие стиль текста и его положение. Например, свойства font-size, font-family, font-style и font-weight задают характеристики шрифта, используемого для отображения текста. Свойство text-anchor определяет, как позиционируется текст относительно текстового якоря. В приведённом примере текст центруется относительно текстового якоря.

Текст можно размещать вдоль произвольной траектории. Следующий код размещает буквы "S","V","G", окрашенные в черный, красный и желтый цвет соответственно, вдоль кривой линии:

```
<defs>
  <path id="путь1" d="M -100 0 c 0 -100 200 -100 200 0"/>
</defs>
<g transform="translate(200, 250)">
```

```
  <use xlink:href="#путь1" style="stroke:yellow; stroke-width:40; fill:none;" />
  <use xlink:href="#путь1" style="stroke:black; stroke-width:1; fill:none;" />
  <text style="font-family:Verdana; font-size:80; font-weight:800;
                                                    fill:blue; text-anchor:middle">
    <textPath xlink:href="#путь1" startOffset="145" >
      <tspan style="fill:black">S</tspan>
      <tspan style="stroke:black; fill:white;">V</tspan>
      <tspan style="fill:red">G</tspan>
    </textPath>
  </text>
</g>
```

Для размещения текстового блока вдоль границы геометрической формы или траектории нужно использовать элемент <path> внутри элемента <defs>. Затем можно сконструировать блок текста (он должен размещаться внутри группирующего элемента <g>), который будет отображаться вдоль траектории. Элемент <textPath> указывает на нужную траекторию с помощью атрибута xlink:href. Если траектория также должна отображаться, можно использовать запись use xlink:href, что и сделано в примере. В элементе <textPath> могут быть заданы различные дополнительные атрибуты, позволяющие управлять отображением текста на траектории, кроме тех свойств, которые можно установить для элемента <text>. Атрибут startOffset определяет, в каком месте начинается текст относительно начальной точки траектории. Элемент <tspan> используется для окрашивания букв в различные цвета. SVG предоставляет средства контроля над направлением текста и его ориентацией. Следующий код изменяет направление текста "ориентация" с горизонтального на вертикальное:

```
<text x="100" y="75" style="font-family:Verdana; font-weight:800; font-size:50;
 fill:white; writing-mode:tb; glyph-orientation-vertical:0;
                                                    text-anchor:middle">ориентация</text>
```

Атрибут writing-mode управляет первичным направлением текста. Его значение "tb" описывает направление сверху-вниз. Среди других возможных значений – "rl" (справа-налево) и "tb-rl" (сверху-вниз и справа-налево).

Для размещения текста вверх ногами достаточен следующий код:

```
<defs> <path id="путь3" d="M 100 150 l -200 0" /> </defs>
<g transform="translate(200, 250)">
  <use xlink:href="#путь3" style="stroke:yellow; stroke-width:40" />
  <use xlink:href="#путь3" style="stroke:black; stroke-width:1" />
  <text style="font-family:Verdana; font-size:40; font-weight:900;
                                    fill:black; stroke:none; text-anchor:middle">
    <textPath xlink:href="#путь3" xml:space="default" startOffset="100">
                                    Этот текст размещён вверх ногами</textPath
  </text>
</g>
```

Элемент <defs> задает прямую линию, которая идет справа налево, она изменяет направление расположенного на ней текста: текст пишется вверх ногами справа налево. Этот код демонстрирует размещение текста на траектории для изменения его направления.

В SVG каждый элемент <text> создает одну строку текста. SVG не делает автоматической разбивки строки или переноса слов. Применение элемента <tspan> позволяет разбивать строки внутри текстового блока. Следующий код показывает, как можно использовать элемент <tspan> для управления относительным положением внутри текстового блока:

```
<text x="0" y="0" style="font-family:Verdana; font-size:50; font-style:italic;
      fill:white; stroke:black; writing-mode:lr; text-anchor:middle;"
                                    transform="translate(-100, 75) rotate(-90)">
  <tspan dy="0"> Круто! </tspan>
  <tspan dy="-25" style="font-size:10; stroke:none; fill:black;"> SVG </tspan>
</text>
```

Атрибут dy элемента <tspan> управляет относительным вертикальным положением текущего фрагмента текста ("Круто!") относительно начальной точки текстового блока. Установив атрибут dy для второго элемента <tspan>, получаем эффект верхнего индекса, т.е. "SVG" является верхним индексом для слова "Круто!". Элемент <tspan> используется для позиционирования и управления отображением текста. Он разделяет

различные фрагменты текста и применяет к ним различные характеристики положения и отображения.

5. Анимация

SVG-анимацию, т.е. изменение изображения с течением времени, можно осуществить несколькими способами. Во-первых, использовать svg-элементы анимации (ЭА), которые позволяют размечать пути движения, эффекты постепенного появления и исчезновения изображения, увеличение, уменьшение, перемещение, вращение, изменение цвета графических объектов и т.д. Во-вторых, применять SMIL-элементы к анимируемым svg-элементам (АЭ) для создания анимационных эффектов. В-третьих, использовать DOM-интерфейс для осуществления сложных анимаций путём создания сценариев (программ, скриптов), если эту задачу не удаётся или сложно решить первыми двумя способами.

ЭА предложены рабочей группой Synchronized Multimedia (SYMM), разработчиками спецификаций языка разметки Synchronized Multimedia Integration Language (SMIL 1.0). Рабочая группа SYMM, в сотрудничестве с рабочей группой SVG, создала спецификацию SMIL-анимации, которая представляет универсальный набор XML-признаков анимации. SVG включает особенности анимации, определенные в SMIL и имеет некоторые расширения, определённые в SVG. SMIL использует анимированное или статическое SVG-содержание как медиа-компоненты. SMIL-компоненты используются вместе с SVG и другими приложениями XML для достижения медиа-эффектов.

После загрузки SVG-документа или svg-фрагмента xhtml-документа начинается отсчёт анимационного времени, и реализуется размеченная анимация для всех вложенных svg-элементов. Анимация осуществляется через изменение атрибутов (свойств) АЭ, которые называются целевыми атрибутами (ЦА). Если ЦА наследован потомком, то анимация предка (элемента типа g) распространяется к потомкам, т.е. элементы потомки наследуют анимируемые атрибуты от предков.

В SVG используются пять следующих ЭА:

animate – изменяет числовые (скалярные и векторные) ЦА АЭ с течением времени, т.е. анимирует отдельное свойство АЭ.

animateColor – изменяет цвет АЭ.

animateMotion – перемещает АЭ по заданному пути, может иметь атрибуты path (описывает путь перемещения АЭ), keyPoints (управляет скоростью анимации), rotate (вращает АЭ так, что его ось X изменяет угол наклона к траектории перемещения) и т.д. Элемент mpath как потомок ЭА animateMotion, содержит ссылку на путь перемещения АЭ.

animateTransform – изменяет один из параметров (например, rotate) ЦА АЭ transform.

set – изменяет нечисловые ЦА АЭ, например, свойство “visibility” (видимость), т.е. осуществляет дискретную анимацию.

Следующие два атрибута ЭА идентифицируют ЦА АЭ:

attributeName = “имяАтрибута” – определяет имя ЦА АЭ. Например, для изменения цвета фигуры (заливки) атрибут будет выглядеть так: attributeName="fill".

attributeType = "CSS | XML | auto" – определяет тип ЦА. Возможные значения: CSS – анимация свойств, относящихся к спецификации CSS (шрифт, цвет шрифта, кернинг и другие). XML – анимация свойства, относящегося к SVG-графике (перенос, вращение, искажение и другие). auto – значение по умолчанию, включающее в себя значения свойств CSS и XML.

Атрибут xlink:href = “uri” ЭА указывает на АЭ, который должен находится в текущем фрагменте SVG-документа. Если этот атрибут отсутствует, то в качестве АЭ выступает родитель (предок) текущего ЭА.

Следующие девять атрибутов (общие для всех ЭА) управляют синхронизацией анимации, например, размечают события запуска и прекращения анимации, управляют повторением анимации, сохраняют конечное состояние анимации после её завершения.

1. begin=”begin-value-list” – список (через точку с запятой с возможными пробельными символами) параметров (один или несколько), один из которых запускает анимацию. Значением (begin-value) списка может быть: offset-value | syncbase-value | syncoToPrev-value | event-value | repeat-value | accessKey-value | media-marker-value |

wallclock-sync-value | indefinite. Опишем смысл этих значений, используя форму Бэкуса-Науэра, где символы означают следующее: S – пробельный символ; * – ноль или больше; + – один или больше; ? – ноль или 1; ( ) – группирование; | – различные варианты; “текст” – двойные кавычки вокруг стринговых величин.

1.1 offset-value ::= Clock-value. Анимация начинается через указанное время после открытия документа.

1.2 syncbase-value ::= (Id-value "." ("begin" | "end")) (S? ("+"|"-") S? Clock-value)? Анимация определена относительно начала или конца другой анимации, на которую указывает идентификационная ссылка, с возможным дополнительным временном смещении.

1.3 syncoToPrev-value ::= ("prev.begin" | "prev.end") (S? ("+"|"-") S? Clock-value)? Начало анимации определяется (с возможным временным смещением) относительно начала или конца предыдущего ЭА, имеющего общий родительский элемент с данным элементом, или, если нет такого элемента, то общий фрагмент SVG-документа.

1.4 event-value ::= (Id-value ".")? (event-ref) (S? ("+"|"-") S? Clock-value)? Начало анимации определяется (с возможным временным смещением) относительно указанного события того элемента, на которое сделана идентификационная ссылка. Ссылка может отсутствовать. У каждого элемента есть свой список идентификаторов событий, доступных для данного элемента.

1.5 repeat-value ::= (Id-value ".")? "repeat(" integer ")" (S? ("+"|"-") S? Clock-value)? Начало анимации определяется (с возможным временным смещением) относительно заданного повторения другой анимации, на которую указывает идентификационная ссылка, которая может отсутствовать.

1.6 accessKey-value ::= "accessKey("character")" (S? ("+"|"-") S? Clock-value)? Анимация начинается (с возможным временным смещением) после того как пользователь нажимает клавишу (символ) character.

1.7 media-marker-value ::= Id-value".marker(“marker-name”)". Описывает начало анимации, как именованную метку времени, определяемую элементом данных (media element). Используется, когда SVG выполняется как компонент других XML-языков, поддерживающих типы данных с именованными маркерами, например, SMIL.

1.8 wallclock-sync-value ::= wallclock("wallclock-value"). Начало анимации привязано к текущему реальному времени (синтаксис времени определён в ISO8601).

1.9 indefinite. Начало анимации определяется вызовом метода "beginElement()" или гиперссылкой, указывающей на АЭ.

2. end=”end-value-list”. Список значений параметров, одно из которых завершит анимацию, где end-value может принимать те же значения что и begin-value. Значение "indefinite" определяет конец анимации по "endElement()".

3. dur=”Clock-value” | “media” | “indefinite”. Определяет время простой анимации (ВПА). Clock-value – величина ВПА (больше нуля). “media” – определяет ВПА, как встроенную продолжительность медиа-носителей и допустимо только для элементов, определяющих носителей. indefinite – ВПА равно бесконечности. Если анимация не имеет атрибута dur, то ВПА не определено.

4. min=”Clock-value” | "media". Определяет минимальную активную длительность анимации. Clock-value – минимальное значение длительности в текущем времени. media – определяет минимальное значение активной продолжительности как встроенную продолжительность носителей.

5. max=”Clock-value” | "media" – определяет максимальную активную длительность. Clock-value – максимальное значение активной длительности в текущем времени. media – определяет максимальное значение активной длительности как встроенную продолжительность носителей.

6. restart : "always" | "whenNotActive" | "never". Определяет режим перезапуска анимации. always – анимация может быть перезапущена в любое время (значение по умолчанию). whenNotActive – анимация может быть перезапущена только когда она не активна, т. е. после конца активации, и попытки перезапускать анимацию в течении её активной деятельности игнорируются. never – анимация не может быть перезапущена до окончания продолжительности исходного контейнера времени. Если исходный контейнер времени – фрагмент SVG-документа, тогда анимация не может быть перезапущена до окончания длительности документа.

7. repeatCount=”число” | “indefinite”. Определяет число повторений функции анимации. numeric – числовое значение “с плавающей точкой”, которое определяет число

повторений. Значения должны быть больше нуля. “indefinite” – анимация определена до конца документа.

8. repeatDur=”Clock-value” | "indefinite". Определяет продолжительность повторений. indefinite (неопределенный) – анимация повторяется до закрытия документа.

9. fill="freeze" | "remove". Определяет режим закрепления состояния АЭ. freeze (закрепление) – эффект анимации "закреплен" до закрытия документа или до перезапуска анимации. remove (удалить) – достигнутое состояние АЭ удаляется в конце анимации (значение по умолчанию).

Следующие девять атрибутов ЭА animate, animateMotion, animateColor, animateTransform определяют значения ЦА АЭ. Они обеспечивают управление синхронизацией ключевых значений ЦА, назначают метод интерполяции между этими значениями и регулируют скорость изменения значений.

1. from = "значение" – определяет начальное значение ЦА.

2. to = "значение" – определяет конечное значение ЦА.

3. by = "значение" – определяет смещение относительно достигнутого значения ЦА.

4. values = "список значений" – список (через точку с запятой) значений ЦА (возможны пробельные символы до и после точки с запятой).

5. calcMode = "discrete | linear | paced | spline". Определяет режим интерполяции для значений ЦА, например, из списка values. discrete (дискретный) – значения воспроизводятся без интерполяции. paced – темповая интерполяция, создающая ровный темп (постоянную скорость) изменения значений ЦА. linear – линейная интерполяция значений. Если ЦА не поддерживает линейную интерполяцию (например, для строковых величин), или если атрибут values имеет только одно значение, то атрибут calcMode игнорируется и используется дискретная интерполяция. spline – интерполирует значения согласно кубической функции Безье, где контрольные точки для каждого временного интервала определены в атрибуте keySplines, а время определено в атрибуте keyTimes.

6. keyTimes = "список". Список (через точку с запятой) значений временных долей соответствующих различным значениям ЦА в списке values. Значения в списке keyTimes определены между 0 и 1 (включительно) с плавающей запятой, представляя

долевую разбивку ВПА. Каждое следующее значение времени должно быть больше или равняться предыдущему значению времени. Семантика списка keyTimes зависит от режима интерполяции. Для linear и spline интерполяции в первый момент значение в списке должно быть 0, а в последний момент значение в списке должно быть 1. Для дискретной анимации в первый момент значение в списке должно быть 0. Если режим интерполяции – "paced", то атрибут keyTimes игнорируется.

7. keySplines = "список". Список (через точку с запятой или пробел) контрольных точек кубической функции Безье, связанных со списком keyTimes. Функция Безье управляет скоростью анимации внутри и на концах каждого временного интервала из списка keyTimes. Каждое значение списка keySplines – это набор четырёх вещественных чисел x1 y1 x2 y2, т.е. координат контрольных точек для начала и конца соответствующего временного интервала из списка keyTimes. Все значения списка keySplines, т.е. координаты контрольных точек должны быть в диапазоне от 0 до 1. Наборов контрольных точек должно быть на единицу меньше, чем моментов времени в keyTimes. Атрибут keySplines игнорируется, если значение calcMode не равно "spline".

8. additive = "replace | sum". Значение sum определяет, что анимация аддитивная, т.е. добавляет к начальному значению ЦА и менее приоритетным анимациям достигнутое значение. replace – определяет, что анимация не изменит начальные значение ЦА и других менее приоритетных анимаций (значение по умолчанию). Учитывает значения атрибутов анимации by и to.

9. accumulate = "none | sum". Для повторяющихся анимаций позволяет положиться на достигнутые значения ЦА, накапливающиеся с каждым повторением. sum – определяет что анимация кумулятивная, т.е. каждый повтор анимации (после первой) надстроен на достигнутом значении ЦА от предыдущей анимации. none (значение по умолчанию) – определяет что анимация некумулятивная, т.е. при повторении анимации ЦА начинает изменяться от своего начального значения. Этот атрибут игнорируется, если ЦА не поддерживает суммирование или если ЭА не повторяется.

Кратко рассмотрим ЭА.

ЭА “animate”

ЭА animate позволяет осуществлять анимацию АЭ через изменение с течением времени одного ЦА (свойства). При анимации свойств, названия которых соответст-

вуют спецификации CSS, следует указывать значение CSS атрибута attributeType явным образом. В следующих фрагментах svg-кода анимируется один или два атрибута прямоугольника.

Фрагмент 1.

```
<rect x="75" y="25" width="50" height="50"  fill="none" stroke="red">
  <animate  attributeType="CSS" attributeName="stroke-width"
                                    from="0" to="45" dur="10s" repeatCount="indefinite"/>
</rect>
```

Изменение толщины контура прямоугольника.

Фрагмент 2.

```
<rect y="40" width="20" height="20" fill="red">
  <animate attributeName="x" from="0" to="200" dur="10s" repeatCount="indefinite"/>
</rect>
```

Изменение абсциссы прямоугольника.

Фрагмент 3.

```
<rect  width="20" height="20" fill="red">
  <animate attributeName="x" from="0" to="200" dur="10s" repeatCount="indefinite"/>
  <animate attributeName="y" from="0" to="100" dur="10s" repeatCount="indefinite"/>
</rect>
```

Одновременное увеличение абсциссы и ординаты, в результате чего прямоугольник будет двигаться по диагонали.

Фрагмент 4.

```
<rect x="5" y="5" width="20" height="20" fill="red">
  <animate attributeName="width" from="0" to="195" dur="10s" repeat-
Count="indefinite"/>
  <animate attributeName="height" from="0" to="95" dur="10s" repeatCount="indefinite"/>
</rect>
```

Одновременное увеличение ширины и высоты прямоугольника.

ЭА “set”

ЭА set обеспечивает простые установки значения атрибута на ВПА. Поддерживает все типы атрибутов, включая строки и булевы значения. Атрибуты добавления (addi-

tive) и накопления (accumulate) игнорируются. Атрибут to определяет конечное значение для ЦА. Значение аргумента должно соответствовать типу ЦА. ЭА set позволяет изменять анимируемые свойства не плавно, а скачком.

Фрагмент 5.

```
<!--Анимация при встраивании атрибута set внутри тега ellipse-->
<ellipse cx="100" cy="50"  rx="60" ry="40" fill="lightpink"
                                                    stroke="lightcoral" stroke-width="4">
  <set attributeName="ry" to ="20" begin="1s" dur="2s"/>
</ellipse>
```

В результате выполнения этого кода вертикальный радиус элемента эллипс изменит свой размер на две секунды. Анимация начнется спустя одну секунду после загрузки ролика.

Фрагмент 6.

```
<!--Анимация с применением ссылки-->
<ellipse id="эллипс" cx="100" cy="150"  rx="60"
                                    ry="40" fill="yellow" stroke="orange" stroke-width="4"/>
<set  xlink:href="#эллипс" attributeName="ry" to ="20" begin="1s" dur="2s"/>
```

В этом способе создания анимации к уже имеющемуся элементу обращаются по его id для изменения свойств. Этот способ может применяться для всех видов анимации.

Эллипс в течение секунды после загрузки отображается в исходном виде. Далее, в течение двух секунд происходит изменение вертикального радиуса.

Фрагмент 7.

```
<text x="10" y="50" style="fill:yellow; stroke:red;
                        font-family:Times New Roman; font-size:60; visibility:hidden;">
                                SVG
  <set attributeName="visibility" attributeType="CSS" to="visible" begin="1s"
                                                                    fill="freeze"/>
</text>
<text x="10" y="110" style="fill:yellow; stroke:red;
                        font-family:Times New Roman; font-size:60; visibility:hidden;">
```

графика

```
<set attributeName="visibility" attributeType="CSS" to="visible" begin="2s"
                                                                fill="freeze"/>
</text>
```

Последовательное появление текста. Атрибут visibility элемента text отвечает за отображение текста в SVG-документе.

ЭА “animateColor”

Определяет преобразование цвета с течением времени. Атрибуты from, by и to задают значения цвета. Атрибут values для ЭА состоит из списка значений цвета, отделённых точкой с запятой. Свойство “color-interpolation” применяется для интерполяции цвета.

Фрагмент 8.

```
<ellipse cx="100" cy="50"  rx="80" ry="40" fill="red">
  <animateColor attributeName="fill" from="red" to="yellow" dur="3s"
                                                    repeatCount="indefinite"/>
</ellipse>
```

Атрибуты внутри этого элемента такие же, как и в случае использования элемента animate.

Фрагмент 9.

```
<ellipse cx="100" cy="50"  rx="80" ry="40" fill="red">
  <animateColor attributeName="fill"
                                from="red" to="yellow" begin="2s" dur="8s" fill="freeze"/>
</ellipse>
```

Последовательный переход красного цвета в желтый. Анимация начинается через две секунды после загрузки ролика (begin="2s"). После завершения фигура остается желтой (fill="freeze").

Фрагмент 10.

```
<ellipse cx="100" cy="50"  rx="80" ry="40" fill="red">
  <animateColor attributeName="fill" from="red" to="orange" begin="2s" dur="3s"/>
  <animateColor attributeName="fill" from="orange" to="yellow" begin="prev.end"
                                                                    dur="3s"/>
  <animateColor attributeName="fill" from="yellow" to="green" begin="prev.end"
                                                                    dur="3s"/>
  <animateColor attributeName="fill"
                              from="green"      to="deepskyblue"      begin="prev.end"
dur="3s"/>
  <animateColor attributeName="fill"
                               from="deepskyblue" to="blue" begin="prev.end" dur="3s"/>
  <animateColor attributeName="fill"
                           from="blue" to="purple" begin="prev.end" dur="3s" fill="freeze"/>
</ellipse>
```

Последовательная серия изменения цвета фигуры. Следующий этап анимации начинается тогда, когда заканчивается предыдущий (begin="prev.end").

Фрагмент 11.

```
<text x="10" y="50"
                style="fill:yellow; stroke:red; font-family:Times New Roman; font-size:60;">
                                    SVG - графика
  <animateColor attributeName="fill"
                               from="yellow" to="red" begin="2s" dur="8s" fill="freeze"/>
</text>
```

Изменение цвета заливки текста. Атрибут fill включает в себя также цвет контура текста.

ЭА “animateTransform”

ЭА animateTransform изменяет атрибут transform АЭ. Атрибут type = "translate | scale | rotate | skewX | skewY" этого ЭА указывает тип используемого преобразования. Для значения " translate", каждая индивидуальная величина выражена как tx [, ty]. Для " scale " – как sx [, sy]. Для "rotate" – как rotate-angle [cx cy]. Для "skewX" и "skewY" – как skew-angle. Атрибуты from, by и to указывают диапазоны изменения этих величин.

В списке values величины отделяются точкой с запятой. Когда анимация активна, эффект несуммирующего ЭА animateTransform (additive="replace") должен заместить величину данного атрибута преобразованием, определенным ЭА animateTransform. Эффект суммирования (additive="sum") должен пост-умножить матрицу преобразования, соответствующую преобразованию, определенному этим ЭА animateTransform. Для атрибута calcMode значение по умолчанию “paced” и движение происходит с постоянной скоростью вдоль заданного пути. Элемент animateTransform позволяет "оживлять" трансформации. Как и для прочих элементов анимации нужно указывать начальные и конечные значения.

Фрагмент 12.

```
<rect x="25" y="25" width="50" height="25" fill="skyblue" stroke="red" stroke-width="2">
  <animateTransform attributeName="transform" attributeType="XML" type="translate"
                                    from="0,0" to="100,75" begin="1s" dur="2s" fill="freeze"/>
</rect>
```

В этом случае прямоугольник будет перемещаться из точки с координатами 0,0 в точку с координатами 100,75 в результате применения команды translate. Для элемента animateTransform указывается атрибут attributeName="transform" поскольку в анимации используется встроенная команда SVG – графики.

Фрагмент 13.

```
<rect x="25" y="25" width="50" height="25" fill="skyblue" stroke="red" stroke-width="2">
  <animateTransform attributeName="transform" attributeType="XML"
                type="translate" from="0,0" to="100,75" begin="1s" dur="2s" fill="freeze"/>
</rect>
```

Перемещение прямоугольника. Анимация команды translate.

Фрагмент 14.

```
<!--Движущийся прямоугольник-->
<rect x="25" y="25" width="50" height="25" fill="blue" stroke="red" stroke-width="2">
 <animateTransform attributeName="transform" attributeType="XML"
                type="translate" from="0,0" to="100,75" begin="1s" dur="2s" fill="freeze"/>
```

```
</rect>
<!--Первый неподвижный прямоугольник-->
<rect x="25" y="25" width="50" height="25" fill="skyblue" stroke="red" stroke-width="2"/>
 <!--Второй неподвижный прямоугольник-->
<rect x="25" y="25" width="50" height="25" fill="palegreen" stroke="red"
                                                stroke-width="2" transform="translate(100,75)"/>
<!--Перемещение прямоугольника на фоне неподвижных объектов-->
<rect x="25" y="50" width="50" height="25"
                                                fill="palegreen" stroke="red" stroke-width="2">
  <animateTransform attributeName="transform" type="scale"
                                        from="1,1" to="2,1" begin="0" dur="2s" fill="freeze"/>
</rect>
```

Вытягивание прямоугольника в горизонтальном направлении. Анимация команды scale.

Фрагмент 15.

```
<!--Прямоугольник - рамка рисунка-->
<rect x="5" y="5" width="390" height="390" fill="none"  stroke="blue" stroke-width="2"/>
<!--Исходная фигура-->
<g>
  <rect x="200" y="200" width="100" height="50"
                                                fill="yellow"  stroke="red" stroke-width="4"/>
  <rect x="200" y="200" width="50" height="25"
                                                  fill="green" stroke="red" stroke-width="4"/>
</g>
<!--Вращающаяся фигура-->
<g>
  <rect x="200" y="200" width="100" height="50"
                                                fill="yellow"  stroke="red" stroke-width="4"/>
  <rect x="200" y="200" width="50" height="25"
                                                  fill="green" stroke="red" stroke-width="4"/>
```

```
 <animateTransform attributeName="transform" attributeType="XML"
                  type="rotate" from="0" to="360" begin="2s" dur="10s" fill="freeze"/>
</g>
```

Вращение прямоугольника относительно начала системы отсчета рисунка. Анимация команды rotate. Для просмотра невидимой части вращения рисунок можно перетаскивать (Alt + кнопка мыши).

Фрагмент 16.

```
<!--Исходная фигура-->
<g>
 <rect x="200" y="200" width="100" height="50"
                                    fill="yellow" stroke="red" stroke-width="4"/>
 <rect x="200" y="200" width="50" height="25"
                                     fill="green" stroke="red" stroke-width="4"/>
</g>
<!--Вращающаяся фигура-->
<g>
 <rect x="200" y="200" width="100" height="50"
                                    fill="yellow" stroke="red" stroke-width="4"/>
 <rect x="200" y="200" width="50" height="25"
                                     fill="green" stroke="red" stroke-width="4"/>
 <animateTransform attributeName="transform" attributeType="XML"
                                type="rotate" from="0, 200, 200" to="360, 200, 200"
                                     begin="2s" dur="10s" fill="freeze"/>
</g>
```

Вращение объекта относительно его левой верхней точки.

Фрагмент 17.

```
<!--Исходная фигура-->
<g>
 <rect x="10" y="10" width="100" height="50"
                                    fill="yellow" stroke="red" stroke-width="4"/>
 <rect x="10" y="10" width="50" height="25"
```

```
                                                fill="green" stroke="red" stroke-width="4"/>
</g>
<!--Трансформированная фигура -->
<g>
  <rect x="10" y="10" width="100" height="50"
                                                fill="yellow" stroke="red" stroke-width="4"/>
  <rect x="10" y="10" width="50" height="25"
                                                fill="green" stroke="red" stroke-width="4"/>
  <animateTransform attributeName="transform" attributeType="XML"
                          type="skewY" from="0" to="45" begin="2s" dur="10s" fill="freeze"/>
</g>
```

Искажение объекта в вертикальном направлении. Анимация команды skewY.

Фрагмент 18.

```
<g>
  <g>
    <circle cx="500" cy="400" r="300" fill="lightskyblue" stroke="blue" stroke-
width="30"/>
    <ellipse cx="500" cy="500" rx="150" ry="100" fill="none"
                                                stroke="blue" stroke-width="30"/>
    <rect x="300" y="370" width="400" height="130" fill="lightskyblue"/>
    <ellipse cx="350" cy="300" rx="100" ry="50" fill="white"
                                                stroke="blue" stroke-width="30"/>
    <ellipse cx="650" cy="300" rx="100" ry="50" fill="white"
                                                stroke="blue" stroke-width="30"/>
    <circle cx="350" cy="300" r="25" fill="black"/>
    <circle cx="650" cy="300" r="25" fill="black"/>
    <!--Увеличение фигуры в два раза -->
    <animateTransform attributeName="transform" type="scale"
                                          from="1" to="2" begin="0" dur="1s" fill="freeze"/>
  </g>
```

```
<!-- Вращение фигуры относительно нового центра с координатами 1000,800,
                    полученных в результате удвоения старых: cx="500"*2  cy="400"*2 -->
<animateTransform attributeName="transform" attributeType="XML" type="rotate"
  from="0, 1000, 800" to="360, 1000, 800" begin="1" dur="1s" repeatCount="indefinite"/>
</g>
<g>
  <g>
    <text x="-900" y="2000" style="fill:yellow; stroke:red;
                                                font-family:Times New Roman; font-size:500;">
                                        SVG – графика
     <!-- Изменение цвета текста -->
     <animateColor attributeName="fill"
                        from="yellow" to="red" begin="2s" dur="2s" repeatCount="indefinite"/>
 </text>
<!-- Увеличение ширины  текста в 1,2 раза -->
 <animateTransform attributeName="transform" type="scale"
                        from="1,1" to="1.2,1" begin="2" dur="1s" repeatCount="indefinite"/>
</g>
<!-- Увеличение высоты  текста в 1,2 раза -->
<animateTransform attributeName="transform" type="scale"
                        from="1,1" to="1,1.2" begin="2" dur="1s" repeatCount="indefinite"/>
</g>
```

При загрузке ролика мы видим текст и увеличивающуюся в течении секунды группу объектов. В течении следующей секунды группа начинает вращаться, а спустя еще одну секунду начинает увеличиваться текст с одновременным изменением всего цвета.

ЭА “animateMotion”

ЭА animateMotion перемещает по описанному пути один из следующих АЭ: g, defs, use, image, switch, path, rect, circle, ellipse, line, polyline, polygon, text, clipPath, mask, a, foreignObject. У него могут быть следующие атрибуты:

from, by, to и values. Содержат координаты (x, y). Например, to="33,15" определяет конечную точку прямолинейного пути с координатами x=33, y=15 его конца. Величины координат отделены, по крайней мере, одним пробелом или запятой. Дополнительные пробелы разрешаются. Например, " 10,20; 30,20; 30,40 " или " 10mm, 20mm; 30mm, 20mm; 30mm, 40mm". Каждая пара координат отделяется друг от друга точкой с запятой. Атрибуты from, by, to и values определяют форму траектории на плоскости.

path = "ДП". Описывает траекторию перемещения. Влияние траектории анимации заключается в присоединении дополнительной матрицы преобразования СК (МПСК) к текущей МПСК АЭ. В результате текущая пользовательская СК перемещается относительно осей X и Y по рассчитанным координатам x и y.

calcMode = "discrete | linear | paced | spline". Определяет вид интерполяции точек списка values, описывающих траекторию перемещения. Заданный по умолчанию режим интерполяции для “animateMotion” – это “paced”. Он обеспечивает движение с постоянной скоростью по указанному пути.

keyPoints = "<list-of-numbers>". keyTimes = "<list-of-numbers>". Список keyPoints (разделённых точкой с запятой) вещественных чисел (между 0 и 1), указывающий доли перемещения АЭ по заданному пути в соответствии со временем, установленном в списке keyTimes. Каждая доля продвижения в списке keyPoints соответствует временному интервалу в списке keyTimes. В списке keyPoints должно быть столько же величин сколько их в списке keyTimes.

rotate = "<angle> | auto | auto-reverse". auto указывает, что АЭ вращается вместе с касательной к траектории движения. auto-reverse указывает, что касательная к траектории повёрнута на 180 градусов. Угловой наклон к траектории можно задавать углом angle относительно оси абсцисс текущей СК пользователя. Атрибут rotate присоединяет дополнительную МПСК к текущей пользовательской СК так, что преобразование поворота следует после преобразования смещения вдоль траектории.

Определить путь АЭ можно двумя способами. Во-первых, атрибут пути path можно поместить прямо в ЭА 'animateMotion', используя в качестве значений любые ДП, во-вторых, его потомок mpath может сослаться на внешний элемент path, определяющий траекторию движения АЭ. Причём ЭА mpath отменяет действие атрибута path ЭА animateMotion.

ЭА “animateMotion” может иметь атрибуты добавления (additive) и накопления (accumulative).

Несколько ЭА 'animateMotion' могут одновременно ссылаться на один АЭ действуя аддитивно относительно друг друга. Преобразования, которые обеспечивают ЭА 'animateMotion', всегда дополнительны к любому преобразованию ЦА АЭ или к любым ЭА 'animateTransform'.

По точкам пути рассчитывается дополнительная МПСК которая вызывает перемещение АЭ относительно осей текущей СК пользователя. Это преобразование применяется после любых преобразований, вызываемых атрибутом transform АЭ или любой анимацией этого атрибута, вызываемого ЭА animateTransform.

Если calcMode имеет значение discrete, linear или spline, и атрибут keyPoints отсутствует, то количество значений определяется равным количеству точек пути, если в пути нет команды “move to”. Команда “move to” в пути (т.е. не в начале описания пути) не считается дополнительной точкой при расчёте длительности или при привязке значений keyTimes, keySplines и keyPoints. При значении paced для calcMode, считается, что все команды “move to” имеют нулевую длину (т.е. выполняются мгновенно), и не учитываются при расчёте шага.

Для более гибкого контроля скорости вдоль траектории, атрибут keyPoints предоставляет возможность задавать скорость для каждого значения списка keyTimes. Если задан атрибут keyPoints, значения keyTimes ассоциируются со значениями keyPoints, а не с точками, заданными атрибутом values или атрибутом path.

Для ЭА 'animateMotion' действуют следующие правила замены. Относительно определения траектории – элемент mpath заменяет атрибут path, который заменяет values, который заменяет from/by/to. Относительно определения точек, соответствующих значениям атрибута keyTimes, атрибут keyPoints заменяет атрибут path, который заменяет values, который заменяет from/by/to.

В любое время t на траектории анимации длительностью dur рассчитываемые координаты (x, y) определяются точкой (x, y), которая смещена на величину t/dur относительно начала траектории.

Правила отмены ЭА animateMotion следующие. ЭА mpath, вложенный в ЭА animateMotion, отменяет действие атрибута path ЭА animateMotion, который отменяет действие атрибутов from/by/to.

Одна из самых интересных возможностей SVG-анимации – движение объекта вдоль произвольной траектории. В качестве объекта может быть применяться простой элемент, группа элементов, текст или растровое изображение. Рассмотрим фрагменты кода.

Фрагмент 19.

```
<rect x="-10" y="-10" width="20" height="20" fill="red">
  <animateMotion path="M20 100 C20 0 180 0 180 100" dur="10s" rotate="auto"
                                                    repeatCount="indefinite"/>
</rect>
```

Здесь красный квадратик будет двигаться вдоль траектории, определяемой элементов path. Для центровки квадратика относительно траектории заданы смещенные на половину ширины значения координат x="-10" y="-10". Атрибут rotate="auto" указан для автоматического поворота объекта вокруг своей оси при движении. Сама траектория не отображается в окне документа.

Фрагмент 20.

```
<path d="M20 100 C20 0 180 0 180 100" style="fill: none; stroke:blue;stroke-width:2;"/>
<rect x="-10" y="-10" width="20" height="20" fill="red">
  <animateMotion path="M20 100 C20 0 180 0 180 100" dur="10s" rotate="auto"
                                                    repeatCount="indefinite"/>
  <!-- А вот так движение повторится три раза и затем остановится:-->
  <animateMotion path="M20 100 C20 0 180 0 180 100" dur="10s" rotate="auto"
                                                  repeatCount="3" fill="freeze"/>
</rect>
```

Движение простого элемента вдоль траектории path. Здесь (и далее) для отображения траектории был добавлен отдельный элемент path.

Фрагмент 21.

```
<!--Первая траектория-->
<path d="M20,20 180,20 180,180 20, 180 20,20 z"
```

```
                                        style="fill: none; stroke:blue;stroke-width:2;"/>
<!--Прямоугольник-->
<rect x="-10" y="-10" width="20" height="20" fill="red">
  <animateMotion path="M20,20 180,20 180,180 20, 180 20,20 z"
                                        dur="10s" rotate="auto" repeatCount="indefinite"/>
</rect>
<!-- Вторая траектория-->
<path d="M20,220 180,220 180,380 20, 380 20,220 z"
                                        style="fill: none; stroke:green;stroke-width:2;"/>
<!-- Текст -->
<text style="font-family:Times New Roman; stroke:orangered; fill:lime;font-size:30;">
                                        текст
  <animateMotion path="M20,220 180,220 180,380 20, 380 20,220 z"
                                        dur="15s" rotate="auto" repeatCount="indefinite"/>
</text>
```

Одновременное движение двух объектов – элемента rect и текста

Фрагмент 22.

```
<!--Первая траектория-->
<path d = "M20,20 180,20 180,180 20, 180 20,20 z"
                                        style="fill: none; stroke:blue;stroke-width:2;"/>
<!-- Вторая траектория-->
<path d = "M180,180 340,180 340,340 180, 340 180,180 z"
                                        style="fill: none; stroke:green;stroke-width:2;"/>
<!-- Текст на траектории -->
<text style="font-family:Times New Roman; stroke:orangered; fill:lime;font-size:30;">
                                        Текст
  <animateMotion
          path="M20,20 180,20 180,180 340,180 340,340 180,340 180,180 20,180 20,20 z"
                                        dur="15s" rotate="auto" repeatCount="indefinite"/>
</text>
```

Движение текста вдоль сложной траектории

Фрагмент 23.

```
<!--Первая траектория-->
<path d = "M20,20 180,20 180,180 20, 180 20,20 z"
                                        style="fill: none; stroke:blue;stroke-width:2;"/>
<!-- Вторая траектория-->
<path d = "M180,180 340,180 340,340 180, 340 180,180 z"
                                       style="fill: none; stroke:green;stroke-width:2;"/>
<!-- Объект на траектории -->
<path d="M0 0 v -5 h20 v-10 l10 15 l-10 15 v-10 h-20 v-5"
                                         style="fill: red; stroke:black;stroke-width:1;">
  <animateMotion
          path="M20,20 180,20 180,180 340,180 340,340 180,340 180,180 20,180 20,20 z"
                                      dur="15s" rotate="auto" repeatCount="indefinite"/>
</path>
```

Движение элемента path вдоль траектории, определяемой другим элементом path.

Фрагмент 24.

```
<!--Первая траектория-->
<path d = "M20,20 180,20 180,180 20, 180 20,20 z"
                                        style="fill: none; stroke:blue;stroke-width:2;"/>
<!-- Вторая траектория-->
<path d = "M180,180 340,180 340,340 180, 340 180,180 z"
                                       style="fill: none; stroke:green;stroke-width:2;"/>
<!-- Объект на траектории -->
<image height="50" xlink:href="ducky.png" width="46">
  <animateMotion
          path="M20,20 180,20 180,180 340,180 340,340 180,340 180,180 20,180 20,20 z"
                                      dur="15s" rotate="auto" repeatCount="indefinite"/>
</image>
```

Движение растрового изображения. В этом примере (в отличие от предыдущих), привязка к траектории осуществляется по умолчанию к левому верхнему углу изображения.

6. Примеры использования ЭА animate

Напомним, что ЭА animate позволяет осуществлять анимацию svg-элемента через изменение с течением времени одного из его атрибутов (свойств). Изменяемый атрибут называют целевым атрибутом (ЦА). В качестве анимируемого элемента (АЭ) в примерах рассматривается графический элемент <*circle*>, а в качестве ЦА взят его атрибут *cx*. АЭ может быть предком ЭА, как в примере А, или ЭА обращается к АЭ по ссылке, как в примере В.

Пример А

```
<?xml version="1.0" standalone="no"?>
<svg viewBox="0 0 2000 1500" xmlns="http://www.w3.org/2000/svg" version="1.1">
 <circle cx="200" cy="200" r="50">
  <animate attributeName="cx" from="10" to="1000" dur="3s"/>
 </circle>
</svg>
```

Пример В

```
<?xml version="1.0" standalone="no"?>
<svg viewBox="0 0 2000 1500" xmlns="http://www.w3.org/2000/svg" version="1.1"
                                                    xmlns:xlink="http://www.w3.org/1999/xlink>
 <circle id="круг" cx="200" cy="200" r="50"/>
 <animate xlink:href="# круг" attributeName="cx" from="10" to="1000" dur="3s"/>
</svg>
```

Во всех примерах используется 1-й способ обращения к АЭ (пример А), и в тексте разметки примеров первая строка и повторяющиеся элементы не приводятся.

Примеры простой анимации

Пример 1. *from-to* анимация.

```
<animate attributeName="cx" from="10" to="1000" dur="3s"/>
```

ЦА *cx* изменяется от значения 10 до значения 1000 в течении 3 сек. Здесь *attributeName* определяет имя ЦА, *from* определяет начальное значение ЦА, *to* определяет конечное значение ЦА, *dur* определяет время простой анимации (ВПА). По умолчанию значения величин *from, to* измеряются в пикселах, *dur* – действительное число, измеряемое в секундах. Атрибут *dur* может принимать значения *clock-value, "media"*, или *"indefinite"*. *clock-value* – значение ВПА. *"media"* – ВПА соответствует внутренней продолжительности данных и применяется только для элементов, определяющих данные. *"indefinite"* – ВПА неограниченно, т.е. действует до закрытия документа.

Пример 2. *from-by* анимация.

```
<animate attributeName="cx" from="10" by="990" dur="2.5s"/>
```

Атрибут *by* определяет смещение конечного значения ЦА относительно начального значения, т.е. ЦА изменяется от начального значения 10 до значения 10+990=1000 в течении 2.5 сек.

Пример 3. *by* анимация.

```
<animate attributeName="cx" by="800" dur="3s"/>
```

В качестве начального значения ЦА берётся его исходное значение, т.е. значение 200 атрибута *cx* АЭ *circle*. К этому значению прибавляется число 800, т.е. через 3 секунды ЦА примет значение 1000.

Пример 4. *to* анимация.

```
<animate attributeName="cx" to="1000" dur="3s"/>
```

В качестве начального значения ЦА берётся его исходное значение, т.е. значение 200 атрибута *cx*. Конечным является значение 1000 атрибута *to* ЭА, т.е. от значения 200 через 3 секунды ЦА увеличивается до значения 1000.

Пример 5. Анимация дискретного ЦА.

```
<circle cx="200" cy="200" r="50" fill="black">
  <animate attributeName="fill" to="blue" dur="3s"/>
<circle/>
```

Цвет заливки круга меняется дискретно от чёрного до голубого в течении 3-х секунд, и в начале 4-й секунды заливка вновь принимает чёрный цвет.

Пример 6. Анимация по списку (числовых) значений.

```
<animate attributeName="cx" values="400;1000;400" dur="6s" calcMode="linear"/>
```

Атрибут *values* определяет список (через точку с запятой) целевых значений ЦА. В примере ЦА будет изменяться в течение 6-ти секунд от значения 400 до 1000 и обратно от 1000 до 400. В первые 3 секунды значение *cx* возрастает по линейному закону от 400 до 1000, и в следующие 3 секунд уменьшается по линейному закону от 1000 до 400. По истечении 6-ти секунд ЦА примет значение 200, что соответствует значению атрибута *cx* АЭ.

Пример 7. Анимация по списку строковых значений.

<animate attributeName="fill" dur="10s" values="black; white; red; yellow" />

Цвет линии в течении 10 сек. меняется от чёрного до белого, от него к красному и далее к жёлтому. В начале 11-й секунды круг исчезает, так как его заливка имеет белый цвет.

Пример 8. Линейная анимация по списку значений (с равными временными отрезками).

<animate attributeName="cx" values="40;1500;1200" dur="6s" calcMode="linear"/>

Значение ЦА изменяется от 40 до 1500 в первые 3 секунды и затем с 1500 до 1200 за вторые 3 сек., т.е. время на каждое изменение отводится одинаковое. Поэтому в течение второй половины анимации величина ЦА изменяется (уменьшается) медленнее. Атрибут *calcMode* описывает режим интерполяции заданных значений ЦА. Его значения *discrete, linear, paced, spline* определяют режим интерполяции для значений ЦА (по умолчанию *linear*). *discrete* (дискретный) – значения воспроизводятся без интерполяции. *linear* – линейная интерполяция. *paced* – определяет темповую интерполяцию, т.е. интерполяцию для создания ровного темпа (постоянной скорости) изменения значений ЦА. *spline* – интерполирует значения в списке *values* согласно функции времени, определенной кубической функцией Безье.

Пример 9. Темповая анимация по списку значений ЦА (с постоянной скоростью).

<animate attributeName="cx" values="40;1500;1200" dur="6s" calcMode="paced"/>

Реализуется постоянный темп (скорость) изменения ЦА от значения к значению. Поэтому увеличение ЦА на величину 1460 протекает дольше чем уменьшение на величину 300, так как скорость изменения ЦА от значения 40 до 1500 равна скорости изменения ЦА от 1500 до 1200.

Пример 10. Дискретная анимация по списку значений.

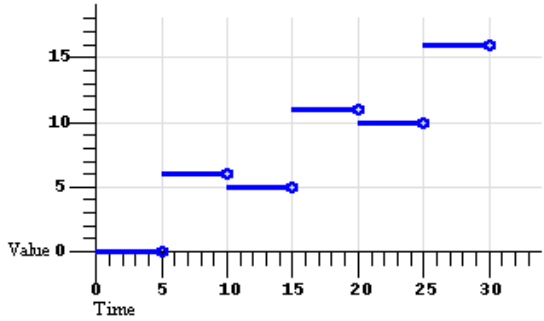
<circle cx="200" cy="200" r="50" fill="black">
<animate attributeName="fill" dur="10s"
values="black; white; red; yellow" calcMode="discrete"/>
<circle/>

Цвет заливки меняется дискретно и застывает на чёрном цвете, т.к. ЦА *fill* АЭ имеет значение *black*.

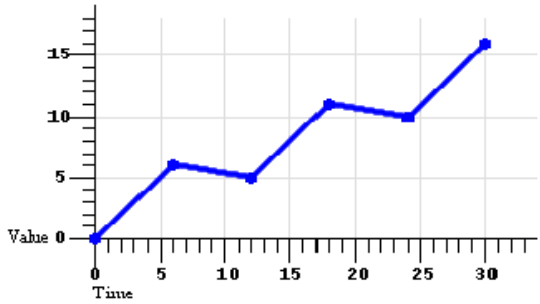
Пример 11. Анимация по списку значений для возможных значений атрибута calcMode ЭА.

<animate dur="30s" values="0; 6; 5; 11; 10; 16" calcMode="[as specified]" />

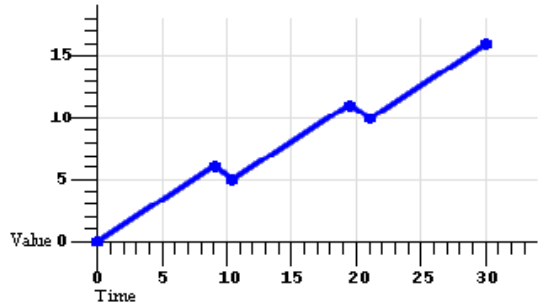
Анимация изменяет значения за 30 секунд в зависимости от способа интерполяции, заданного в *calcMode*:



а)



б)



в)

Рис. 1. Дискретная (а), линейная (б) и темповая (в) анимация

Пример 12. Линейная анимация по списку значений с временной разбивкой.

<animate attributeName="cx" dur="10s" values="0; 1000; 500"
keyTimes="0; .8; 1" calcMode="linear"/>

Величины *keyTimes* заставляют ЦА расти от 0 до 1000 за 8 секунд, что соответствует 80% ВПА (10 сек.), и уменьшаться от 1000 до 500 за оставшиеся 2 секунды, т.е. до конца анимации. Значения в списке *keyTimes* определены между 0 и 1 (включительно) с плавающей запятой, представляя пропорциональное смещение времени. Каждое следующее значение времени должно быть больше или равняться предыдущему значению времени. Семантика списка *keyTimes* зависит от режима интерполяции. Для *linear* и *spline* интерполяции в первый момент значение в списке должно быть 0, а в последний момент значение в списке должно быть 1. Для дискретной анимации в пер-

вый момент значение в списке должно быть 0. Если режим интерполяции – *"paced"*, атрибут *keyTimes* игнорируется.

Пример 13. Дискретная анимация по списку значений с временной разбивкой.

```
<animate attributeName="cx" dur="10s" values="0; 1000; 500"
                                        keyTimes="0; .8; 1" calcMode="discrete"/>
```

Величины *keyTimes* заставляют ЦА измениться от 0 до 1000 дискретно за 8 секунд, что соответствует 80% ВПА (10 сек.), и уменьшаться от 1000 до 500 через 2 сек.

Пример 14. Сплайновая анимация по списку значений.

```
<animate attributeName="c2" dur="10s" values="0; 1000; 500"
                        keyTimes="0; .8; 1" calcMode="spline" keySplines=".5 0 .5 1; 0 0 1 1" />
```

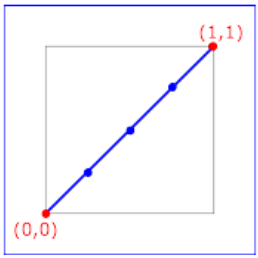
Атрибуты *keySplines* и *keyTimes* обеспечивают управление скоростью анимации между значениями ЦА из списка *values*. ЦА имеет значение "0" в начале анимации, "1000" по истечении 80% ВПА (через 8 сек.) и "500" в конце анимации. Значения *keySplines* определяют кубическую кривую Безье для интерполяции перехода между величинами 0 – 1000 и 1000 – 500. В примере сплайн вызывает плавное начало и окончание эффекта увеличения ЦА от 0 до 1000 за 8 сек. (т.е. между *keyTimes* 0 и .8, и *values* "0" и "1000"), и линейную интерполяцию для уменьшения ЦА от 1000 до 500 за 2 сек. (т.е. между *keyTimes* .8 и 1, и *values* "1000" и "500"). *keySplines = "<list>"* – список (через точку с запятой) контрольных точек функций Безье связанный со списком *keyTimes*. *keySplines* определяет кубические функции Безье, которые управляют изменением скорости (ускорением) на концах и внутри соответствующего временного интервала списка keyTimes. Каждое описание контрольных точек – это набор значений *x1 y1 x2 y2*, отделённых точкой с запятой. Значения координат должны быть в диапазоне от 0 до 1. Наборов контрольных точек должно быть на единицу меньше, чем моментов времени в *keyTimes*. Атрибут *keySplines* игнорируется, если *calcMode* не установлен на *"spline"*. Если ЦА не поддерживает линейную интерполяцию (например, для строк), или если атрибут *values* имеет только одно значение, то атрибут *calcMode* игнорируется и используется дискретная интерполяция.

Пример 15. Анимация по списку 2-х значений ЦА с типовыми наборами атрибута keySplines.

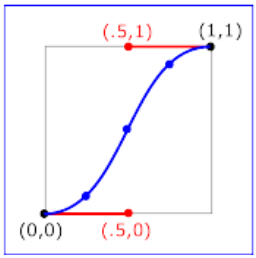
```
<animate dur="4s" values="10; 20" keyTimes="0; 1"
```

calcMode="spline" keySplines=*{значения из таблицы 2}* />

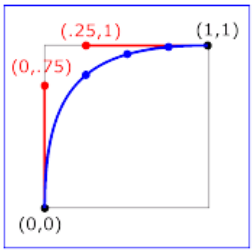
Значения анимируемой величины в таблице для каждого из четырёх значений атрибута keySplines.



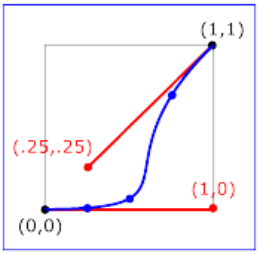
keySplines="0 0 1 1"



keySplines=".5 0 .5 1"



keySplines="0 .75 .25 1"



keySplines="1 0 .25 .25" (по умолчанию)

Рис.2.

Каждый график на рисунке иллюстрирует эффект действия атрибута *keySplines* для одного интервала значений ЦА, т.е. между соответствующими парами величин в списках *keyTimes* и *values*. Горизонтальную ось можно считать входным временем, а вертикальную - выходным временем. В следующей таблице приведены соответствующие значения ЦА.

Значение keySplines	Начальное значение ЦА	Значения ЦА после 1 сек.	Значения ЦА после 2 сек.	Значения ЦА после 3 сек.	Конечное значение
0 0 1 1	10.0	12.0	15.0	17.5	20.0
.5 0 .5 1	10.0	11.0	15.0	19.0	20.0
0 .75 .25 1	10.0	18.0	19.3	19.8	20.0

1 0 .25 .25	10.0	10.1	10.6	16.9	20.0

Пример 16. Сплайновая from-to анимация.

```
<animate attributeName="cx" from="10" to="1000" dur="10s"
                                keyTimes="0.0; 0.8" calcMode="spline" keySplines=".5 0 .5 1" />
```

Атрибуты *keyTimes* и *keySplines* могут также быть использованы с *from/to/by.* Значение ЦА меняется с 10 до 1000 по закону, описанному значениями списков *keySplines* и *keyTimes*.

Примеры сложной анимации

По умолчанию, когда анимация заканчивается, величина ЦА принимает исходное значение. Атрибут *fill* используется для закрепления значения ЦА АЭ после окончания ВПА. Атрибут *fill* может принимать значения: *"freeze"* или *"remove"*. *freeze* (закрепление) – эффект анимации "закреплен" до закрытия документа или до перезапуска анимации. *remove* (удалить) – эффект анимации удаляется в конце анимации (значение по умолчанию). Атрибут *begin* указывает время по истечении которого (после открытия документа) начнётся анимация. Анимацию можно повторять, используя атрибуты *repeatCount*, *repeatDur*. Анимация может быть определена как кумулятивная или некумулятивная, когда она повторяющаяся. Анимация может быть определена как аддитивная или неаддитивная, когда объединяется с основным значением ЦА. Аддитивное и кумулятивное поведение повторяющейся анимации контролируется атрибутами *additive* и *accumulate* соответственно.

Пример 1. Анимация с закреплением.

```
<circle cx="200" cy="500" r="50">
 <animate begin="2s" dur="3s" attributeName="cx" by="1000" fill="freeze" />
</circle>
```

Рассматриваемая анимация начинается через 2 секунды после открытия документа и продолжается 3 секунды. Значение ЦА зафиксируется на величине 1200 через 3 секунды после начала изменения.

Пример 2. Анимация с заданным числом повторений

```
<line y1="200" x2="3" y2="200" stroke="black" stroke-width="10">
 <animate begin="5s" dur="10s" attributeName="x2" by="100"
                                repeatCount="2.5" fill="freeze" />
```

</line>

ВПА – 3 сек. Атрибут repeatCount задаёт количество повторений простой анимации (2.5 раза), т.е. анимация повторяется два раза полностью и затем половину ВПА. Величина ЦА в конце активной продолжительности анимации принимает значение 53. RepeatCount : numeric | "indefinite" - управляет повторами анимации. Значение numeric (вещественное число) устанавливает количество повторов анимации. Дробная часть представляет часть длительности одного повтора анимации. "*indefinite*" - анимация повторяется до конца (закрытия) документа. Время активной анимации (ВАА) – 30 сек.

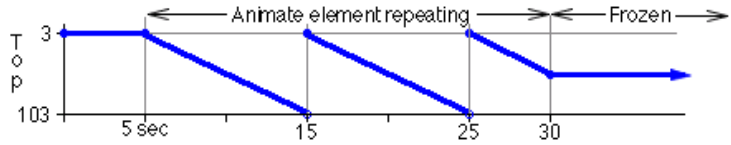


Рис.3.Комбинированная анимация с использованием атрибутов repeat и fill="freeze"

Пример 3. Анимация, в которой ВПА меньше ВАА.

<animate attributeName="x2" from="0" to="10" dur="2.5s" repeatDur="7s"/>

Анимация длится в течении 7 сек. Проигрывается полностью два раза (по 2.5 сек.). В последней (частичной) итерации ЦА изменится в диапазоне от 0 до 8. *RepeatDur* : *clock-value* | "indefinite" – управляет общим временем анимации, т.е. в данном случае - ВАА. *clock-value* – устанавливает длительность ВАА. "*indefinite*" – анимаця повторяется бесконечно (т.е. до конца документа).

Пример 4. Анимация, в которой ВПА больше ВАА.

<animate attributeName="x2" from="10" to="20" dur="10s" repeatDur="5s"/>

ВПА больше, чем длительность, определенная repeatDur. Активная длительность прервет ВПА. ЦА изменится от 10 до15 в течение 5 сек.

Пример 5. Аддитивная анимация.

<line y1="200" x2="10" y2="200" stroke="black" stroke-width="10">
<animate begin="5s" dur="3s" attributeName="x2" by="100" fill="freeze" />
</line>

ЦА увеличивается с 15 пикс. до 115 пикс. за 10 сек. Если бы анимация не была объявлена аддитивной, то ЦА изменился бы от 5 до 105 пикс. в течение 10 сек. *Additive* = "replace | sum" - определяет, является ли анимация аддитивной. *sum* - значение анимации складывается с базовым значением атрибута и с другими значениями анимаций с меньшим приоритетом. *replace* (по умолчанию) - значение анимации замещает базовое значение атрибута и значения анимаций с меньшим приоритетом.

Пример 6. Неограниченная во времени кумулятивная анимация.

```
<animate dur="10s" repeatDur="indefinite" attributeName="x2" from="20" by="10"
                                    accumulate="sum" begin="accessKey(d)" end="click"/>
```

Атрибут accumulate не следует путать с атрибутом additive. Атрибут additive определяет, как анимация объединяется с другими анимациями и базовой величиной ЦА. Часто авторы считают, что кумулятивная анимация будет или должна быть аддитивной, но это не так. В этом примере анимация кумулятивная, но не аддитивная. Поэтому ЦА изменится с 20 до 30 пикс., затем с 30 до 40 и т.д.. Из-за свойства кумулятивности второе повторение анимации стартует с значения x2=30 (как результата 1-й итерации). Анимация начинается после нажатия клавиши d и заканчивается после “клика” мышкой.

Пример 7. Аддитивная кумулятивная анимация с повторением.

```
<line y1="200" x2="20" y2="200" stroke="black" stroke-width="10">
  <animate attributeName="x2" from="0px" to="10px" dur="10s"
                                    additive="sum" accumulate="sum" repeatCount="5"/>
</line>
```

ЦА растёт с каждым повторением анимации. В конце первого повторения ЦА имеет значение 30 пикселей. В конце второго повторения ЦА равен 40 пикселей. В конце пятого повторения - 70 пикселей.

Пример 8. Анимация двоичного свойства

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

Пример 9. Кумулятивная анимация списка значений.

```
<animate attributeName="x2" dur="5s" values="0; 300; 200"
                                    accumulate="sum" repeatCount="3" />
```

ЦА в течение 5 секунд сначала увеличивается от 0 до 300 пикселов, затем уменьшается до 200 пикс. Это повторяется 3 раза. А именно, следующие 5 секунд ЦА растёт с 200 до 500 и падает до 400. Далее за 5 секунд ЦА растёт с 400 до 700 и уменьшается с 700 до 600 пикселов. По истечении 15 сек ЦА возвращается к первоначальному значению.

Пример 10. Аддитивная кумулятивная анимация списка значений.

```
<line y1="200" x2="3" y2="200" stroke="black" stroke-width="10">
  <animate attributeName="x2" dur="5s" values="0; 15; 10" additive="sum"
       accumulate="sum" repeatCount="10" />
</line>
```

Пульсирующий рост ЦА до 100 пикс. за 50 сек. Каждый раз ЦА сначала увеличивается на 15 пикс. и затем уменьшается до 10 пикс. Анимация повторяется и строится на предыдущей величине. В конечном счете увеличится на 120 (20+100) пикс. после всех 10 повторений.

7. Примеры использования ЭА animateMotion

ЭА animateMotion позволяет перемещать ЦЭ по заданному пути. В качестве ЦЭ в примерах рассматривается ГЭ rect (прямоугольник). Путь перемещения задаётся либо атрибутами (from, to, by, values, path) ЭА animateMotion, либо ЦЭ может быть предком ЭА, как в примере А, или ЭА обращается к ЦЭ по ссылке, как в примере В.

Пример А

```
<?xml version="1.0" standalone="no"?>
<svg  viewBox="0 0 2000 1500" xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect width="200" height="200">
    <animateMotion from="10, 20" to="2000, 1000" dur="3s"/>
  </rect>
</svg>
```

Пример В

```
<?xml version="1.0" standalone="no"?>
```

```
<svg viewBox="0 0 2000 1500" xmlns="http://www.w3.org/2000/svg" version="1.1">
 <line  y1="200" x2="10" y2="200" />
 <animate attributeName="x2" from="10" to="1000" dur="3s"/>
</svg>
```

Во всех примерах используется 1-й способ обращения (пример А) к ЦЭ, и в тексте разметки примера приводится только ЭА.

Примеры простой анимации

Пример 1. *from-to* перемещение вдоль прямой

```
<animateMotion from="10, 20" to="2000, 1000" dur="3s"/>
```

ЦЭ перемещается вдоль прямой с началом в точке x=10 y=20 и с концом в точке x=2000 y=1000 в течении 3 сек. Здесь from определяет координаты начала прямой, to определяет координаты конца прямой, dur определяет ВПА. По умолчанию, координаты атрибутов from, to измеряются в пикселах.

Пример 2. *from-by* перемещение вдоль прямой

```
<animateMotion from="10, 10" by="2000, 1000" dur="3s"/>
```

Атрибут by определяет относительное смещение конечного значения ЦА, т.е. ЦА изменяется от значения 10 до значения 10+90=100 в течении 2.5 сек.

Пример 3. *by* перемещение вдоль прямой

```
<animateMotion from="10, 10" by="2000, 1000" dur="3s"/>
```

В качестве начального значения ЦА берётся его исходное значение, т.е. значение 10 атрибута x2 АЭ line. К этому значению прибавляется число 1000, т.е. через 3 секунды ЦА примет значение 1010.

Пример 4. *to* перемещение вдоль прямой

```
<animateMotion from="10, 10" by="2000, 1000" dur="3s"/>
```

В качестве начального значения ЦА берётся его исходное значение, т.е. значение 10 атрибута x2 АЭ line. Конечным является значение 1000 атрибута to ЭА, т.е. от значения 10 через 3 секунды ЦА примет значение 1000.

Пример 5. Перемещение по списку значений пути

```
<animateMotion values="100, 200; 100, 400; 400, 400" dur="3s"/>
```

Используется синтаксис values для изменения ЦА в течение 2-х секунд от значения 40 до 100 и обратно до 40. В первую секунду значение x2 меняется по закону f(t) = 40

+ 60*t/5, 0<=t<1, и в следующую секунду по закону f(t) = 100 - 60*(t-5)/5, 1<= t <= 2. По истечении 2-х секунд ЦА примет значение 200, что соответствует значению атрибута x2 АЭ.

Пример 6. Перемещение по заданному пути

```
<animateMotion path="m 0 0 c 30 50 70 50 100 0 z" dur="3s" />
```

Пример 7. Перемещение по ссылке на заданный путь

```
<path id="путь" d="M100,250 C 100,50 400,50 400,250"
                                        fill="none" stroke="blue" stroke-width="10" />
<animateMotion dur="6s" repeatCount="3" rotate="auto">
  <mpath xlink:href="#путь"/>
</animateMotion>
```

Пример 8. Перемещение по заданному пути с поворотом

```
<rect width="200" height="20">
  <animateMotion values="100, 200; 100, 400; 400, 400" dur="3s" rotate="auto"/>
</rect>
```

Пример 9. Перемещение по списку значений пути с линейной нтерполяцией

```
<animateMotion values="100, 200; 100, 400; 400, 400" dur="3s" calcMode="linear"/>
```

Значение ЦА изменяется от 40 до 1500 в первые 3 секунды и затем с 1500 до 1200 за вторые 3 сек., т.е. время на каждое изменение отводится одинаковое. Поэтому в течение второй половины анимации величина ЦА изменяется (уменьшается) медленнее.

Пример 10. Перемещение по списку значений пути с темповой интерполяцией

```
<animateMotion values="100, 200; 100, 400; 400, 400" dur="3s" calcMode="paced"/>
```

Реализуется ровный темп изменения ЦА, т.е. скорость увеличения ЦА равна скорости его уменьшения. Увеличение ЦА на величину 1460 протекает дольше, чем уменьшение на величину 300.

Пример 11. Перемещение по списку значений пути с темповой интерполяцией

```
<animateMotion values="100, 200; 100, 400; 400, 400" dur="3s" calcMode=" discrete"/>
```

Пример 12. Перемещение по списку значений пути с темповой интерполяцией

```
<animateMotion values="100, 200; 100, 400; 400, 400" dur="3s"
                                        keyTimes="0; .8; 1" calcMode="linear" />
```

Величины keyTimes заставляют ЦА расти от 0 до 100 за 8 секунд, что соответствует 80% ВПА (10 сек.), и уменьшаться от 100 до 50 за оставшиеся 2 секунды, т.е. до конца анимации.

Пример 13. Перемещение по списку значений пути с темповой интерполяцией

<animateMotion values="100, 200; 100, 400; 400, 400" dur="3s"
keyTimes="0; .8; 1" calcMode="spline" keySplines=".5 0 .5 1; 0 0 1 1" />

Атрибуты keySplines и keyTimes обеспечивают управление ускорением изменения значений атрибута values, а, следовательно, и ускорением анимации. ЦА имеет значение "0" в начале анимации, "1000" по истечении 80% ВПА, т.е. через 8 сек. и "500" в конце анимации. Значения keySplines определяют кривую для темповой интерполяции перехода между величинами 0 – 1000 и 1000 - 500. В примере сплайн вызывает плавное начало и окончание эффекта увеличения ЦА от 0 до 1000 между временем 0 и 8 сек. (т.е. между keyTimes 0 и .8, и values "0" и "1000"), но строгую линейную интерполяцию для уменьшения ЦА от 1000 до 500 между 8 сек. и окончанием (т.е. между keyTimes .8 и 1, и values "1000" и "500").

Примеры сложной анимации

Пример 1. Многократное перемещение с реверсом.

<animateMotion by="2000, 1000" dur="3s" autoReverse="true" repeatCount="3"/>

by="2000,1000" – указывает целевые координаты перемещаемого ГЭ, т.е. прямоугольник перемещается вдоль прямой из исходного положения (x=0, y=0) в точку с координатами 2000 пикселей на оси X и 1000 пикселей на оси Y. dur="3s" задаёт ВПА 3 секунды, т.е. это продолжительность однократного перемещения. autoReverse="true" – значение реверса (возврата) «истина», т.е. включен возврат в исходное положение (обратная прокрутка перемещения). repeatCount="2" – повтор операции 2 раза.

Пример 2. В этом и следующих примерах изменён только элемент animateMotion примера 1. Однократное перемещение из примера 1 повторяется в течении 9 секунд или прерывается, если пользователь щелкнет мышкой на изображении прямоугольника.

<animateMotion by="2000,1000" dur="3s" autoReverse="true"
repeatDur="9" end="click" />

repeatDur="9" – повтор операции в течение 9 секунд, т.е. будет выполнятся 3 цикла. end="click" – остановка ППВ при нажатии кнопки мыши на прямоугольнике.

Пример 3. Перемещение полностью совпадает с примером 2, но только начало движения запускается щелчком мыши на изображении прямоугольника. По завершении процесса прямоугольник остаётся в своём достигнутом положении. При повторном нажатии кнопкой мыши на изображении движение начинается из начальной точки.

<animateMotion by="2000,1000" dur="3s" autoReverse="true"
repeatDur="9" begin="click" fill="freeze"/>

Пример 4. Данный пример повторяет пример 3, однако запуск движения происходит при нажатии клавиши «d» на клавиатуре (предварительно необходимо один раз кликнуть), а останов происходит при щелчке мышью на прямоугольнике. Повторное нажатие клавиши ”d” запускает процесс заново.

<animateMotion by="2000,1000" dur="3s" autoReverse="true" repeatDur="9"
begin="accessKey(d)" end="click" fill="freeze"/>

Пример 5. Прямоугольник совершает перемещение в течение 8 секунд, при этом в течение первых двух секунд он разгоняется, затем в течение следующих 4 секунд движется с постоянной скоростью, затем за оставшиеся 2 секунды замедляет движение (не все плагины поддерживают эту функцию).

<animateMotion by="2000,1000" dur="8s" accelerate=".25" decelerate=".25"/>

accelerate = ".25" означает, что первую четверть времени объект ускоряется.

decelerate = ".25" означает, что последнюю четверть времени объект замедляет свое движение.

Пример 6. Прямоугольник движется вдоль заданного пути (дуга) в течение 5 сек. и возвращается на исходную позицию. Движение повторяется 4 раза.

<animateMotion path="m 0 0 c 30 50 70 50 100 0 z" dur="5s"
accumulate="sum" repeatCount="4" />

path = "<path-data>" – траектория движения, выражается и интерпретируется так же, как и атрибут d элемента 'path'. Влияние траектории анимации заключается в присоединении дополнительной матрицы трансформации к текущей матрице трансформации соответствующего объекта. В результате текущая пользовательская система координат перемещается относительно осей X и Y по рассчитанным координатам x и y.

Пример 7. Прямоугольник передвигается последовательно по трем заданным траекториям. Конечная точка предыдущего пути является начальной точкой следующего пути. По окончании движения прямоугольник фиксируется в последней точке пути.

```
<animateMotion begin="0" dur="5s" path="m 0 0 c 30 50 70 50 100 0 z"
                                                    additive="sum" fill="freeze" />
<animateMotion begin="5s" dur="5s" path="m 0 0 L100 0 L100 100"
                                                    additive="sum" fill="freeze" />
<animateMotion begin="10s" dur="5s" path="m 0 0 c 30 50 70 50 100 0"
                                                    additive="sum" fill="freeze" />
```

fill : "freeze" | "remove" Атрибут может принимать следующие значения: *freeze* – эффект анимации фиксируется на значении, соответствующем последнему моменту t. *remove* (значение по умолчанию) – эффект анимации удаляется по окончании действия анимации. *Begin* – определяет момент t, когда элемент становится активным.

Пример 8. Прямоугольник движется горизонтально в течение 5 сек, совершая при этом колебательные движения в течении 3 сек. За время движения прямоугольник перемещается на 1000 пикс. Одно колебательное движение имеет амплитуду 100 пикс. и длится 1 сек.

```
<animateMotion from="0,0" to="1000,0" dur="5s" fill="freeze" />
<animateMotion values="0,0; 0,100; 0,0" dur="1s" repeatDur="3s" additive="sum"/>
```

Атрибут *values* должен состоять из списка пар координат x, y. Значения координат должны разделяться, как минимум, одним пробелом или запятой. Дополнительные пробелы вокруг разделяющего символа допускаются. Например values="10,20;30,20;30,40" или values="10mm,20mm; 30mm, 20mm;30mm,40mm". Каждая координата представляет длину.

```
<animateMotion path="m 0 0 c 30 50 70 50 100 0 z" dur="3s"
                                                    accumulate="sum" repeatCount="4" />
```

Изображение движется из исходного положения вдоль дуги в течение 5 сек. Анимация повторяется 4 раза, и каждый раз очередная дуга начинается там, где заканчивается предшествующая, как показывает рис. 4.

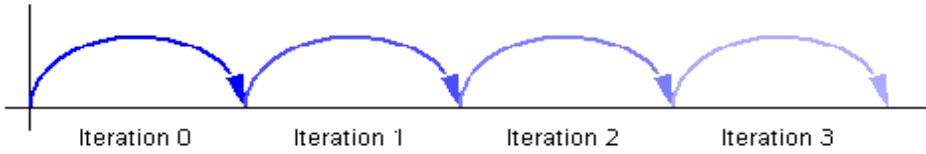


Рис. 4.

8. Интерактивность

Интерактивность в SVG, т.е. визуальные изменения SVG-содержимого по действиям пользователя, обеспечивается следующими способами. Во-первых, нажатием кнопок устройств ввода (например, кнопок мыши) для запуска анимации, бегущей строки текста или программы (например, скрипта или апплета). Во-вторых, наведением указателя мыши на графический элемент (ГЭ), который является потомком анкерного элемента “a”, для вызова новой Web-страницы или другого ГЭ. В-третьих, масштабированием изображения ГЭ с использованием его атрибута zoomAndPan. В-четвёртых, изменением вида курсора при изменении его местоположения. При этом наступившие события могут быть перехвачены, во-первых, скриптами, если таковые внедрены в svg-документ и являются слушателями этих событий, во-вторых, ГЭ, если они имеют соответствующие атрибуты. В последнем случае запускаются либо скрипты, либо осуществляется управление (запуск и/или останов) анимации или вызов Web-страницы и т.д.

SVG-события подразделяются на три группы: атрибуты событий анимации (animationOnEvents) – onbegin, onend, onrepeat; атрибуты события документа (documentEvents) – onunload, onabort, onerror, onresize, onscroll, onzomm; атрибуты события графических и контейнерных элементов – onfocusin, onfocusout, onactivate, onclick, onmausedown, onmauseup, onmauseover, onmausemove, onmauseout, onload.

Далее перечислены некоторые svg-события. После названия события следует (в скобках) его перевод и соответствующее название атрибута события, которое может иметь ГЭ. Затем в скобках указывается DOM2-имя события, используемое при определении приёмника события DOM2 (DOM2 events listeners). Через тире дано описание события.

focusin (фокус включен) onfocusin (DOMFocusIn) – Происходит, когда ГЭ получает фокус, например, когда ГЭ становится выбранным.

focusout (фокус выключен) onfocusout (DOMFocusOut) – Происходит, когда ГЭ теряет фокус, например, когда указатель покидает ГЭ.

activate (активация) onactivate (DOMActivate) – Происходит, когда ГЭ активизирован, например, через нажатие клавиши "мыши " или клавиатуры. Численный аргумент определяет тип происходящей активации: 1 – для простой активации (например, одинарный клик «мышью» или нажатие клавиши Enter), 2 – для гиперактивации (например, двойной клик «мышью» или нажатие клавиш Shift и Enter).

click (клик) onclick (DOMMouseEvent) – Происходит, когда клавишей мыши щелкают по ГЭ. Щелчок определен как mousedown и mouseup по той же самой области экрана. Последовательность этих событий: mousedown, mouseup, click. Если множественные щелчки происходят в той же самой области экрана, то происходит повторение последовательности с присвоением признака detail каждому повторению.

mousedown (нажата над) onmousedown – Происходит, когда кнопка мыши нажата на ГЭ.

mouseup (отпущена над) onmouseup – Происходит, когда кнопка мыши отпущена на ГЭ.

mouseover (мышь наезжает) onmouseover – Происходит, когда курсор мыши наведён на ГЭ.

mousemove (мышь движется) onmousemove – Происходит, когда курсор мыши перемещается, по ГЭ.

mouseout (мышь съезжает) onmouseout – Происходит, когда курсор мыши «съезжает» с ГЭ.

SVGZoom (масштабирование SVG) onzoom (нет) – Происходит, когда масштаб представления (вида) документа изменяется или через непосредственное пользовательское взаимодействие или любое изменение свойства 'currentScale (текущая шкала)', располагаемого на поверхности раздела элемента SVG.

beginEvent (событие начала анимации) onbegin (нет) – Происходит при запуске элемента анимации.

endEvent (событие окончания анимации) onend (нет) – Происходит при окончании элемента анимации.
repeatEvent (событие повтора анимации) onrepeat (нет) – Происходит при повторе простой анимации, когда заданы атрибуты (количественный или временной) повтора. Это используется каждый раз при повторении ГЭ, после первого повтора.

Как и в DOM2 в SVG пока отсутствует набор событий клавиш. Это ожидается в следующих версиях DOM и SVG.

В набор возможных пользовательских событий входят: события указателя (мыши), события клавиатуры и события документа (фрагмента). В ответ на эти действия пользователя возможны запуск анимации, гиперссылка на другую страницу в Сети, выдвижении на первый план части документа (например, изменение цвета ГЭ, которые находятся под курсором), инициация "roll-over" (например, это заставляет некоторые, предварительно скрытые ГЭ, появляться около курсора, т.е. показ ранее скрытых элементов) или запуска сценария (апплета), который связывается с отдаленной базой данных и т.д.

Для каждого события указателя (перемещения, клики и другие действия мыши) должен быть определён *целевой* элемент (ЦЭ). Это, как правило ГЭ, графическое изображение которого находится под указателем во время события. Когда элемент невидим (то есть, когда свойство «display» у этого элемента или у одного из его предков имеет значение «none»), он не может быть ЦЭ события указателя.

Наличие или отсутствие события зависит от следующего:
– Если нет ГЭ, чьё графическое изображение находится под указателем, (то есть, нет никакого ЦЭ), то событие не состоялось;
– ЦЭ есть, если есть предок ЦЭ, который определил появление события, захватывая данное событие, то событие послано этому элементу предка.
– Если ЦЭ имеет соответствующий определитель события для данного события, то это событие послано ЦЭ.
– Если каждый предок ЦЭ (начинающийся непосредственно с этого элемента) проверен, чтобы узнать, имеет ли он соответствующий определитель события. Если найден предок с соответствующим определителем события, то это событие послано найденному предку этого элемента.

– Иначе событие отменено.

Когда событие пузырения (EVBUBBLE) активно, пузырение происходит до всех прямых предков ЦЭ. Элементы потомки получают события перед их предками. Таким образом, если элемент “path” является потомком элемента “g”, и они оба подпадают под событие клика, то это событие будет послано элементу “path”, перед “g”.

Свойство 'pointer-events' определяет, что при определённых обстоятельствах данный ГЭ может быть ЦЭ для события указателя. Обстоятельства, при которых событие может быть обработано: пользователя связывают с помощью интерфейса событиями, типа кликов "мышью"; динамические псевдоклассы (то есть :hover :active и :focus) (CSS2); гиперссылки.

Событие указателя мыши (pointer-events) может иметь следующие значения – visiblePainted, visibleFill, visibleStroke, visible, painted, fill, stroke, all, none, inherit. Начальное значение – visiblePainted. Оно применяется к ГЭ, *наследуется и анимируется.* Рассмотрим эти значения подробнее.

visiblePainted (Видимый Окрас) – Данный элемент может быть ЦЭ для событий указателя, когда свойство 'visibility (видимость)' установлено, и когда указатель на окрашенной области, то есть fill (заливка) элемента установлена не в значение 'none', или на периметре элемента, и свойство 'stroke (контур)' установлено не в 'none (отсутствует)'.

visibleFill (Видимая Заливка) – Данный элемент может быть ЦЭ для событий указателя, когда свойство ' visibility' установлено, и когда указатель на внутренней заливке элемента.

visibleStroke (Видимый контур) – Данный элемент может быть ЦЭ для событий указателя, когда свойство 'visibility' установлено в значение видимый, и когда курсор – на периметре, то есть на контуре элемента.

visible (Видимость) – Данный элемент может быть ЦЭ для событий указателя, когда свойство 'visibility' установлено в значение видимый, и курсор - или внутри (на заливке), или на периметре (на контуре) элемента.

painted (Окрас) – Данный элемент может быть ЦЭ для событий указателя, когда указатель – на "окрашенной" области (на заливке) элемента, и свойство 'fill' установ-

лено не в значение 'отсутствует', или на контуре элемента, и свойство 'контура' установлено не в значение 'отсутствует'.

fill (Заливка) – Данный элемент может быть ЦЭ для событий указателя, когда курсор внутри (на заливке) элемента.

stroke (Контур) – Данный элемент может быть ЦЭ для событий указателя, когда курсор на контуре элемента.

all (Всё) – Данный элемент может быть ЦЭ для событий указателя всякий раз, когда курсор – или внутри (то есть, на заливке) или на периметре (то есть, на контуре) элемента. Значения свойств 'заливка', 'контур' и 'видимость' не влияют на обработку явления.

none (Отсутствует) – Данный элемент не получает события указателя.

Масштабирование представляет собой полное, однородное преобразование фрагмента svg-документа, где действие увеличения определяет масштаб всех ГЭ. Действие увеличения имеет эффект дополнительной шкалы и производит преобразование, помещенное на наиболее удаленном уровне фрагменте svg документа (то есть вне наиболее удаленного 'svg' элемента).

Конвертация представляет собой перевод (трансляцию) преобразования фрагмента svg-документа в ответ на события интерфейса пользователя.

Некоторые диалоговые окружающие среды показа имеют способность модифицировать появление указателя, который также известен как курсор. Существуют три типа курсоров: Стандартные встроенные курсоры; Определённые платформой традиционные курсоры; Независящие от платформы традиционные курсоры.

Свойство cursor

Основное назначение свойства курсора – привлечь внимание пользователя к ГЭ, на котором установлен курсор.

Свойство 'cursor' используется, для определения курсора, который нужно использовать. Традиционные курсоры упомянуты как <uri> и могут обращаться или к внешнему ресурсу типа курсора, определяемого платформой, или к элементу 'cursor', который может использоваться для определения курсора, независимого от платформы.

Свойство cursor может иметь значения – [<uri> ,]* auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help | inherit. Начальное значение – auto. Применяется к ГЭ или контейнеру. Наследуется и анимируется. Эти свойства определяют тип курсора, которым будет обозначено устройство обращения («мышь»). Величины имеют следующие значения:

auto (Авто) – обычный для ГЭ вид курсора.

crosshair (Перекрестие) – задаёт курсор в виде перекрестия, напоминающего знак "+".

default (Свободный, т.е. ни на что не наведённый) – зависящий от платформы «свободный» курсор, часто представляемый как стрелка.

pointer (Указатель) – формирует привычный курсор в виде стрелки.

move (Передвижение) – обозначает, что ЦЭ можно перемещать, имеет вид пересекающихся линий с двунаправленными стрелками.

e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize – курсор с указанием направлений сдвигов (например, курсор 'se-resize ' используется, когда перемещение начинается от юго-восточного угла перемещаемого ГЭ).

text (Текст) – указывает, что текст может быть выделен, часто представляется как символ I.

wait (Ожидание) – Указывает, что система (программа) занята, часто представляется как часы или песочные часы.

help (Помощь) – задаёт курсор, информирующий пользователя о возможности получения справочной информации (часто представляется знаком вопроса или воздушным шаром).

inherit – задаёт наследование свойства от родителя.

<uri> – определяет курсор в зависимости от ресурса, обозначенного uri. Если пользовательское средство не может оперировать первым курсором списка курсоров, оно должно пытаться сделать это вторым, и т.д. Если пользовательское средство не может оперировать ни одним определенным курсором, оно должно использовать групповой курсор в конце списка.

P { cursor : url("mything.cur"), url("second.csr"), text; }

Атрибуты элемента < cursor > (курсор):

x = "координата" – координата X положения в системе координат (СК) курсора, которая представляет точное указанное положение. Если признак не определен, эффект – как будто было указано значение «0». Возможность анимирования: да.

y = "координата" координата Y положения в СК курсора, которая представляет точное указанное положение. Если признак не определен, эффект – как будто было указано значение «0». Возможность анимирования: да.

xlink:href = "uri" uri ссылается к файлу или элементу, который содержит изображение курсора. Возможность анимирования: да.

Свойство key-equivalent

Свойство key-equivalent является CSS-аналогом атрибута accesskey, который можно задавать в дескрипторах input и anchor. Он связывает клавишу со ссылкой или с интерактивным элементом формы, после нажатия клавиши выбирается соответствующий элемент. Свойство key-equivalent позволяет указывать клавишу (комбинацию клавиш), нажатие которой (которых) генерирует то или иное событие. Возможны следующие значения – none (запрещает связывать клавишу с элементом), inherit (задаёт наследование свойства), набор клавиш и системный эквивалент. Для задания сочетания клавиш используется символ “-”, например, ctrl-x описывает нажатие клавиши <Ctrl> затем (удерживая <Ctrl>) <X>. Свойство key-equivalent не чувствительно к регистру. Значение системного эквивалента, т.е. какая клавиша в данной операционной системе эквивалентна, например, system-cancel, трактуется разработчиком браузера.

Псевдоклассы focus и hover информируют о том, что курсор мыши находится над элементом или элемент владеет фокусом ввода. Эти псевдоклассы всегда связываются с дескриптором. Псевдокласс focus изменяет курсор, сообщая о том, что фокус ввода принадлежит соответствующему элементу, т.е. что этот элемент способен воспринимать действия пользователя. Псевдокласс hover сообщает пользователю, что элемент, на котором находится курсор, может быть активизирован тем или иным способом. Эти псевдоклассы непосредственно связаны с псевдоэлементами link, visited и active.

9. Примеры кода

Для каждого из предшествующих теоретических разделов (до анимации) здесь приведены практические примеры с результатами их визуализации. Каждому примеру соответствует от одного до нескольких листингов. После названия листинга в скобках приведено название (с расширением) файла, в котором сохранен текст листинга.

9.1 Примеры кода для раздела 1

Пример 1

Листинг 1. *Чёрный квадрат (рис. 1) (пример_1.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="500" height="500"
                                xmlns="http://www.w3.org/2000/svg">
  <!-- собственная стилизация -->
  <rect x="100" y="100" width="200" height="200" style="black"/>
</svg>
```

Рис. 1

Пример 2

Листинг 1. *Красный квадрат с голубой границей в 10 ед. (рис. 2) (пример_2.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg width="10cm" height="5cm" viewBox="0 0 1000 500"
                                xmlns="http://www.w3.org/2000/svg" version="1.1">
  <!--локальное стилевое оформление-->
  <style type="text/css">
    rect { fill: red; stroke: blue; stroke-width: 10 }
  </style>
  <rect x="200" y="100" width="600" height="300"/>
</svg>
```

Рис. 2

Пример 3

Листинг 1. *Красный круг с голубой границей в 3 мм. Стилевое оформление глобальное (рис. 3) (пример_3.svg)*

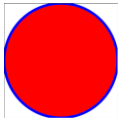


Рис. 3

```
<?xml version="1.0" encoding="windows-1251"?>
<!--глобальная стилизация-->
<?xml-stylesheet href="пример_3.css" type="text/css"?>
<svg version="1.1" width="500" height="500" viewBox="0 0 500 500"
                                    xmlns="http://www.w3.org/2000/svg">
  <circle cx="250" cy="250" r="250" />
</svg>
```

Листинг 2. *Стилевой файл (пример_3.css)*

```
circle { fill: rgb(100%,0%,0%);  stroke: rgb(0%,0%,100%);  stroke-width: 3mm }
```

Пример 4

Листинг 1. *Ссылки эллипса на прямоугольник и обратно (рис. 4) (пример_4.1.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<?xml-stylesheet href="пример_4.1.css" type="text/css"?>
<svg width="10cm" height="5cm" xmlns:xlink="http://www.w3.org/1999/xlink"
                                    xmlns="http://www.w3.org/2000/svg" version="1.1">
  <a xlink:href="пример_4.2.svg">
    <rect x="200" y="100" width="600" height="300"/>
  </a>
</svg>
```

Рис. 4

Листинг 2. *пример_4.1.css*

```
rect { fill: rgb(100%,0%,0%); stroke: rgb(0%,100%,0%); stroke-width: 10 }
```

Листинг 3. *пример_4.2. svg*

```
<?xml version="1.0" encoding="windows-1251"?>
```

```
<svg width="5cm" height="3cm" xmlns="http://www.w3.org/2000/svg"
                                        xmlns:xlink="http://www.w3.org/1999/xlink">
  <!--Собственное стилевое оформление без использования атрибута style-->
  <rect x=".01" y=".01" width="4.98" height="2.98" fill="none"
                                                  stroke="blue"  stroke-width=".03"/>
  <a xlink:href="пример_4.1.svg">
    <ellipse cx="2.5" cy="1.5" rx="200" ry="100" fill="red" />
  </a>
</svg>
```

9.2 Примеры кода для раздела 2

Пример 5

Листинг 1. *В ДП следующей разметки описан треугольник (рис. 5) (пример_5.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
      xmlns="http://www.w3.org/2000/svg" version="1.1">
  <path d="M 100 100 L 300 100 L 200 300 z"
                        fill="red" stroke="blue" stroke-width="3"/>
</svg>
```

Рис. 5

Пример 6

Листинг 1. *Вращательный шарнир (рис. 6) (пример_6.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="300px" height="200px"
                        viewBox="-50 -25 300 200" xmlns="http://www.w3.org/2000/svg">
 <path d="m10,20 l 0,-20 m-10,0 l 20,0 m-10,20 l 0,20 m-10,0 l 20,0 m-5,-5 l0,-30 m-10,0
                                                        l 0,30" stroke="black"/>
</svg>
```

Пример 7

Листинг 1. *Деталь шарнира (рис. 7) (пример_7.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1"  width="300px" height="200px"
                        viewBox="-50 -25 300 200" xmlns="http://www.w3.org/2000/svg">
 <path d="m0,0 l 20,0 a 10,10 0 0,1 20,0 l 20,0" stroke='rgb(0,0,0)' fill='none' />
</svg>
```

Пример 8

Листинг 1. *Ряд наклонных линий (рис. 8) (пример_8.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1"  width="300px" height="200px"
                        viewBox="-50 -25 300 200" xmlns="http://www.w3.org/2000/svg">
  <path d="m0,8 l 8,-8 m0,8 l 8,-8 m0,8 l 8,-8 m0,8 l 8,-8 m0,8 l 8,-8 m0,8 l 8,-8 m0,8 l
                                                                    8,-8"
                                                                    stroke='rgb(0,0,0)'/>
</svg>
```

Пример 9

Листинг 1. *Упругий элемент (рис. 9) (пример_9.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1"  width="300px" height="200px"
                        viewBox="-50 -25 300 200" xmlns="http://www.w3.org/2000/svg">
 <path id="упругость180"  d="M0,0 l 2,13 10,-5 -2,13 10,-5 0,13 6,-10 3,13 6,-10
                                  M70,0  l -2,13 -10,-5 2,13 -10,-5 0,13 -6,-10 -3,13 -6,-10"
                                                    stroke='rgb(0,0,0)' fill='none' />
</svg>
```

Пример 10

Листинг 1. *Горизонтальная линия со стрелками на концах (рис. 10) (пример_10.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="300px" height="200px"
```

```
viewBox="-50 -25 300 200" xmlns="http://www.w3.org/2000/svg">
  <path id="размер" d="m0,0 l-10,-5 m10,-5 l-10,5 80,0 -10,-5 m10,5 l -10,5"
                                    stroke='rgb(0,0,0)' fill='none'/>
</svg>
```

Рис. 6

Рис. 7

Рис. 8

Рис. 9

Рис. 10

9.3 Примеры кода для раздела 3

Пример 11

Листинг 1. *Следующий пример показывает, что исходная СК имеет начало в верхнем левом углу с осью X, направленной направо и осью Y, направленной вниз (рис.15). Пользовательская СК равняется родительской СК, единицы измерения установлены в "пикселах" (рис. 11) (пример_11.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="300px" height="100px" xmlns="http://www.w3.org/2000/svg">
<g fill="none" stroke="black" stroke-width="3">
  <line x1="0" y1="1.5" x2="300" y2="1.5"/>
  <line x1="1.5" y1="0" x2="1.5" y2="100"/>
 </g>
 <g fill="red" stroke="none" >
  <rect x="0" y="0" width="3" height="3" />
  <rect x="297" y="0" width="3" height="3" />
  <rect x="0" y="97" width="3" height="3" />
 </g>
<!-- конец фрагмента - начало -->
 <g font-size="14" font-family="Verdana" >
  <text x="10" y="20">(0,0)</text> <text x="240" y="20">(300,0)</text>
  <text x="10" y="90">(0,100)</text>
```

</g>
</svg>

Рис. 11

Пример 12

Листинг 1. *Устанавим новое ПП преобразованием transform="translate(50,50)" в третьем элементе g (рис.16). Новое ПП смещено относительно исходной СК на 50 единиц вправо и вниз. Здесь и в следующих примерах первые 9-ть строк взяты из 1-го примера (рис. 12) (пример_12.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="600px" height="100px" xmlns="http://www.w3.org/2000/svg">
 <g>
  <text x="30" y="30" font-size="20" font-family="Times New Roman">
                                        АБФ (исходная система координат) </text>
 </g>
 <!-- Перенос СК на 50 ед. По каждой оси. -->
 <g transform="translate(50,50)">
  <g fill="none" stroke="red" stroke-width="3" >
   <!-- Нарисовать линии по 50 ед. вдоль осей новой СК -->
   <line x1="0" y1="0" x2="50" y2="0" stroke="red" />
   <line x1="0" y1="0" x2="0" y2="50" />
  </g>
  <text x="30" y="30" font-size="20" font-family="Times New Roman">
                                        АБФ (перемещенная система координат) </text>
 </g>
```

</svg>

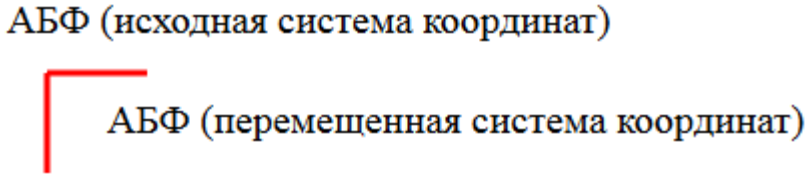


Рис. 12

Пример 13

Листинг 1. *Рассмотрим вращение и масштабирование. Здесь определены две новые СК:*

первая является результатом перемещения на 50 единиц по X и на 30 по Y, также поворотом на 30°, другая – результат перемещения на 200px по X и 40px по Y, а так же масштабирования в 1,5 раза (рис. 13) (пример_13.svg)

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="600px" height="200px" xmlns="http://www.w3.org/2000/svg">
<!-- Перенос и поворот новой СК. -->
 <g transform="translate(50,30)">
  <g transform="rotate(30)">
   <g fill="none" stroke="red" stroke-width="3">
    <line x1="0" y1="0" x2="50" y2="0"/>
    <line x1="0" y1="0" x2="0" y2="50"/>
   </g>
   <text x="0" y="0" font-size="20" font-family="Times New Roman" fill="blue">
                                                              АБФ (поворот) </text>
  </g>
 </g>
 <!-- Перенос и масштабирование новой СК. -->
 <g transform="translate(200,40)">
  <g transform="scale(1.5)">
   <g fill="none" stroke="red" stroke-width="3">
```

```
   <line x1="0" y1="0" x2="50" y2="0"/>
   <line x1="0" y1="0" x2="0" y2="50"/>
  </g>
  <text x="0" y="0" font-size="20" font-family="Times New Roman" fill="blue">
                                                АБФ          (масштабирование)
</text>
 </g>
</g>
</svg>
```

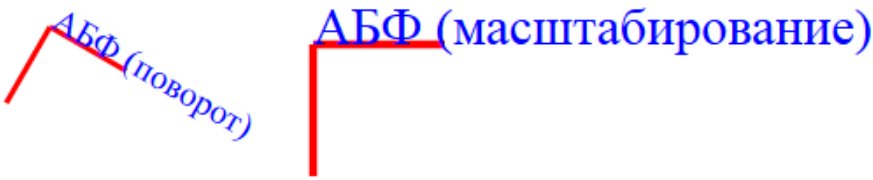


Рис. 13

Пример 14

Листинг 1. *Здесь определены две СК, которые наклонены относительно исходной СК (рис. 14) (пример_14.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="600px" height="350px" xmlns="http://www.w3.org/2000/svg">
<!-- Перенос и скос новой СК. -->
 <g transform="translate(30,30)">
  <g transform="skewX(30)">
   <g fill="none" stroke="red" stroke-width="3" >
    <line x1="0" y1="0" x2="50" y2="0"/>
    <line x1="0" y1="0" x2="0" y2="50"/>
   </g>
   <text x="0" y="0" font-size="20" font-family="Times New Roman" fill="blue">
                                                АБФ (сдвиг вдоль оси абсцисс) </text>
  </g>
```

```
 </g>
 <g transform="translate(200,30)">
  <g transform="skewY(30)">
   <g fill="none" stroke="red" stroke-width="3" >
    <line x1="0" y1="0" x2="50" y2="0" />
    <line x1="0" y1="0" x2="0" y2="50" />
   </g>
   <text x="0" y="0" font-size="20" font-family="Times New Roman" fill="blue">
                                                АБФ (сдвиг вдоль оси ординат)
</text>
  </g>
 </g>
</svg>
```

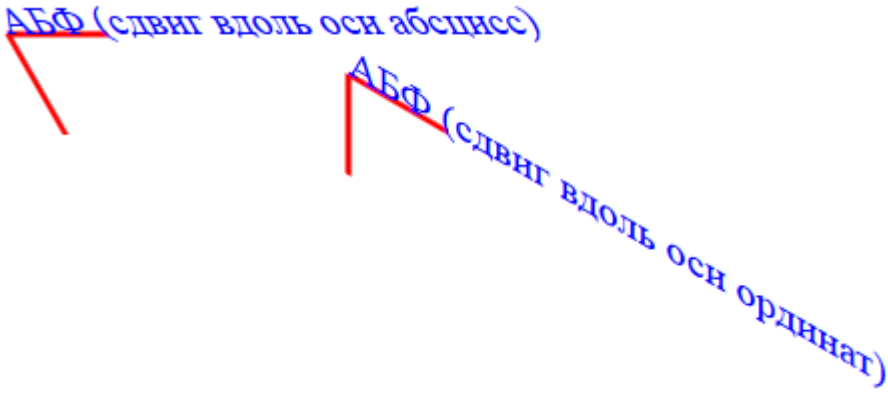


Рис. 14

Пример 15

Листинг 1. *Вложенные преобразования (рис. 15) (пример_15.svg)*

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" width="400px" height="150px" xmlns="http://www.w3.org/2000/svg">
<!-- 1. перенос -->
 <g transform="translate(50,90)">
  <g fill="none" stroke="red" stroke-width="3" >
```

```
 <line x1="0" y1="0" x2="50" y2="0" />
 <line x1="0" y1="0" x2="0" y2="50" />
</g>
<text x="14" y="-3" font-size="16" font-family="Times New Roman" >Сдвиг(1) </text>
<!-- 2. поворот -->
<g transform="rotate(-45)">
 <g fill="none" stroke="green" stroke-width="3" >
  <line x1="0" y1="0" x2="50" y2="0" />
  <line x1="0" y1="0" x2="0" y2="50" />
 </g>
 <text x="-3" y="-3" font-size="16" font-family="Times New Roman" >Поворот(2) </text>
 <!-- 3. другой перенос -->
 <g transform="translate(130,160)">
  <g fill="none" stroke="blue" stroke-width="3" >
   <line x1="0" y1="0" x2="50" y2="0" />
   <line x1="0" y1="0" x2="0" y2="50" />
  </g>
  <text x="-3" y="-3" font-size="16" font-family="Times New Roman" >Сдвиг(3) </text>
 </g>
</g>
</svg>
```

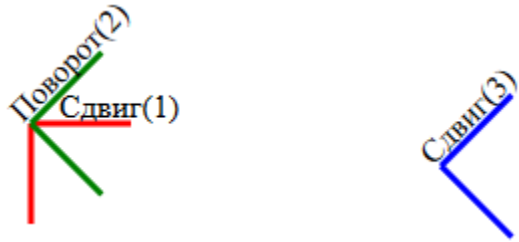


Рис. 15

9.4 Примеры кода для раздела 4

Пример 16

Листинг 1. *Радиус-вектор точки O_2 (рис. 16) (пример_16.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1"  width="300px" height="100px"
                                                xmlns="http://www.w3.org/2000/svg">
 <g transform="translate(30,50)">
  <text>O</text>
  <text x="12">O</text>
  <text x="24" y="3" font-size="10">2</text>
  <line x1="1" y1="-14" x2="22" y2="-14" stroke="black"/>
 </g>
</svg>
```

Пример 17

Листинг 1. *Производная по времени вектора $\overline{O_3O_4}$ (рис. 17) (пример_17.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="300px" height="100px"
                                                xmlns="http://www.w3.org/2000/svg">
 <g transform="translate(30,50)">
  <text>O</text>
  <text x="12" y="3" font-size="10">3</text>
  <text x="17">O</text>
  <text x="28" y="3" font-size="10">4</text>
  <line x1="1" y1="-14" x2="28" y2="-14" stroke="black"/>
  <text x="13" y="-17">.</text>
 </g>
</svg>
```

Пример 18

Листинг 1. *Вектор силы (рис. 18) (пример_18.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="300px" height="100px"
                                        xmlns="http://www.w3.org/2000/svg">
 <g transform="translate(30,50)">
  <text>F</text> <text x="9" y="3" font-size="10">i</text>
  <line y1="-14" x2="10" y2="-14" stroke="black"/>
 </g>
</svg>
```

Пример 19

Листинг 1. *Обобщенное ускорение (рис. 19) (пример_19.svg)*

```
<?xml version="1.0" encoding="windows-1251"?>
<svg version="1.1" width="300px" height="100px"
                                        xmlns="http://www.w3.org/2000/svg">
 <g transform="translate(30,50)">
  <text>q</text>
  <text x="9" y="5" font-size="10">i</text>
  <text y="-10">..</text>
 </g>
</svg>
```

Рис. 16

Рис. 17

Рис. 18

Рис. 19

Литература

1. Frost, J. Learn SVG: The Web Graphics Standart / J. Frost, S. Goessner, M. Hirtzler, 2003 – 518 pages.
2. Борисенко, А. А. Web-дизайн. Просто как дважды два / А. А. Борисенко. - М. : Эксмо, 2008. – 320 с.
3. Дунаев, В. HTML, скрипты и стили / В. Дунаев. – СПб. : БХВ-Петербург, 2008. – 1024 с.
4. Дунаев, В. Web-программирование для всех / В. Дунаев. – СПб. : БХВ-Петербург, 2008. – 560 с.
5. Кайгородцев, М. И. Расширяемый XML-ориентированный программный комплекс для моделирования систем и процессов / М. И. Кайгородцев, И. А. Масалимова // Ракетодинамика. Энергетика. Информатика : сборник научных трудов. – Челябинск : Издат. Центр ЮУрГУ, 2012. - С. 185-190.
6. Frost, J. Building Web Applications with SVG / J. Frost, D. Dailey, D. Strazullo/ - 2012. – 294 p.
7. Eisenberg, J. SVG Essentials, 2nd Edition / J. Eisenberg, A. Bellamy-Royds. – USA : O'Reilly Media, 2014.
8. Bellamy-Royds, A. SVG Colors, Patterns and Gradients / A. Bellamy-Royds. – USA : O'Reilly Media, 2015.
9. Bellamy-Royds, A. SVG Text Layout / A. Bellamy-Royds. – USA : O'Reilly Media, 2015.
10. Масштабируемая векторная графика = Scalable Vector Graphics : учебный курс / INTUIT.ru. – Режим доступа : www.intuit.ru/department/graphics/svg/. – свободный.
11. Сущность SVG. / J. David Eisenberg. – Режим доступа : www.xiper.net/learn/svg/svg-essentials/table-of-content.html/ – свободный.
12. SVG Tutorial / Jakob Jenkov. – Режим доступа : tutorials.jenkov.com/svg/index.html. – свободный.

Оглавление

НАУЧНОЕ ИЗДАНИЕ

Телегин Александр Иванович
Тимофеев Дмитрий Николаевич
Читалов Дмитрий Иванович
Пудовкина Светлана Геннадьевна

SVG-РАЗМЕТКА ДВУХМЕРНОЙ ГРАФИКИ

ОПЫТ ИСПОЛЬЗОВАНИЯ SVG В СОЗДАНИИ ДВУХМЕРНОЙ ГРАФИКИ

Издание Электротехничекого факультета ЮУрГУ
456318, г. Миасс, пр. Октября, 16.