**Вадим Дунаев**

# Основы SVG

Здесь рассматриваются основные понятия и применение SVG без претензий на исчерпывающую полноту. Тем не менее, приведенный материал можно использовать в качестве элементарного учебника для начинающих. Нумерация разделов согласована с моей книгой "**HTML, скрипты и стили**", ***3-е издание, 2011г.***.

## 13.1. Что такое SVG

SVG (Scalable Vector Graphics — масштабируемая векторная графика) — язык описания двухмерной векторной графики, являющийся словарем данных языка XML. Это означает, что приложения на языке SVG создаются по общим правилам XML с использованием дескрипторов (тегов), которые могут иметь атрибуты. SVG позволяет описывать собственно векторные изображения (линии и фигуры), форматированный текст, а также включать в документ и работать с растровыми изображениями: масштабировать и трансформировать их подобно векторным объектам. Кроме того, он дает возможность создавать анимационную графику, интерактивные приложения, управлять поведением и внешним видом с помощью JavaScript и CSS.

Статические и анимационные графические изображения, а также другие объекты описывают в формате SVG с помощью специальных дескрипторов, подобных HTML-тегам с атрибутами, и сохраняют в текстовом файле с расширением svg. Возможно также сохранение в сжатом виде (с применением gzip-компрессии), тогда расширение файла — svgz. Знакомые с языками разметки гипертекста HTML, XHTML или XML легко поймут и язык SVG. В SVG также применяются теги с атрибутами, хотя и специфические. С их помощью указывается, что именно следует отобразить, и с какими параметрами. Например, вы можете с помощью специального тега просто указать, что необходимо отобразить прямоугольник с закругленными углами, окрашенный в тот или иной цвет.

Итак, SVG-графика с точки зрения практического применения это, прежде всего, SVG-код в текстовом файле, в начале которого необходимо указать тип документа. Листинг 13.1 иллюстрирует типичный шаблон для SVG-файла.

**Листинг 13.1. Шаблон SVG-файла**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
    xmlns="http://www.w3.org/2000/svg" version="1.1"
    x="горизонтальная координата" y="вертикальная координата"
    width="ширина" height="высота">
… здесь находится собственно SVG-код
</svg>
```

Если в SVG-документе планируются ссылки на внешние ресурсы или внутренние объекты, то в тег `<svg>` следует добавить еще один атрибут:

```
xmlns:xlink="http://www.w3.org/1999/xlink"
```

В первой строке SVG-документа находится стандартное объявление XML-документа с указанием кодировки содержимого UTF-8. Файл с SVG-кодом нужно сохранить в той кодировке, которую вы указываете в дескрипторе `<?xml…>` с помощью атрибута `encoding`. Возможны и другие кодировки, например, windows-1251.

Далее следует дескриптор `<!DOCTYPE…>` определения типа документа (в данном случае указан SVG версии 1.1), затем — контейнерный тег `<svg>`, внутри которого размещаются специфические для SVG дескрипторы (теги). С помощью атрибутов `xmlns` и `version` указывают пространство имен и версию SVG, а посредством атрибутов `x`, `y`, `width` и `height` — координаты и размеры с указанием единиц измерения (px, cm, mm, in, %) прямоугольной области, в которой следует отобразить содержимое, заданное последующими тегами. Содержимое, выходящее за указанные рамки, при отображении будет усечено. Если атрибуты `width` и `height` опущены, то показывается все содержимое файла SVG.

SVG-код (текстовый файл с расширением svg) можно открыть непосредственно в браузере. В листинге 13.2 в качестве примера приведен SVG-документ, в котором определены прямоугольник и круг, а результат его загрузки в браузер показан на рис. 13.1. Здесь прямоугольник и круг заданы тегами `<rect>` и `<circle>` соответственно. Параметры данных векторных фигур указаны посредством атрибутов, на которых мы сейчас не будем останавливаться.

**Листинг 13.2. Пример SVG-документа с определением прямоугольника и круга**

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
   width="200px" height="100px"
   xmlns="http://www.w3.org/2000/svg" version="1.1"
>
  <rect x="50" y="40" width="100px" height="50px" fill="#00ffff"/>
  <circle cx="40px" cy="45px" r="30px" fill="red"/>
</svg>
```

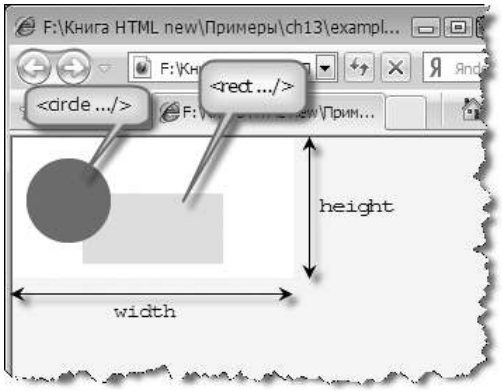



**Рис. 13.1.** Пример отображения в браузере SVG-документа с определением прямоугольника и круга (листинг 13.2)

SVG-документ можно не только непосредственно загрузить и отобразить в окне браузера, но и вставить в (X)HTML-документ. Для этого используются теги `<object>` или `<embed>` (листинг 13.3).

**Листинг 13.3. Примеры вставки SVG-документа в HTML-документ**

```
<object data="URL-адрес SVG-файла" type="image/svg+xml">
Для просмотра SVG необходим плагин.
 <a href="http://www.adobe.com/svg/viewer/install/">
  Загрузить Adobe SVGViewer
</a>
</object>
<embed src="URL-адрес SVG-файла" type="image/svg+xml" pluginspage=
"http://www.adobe.com/svg/viewer/install/" />
```

В данных тегах с помощью атрибутов `width` и `height` можно указать размеры области отображения содержимого SVG-файла на Web-странице, выделяемой

для элемента <object> или <embed>. Несогласованность значений атрибутов width и height тегов <svg> и <object> или <embed> может привести к неполному отображению SVG-содержимого.

Как уже отмечалось в *главе 11*, вставлять в (X)HTML-документ внешнее содержимое рекомендуется с помощью тега <object>. Однако применение тега <embed> представляется сейчас более надежным. Например, вставка SVG посредством тега <object> в Internet Explorer 8 с плагином Adobe SVGViewer хорошо работает на локальном компьютере, но при загрузке файла из Интернета возникают проблемы. Причина этого в плагине, поскольку при использовании RENESIS Player Plugin данный дефект отсутствует. Далее при описании особенностей отображения SVG в Internet Explorer будет предполагаться, что к нему подключен Adobe SVGViewer.


**Рис. 13.2.** Вставка SVG-документа в HTML-документ (листинг 13.4)

В листинге 13.4 приведен пример вставки SVG-документа в (X)HTML-документ с помощью тега <embed>, а на рис. 13.2 — его вид в Internet Explorer и Firefox. По умолчанию браузеры Firefox и Opera отображают SVG-документ на прозрачном фоне, если только он не был задан специальным образом в самом SVG-документе. Браузеры Internet Explorer, Safari и Chrome показывают его по умолчанию на белом фоне. В Internet Explorer сделать фон про-

зрачным можно с помощью атрибута `wmode="transparent"`, а в Safari и Chrome — нет. Таким образом, для обеспечения межбраузерной совместимости следует позаботиться о фоне SVG-документа в нем самом (см. *разд. 13.5*).

**Листинг 13.4. Пример вставки SVG-документа в (X)HTML-документ**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Вставка SVG</title>
<style>
  body {background:grey}
</style
</head>
<body>
<h1>Вставка SVG-графики</h1>
<embed src="example.svg" type="image/svg+xml"
  pluginspage="http://www.adobe.com/svg/viewer/install/"
  width="200" height="100" />
</body>
</html>
```

Теперь снова вернемся к SVG-документу, а именно к установке размеров. Как уже отмечалось, в теге `<svg>` с помощью атрибутов `width` и `height` можно задать ширину и высоту прямоугольной области, в которой размещается все его содержимое. По умолчанию единицы измерения — пикселы, но их можно задать и явно (px, cm, mm, in, %), как это делается в CSS. Графические объекты, определяемые в SVG-документе, также могут иметь размеры и координаты позиционирования. Все что не помещается в отведенное атрибутами `width` и `height` место, просто отсекается. Однако тег `<svg>` имеет еще и необязательный атрибут `viewBox`, с помощью которого можно масштабировать содержимое SVG-документа.

Атрибут `viewBox` принимает в качестве значения строку, содержащую четыре параметра, указанные через пробел или запятую с пробелом: горизонтальная координата, вертикальная координата, ширина и высота. Это параметры прямоугольной области в пикселах (например, `viewBox="0 0 200 100"`) относительно которой устанавливаются размеры всех элементов SVG-документа. Если значения ширины и высоты в `viewBox` равны значениям атрибутов `width` и `height`, то эффект масштабирования не возникнет, а если нет — содержимое SVG-документа масштабируется (рис. 13.3).

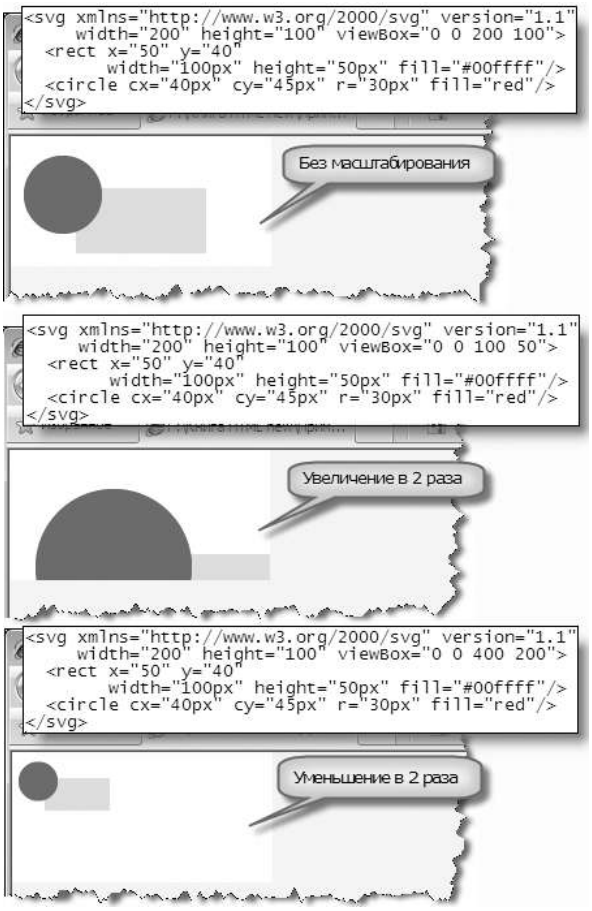



**Рис. 13.3.** Масштабирование содержимого SVG-документа

В SVG-коде рекомендуется применять контейнерный тег `<title>`, в котором указывается название документа. Текст комментария следует заключить в контейнерный тег `<desc>` (сокращение от description), например,

```
<desc>Пример фигуры </desc>
```

Вместе с тем, допустимы и обычные дескрипторы `<!-- Это комментарий -->`.

Все теги должны удовлетворять синтаксису XML: неконтейнерные теги должны завершаться косой чертой перед закрывающей уголковой скобкой. Впрочем, для таких тегов допустима и запись в виде контейнера. Например, возможны два варианта записи тега для прямоугольника:

```
<rect x="0" y="0" width="100" height="50"/>
<rect x="0" y="0" width="100" height="50"></rect>
```

В следующих разделах будут рассмотрены лишь основные (далеко не все) средства создания и приемы работы с SVG-объектами. В частности, такой мощный инструмент для создания дополнительных графических эффектов как фильтры, здесь не рассматривается.

## 13.2. Создание простых фигур

Начать изучение SVG лучше всего с рисования простых фигур, таких как прямоугольники, овалы, линии и т. п. Сложные фигуры получают комбинацией простых.

Фигуры могут быть замкнутыми (прямоугольник, круг) и разомкнутыми (прямые и кривые линии). Как объекты векторной графики, замкнутые фигуры состоят из внутренней части, называемой еще областью заливки, и внешнего контура (обводки). Линии можно представить себе как фигуры, состоящие только из контура.

Все графические объекты задают посредством специальных тегов с атрибутами. Для объектов, имеющих область заливки, с помощью атрибута `fill` можно назначить ее цвет. Параметры обводки определяют посредством атрибутов `stroke` (цвет) и `stroke-width` (толщина). Цвет задается как шестнадцатеричное значение, названием или как `rgb(r,g,b)`, где параметры — десятичные значения яркостей красной, зеленой и синей составляющих цвета в диапазоне от 0 до 255.

Вот три способа задания красного цвета заливки:

`fill="#ff0000"`, `fill="red"` и `fill="rgb(255,0,0)"`

Браузеры Firefox и Opera допускают еще задание цвета посредством значения `rgba(r,g,b,a)` из спецификации CSS3, где `a` — уровень непрозрачности от нуля до единицы. Значение `none` применяется, когда требуется указать отсутствие цвета. Область без цвета прозрачна. Если атрибут `fill` опустить, то заливка будет черной. Если не указывать атрибут `stroke`, то обводки не будет.

Чтобы указать уровень непрозрачности от 0 до 1, применяют атрибуты `fill-opacity` и `stroke-opacity` соответственно для области заливки и обводки. Например, атрибут `fill-opacity="0.5"` задает полупрозрачную область заливки. Имеются параметры (координаты и размеры), задаваемые числами, которые не обязательно должны быть целыми. Если размерность опущена, то подразумеваются пикселы.

Рассмотрим сначала создание простейших фигур, таких как прямоугольник, круг, эллипс (овал), отрезок прямой линии и т. п. Это так называемые графические примитивы, из которых можно построить более сложные изображения.

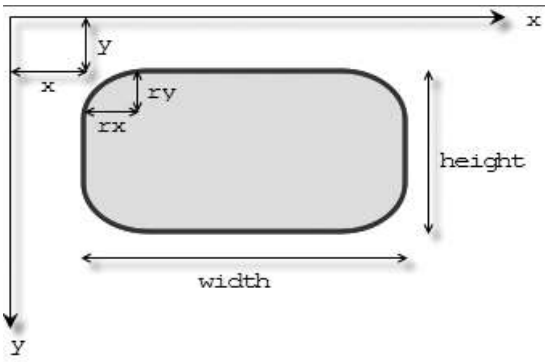
## 13.2.1. Прямоугольник

Прямоугольник задается тегом `<rect>` с возможными атрибутами:

- `x`, `y` — соответственно горизонтальная и вертикальная координаты верхнего левого угла;
- `width`, `height` — ширина и высота сооветственно;
- `rx`, `ry` — радиусы скругления углов; значения не должны превышать половины соответственно ширины и высоты прямоугольника (в противном случае может получиться весьма причудливая фигура); если указан только один из данных атрибутов, другой автоматически принимает такое же значение; если данные атрибуты не указаны, то углы будут прямыми.

Параметры области заливки и обводки задают атрибуты `fill`, `fill-opacity` и `stroke`, `stroke-width`, `stroke-opacity` соответственно.

На рис. 13.4 показан пример создания прямоугольника с закругленными углами. Если задать значения атрибутов `rx` и `ry` равными половинам значений атрибутов соответственно `width` и `height`, то получится овал. Если при этом значения `width` и `height` равны, то получается круг.




```
<rect x="50" y="40"
      width="200px" height="100px"
      rx="40" ry="30"
      fill="#00ffff"
      stroke="blue" stroke-width="3"
/>
```

**Рис. 13.4.** Прямоугольник

## 13.2.2. Круг

Круг задается атрибутом `<circle>` с возможными атрибутами:

- `cx`, `cy` — горизонтальная и вертикальная координаты центра круга;
- `r` — радиус круга.

На рис. 13.5 показан пример создания круга.

## 13.2.3. Эллипс

Эллипс задается тегом `<ellipse>` с возможными атрибутами:

- `cx`, `cy` — горизонтальная и вертикальная координаты центра эллипса;
- `rx`, `ry` — длины горизонтальной и вертикальной полуосей эллипса (горизонтальный и вертикальный радиусы).

На рис. 13.6 показан пример создания эллипса. При равенстве значений атрибутов `rx` и `ry` получается круг.

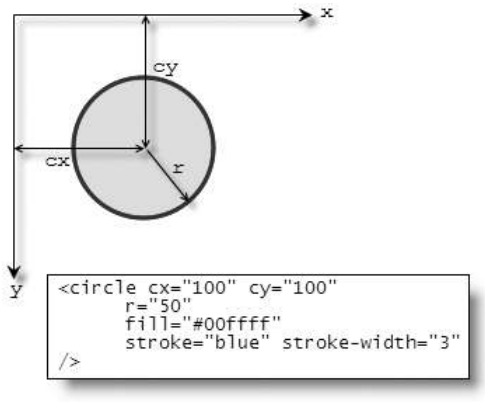



**Рис. 13.5.** Круг

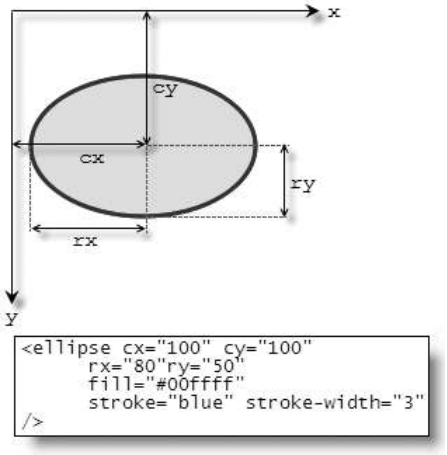



**Рис. 13.6.** Эллипс

## 13.2.4. Многоугольник

Многоугольник задается тегом `<polygon>` с атрибутом `points`, принимающим в качестве параметра строку, которая содержит координаты вершин. Каждой вершине соответствует пара из горизонтальной и вертикальной координат, которые разделяются пробелом или запятой. Пары координат соседних вершин разделяют запятой. По существу, атрибут `points` задает ломаную прямую периметра многоугольника, которая замыкается автоматически. Так что, совмещать начальную и конечную точки контура вручную нет

необходимости. Как и все рассмотренные ранее замкнутые фигуры, многоугольник имеет область заливки и обводку.

В листинге 13.5 приведен SVG-документ, в котором заданы пяти- и четырехугольник, а на рис. 13.7 — вид данного документа в окне браузера. Стрелками указан порядок следования вершин данных многоугольников.

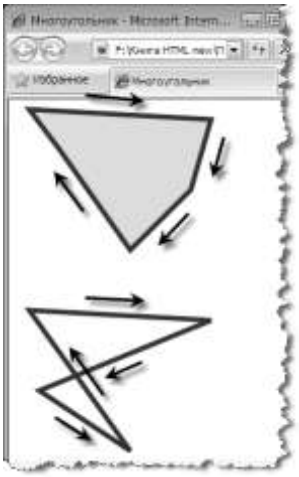


**Рис. 13.7.** Многоугольники

**Листинг 13.5. Создание многоугольников**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="300" height="360" >
  <title>Многоугольник</title>
  <polygon points="20,10 200,20 180,90 120,150 50,100"
     fill="#00ffff"
     stroke="blue" stroke-width="5"
  />
  <polygon points="20,210 200,220 30,290 120,350"
     fill="none"
     stroke="blue" stroke-width="5"
  />
</svg>
```

## 13.2.5. Линии

Вообще говоря, линии в векторной графике понимаются как объекты без области заливки. Однако в SVG это не так. Если линия непрямая, то область, ограниченная ею и отрезком прямой, соединяющим ее концы, по умолчанию залита черным цветом. Поэтому, если вы хотите нарисовать только линию, следует указать для нее отсутствие заливки (атрибут `fill="none"`).

Цвет и толщину линии задают атрибуты `stroke` и `stroke-width`.

Отрезок прямой линии задается тегом `<line>` с указанием координат начала и конца с помощью атрибутов:

- `x1`, `y1` — горизонтальная и вертикальная координаты начальной точки отрезка;
- `x2`, `y2` — горизонтальная и вертикальная координаты конечной точки отрезка.

Если не задавать цвет атрибутом `stroke`, то линия не будет видна.

На рис. 13.8 показан пример создания отрезка прямой линии. Задав достаточно большую толщину отрезка, можно получить прямоугольник.
Ломаную прямую линию (полилинию), состоящую из отрезков прямых, задает тег `<polyline>` с атрибутом `points`, принимающим в качестве параметра строку, которая содержит координаты концов отрезков. Синтаксис такой же, как и для тега `<polygon>`, определяющего многоугольник (см. *разд. 13.2.4*). Однако, в отличие от многоугольника, полилиния автоматически не замыкается. Тем не менее, по умолчанию внутренняя область, получающаяся при мысленном замыкании линии, представляется как область заливки черного цвета. Разумеется, с помощью атрибута `fill` можно задать другой цвет заливки или отменить его совсем.

На рис. 13.9 показаны два варианта незамкнутой полилинии. В первом варианте отображается область заливки, поскольку не указан атрибут `fill`. Чтобы область заливки не отображалась, во втором варианте был применен атрибут `fill="none"`.

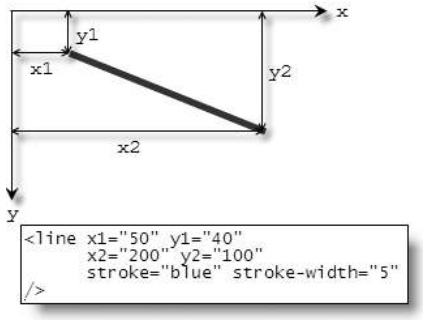



**Рис. 13.8.** Отрезок прямой линии

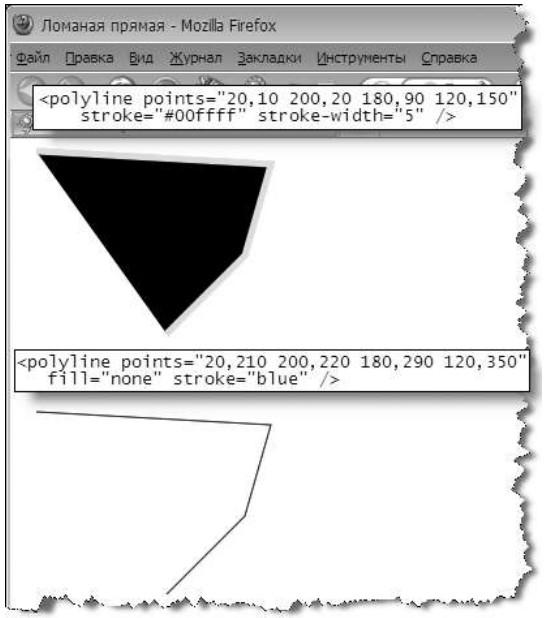



**Рис. 13.9.** Полилинии

Как уже отмечалось, цвет, толщина и степень прозрачности линий самих по себе, а также обводки (контура) фигур определяют атрибуты `stroke`, `stroke-width` и `stroke-opacity` соответственно. Вместе с тем, имеются и дополнительные атрибуты:

- `stroke-dasharray` — определяет стиль линии (пунктирная, штриховая и штрих-пунктирная); принимает в качестве значения одно, два или три числа, разделенных запятыми; единственное число указывает размер точек пунктирной (точечной) линии, пара чисел определяет соответственно размер штрихов и интервал между ними, а тройка чисел — размер точек, интервал между ними и размер штрихов между точками.

**Примеры**

```
<line x1="10" y1="10" x2="100" y2="10"
    stroke="black" stroke-width="2"
    stroke-dasharray="3, 10"/>

<rect x="" y="" width="" height=""
    fill="none" stroke="black"
    stroke-dasharray="3, 10,15"/>
```

- `dashoffset` — определяет отступ в процентах при задании линии с использованием атрибута `stroke-dash`; значение по умолчанию 0.

**Пример**

```
<line x1="10" y1="10" x2="100" y2="10"
    stroke="black" stroke-width="2"
    stroke-dasharray="3, 10"
    stroke-offset="25%"/>
```

- `stroke-linecap` — определяет вид концов линии; возможны значения `butt` (обычный, принятый по умолчанию), `round` (округленный) и `squarel` (с выступом)

**Пример**

```
<line x1="10" y1="10" x2="100" y2="10"
    stroke="black" stroke-width="2"
    stroke-dasharray="3, 10"
    stroke-linecap="round"/>
```

- `stroke-linejoin` — определяет вид соединений концов линий; возможны значения `mitre` (обычный, принятый по умолчанию), `round` (скругленный), `bevel` (рубленый).

**Пример**

```
<polyline  points="20,80 50,20 80,80"
    stroke="black" stroke-width="10"
    stroke- linejoin="bevel"/>
```

- `stroke-mitrelimit` — определяет ограничение на длину соединения двух линий, заданного с помощью атрибута `stroke-linejoin="mitre"`; значение по умолчанию 4.

## 13.3. Создание сложных фигур (тег <*path*>)

Сложные фигуры можно создавать как комбинации простых, рассмотренных в *разд. 13.2*. Вместе с тем, во многих случаях более удобен специальный тег `<path>`, определяющий некоторую траекторию (маршрут, путь), состоящую

из отдельных отрезков прямых, дуг и даже кривых Безье. Траекторию строят от некоторой опорной точки по частям, от одной точки до следующей, причем с помощью специальных команд указывают тип линии на том или ином участке. Последовательность команд и координат точек задается как значение атрибута `d` тега `<path>`. На рис. 13.10 показан пример некоторой линии, построенной из нескольких кривых Безье и отрезка прямой. Здесь `M`, `C` и `L` — команды, задающие соответственно опорную точку, линию Безье и отрезок прямой.

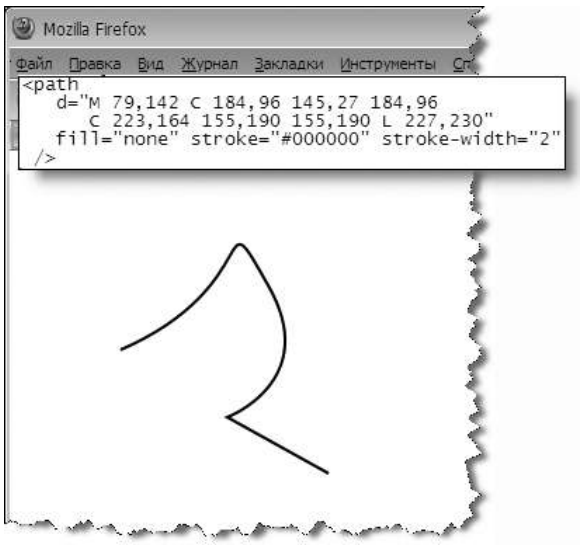



**Рис. 13.10.** Кривая, построенная с помощью тега `<path>`

Различают абсолютные и относительные команды. Координаты, указанные для абсолютных команд, отсчитываются в системе координат SVG-документа, а координаты относительных команд являются смещениями относительно текущей опорной точки. Абсолютные команды обозначают прописными буквами, а относительные — строчными.

Наиболее распространенные команды для рисования прямых линий и простых фигур (форм):

- `M`, `m` — устанавливает начальную опорную точку с координатами `x`, `y`, указанными в качестве параметров;
- `L`, `l` — рисует отрезок прямой линии от текущей опорной точки до точки, заданной параметрами `x`, `y`; конец отрезка становится текущей опорной точкой;

- `H`, `h` — рисует отрезок горизонтальной прямой линии от текущей опорной точки до точки, координата `x` которой задана параметром; конец отрезка становится текущей опорной точкой;
- `V`, `v` — рисует отрезок вертикальной прямой линии от текущей опорной точки до точки, координата y которой задана параметром; конец отрезка становится текущей опорной точкой;
- `Z`, `z` — завершает траекторию, рисует прямую линию от текущей до начальной опорной точки, заданной с помощью команды `M` или `m` (т. е. замыкание траектории).

Например, следующая запись определяет отрезок прямой линии от точки с координатами (50, 100) до точки с координатами (120, 150):

```
<path d="M 50, 100 L 120,150"/>
```

Такую же линию можно определить и с помощью аналогичной относительной команды `l`, параметры которой задают смещения относительно текущей опорной точки (в данном случае — начальной):

```
<path d="M 50, 100 l 70, 50"/>
```

При использовании относительных команд параметры, указывающие смещения, могут быть отрицательными числами.

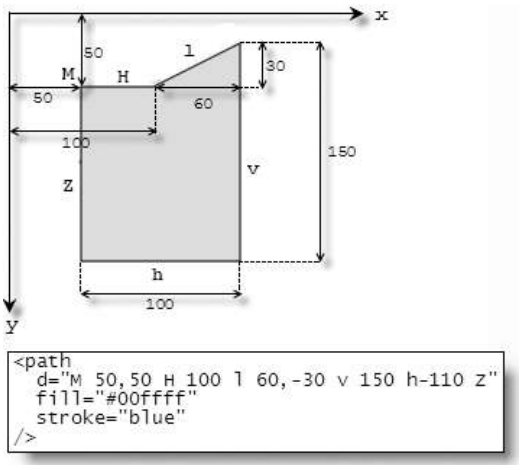



**Рис. 13.11.** Пример использования абсолютных и относительных команд для рисования фигур

На рис. 13.11 показан пример многоугольника, нарисованного с помощью абсолютных и относительных команд. Сначала команда `M50,50` устанавливает опорную точку с указанными координатами в пикселах в абсолютной системе

координат SVG-документа. Затем от нее с помощью абсолютной команды `H100` проводится горизонтальная линия до точки с горизонтальной координатой `100px` в той же абсолютной системе координат. Конец этого отрезка становится текущей опорной точкой. Далее следуют относительные команды. Так, команда `l 60,-30` рисует отрезок прямой от текущей опорной точки до точки, сдвинутой от последней по горизонтали на `60px` и по вертикали — на `-30px`. В абсолютной системе конец данного отрезка имеет координаты 110 и 20 пикселов соответственно по горизонтали и вертикали. Далее следуют относительные команды рисования вертикальной и горизонтальной прямых, а затем команда замыкания получившейся ломаной прямой, т. е. команда проведения прямой линии от текущей опорной точки до начальной опорной точки, заданной командой `M50,50`.

Атрибут `d` тега `<path>` допускает сокращение в записи значения: запятые можно не указывать, символы повторяющихся команд можно опускать, в дробном числе, меньшем по модулю единицы, ноль можно опускать (например, вместо 0.5 писать .5).

На рис. 13.12 показан многоугольник в виде звезды, созданный посредством последовательности отрезков прямых. Команда `L` здесь явно упоминается только один раз.

Для рисования эллиптических кривых и фигур в атрибуте `d` тега `<path>` применяется команда `A` (`a`), в абсолютном и относительном своих вариантах. Кривая рисуется от текущей опорной точки до заданной конечной точки и доопределяется параметрами данной команды.

Алгоритм рисования эллиптической кривой поясняет рис. 13.3. Кривая создается как дуга между точками пересечения двух одинаковых по размерам эллипсов. Одна точка пересечения это текущая опорная точка, а вторая — заданная посредством параметров `x` и `y` конечная точка. Размеры эллипсов задают горизонтальный `rx` и вертикальный `ry` радиусы, а ориентацию — параметр `x-axis-rotation`. Центры эллипсов вычисляются автоматически.

Очевидно, в системе из двух пересекающихся эллипсов можно выделить четыре дуги. Нужную из них выбирают с помощью двух параметров, принимающих значения 0 и 1: `large-arc-flag` (выбор большой или малой дуги) и `sweep-flag` (выбор эллипса).

Параметр `x-axis-rotation` определяет поворот в градусах горизонтальной оси эллипсов относительно горизонтальной оси текущей системы координат.

Формат записи команды `A`:

```
A rx,ry x-axis-rotation large-arc-flag,sweep-flag x,y
```

Здесь запятые можно заменить пробелами. Возможен также относительный вариант данной команды (`a`).

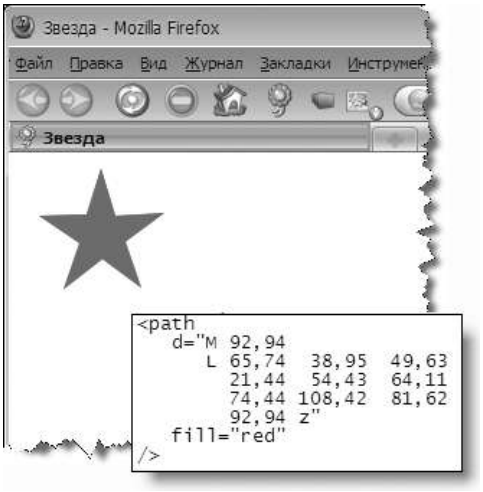



**Рис. 13.12.** Звезда, нарисованная посредством `<path>`

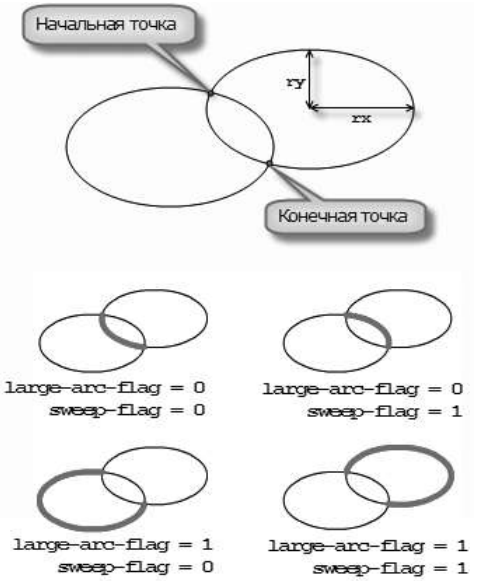



**Рис. 13.13.** Определение вида эллиптической кривой

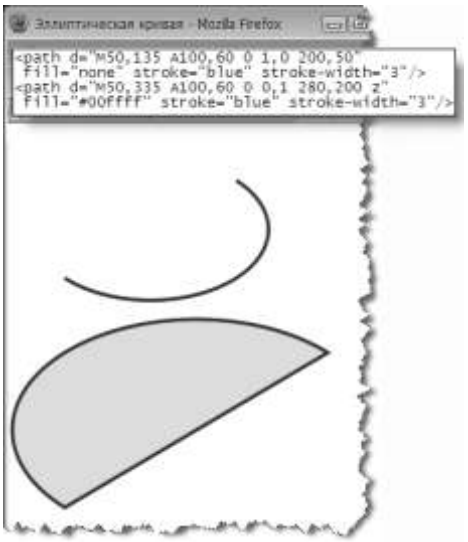



**Рис. 3.14.** Примеры эллиптических кривых

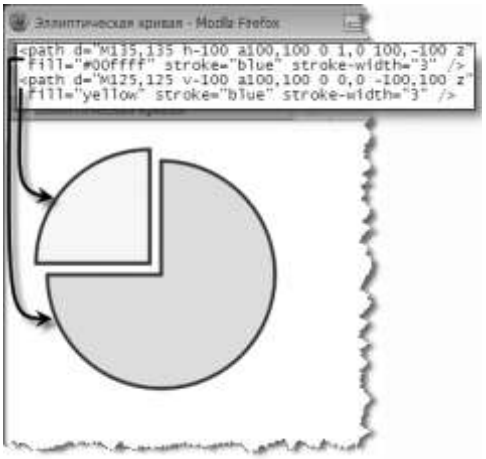



**Рис. 13.15.** Круговая диаграмма

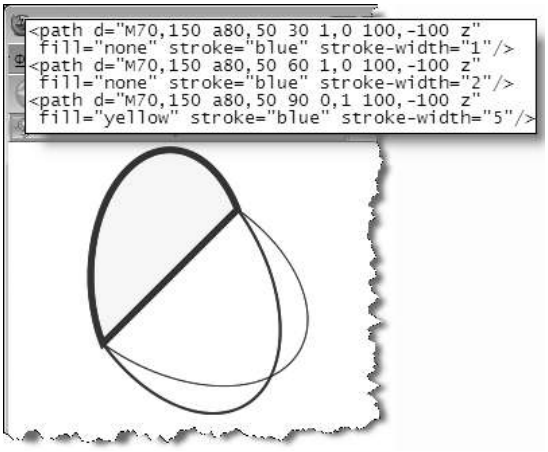



**Рис. 13.16.** Влияние параметра поворота горизонтальной оси на вид эллиптической кривой

На рис. 13.14 показаны примеры двух эллиптических кривых, нарисованных с применением абсолютного варианта команды, а на рис. 13.15 — две фигуры (что-то вроде круговой диаграммы), созданные с применением относительного варианта команды и команд рисования отрезков прямых. На рис. 13.16 показано влияние параметра `x-axis-rotation` на вид кривой. В данном примере он принимает значения 30, 60 и 90 градусов. Последний случай рассмотрен при иных значениях параметров `large-arc-flag` и `sweep-flag`.

Чтобы нарисовать замкнутую фигуру, необходимо обеспечить некоторое, пусть незаметное глазу, расстояние между начальной и конечными точками. Если эти точки совпадают (предельный случай), то фигура не будет отображаться.

Например, круг можно создать следующим образом:

```
<path d="M200,200 A50,50 0 1,1 200.01,200.01"
      stroke="black" fill="none"/>
```

Здесь координаты начальной и конечной точек отличаются на 0,01 px.

Для рисования сложных и гладких кривых в векторных графических редакторах используются так называемые кривые Безье. На рис. 13.10 показан пример кривой, фрагменты которой являются кривыми этого типа. В записи значения атрибута `d` тега `<path>` можно применять специальные команды `C` (`c`), `S` (`s`) для рисования кривых Безье третьего прядка и команды `Q` (`q`), `T` (`t`) для рисования кривых Безье четвертого порядка. Однако вручную написать код, реализующий кривую данного типа, довольно трудно. Обычно сложные фигуры на основе кривых Безье, создают в векторных редакторах (например, CorelDraw, Adobe Illustrator, Inkscape и др.), сохраняют изображение в форма-

те SVG, а затем заимствуют нужный фрагмент кода, для копирования в свой SVG-документ. В данной книге мы не будем рассматривать команды, с помощью которых строятся кривые Безье.

## 13.4. Вставка растровых изображений

В SVG-документ можно вставлять растровые изображения форматов JPEG, PNG и GIF из внешних файлов. Это делается с помощью тега `<image>` со следующими атрибутами:

- `xlink:href` — URL-адрес графического файла;
- `x`, `y` — горизонтальная и вертикальная координаты верхнего левого угла изображения;
- `width`, `height` — ширина и высота изображения.

Пример вставки графического изображения приведен в листинге 13.6. Обратите внимание на атрибуты `xmlns`, задающие пространство имен.

**Листинг 13.6. Вставка растрового графического изображения**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="300" height="250" >
<title>Графическое изображение</title>
<desc>Вставка картинки из файла </desc>
<image xlink:href="mypicture.jpg"
 x="50" y="10" width="100px" height="200px"/>
</svg>
```

Атрибуты `width` и `height` обязательны; если их не указывать, то картинка не будет отображаться. Значения этих атрибутов могут отличаться от оригинальных размеров графического изображения. В таком случае браузеры Firefox, Opera, Safari и Chrome отображают его с сохранением оригинальных пропорций, но, возможно, нарушая заданное позиционирование. Internet Explorer с плагином Adobe SVGViewer может исказить пропорции, выдерживая заданные размеры. Чтобы добиться инвариантности отображения картинки, следует устанавливать значения атрибутов `width` и `height` с соблюдением оригинальных пропорций.

## 13.5. Применение CSS

В предыдущих разделах параметры внешнего вида фигур задавались с помощью атрибутов, таких как `fill`, `fill-opacity`, `stroke`, `stroke-width`, `stroke-opacity`. Вместе с тем, их можно определять посредством каскадных таблиц стилей различными способами:

- с помощью атрибута `style` элемента SVG-документа;
- заданием таблицы стилей внутри SVG-документа;
- определением таблицы стилей во внешнем файле и указанием ссылки на него в SVG-документе.

Определения стилевых параметров имеют такой же синтаксис, что и применительно к (X)HTML-документам: имя и значения параметра разделяют двоеточием, а сами определения различных параметров — точкой с запятой. Однако имена стилевых параметров в SVG совпадают с соответствующими именами атрибутов SVG-тегов. По крайней мере, они не обязательно такие же, что и в CSS для (X)HTML-документов. Например, в CSS для (X)HTML-документов заливка (фон) элемента определяется параметром `background`, а для SVG-документа — параметром `fill`.

Следующий SVG-тег задает прямоугольник с параметрами, определенными с помощью атрибутов:

```
<rect
   x="50" y="30"
   width="100" height="50"
   fill="red"
   stroke="blue" stroke-width="3"
/>
```

То же самое можно сделать с помощью атрибута `style`:

```
<rect
   x="50" y="30"
   width="100" height="50"
   style="fill:red; stroke:blue; stroke-width:3px"
/>
```

Правила стилей можно оформить отдельным блоком кода, вложенным в дескриптор `<![CDATA[ … ]]>`, который, в свою очередь, вложен в контейнер `<style>`.

В качестве элементарных селекторов правил CSS допустимы имена тегов, значения атрибутов `id`, а также имена классов. Связать элемент с набором стилевых параметров, селектором которого является имя класса, позволяет атрибут `class`.

Пример использования CSS внутри SVG-документа приведен в листинге 13.7. Здесь селектором определения набора стилевых параметров служит `#myrect` — значение атрибута `id="myrect"` тега `<rect>`.

**Листинг 13.7. Пример использования CSS в SVG-документе**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="300" height="250" >
<title>Применение CSS</title>
<style type="text/css">
<![CDATA[
  #myrect {
     fill:#00ffff;
     stroke:blue; stroke-width:3px
  }
]]>
</style>
<rect id="myrect"
   x="50" y="30"
   width="100" height="50"
/>
</svg>
```

Наборы правил CSS можно сохранить в отдельном файле, а в SVG-документе в специальном XML-дескрипторе перед дескриптором `<!DOCTYPE…>` указать ссылку на него (листинг 13.8).

**Листинг 13.8. Пример использования ссылки на внешнюю CSS**

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet href="css/mystyle.css" type="text/css"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="400" height="500" >
<title>Применение CSS</title>
<rect id="myrect"
   x="50" y="30"
```

```
   width="100" height="50"
/>
</svg>
```

Чтобы определить фон SVG-документа, достаточно задать стилевой параметр `background` для тега `<svg>`, например,

```
<svg style="background:blue" …/>
```

# 13.6. Группировка элементов

Несколько элементов SVG-документа можно объединить в одну группу. Это удобно при назначении всем элементам группы одинаковых стилевых параметров или выполнения преобразований над группой как единым целым. Для группировки элементов применяется контейнерный тег `<g>`. Он может содержать в себе и другие контейнеры `<g>`.

На рис. 13.17 показан пример, в котором сгруппированы прямоугольник, круг и треугольник. Стилевые параметры, общие для всех фигур, заданы посредством атрибута `style` тега `<g>` группы: цвета заливки и обводки, а также уровень прозрачности заливки. Но для прямоугольника определен специфический цвет заливки с помощью своего атрибута `style= "fill:blue"`.

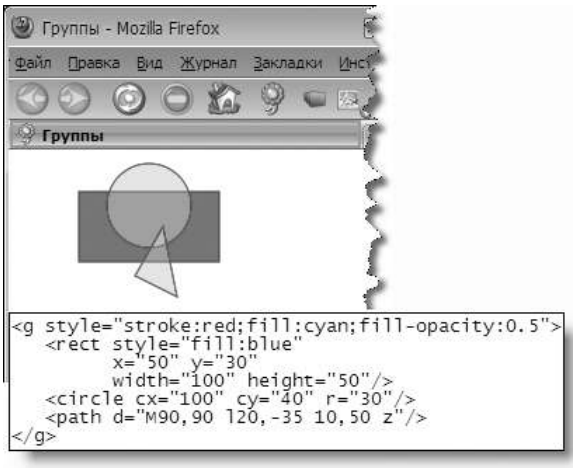



**Рис. 13.17.** Применение группировки элементов для назначения общих параметров

Другой пример группировки элементов будет рассмотрен в следующем разделе.

# 13.7. Третье измерение, определения и клонирование элементов

Графические элементы позиционируются на плоскости с помощью атрибутов `x` и `y`. При этом они могут перекрывать друг друга. Их положение выше/ниже определяется по умолчанию порядком упоминания в коде подобно HTML-элементам в нормальном потоке. Чем ниже (позднее) в коде упоминается элемент, тем ближе он к переднему плану. Например, фигуры на рис. 13.17 располагаются в таком порядке: внизу находится прямоугольник, над ним — круг, а на переднем плане — треугольник. В данном примере заливка фигур полупрозрачная (`fillopacity:0.5`) и поэтому они все видны. В противном случае вышерасположенные фигуры частично закрывали бы находящихся ниже. Тем не менее, положением элементов в третьем измерении можно управлять худо-бедно с помощью тегов `<defs>` и `<use>`.

В контейнер `<defs>` включают теги элементов. При этом они не отображаются, а только определяются для будущего использования. Затем с помощью тега `<use>` можно отобразить в нужном месте любой элемент, определение которого дано в контейнере `<defs>`. Более того, любой ранее определенный элемент можно клонировать (воспроизводить) сколько угодно раз и в любом месте пространства документа.

В листинге 13.9 приведен код, в котором в контейнере `<defs>` определена группа (контейнер `<g>`), содержащая три элемента: прямоугольник, круг и треугольник. Для каждого из этих элементов, включая и группу, указан атрибут `id` (идентификатор элемента). Этот атрибут понадобится для ссылок в теге `<use>` для клонирования элементов. Далее, с помощью тегов `<use>` отображается сначала вся группа в двух вариантах, а затем отдельные ее элементы, причем в порядке, отличном от того, в каком они упоминались в контейнере определений `<defs>`. Обратите внимание на позиционирование элементов с учетом координат, заданных атрибутами `x` и `y` тегов `<use>`. Эти атрибуты задают новую систему координат, относительно которой позиционируются элементы, перечисленные в контейнере `<defs>`. Так, каждый элемент, определенный в контейнере `<defs>`, имеет свои собственные координаты позиционирования, но его отображаемый клон позиционируется относительно координат, заданных атрибутами `x` и `y` соответствующего тега `use`: координаты просто суммируются.

Поскольку в тегах `<use>` присутствуют ссылки на элементы по их `id`, в теге `<svg>` следует указать соответствующее пространство имен:

```
xmlns:xlink =" http://www.w3.org/1999/xlink"
```

Вид данного документа в браузере показан на рис. 13.18. Чтобы легче понять, как осуществляется позиционирование, указаны координаты четырехугольника.

**Листинг 13.9. Применение тегов <def> и <use>**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="500">
<title>defs &amp; use</title>
<desc> Определение элементов </desc>
<defs>
  <g id="mygroup" style="stroke:red">
     <rect id="myrect" style="fill:blue"
           x="50" y="30"
           width="100" height="50"
     />
     <circle id="mycircle" style="fill:cyan"
           cx="100" cy="40" r="30"
     />
     <path id="mypath" style="fill:cyan"
           d="M90,90 l20,-35 10,50 z"
     />
  </g>
</defs>
<desc> Рисуем первый клон группы </desc>
<use x= "0" y= "0" xlink:href="#mygroup" />
<desc> Рисуем второй клон клон группы </desc>
<use x="150" y="50" xlink:href="#mygroup" />

<desc> Рисуем отдельно элементы </desc>
<use x="50"  y="150" xlink:href="#mycircle"/>
<use x="85"  y="150" xlink:href="#mypath"  />
<use x="30"  y="180" xlink:href="#myrect"  />
<use x="-30" y="250" xlink:href="#myrect"  />
</svg>
```

В теге <use> можно указать атрибуты стиля и/или стилевые параметры CSS, чтобы задать внешний вид клона элемента, отличный от ранее определенного в <defs>, например,

```
<use x="50" y="250" xlink:href="#myrect"
      style="fill:red; stroke:black"/>
```

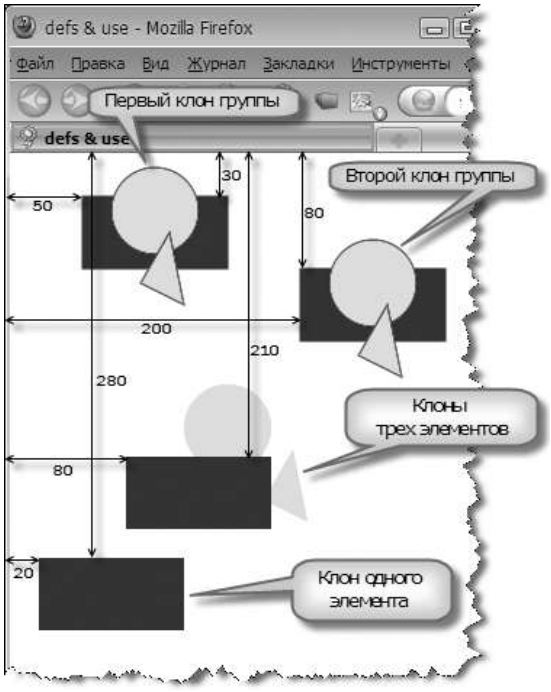



**Рис. 13.18.** Клонирование элементов, определенных в контейнере <defs> (листинг 13.7)

## 13.8. Градиентная заливка

Атрибут или стилевой параметр `fill` определяет цвет заливки элемента сплошным или, как еще говорят, твердым цветом. Градиентная заливка (далее просто градиент) — это окрашивание внутренней области элемента двумя и более цветами с плавными переходами между ними. Возможно также указание направления перехода. В SVG существуют линейные и радиальные градиенты.

## 13.8.1. Линейный градиент

Линейный градиент определяется контейнерным тегом <linearGradient>, в котором размещаются теги и атрибуты, задающие свойства градиента, такие как цвета, между которыми должен происходить переход, направление перехода и др.

Градиент определяют в контейнере <defs> (см. *разд. 13.7*), а в теге элемента указывают ссылку на него, чтобы применить к нему данную градиентную заливку. В листинге 13.10 приведен пример заливки прямоугольника линейным градиентом, а на рис. 13.19 — его вид в браузере.

В контейнере <linearGradient> размещают теги <stop>, посредством которых задают характеристики градиента. В теге <stop> атрибут stop-color определяет опорный (сплошной) цвет, который должен далее использоваться при создании плавного перехода к другому опорному цвету, заданному в следующем теге <stop>. Атрибут offset тега <stop> задает координату (в процентах), начиная с которой сплошной цвет начинает плавно переходить в следующий опорный цвет. Таким образом, атрибуты stop-color и offset тега <stop> определяют, какой цвет должен преобразовываться, и с какого места (в процентах от ширины заливаемой фигуры) должно начаться преобразование.

Элемент, к которому применяется градиентная заливка, должен иметь атрибут fill="url(#*значение id градиента*)", содержащий ссылку на требуемый градиент (значение атрибута id тега <linearGradient>).

В рассматриваемом примере применяются два тега <stop>, задающие черный и белый опорные цвета. Область черного цвета занимает от левого края фигуры (прямоугольника) по горизонтали 25%. Область белого цвета простирается от 75% по горизонтали до правого края фигуры. Промежуточная область от 25% до 75% залита переходным цветом, т. е. плавно изменяющимся от черного к белому. Разумеется, опорные цвета и координаты области переходного цвета могут быть различными.

**Листинг 13.10. Линейный градиент**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="500">
<title>Линейный градиент</title>
```

```
<defs>
  <linearGradient id="mygradient">
    <stop offset="25%" stop-color="black"/>
    <stop offset="75%" stop-color="white"/>
  </linearGradient>
</defs>
<rect x="50" y="50" width="4cm" height="2cm" stroke="black"
      fill="url(#mygradient)"
/>
</svg>
```

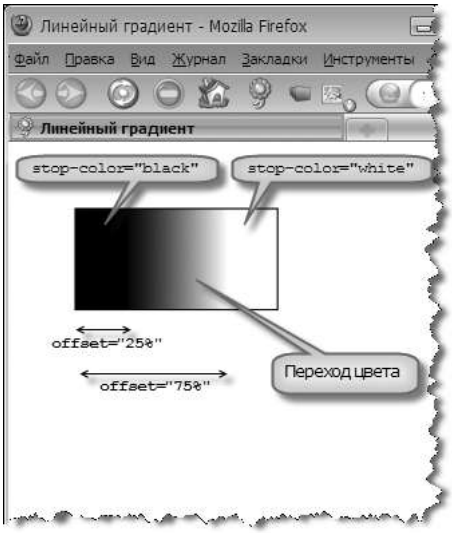



**Рис. 13.19.** Заливка прямоугольника линейным градиентом (листинг 13.10)

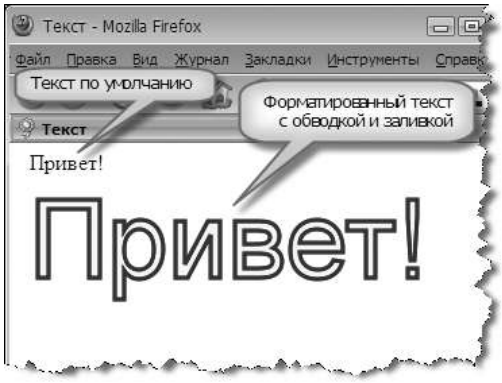



**Рис. 13.20.** Примеры линейных градиентов

Варьируя границами областей, задаваемых атрибутами `offset`, можно получать переходы цветов с различной плавностью, вплоть до вырожденного случая, когда плавного перехода нет совсем. Кроме того, можно создавать градиенты с более чем одной областью перехода, применяя более двух тегов `<stop>`. На рис. 13.20 приведено несколько примеров. Так наверху показан градиент с областью перехода 100% ширины фигуры, в середине — вырожденный случай градиента, когда область перехода занимает 0% ширины фигуры, а внизу — градиент с двумя областями перехода, от черного к белому и от белого к черному.

По умолчанию градиент имеет направление слева направо. Однако с помощью атрибутов `x1,y1` и `x2,y2`, задающих координаты начала и конца вообра-

жаемой прямой линии, можно задать направление градиента явно. Координаты задают в процентах от ширины и высоты элемента, к которому применяется градиентная заливка. Чтобы явно задать направление градиента, совпадающее с направлением по умолчанию (когда атрибуты `x1,y1` и `x2,y2` опущены), достаточно записать так:

```
<linearGradient x1="0%" y1="0%" x2="100%" y2="0%" …>
```

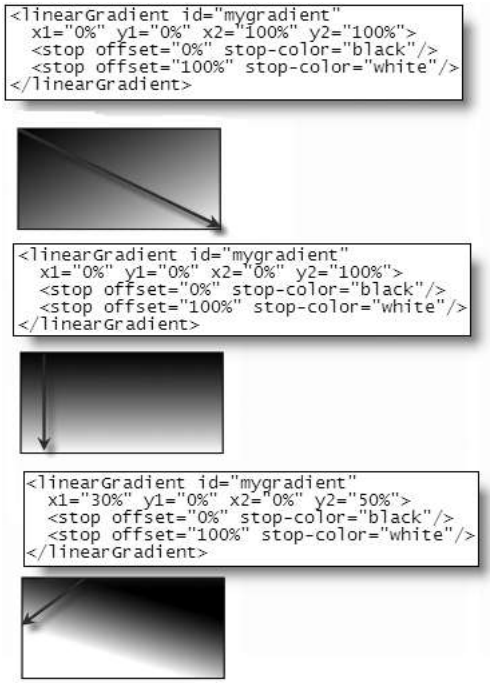



**Рис. 13.21.** Примеры линейных градиентов с негоризонтальными направлениями, указанными стрелками

Впрочем, следующая запись также задает направление градиента слева направо:

```
<linearGradient x1="0%" y1="50%" x2="100%" y2="50%" …>
```

Задать направление сверху вниз можно с пмощью такой записи:

```
<linearGradient x1="0%" y1="0%" x2="0%" y2="100%" …>
```

На рис. 13.21 показано несколько примеров градиентов с негоризонтальными направлениями.

Применяя градиенты с двумя и более областями перехода цветов, а также устанавливая нужное направление, можно создавать эффект объемности фигур, например, изображений кнопок.

Для управления степенью прозрачности цветов в градиентной заливке в теге `<stop>` есть атрибут `stop-opacity`, принимающий значения от нуля до единицы. Примеры использования данного атрибута будут рассмотрены в следующем разделе.

## 13.8.2. Радиальный градиент

Радиальный градиент определяется контейнерным тегом `<radialGradient>`. Задание данного типа градиента аналогично построению линейного градиента, но вместо атрибутов, определяющих линию направления градиента, можно указать координаты центра `cx`, `cy` и радиус `r`. Значения данных атрибутов задают в процентах. Цвет меняется от заданного центра к периферии элемента, к которому градиент применен. Если атрибуты не указывать, переход цветов происходит от геометрического центра элемента.

В листинге 13.11 приведены примеры заполнения радиальным градиентом квадрата, прямоугольника и круга, когда атрибуты `cx`, `cy` и `r` в теге `<radialGradient>` не указаны (рис. 13.22)

**Листинг 13.11. Примеры радиального градиента**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="500">
<title>Радиальный градиент</title>
<defs>
  <radialGradient id="mygradient">
    <stop offset="0%" stop-color="black"/>
    <stop offset="100%" stop-color="white"/>
  </radialGradient>
</defs>
<rect x="10" y="10" width="2cm" height="2cm" stroke="black"
      fill="url(#mygradient)"
/>
<rect x="10" y="3cm" width="4cm" height="2cm" stroke="black"
      fill="url(#mygradient)"
```

```
/>
<circle cx="6cm" cy="2cm" r="1.5cm" height="2cm" stroke="black"
      fill="url(#mygradient)"
/>
</svg>
```

На рис. 13.23 показан вид документа листинга 13.11, но при добавлении атрибутов градиента:

```
<radialGradient id="mygradient"
   cx="25%" cy="30%" r="50%"
>
```

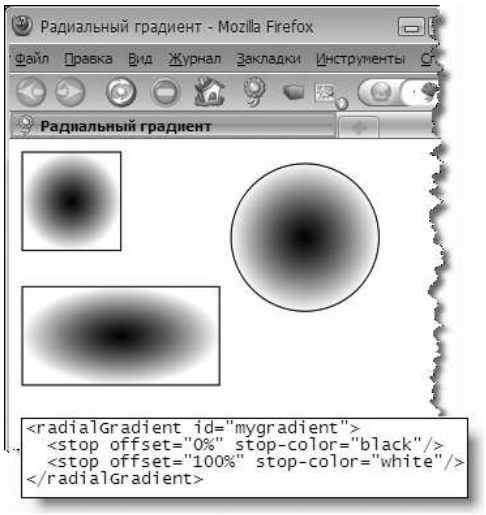



**Рис. 13.22.** Примеры радиального градиента без указания атрибутов `cx`, `cy` и `r` (листинг 13.11)

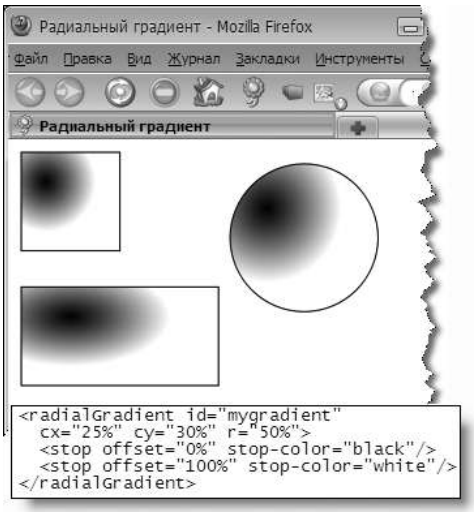



**Рис. 13.23.** Примеры радиального градиента при `cx="25%"`, `cy="30%"` и `r="50%"`

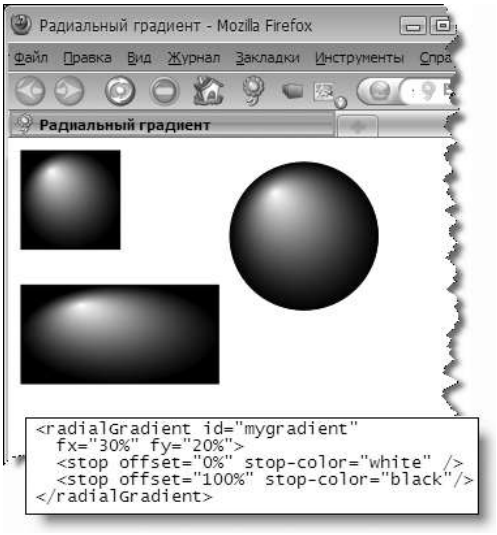



**Рис. 13.24.** Эффект цветового блика

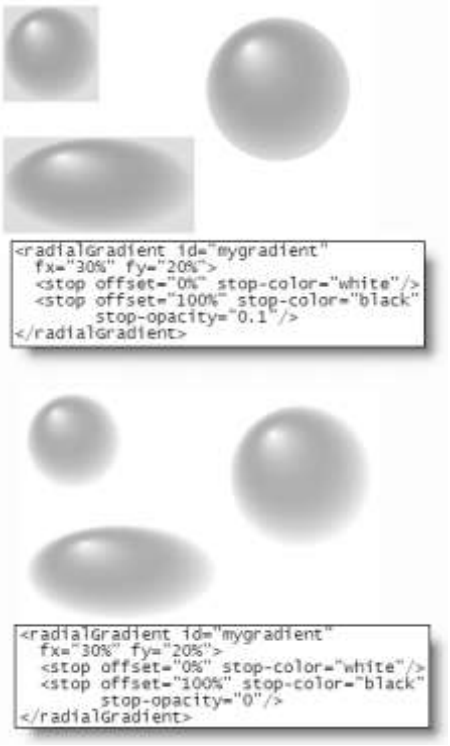






**Рис. 13.25.** Применение атрибута `stop-opacity`

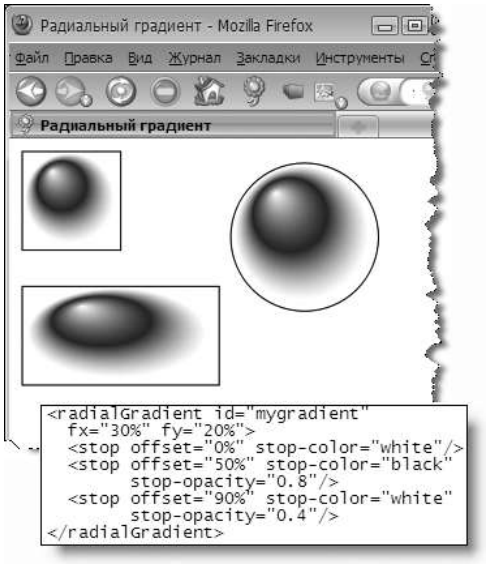



**Рис. 13.26.** Применение трех тегов `<stop>`, эффекта светового блика и управление степенью прозрачности

Для создания эффекта светового блика в теге <radialFradient> применяются атрибуты fx и fy, задающие центр блика (рис. 13.24). Дополнительные эффекты достигаются добавлением в тег <stop> атрибута stop-opacity, задающего степень прозрачности цвета от нуля до единицы (рис. 13.25). Элементы на данных рисунках показаны без обводки.

Листинг 13.12 содержит пример радиального градиента с использованием трех тегов <stop>, создающих две области перехода цветов. Для наглядности элементы отображаются с обводкой. Здесь применяются эффект светового блика и управление степенью прозрачности. Вид данного документа в браузере показан на рис. 13.26.

**Листинг 13.12. Пример радиального градиента**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="500">
<title>Радиальный градиент</title>

<defs>
  <radialGradient id="mygradient"
    fx="30%" fy="20%">
    <stop offset="0%" stop-color="white"/>
    <stop offset="50%" stop-color="black"
          stop-opacity="0.8"/>
    <stop offset="90%" stop-color="white"
          stop-opacity="0.4"/>
  </radialGradient>
</defs>

<rect x="10" y="10" width="2cm" height="2cm" stroke="black"
      fill="url(#mygradient)"
/>
<rect x="10" y="3cm" width="4cm" height="2cm" stroke="black"
      fill="url(#mygradient)"
/>
<circle cx="6cm" cy="2cm" r="1.5cm" height="2cm" stroke="black"
      fill="url(#mygradient)"
/>
</svg>
```

# 13.9. Маски

Маска — это графический объект, накладываемый на другой так, чтобы перекрываемая маской часть была видна, а все остальное — нет. Это мощный инструмент для создания комбинированных изображений (коллажей), имеющийся во всех серьезных графических редакторах. Маску можно создать посредством тегов `<mask>` и `<clipPath>`.

## 13.9.1. Тег *<mask>*

Для создания маски можно применить контейнерный тег `<mask>`, в который включают теги элементов, образующие графический объект маски. В качестве последнего может выступать одна и более фигур, а также текст. При этом фигура маски может быть залита градиентом, что позволяет создавать интересные эффекты.

В листинге 13.13 приведен пример простой маски в виде круга, наложенного на растровое графическое изображение. В теге `<image>` указан атрибут `mask="url(#значение id маски)"`, указывающий на маску, которую требуется применить (в данном случае `mymask`). Вид данного документа в браузере показан на рис. 13.27.

**Листинг 13.13. Пример простой маски**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="500">
<title>Маска</title>
<defs>
  <mask id="mymask">
     <circle cx="140" cy="120" r="100" fill="white"/>
  </mask>
</defs>
<image xlink:href="picture.jpg" mask="url(#mymask)"
    x="0" y="0" width="320px" height="256px"/>
</svg>
```

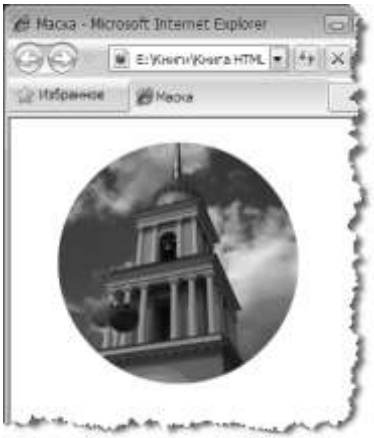



**Рис. 13.27.** Пример маски в виде круга, наложенной на растровое графическое изображение (листинг 13.13)


**Рис. 13.29.** Маска и двух пересекающихся кругов с радиальными градиентами (листинг 13.15)

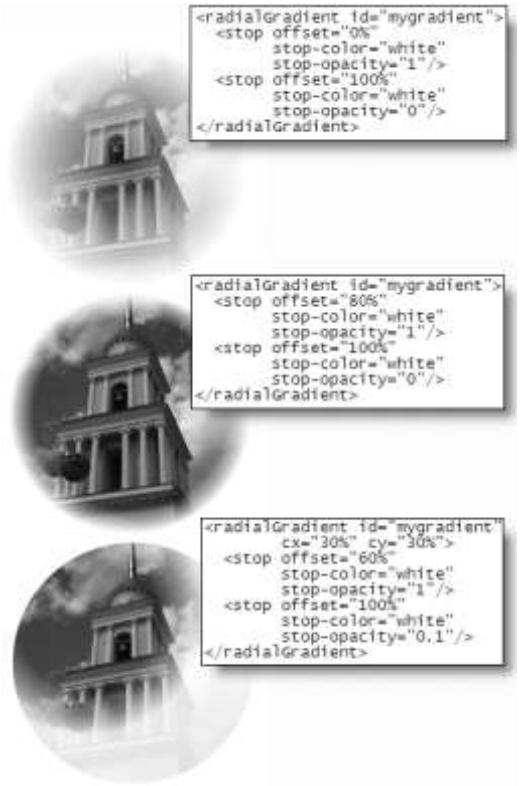



**Рис. 13.28.** Маска с радиальным градиентом (листинг 13.14)

Заполнение (заливка) маски (атрибут `fill` элемента внутри тега `<mask>`) может быть сплошным (любого цвета), с заданием или без степени прозрачности, а также градиентным. Цвет заливки маски не влияет на цветовое представление маскируемого объекта, но косвенно задает ее степень прозрачности. Так, при белом цвете заливки (`white` или `#ffffff`) фигура маски оказывается полностью прозрачной. При других цветах она становится менее прозрачной и совсем непрозрачной при черном цвете заливки или его отсутствии (`fill="none"`). Разумеется, можно задать белый цвет заливки маски, а степень ее прозрачности установить посредством атрибута `fill-opacity`.

В листинге 13.14 приведен пример маски с радиальной градиентной заливкой, а на рис. 13.28 — несколько вариантов вида данного документа в браузере при различных параметрах градиента.

**Листинг 13.14. Пример маски с градиентной заливкой**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="500">
<title>Маска с градиентом</title>
<defs>
  <radialGradient id="mygradient">
    <stop offset="0%" stop-color="white" stop-opacity="1"/>
    <stop offset="100%" stop-color="white" stop-opacity="0"/>
  </radialGradient>
  <mask id="mymask">
     <circle cx="140" cy="120" r="100" fill="url(#mygradient)"/>
  </mask>
</defs>
<image xlink:href="picture.jpg" mask="url(#mymask)"
    x="0" y="0" width="320px" height="256px"/>
</svg>
```

В теге `<mask>` могут быть заданы несколько фигур, определяющих форму маски. Если изображения фигур маски пересекаются, то их области заливки объединяются. Пример маски из двух пересекающихся кругов с радиальным градиентом приведен в листинге 13.15 (рис. 13.29).

**Листинг 13.15. Пример маски, состоящей из двух кругов**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="500">
<title>Маска</title>
<defs>
  <radialGradient id="mygradient"
    cx="50%" cy="50%">
```

```
    <stop offset="80%" stop-color="white" stop-opacity="1"/>
    <stop offset="100%" stop-color="white" stop-opacity="0.2"/>
  </radialGradient>
  <mask id="mymask">
     <circle cx="110" cy="120" r="90" fill="url(#mygradient)"/>
     <circle cx="220" cy="120" r="90" fill="url(#mygradient)"/>
  </mask>
</defs>
<image xlink:href="P-0068.jpg" mask="url(#mymask)"
    x="0" y="0" width="320px" height="256px" />
</svg>
```

В качестве маски можно использовать не только фигуры, но и тексты. Пример такого рода маски приведен в *разд. 13.10*. Там же рассматривается и применение двух масок.

## 13.9.2. Тег <*clipPath*>

В контейнерный тег `<clipPath>` вставляют элементы, задающие область маски подобно тому, как это делалось при использовании тега `<mask>`. Однако, в отличие от `<mask>`, здесь игнорируются все графические свойства фигуры маски (цвет, прозрачность и др.). Фигура маски задает лишь контур последней. Накладывая такую маску на объект, мы просто отсекаем все, что выходит за ее границы, никак не влияя на графические свойства того, что находится внутри. Таким образом, функционально `<clipPath>` — частный случай `<mask>`. Вот пример маски типа `clipPath`:

```
<clipPath>
    <path d="M50 40 500 40 500 600 50 600 z" />
</clipPath>
```

Чтобы применить маску, в атрибуте `clip-path` можно указать ссылку по значению атрибута `id` тега `<clipPath>`. В следующем примере маска в виде замкнутой ломаной линии накладывается на прямоугольник:

```
<clipPath id="myclip">
    <path d="M50 40 500 40 500 600 50 600 z" />
</clipPath>
<rect clip-path="url(#myclip)"
x="10" y="20" width="400" height="700" fill="red"/>
```

# 13.10. Тексты

Тексты (наборы символов) в SVG-документе размещают в контейнерном теге `<text>` и над ними выполняют преобразования как и над графическими объектами. Вместе с тем, они могут быть проиндексированы поисковыми систе-

мами, а также выделены в окне браузера и скопированы в буфер обмена. При этом поддерживается кодировка Unicode, обеспечивающая возможность локализации кода. Иначе говоря, содержимое контейнера `<text>` в SVG ведет себя и как обычный текст, и как графический объект.

Итак, чтобы создать некоторый текст, его следует заключить в контейнер `<text>`. Текстовый элемент позиционируют посредством атрибутов `x` и `y`, задающих горизонтальную и вертикальную координаты воображаемой линии, вдоль и над которой будет располагаться текст. Это абсолютные координаты, отсчитываемые от верхнего левого угла SVG-документа. Если, например, `y="0"`, то текст почти не будет виден (необходимо оставить место с учетом размера шрифта).

Для форматирования текста предусмотрено множество атрибутов, таких как `font-family`, `font-size`, `font-style`, `text-decoration`, `fill`, `stroke` и др. Вместе с тем, можно задействовать стилевые параметры CSS (см. *разд. 13.5*).

Как графический объект, содержимое тега `<text>` кроме набора символов текста имеет обводку (`stroke`) и область заливки (`fill`). Влияние данных параметров заметно при достаточно большом размере шрифта текста.

В листинге 13.16 приведен пример создания двух текстовых блоков. Первый из них имеет параметры, принятые по умолчанию, а параметры второго заданы посредством CSS. Вид в браузере данного документа показан на рис. 13.30.

> ***Примечание***
>
> Следует иметь в виду, что шрифты по умолчанию в различных браузерах могут отличаться. Поэтому рекомендуется явно задавать гарнитуру (`font-family`) и размер (`font-size`).

**Листинг 13.16. Примеры текстовых блоков**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="150">
<title>Текст</title>
<defs>
  <style type="text/css">
  <![CDATA[
    .mytext {
      font-size:80px;
      font-family:"arial";
```

```
      fill:yellow;
      stroke:blue;
      stroke-width:3px;
    }
  ]]>
  </style>
</defs>
<text x="15" y="20">
    Привет!
</text>
<text x="15" y="100" class="mytext">
    Привет!
</text>
</svg>
```

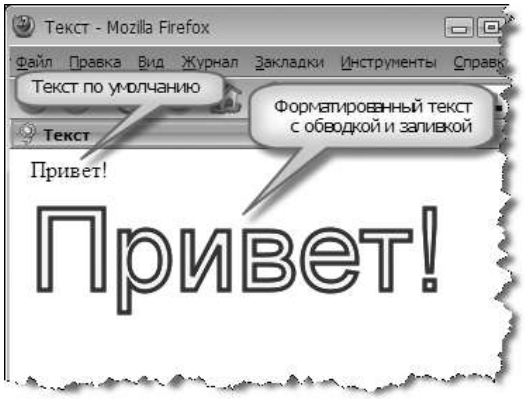



**Рис. 13.30.** Примеры текстовых блоков

К тексту можно применять градиентную заливку подобно тому, как это делается для обычных графических фигур и линий (см. *разд. 13.8*). При этом можно добиться эффекта объемности.

Текст внутри тега `<text>` отображается в виде одной строки, перенос на новую строку данным тегом не поддерживается. Создавать большой текст с помощью нескольких элементов `<text>`, содержащих небольшие текстовые блоки, неудобно. Облегчить формирование большого текстового блока позволяет применение контейнерных тегов `<tspan>` внутри тега `<text>`.

Атрибуты тега `<tspan>`:

- `x`, `y` — горизонтальная и вертикальная координаты строки, отсчитываемые от верхнего левого угла документа; по умолчанию совпадают с одноименными атрибутами содержащего контейнера `<text>`;
- `dx`, `dy` — горизонтальный и вертикальный отступы текста относительно начальной позиции по умолчанию; возможны положительные и отрицательные значения;
- `rotate` — угол поворота (от 0 до 360°) символов текстового фрагмента относительно горизонтали; положительным значениям соответствует поворот по часовой стрелке, а отрицательным — против;
- `baseline-shift` — способ выравнивания относительно базовой горизонтальной линии; возможны значения: `baseline` (по умолчанию), `sup` (выше) и `sub` (ниже); применяется при создании верхних и нижних индексов.

Кроме перечисленных, в теге `<tspan>` можно указывать и другие атрибуты форматирования текста.

В листинге 13.17 приведен пример расположения двух текстовых блоков в четыре строки каждый (рис. 13.31).

**Листинг 13.17. Применение тега `<tspan>`**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="350">
<title>Текст</title>
<text  y="50" fill="red">
    <tspan x="15" >Один два три четыре пять шесть семь</tspan>
    <tspan x="15" dy="1.2em">Один два три четыре пять шесть семь</tspan>
    <tspan x="15" dy="1.2em">Один два три четыре пять шесть семь</tspan>
    <tspan x="15" dy="1.2em">Один два три четыре пять шесть семь</tspan>
</text>
<text  y="150" fill="blue">
    <tspan x="50" dx="30px">Один два три четыре пять шесть семь</tspan>
    <tspan x="50" dy="1.2em" >Один два три четыре пять шесть семь</tspan>
    <tspan x="50" dy="1.2em">Один два три четыре пять шесть семь</tspan>
    <tspan x="50" dy="1.2em">Один два три четыре пять шесть семь</tspan>
</text>
</svg>
```

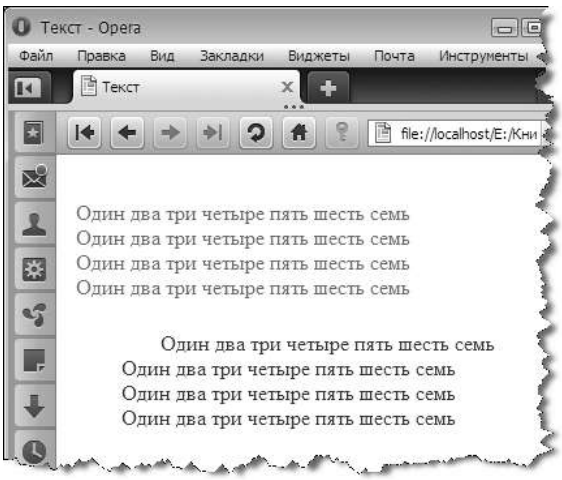



**Рис. 13.31.** Применение <tspan> для форматирования текстов в несколько строк (листинг 13.17)

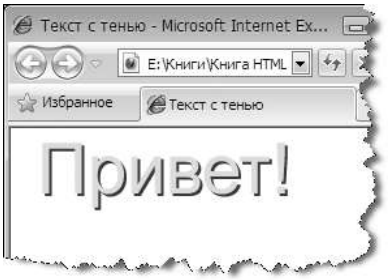



**Рис. 13.32.** Создание эффекта тени (листинг 13.18)

Текстовый блок можно сначала определить в контейнере <defs>, а затем отобразить в нужном месте и с требуемыми параметрами посредством тега <tref>, содержащего ссылку на него. Причем это можно делать неоднократно. В листинге 13.18 показано использование <tref> для создания эффекта тени. Очевидно, для данной цели требуются два экземпляра одного и того же текста, один из которых наложен на другой с небольшим сдвигом. Собственно текст определяется в текстовом блоке внутри контейнера <defs>, а отображаются два его экземпляра посредством другого текстового блока, содержащего теги <tref>. Вид документа в браузере показан на рис. 13.32.

**Листинг 13.18. Применение ссылки на текстовый блок для создания эффекта тени**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="350">
<title>Текст с тенью</title>
<defs>
  <text id="mytext">
    Привет!
```

```
  </text>
</defs>
<text font-size="54px">
  <tref xlink:href="#mytext"
      x="22" y="52"
      fill="black" fill-opacity="0.7"/>
  <tref xlink:href="#mytext"
      x="20" y="50"
      fill="cyan"/>
  </text>
</svg>
```

Однако Firefox 3.6 не поддерживает `<tref>` и для обеспечения межбраузерной совместимости подойдет код с применением `<tspan>` (листинг 13.19).

**Листинг 13.19. Применение <tspan> для создания эффекта тени**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="350">
<title>Текст с тенью</title>
<text x="22" y="52" font-size="54px">
  <tspan fill="black" fill-opacity="0.7">
      Привет!
  </tspan>
  <tspan x="20" dy="-2" fill="cyan">
      Привет!
  </tspan>
</text>
</svg>
```

Текст можно расположить вдоль некоторой траектории, заданной тегом `<path>`. Для этого в контейнер `<text>` вставляют контейнер `<textPath>`, содержащий текст. Тег `<textPath>` содержит еще и ссылку на элемент `<path>`, определяющий траекторию, к которой должен быть привязан текст. Если `<path>` заключен в контейнер `<defs>`, то сама траектория не будет видна. Начальную позицию текста на траектории задает атрибут `startOffset` тега `<textPath>`, принимающий значения от 0% (начало таектории) до 100% (конец траектории).

В листинге 13.20 приведен пример, в котором заданы две траектории и три фрагмента текста. Первые два фрагмента текста расположены на ломаной

прямой, а их пространственное разнесение обеспечено атрибутом `startOffset`. Вторая траектория имеет вид окружности. Обратите внимание на маленький зазор между начальной и конечной точками эллиптической дуги, образующей в данном случае окружность. Без зазора окружность не получилась бы. Это особенность эллиптических дуг, создаваемых командой `A` атрибута `d` тега `<path>` (см. *разд. 13.3*) Вид данного документа показан на рис. 13.33. Теги `<path>` не заключены в контейнер `<defs>` и поэтому траектории, вдоль которых расположены тексты, отображаются.

**Листинг 13.20. Расположение текста вдоль траектории**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="450">
<title>Текст вдоль траектории</title>
<desc>
   Чтобы линии были не видны, элементы path
   следует заключить в контейнер defs
</desc>

  <path id="mypath1" d="M25,75 l60,-40 200,40"
        stroke="grey" fill="none"/>
  <path id="mypath2" d="M50,150 A50,50 0 1,1 50,150.01"
        stroke="grey" fill="none"/>

<text font-size="24"  >
   <textPath xlink:href="#mypath1">
          Добро пожаловать!
   </textPath>
</text>
<text font-size="28" font-family="arial" fill="blue">
   <textPath xlink:href="#mypath1" startOffset="75%">
          Ура!
   </textPath>
</text>
<text font-size="20" font-family="arial" fill="red">
   <textPath xlink:href="#mypath2" startOffset="0%">
          Нехорошо подделывать печати
   </textPath>
```

```
</text>
</svg>
```

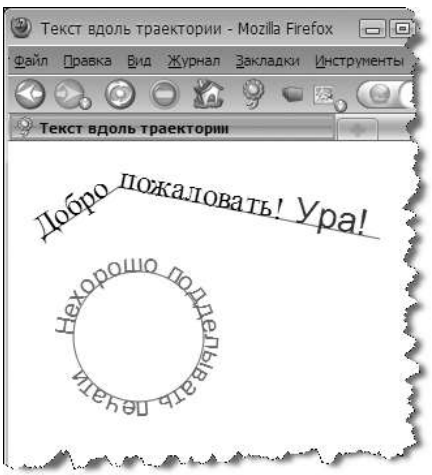



**Рис. 13.33.** Расположение текста вдоль траектории

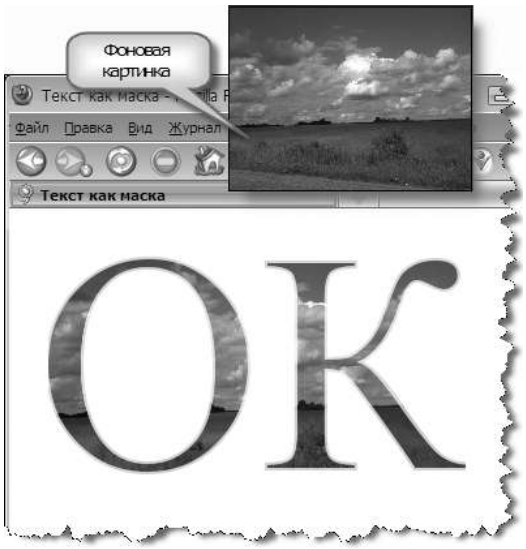



**Рис. 13.34.** Текст в качестве маски (листинг 13.21)

В *разд. 13.9* рассматривалось использование фигур в качестве масок. Текст также можно применить как маску, если его заполнение (`fill`) не очень темное, а размер шрифта достаточно большой, чтобы внутри контура символов удалось разглядеть маскируемый объект, например, растровое графическое изображение.

В листинге 13.21 и на рис. 13.34 приведен пример, в котором строка "OK" маскирует картинку из графического файла. Заливка символов установлена белой для обеспечения их максимальной прозрачности, а контур задан серым, чтобы он был заметен.

**Листинг 13.21. Текст в качестве маски**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
     width="520" height="350">
<title>Текст как маска</title>
<defs>
  <mask id="mymask">
    <text x="20" y="200" font-size="240px"
          fill="white" stroke="grey" stroke-width="2">
       OK
    </text>
  </mask>
</defs>
<image xlink:href="picture.jpg" mask="url(#mymask)"
       x="0" y="0" width="380px" height="256"/>
</svg>
```

В качестве примера применения масок к тексту рассмотрим создание эффекта лупы, при котором через круглую лупу виден увеличенный фрагмент текста (рис. 13.35). Здесь понадобятся две маски: первая — прозрачный прямоугольник с непрозрачным кругом внутри — накладывается на текст с обычным размером шрифта. Вторая маска — прозрачный круг — накладывается на тот же текст, но с бóльшим размером шрифта. Далее текст со второй маской накладывается на текст с первой маской. В завершение выводится контур круга. Соответствующий код приведен в листинге 13.22.

Разумеется, применение нескольких масок возможно не только к текстам.

**Листинг 13.22. Создание эффекта лупы**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="500" >
<title>Маска</title>
<defs>
  <text id="mytext" x="45" y="60"
        style="font-family:Arial">
    <tspan>
        Во глубине сибирских руд
    </tspan>
    <tspan x="45" dy="20">
        Храните гордое терпенье.
```

```
    </tspan>
    <tspan x="45" dy="20">
        Не пропадет ваш скорбный труд
    </tspan>
    <tspan x="45" dy="20">
        И дум высокое стремленье.
    </tspan>
    <tspan x="45" dx="100" dy="20">
        А.С. Пушкин
    </tspan>
  </text>
  <mask id="mymask1">
  <!-- Прозрачный прямоугольник с непрозрачным кругом для лупы внутри -->
    <rect x="0" y="0" width="300" height="300" fill="white"/>
    <circle cx="130" cy="100" r="50" fill="black"/>
  </mask>

  <mask id="mymask2">
    <!-- Прозрачный круг (лупа) -->
    <circle cx="130" cy="100" r="50" fill="white" />
  </mask>
</defs>
<!-- Выводим текст, удалив все, что в круге -->
<use xlink:href="#mytext" mask="url(#mymask1)" style="font-size:12px"/>
<!-- Накладываем текст большего размера, расположенный в круге -->
<use xlink:href="#mytext" mask="url(#mymask2)" style="font-size:18px"/>
<!-- Накладываем контур круга -->
<circle cx="130" cy="100" r="50" fill="none" stroke="red" stroke-
width="4"/>
</svg>
```

В рассмотренном примере к кругу, являющемуся фигурой второй маски, можно применить радиальный градиент, чтобы создать эффект выпуклой линзы. С этой целью в контейнер `<defs>` следует добавить определение градиента, например, такое:

```
<radialGradient id="mygradient" fx="30%" fy="30%" >
    <stop offset="20%" stop-color="white" stop-opacity="0.8"/>
    <stop offset="80%" stop-color="#0000ff" stop-opacity="0.4"/>
    <stop offset="90%" stop-color="#000080" stop-opacity="0.6"/>
</radialGradient>
```

А в теге, задающем контур лупы, вместо атрибута `fill="none"` нужно указать ссылку на градиент:

```
<circle cx="130" cy="100" r="50" fill="url(#mygradient)"
```

```
stroke="red" stroke-width="4"/>
```

Результат показан на рис. 13.36.

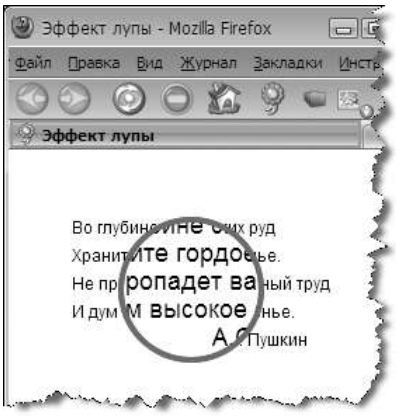



**Рис. 13.35.** Применение двух масок для создания эффекта лупы (листинг 13.22)

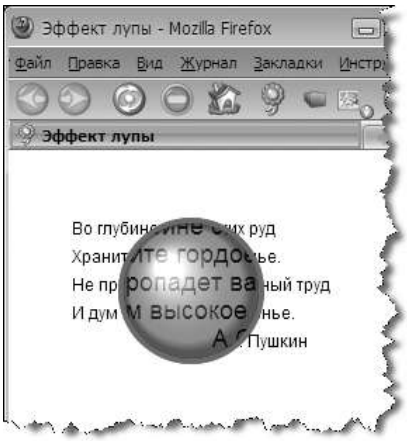



**Рис. 13.36.** Лупа с радиальным градиентом

Несмотря на большие возможности SVG по форматированию текстов, во многих случаях все же более удобно применять HTML. Вставку в SVG-документ фрагментов (X)HTML-кода (и не только) можно обеспечить посредством контейнерного тега `<foreignObject>` (см. *разд. 13.14*).

# 13.11. Трасформации

Трансформации — это геометрические преобразования объектов, такие как перемещение, поворот, масштабирование и др. Для их понимания полезно воспользоваться следующей моделью. Представим себе некоторый объект (например, прямоугольник или круг), нарисованный на прозрачной подложке (кальке, холсте, canvas), которая занимает все пространство SVG-документа. Операция трансформации, которую мы хотим применить к объекту, действует на всю подложку, а не только на данный объект. Так, при масштабировании растягивается или сжимается подложка, а вместе с ней и объект. При этом последний не только изменяется в размерах, но и смещается относительно осей координат, поскольку его исходные расстояния до осей координат также масштабируются. Операция перемещения в точку с координатами *x, y* перемещает в эту точку верхний левый угол подложки, а не верхний левый угол объекта на ней. В результате исходные координаты объекта получат приращения *x* и *y*, а не просто преобразуются в *x* и *y*. Другие операции трансформации действуют аналогично.

Трансформацию в SVG осуществляют посредством атрибута `transform`, в котором указывают вид и параметры операции. Данный атрибут указывают в теге того элемента, который требуется трансформировать. Не забывая, что трансформации подвергается вся прозрачная подложка со всем, что на ней расположено, для краткости мы будем говорить о трансформации объекта.

Вот список видов операций, которые можно задать в качестве значений атрибута `transform`:

- `translate(x,y)` — перенос в точку с горизонтальной координатой x и вертикальной координатой y;
- `rotate(`*угол*`,x,y)` — поворот на заданный угол в градусах; положительным значениям угла соответствует поворот по часовой стрелке, а отрицательным — против; необязательные параметры `x` и `y`, задают еще и перемещение (перенос);
- `skewX(`*угол*`), skewY(`*угол*`)` — наклон соответственно вдоль горизонтальной и вертикальной оси на заданный угол; знак параметра указывает на направление наклона;
- `scale(`$k_x$`,`$k_y$`)` — масштабирование; параметры $k_x$ и $k_y$ задают коэффициенты масштабирования по горизонтали и вертикали; значениям больше единицы соответствует увеличение, значениям меньше единицы — уменьшение, значению, равному единице, — сохранение исходных размеров; если коэффициенты отрицательные, то происходит отражение объекта относительно осей координат;
- `matrix(a,b,c,d,e,f)` — трансформация посредством матрицы; позволяет осуществить все перечисленные ранее преобразования, как по отдельности, так и в композиции.

Например, для переноса подложки с объектом в точку с координатами `x`, `y` в теге объекта следует указать атрибут `transform="translate(x, y)"`.

Перечисленные операции можно выполнять как отдельно, так и композиционно (последовательно). Последующая операция применяется к результату предыдущей, причем итог не зависит от порядка следования операций. Несколько операций указываются в атрибуте `transform` через пробел. Например, для применения операций масштабирования и переноса следует указать атрибут следующего вида:

`transform="scale(`$k_x$`,`$k_y$`) translate(x,y)"`

Операции трансформации применимы как к отдельным объектам (фигуры, растровые изображения, тексты), так и к их группам.

## 13.11.1. Перенос

Операция `translate(x,y)` осуществляет перенос подложки с объектом в указанную параметрами точку. В результате объект, заданный с помощью того или иного тега, перемещается относительно своего исходного положения (рис. 13.37). Например, для переноса четырехугольника можно применить такой код:

```
<rect id="myrect" x="40" y="50" width="50" height="25"
      fill="red" stroke="black"
      transform="translate(100, 150)"/>
```

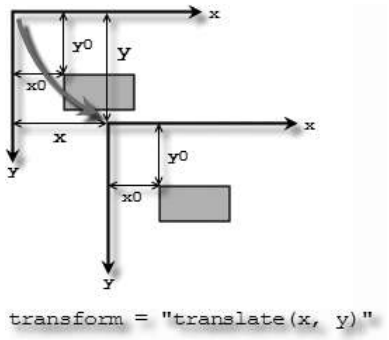



**Рис. 13.37.** Действие операции `translate(x, y)`

В листинге 13.23 приведен пример, в котором определен прямоугольник, а затем в окно браузера с помощью тегов `<use>` выводятся два его клона, ко второму из которых применена операция переноса. В результате получится то, что показано на рис. 13.37.

**Листинг 13.23. Пример применения операции `translate`**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="300" height="200">
<title>Ttanslate</title>
<defs>
  <rect id="myrect" x="40" y="50" width="50" height="25"
      fill="red" stroke="black"/>
</defs>
<!-- Исходный прямоугольник -->
<use xlink:href="#myrect"/>
```

```
<use xlink:href="#myrect" transform="translate(100,100)"/>
</svg>
```

## 13.11.2. Поворот

Операция `rotate(a)` осуществляет поворот подложки с объектом на заданный в градусах угол `a` (рис. 13.38). Например, для поворота графического растрового изображения на 30° можно применить такой код:

```
<image id="myimage" xlink:href="picture.jpg"
  x="100" y="20" width="100" height="140"
  transform="rotate(30)"/>
```

В листинге 13.24 приведен пример, в котором определено графическое растровое изображение, а затем в окно браузера с помощью тегов `<use>` выводятся два его клона, ко второму из которых применена операция поворота. Результат показан на рис. 13.39.

**Листинг 13.24. Пример применения операции rotate**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="500" height="300">
<title>rotate</title>
<defs>
  <image id="myimage" xlink:href="picture.jpg"
    x="100" y="20" width="100" height="142"/>
</defs>
<!-- Исходная картинка -->
<use xlink:href="#myimage"/>
<use xlink:href="#myimage" transform="rotate(30)"/>
</svg>
```


**Рис. 13.38.** Действие операции `rotate(a)`


**Рис. 13.39.** Пример применения операции `rotate` (листинг 13.24)

Операция `rotate` поворачивает подложку и, если объект имел исходные координаты, отличные от нуля, возникает эффект его смещения. Если требуется поворот объекта с сохранением исходного положения его верхнего левого угла, то кроме поворота необходимо еще и установить исходные координаты объекта с помощью второго и третьего параметров операции `rotate` (листинг 13.25, рис. 13.40).

**Листинг 13.25. Поворот с сохранением позиционирования**

```
. <?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="500" height="300">
<title>rotate</title>
<defs>
  <image id="myimage" xlink:href="math_book.jpg"
    x="100" y="20" width="100" height="142"/>
</defs>
<!-- Исходная картинка -->
<use xlink:href="#myimage"/>
```

```
<use xlink:href="#myimage" transform="rotate(30,100,20)"/>
</svg>
```


**Рис. 13.40.** Поворот с сохранением позиционирования (листинг 13.25)

## 13.11.3. Наклон

Операции `skewX(a)` осуществляет наклон (скос) подложки с объектом, как будто вертикальная ось координат наклонилась к горизонтальной на угол `a`. При этом объект деформируется вдоль горизонтали. Операция `skewY(a)` осуществляет наклон (скос) подложки с объектом, как будто горизонтальная ось наклонилась к вертикальной на угол `a`. При этом объект деформируется вдоль вертикали. Действие данных операций показано на рис. 13.41.

Операция `skewX(a)` сохраняет ординаты точек объекта, а абсциссы преобразует по формуле `y·tg(a)`. Операция `skewY(a)`, наоборот, сохраняет абсциссы, а ординаты преобразует по формуле `x·tg(a)`. Угол, задаваемый в градусах, может быть и отрицательным для изменения направления наклона на противоположное.

Например, для наклона прямоугольника вдоль горизонтальной оси на 60° можно использовать следующий код:

```
<rect x="30" y="30" width="100" height="50"
      fill="red" stroke="black"
      transform="skewX(60)" />
```

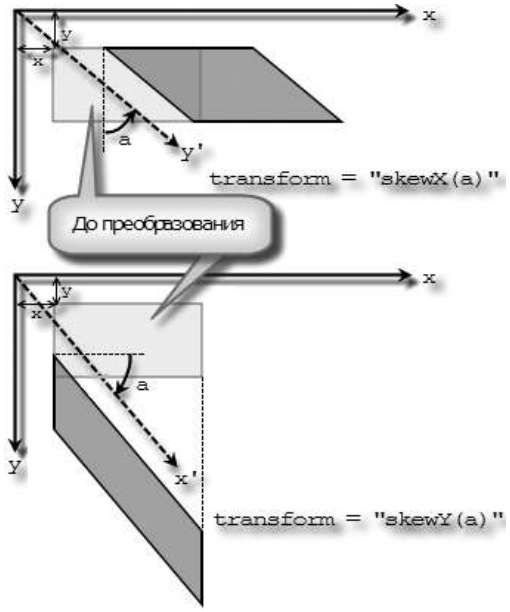



**Рис. 13.41.** Действие операций `skewX(a)` и `skewY(a)`

Если объект имеет исходные координаты `x` и `y`, отличные от нуля, то операции `skewX(a)` и `skewY(a)` будут кроме наклона еще и смещать объект на величины `y·tg(a)` и `x·tg(a)`. Это заметно на рис. 13.41. Для сохранения исходного позиционирования верхнего левого угла достаточно выполнить компенсацию смещения с помощью операции `translate(-y·tg(a), -x·tga)`.

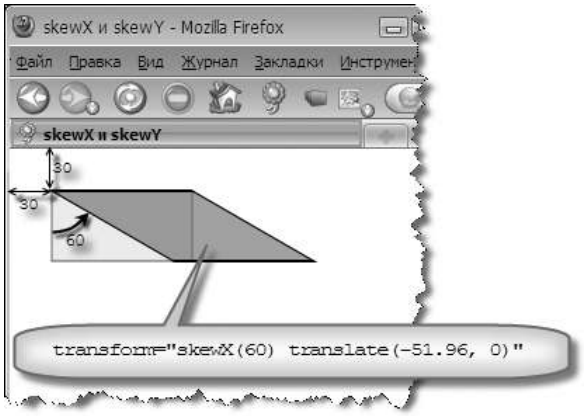



**Рис. 13.42.** Наклон с компенсацией смещения (листинг 13.26)

В листинге 13.26 приведен пример, в котором к прямоугольнику применяется операция `skewX(60)` и компенсируется горизонтальное смещение с помощью операции `translate(-51.96, 0)`, где 51,96 ≈ 30·tg(60°). Для наглядности созданы два клона прямоугольника, причем второй наклонен и сдвинут в исходную точку. Вид данного документа показан на рис. 13.42.

**Листинг 13.26. Пример наклона с компенсацией смещения**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="500" height="300">
<title>skewX и skewY</title>
<defs>
  <rect id="myrect"
    x="30" y="30" width="100" height="50"
    stroke="black" />
</defs>
<!-- Исходный прямоугольник -->
<use xlink:href="#myrect"
    fill="#f0f0f0" stroke-width="0.5"/>
<!-- Наклон и компенсация сдвига -->
<use xlink:href="#myrect"
    fill="#808080" fill-opacity="0.6" stroke-width="2"
    transform="skewX(60) translate(-51.96, 0)" />
</svg>
```

Операции наклона, как и другие трансформации, можно применять не только к отдельным объектам, но и к их группам. В листинге 13.27 приводится пример, в котором прямоугольник объединен в группу с текстом. С помощью `<use>` отображаются два клона данной группы и отдельно текст из нее. При этом результат трансформации похож на эффект перспективы (рис. 13.43).

***Примечание***

Точный эффект перспективы (схождение лучей) нельзя получить с помощью плоских преобразований, к которым относятся все рассматриваемые здесь операции.

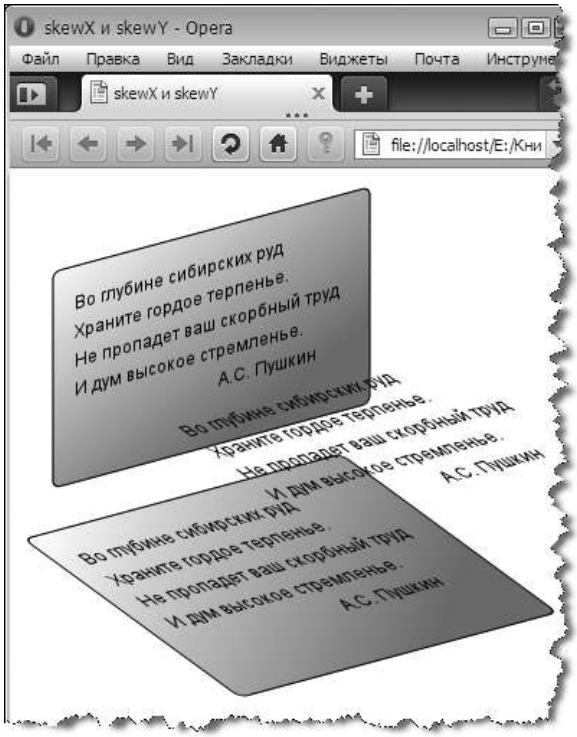



**Рис. 13.43.** Применение операций наклона и переноса к группе объектов (листинг 13.27)

**Листинг 13.27. Применение операций наклона и переноса к группе объектов**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="500" height="400">
<title>skewX и skewY</title>
<defs>
  <linearGradient id="mygradient"
    x1="0%" y1="0%" x2="100%" y2="100%">
    <stop offset="0%" stop-color="white"/>
    <stop offset="80%" stop-color="grey"/>
  </linearGradient>
<!-- Группа -->
```

```
  <g id="mygroup">
   <rect id="myrect" rx="5" ry="5"
    x="30" y="30" width="220" height="150"
    fill="url(#mygradient)"
    stroke="black"  />
    <text id="mytext" x="45" y="60" style="font:12px Arial" >
      <tspan>Во глубине сибирских руд</tspan>
      <tspan x="45" dy="20">Храните гордое терпенье.</tspan>
      <tspan x="45" dy="20">Не пропадет ваш скорбный труд</tspan>
      <tspan x="45" dy="20">И дум высокое стремленье.</tspan>
      <tspan x="45" dx="100" dy="20">А.С. Пушкин</tspan>
    </text>
  </g>
</defs>
<!-- Создание клонов -->
<use xlink:href="#mygroup"
     transform="skewY(-15) translate(0,50)"/>
<use xlink:href="#mygroup" y="180"
     transform="skewY(-15) skewX(45) translate(-280,50)"/>
<use xlink:href="#mytext" x="180"
     transform="skewY(-15) skewX(45) translate(-320,160)"/>
</svg>
```

## 13.11.4. Масштабирование

Операция `skale(`$k_x$`,`$k_y$`)` масштабирует объект — увеличивает или уменьшает по горизонтали и вертикали в соответствии с коэффициентами $k_x$, $k_y$. Значение коэффициента больше единицы определяет увеличение, а меньше единице — уменьшение; значение, равное единице, оставляет объект в данном направлении без изменений. Если указать только один коэффициент, то масштабирование по обоим направлениям будет одинаковым. Так, операции, `scale(2,2)` и `scale(2)` дают одинаковые результаты: увеличение в два раза по горизонтали и вертикали.

Например, для увеличения размеров (ширины и высоты) прямоугольника в два раза можно использовать следующий код:

```
<rect  x="30" y="30" width="50" height="30"
       fill="red"stroke="black"
       transform="scale(2)" />
```

При масштабировании происходит изменение не только размеров самого объекта, но и его координат (рис. 13.44).

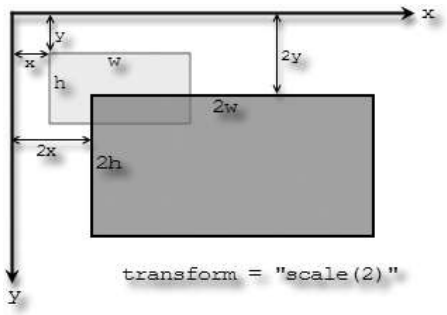



**Рис. 13.44.** Действие операции `scale(x)`

## 13.11.5. Отражение

Симметричное отражение объекта относительно вертикальной или горизонтальной осей системы координат выполняется с помощью двух операций — `scale()` и `translate()`. При этом в операции `scale()` значения параметров отрицательны. Например, `scale(-1,1)` отражает подложку с объектом относительно вертикали, а `scale(1,-1)` — относительно горизонтали. Если же коэффициенты по абсолютной величине отличны от единицы, то происходит еще и масштабирование объекта. Операция `translate()` дополнительно позволяет позиционировать объект в области видимости. На рис. 13.45 показано совместное действие операций `scale()` и `translate()` для создания отражений графического изображения.

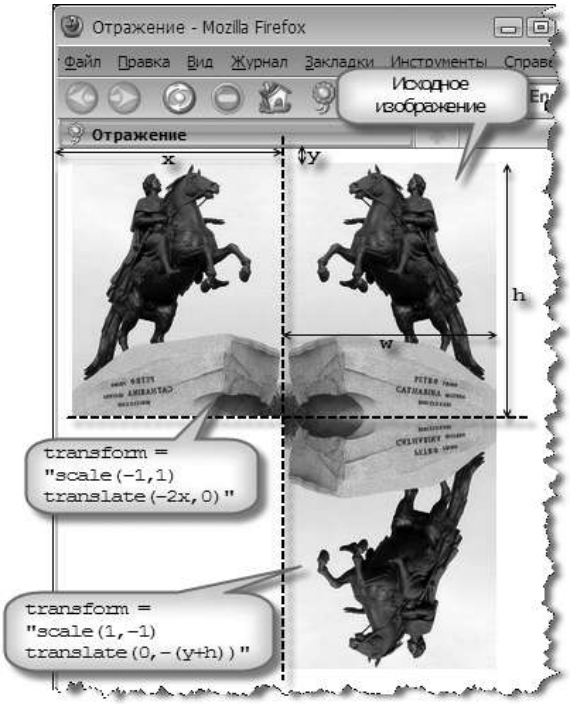



**Рис. 13.45.** Результат операции отражения

С практической точки зрения наиболее полезно, пожалуй, отражение с замещением исходного объекта, т. е. с сохранением исходных координат позиционирования. С этой целью при отражении относительно вертикальной оси (`scale(-1,1)`) следует применить операцию переноса `translate(-(2x+width))`, а при отражении относительно горизонтальной оси (`scale(1,-1)`) — `translate (-2y-height)`. Здесь `x`, `y` — исходные координаты, а `width`, `height` — ширина и высота объекта.

На рис. 13.46 показаны примеры вертикального и горизонтального отражений графического растрового изображения с сохранением исходных координат позиционирования. В листинге 13.28 приведен соответствующий код только для отражения относительно вертикальной оси.


**Рис. 13.46.** Отражение графического изображения с сохранением исходного позиционирования

**Листинг 13.28. Вертикальное отражение с сохранением исходного позиционирования**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="500" height="600">
<title>Отражение</title>
<defs>
  <image id="myimage" xlink:href="petr1.jpg"
   x="30" y="30" width="150px" height="180px"/>
</defs>
```

```
<use xlink:href="#myimage" transform="scale(-1,1) translate(-210,0)"/>
</svg>
```

## 13.11.6. Трансформация посредством матрицы

С математической точки зрения преобразование одной плоской фигуры в другую, расположенную в той же плоскости, можно выполнить с помощью некоторой трехмерной матрицы T размером 3×3, нижняя строка которой имеет вид (0 0 1). Точка с координатами *x*, *y* на плоскости представляется как матрица-столбец (*x y* 0) размером 1×3. Для преобразования точки в другую точку следует матрицу T умножить на матрицу-столбец. Умножение здесь является операцией, определенной в алгебре матриц. В результате получается матрица-столбец, содержащая новые координаты точки. При преобразовании фигуры матрица T применяется ко всем ее точкам.

Изложенное — математическая основа операции трансформации `matrix()`, поддерживаемой SVG. Чтобы применять эту операцию, весьма полезно, но отнюдь не необходимо понимать алгебру матриц. В SVG преобразование с помощью трехмерной матрицы производится посредством атрибута вида

```
transform="matrix(a, b, c, d, e, f)"
```

где `a`, `b`, …, ,`f` — элементы (коэффициенты) матрицы преобразования, которые можно указывать через запятую или пробел. Поскольку для плоских преобразований третьи строки матриц всегда одинаковы — (0 0 1), ее коэффициенты не указывают в качестве параметров `matrix()`. Таким образом, вместо девяти элементов трехмерной матрицы преобразования в `matrix()` достаточно указать только шесть. В будущем, не исключено, появится возможность задания трехмерных преобразований, и тогда будет задействовано большее количество элементов матрицы.

На рис. 13.47 показано соответствие между элементами трехмерной матрицы и параметрами `matrix()`, а также между рассмотренными ранее операциями (`translate`, `rotate` и др.) и их представлениями посредством `matrix`.

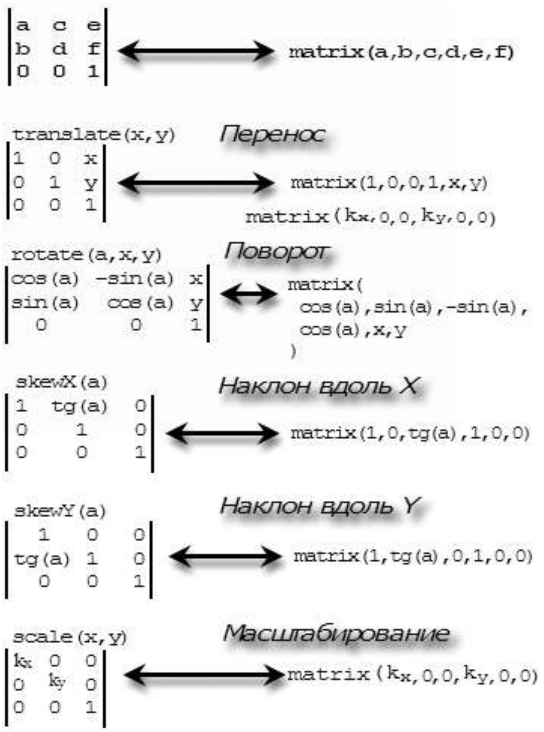



**Рис. 13.47.** Соответствие между различными представлениями трансформаций

Последовательности двух преобразований соответствует произведение двух матриц. Мы не будем рассматривать, как перемножаются две матрицы, а лишь укажем, как вычислять параметры результирующего выражения `matrix()`.

Последовательность двух преобразований задает атрибут

```
transform=
    "matrix(a1, b1, c1, d1, e1, f1)  matrix(a1, b1, c1, d1, e1, f1)"
```

Результат данных двух преобразований эквивалентен одному преобразованию, которое задает такой атрибут:

```
transform="matrix(
  a1a2+c1b2, b1a2+d1b2, a1c2+c1d2, b1c2+d1d2, a1e2+c1f2+e1, b1e2+d1f2+f1
)"
```

Очевидно, зная, как выполнить два преобразования подряд, можно сделать и большее их число.

Рассмотрим несколько типичных примеров представления последовательности из двух элементарных преобразований в виде одного результирующего преобразования:

- Масштабирование и перемещение:

  `transform="matrix(`$k_x$`,0,0,`$k_y$`,0,0) matrix(1,0,0,1,x,y)",`

  где $k_x$, $k_y$ — коэффициенты масштабирования по горизонтали и вертикали; `x`, `y` — координаты точки, в которую требуется переместить объект.

  Данную последовательность из двух операций можно заменить одной:

  `transform="matrix(`$k_x$`,0,0,`$k_y$`,x,y)"`

- Масштабирование и поворот:

  `transform="matrix(`$k_x$`cos(a),`$k_x$`sin(a),-`$k_y$`sin(a),`$k_y$`cos(a),0,0)"`

- Последовательность поворотов на углы `a` и `b`:

  `transform="matrix(cos(a+b),sin(a+b),-sin(a+b),cos(a+b),0,0)"`

  Данное выражение получается в результате применения тригонометрических формул.

Несколько примеров применения матричной формы задания трансформаций рассмотрены в *разд. 13.13.2*, а в листинге 13.58 приведен код функции, вычисляющей произведение двух матриц трансформаций и возвращающей строку, которую можно использовать в качестве значения атрибута `transform`.

# 13.12. Анимация

Анимация — это изменение свойств объекта, растянутое во времени так, чтобы мы могли наблюдать процесс происходящих трансформаций. Свойства объектов, как известно, задаются с помощью атрибутов и/или параметров CSS. Анимация объектов по существу есть изменение значений их атрибутов или стилевых параметров с заданными диапазоном и продолжительностью во времени.

В SVG существует несколько видов анимации, которые задают с помощью специальных тегов:

- `<animate>` — анимация отдельных свойств;
- `<animateTransform>` — анимация трансформации, т. е. изменения формы;
- `<animateMotion>` — анимация положения, т. е. движение;
- `animateColor>` — анимация цвета;
- `<set>` — установка значения свойства.

В каждом из перечисленных видов анимации указывают имя анимируемого свойства и его тип с помощью следующих атрибутов:

- `attributeName` — имя свойства; например, если требуется анимировать горизонтальную координату объекта, то следует использовать атрибут `attributName="x"`;

- `attributeType` — тип свойства; возможные значения:
  - `CSS` (если свойство соответствует спецификации CSS);
  - `XML` (если свойство определяется XML-атрибутом, например, операции трансформации);
  - `auto` (значение по умолчанию, при котором сначала проверяется наличие определения свойства CSS и, если такого нет, то ищется его определение как XML-атрибута).

Описание собственно анимации обеспечивают следующие общие атрибуты:

- `from` — начальное значение свойства;
- `to` — конечное значение свойства;
- `dur` — продолжительность анимации во времени (сокращение от англ. duration); в качестве единиц измерения можно указывать `s` (секунды, по умолчанию), `m` (минуты), `h` (часы); допустимо также задавать продолжительность в формате времени `hh:mm:ss`, например, `02:30:05.3` (2 часа 30 минут 5,3 секунд); если данный атрибут не указывать, то анимация будет воспроизводиться бесконечно, как если указать в качестве значения `infinite`;
- `begin` — время начала воспроизведения анимации от момента окончания загрузки документа; единицы измерения и формат такие же, что и для атрибута `dur`; возможно также указание события, при котором следует начать анимацию;
- `end` — время окончания воспроизведения анимации; единицы измерения и формат такие же, что и для атрибута `dur`; возможно также указание события, при котором следует закончить анимацию;
- `repeatCount` — количество повторных воспроизведений анимации; значение `indefinite` соответствует бесконечному (циклическому) воспроизведению; по умолчанию анимация воспроизводится один раз;
- `repeatDur` — интервал времени, в течение которого анимация может быть воспроизведена повторно; единицы измерения и формат такие же, что и для атрибута `dur`;
- `fill` — определяет состояние по окончании воспроизведения анимации; возможны значения:
  - `remove` (элемент возвращается в исходное состояние и продолжение анимации возможно только с помощью перезапуска, это значение по умолчанию);
  - `freeze` (элемент виден в том состоянии, в котором его застало окончание воспроизведения, при этом возможно продолжение воспроизведения с данного места);

Если атрибут `fill` не указан, то по окончании воспроизведения на экране остается исходное состояние.

- `restart` — определяет возможность повторного воспроизведения; возможны значения:
  - `always` (повторить воспроизведение можно всегда);
  - `whenNotActive` (только после завершения внутреннего цикла воспроизведения);
  - `never` (никогда);
- `min`, `max` — минимальный и максимальный интервалы времени воспроизведения анимации;
- `values` — список из одного или нескольких промежуточных значений анимируемого свойства, указываемых через точку с запятой.

> ***Примечание***
>
> Теги различных видов анимации могут иметь дополнительные специфические для них атрибуты.

Тег, задающий анимацию, вставляется в контейнерный тег анимируемого объекта. Напомню, что теги в неконтейнерной нотации всегда могут быть записаны как контейнерные. Если требуется создать несколько анимационных эффектов для одного и того же объекта, то в тег последнего следует вставить несколько тегов, определяющих соответствующие эффекты. Вместе с тем, можно просто из тега, описывающего анимацию, сослаться на анимируемый элемент по значению его атрибута `id`.

## 13.12.1. Тег *<animate>*

Тег `<animate>` предназначен для определения анимации отдельных свойств объектов, заданных посредством атрибутов и/или параметров CSS.

Например, для перемещения центра круга по горизонтали, достаточно анимировать атрибут `cx` элемента `<circle>`. Код листинга 13.29 изменяет значение `cx` от 100 до 300 в течение 15 с.

**Листинг 13.29. Пример анимации центра круга**

```
<circle cy="70" r="50" fill="red">
  <animate attributeName="cx"
           from="100" to="300"
           dur="15s"
  />
</circle>
```

В данном примере круг, достигнув целевой точки, мгновенно вернется в исходное состояние, т. е. произойдет перемотка ролика к первому кадру. Если требуется, чтобы круг оставался в месте, которого он достиг в конце воспроизведения, то в тег `<animate>` следует добавить атрибут `fill="freeze"` (заморозить).

Код листинга 13.30 подходит для изменения цвета круга с красного на синий.

**Листинг 13.30. Пример анимации цвета**

```
<circle cx="100" cy="70" r="50" fill="red">
  <animate attributeName="fill"
           from="red" to="blue"
           dur="15s"
  />
</circle>
```

Чтобы воспроизведение данной анимации повторялось бесконечно, в тег `<animate>` следует добавить атрибут `repeatCount="indefinite"`. Однако при этом переход от конца к началу анимации (перемотка ролика) происходит мгновенно.

Код, приведенный в листинге 13.31, постепенно изменяет красный цвет на синий, а затем так же постепенно возвращает его к исходному красному, на чем воспроизведение анимации цвета завершается. Здесь применяются две анимации цвета, разнесенные во времени: вторая начинается через одну секунду после окончания первой.

**Листинг 13.31. Пример двойной анимации цвета**

```
<circle  r="50" cx="100" cy="70">
  <animate attributeName="fill"
           from="red" to="blue"
           dur="5s"
           begin="0s"
           fill="freeze"
  />
  <animate attributeName="fill"
           from="blue" to="red"
           dur="5s"
           begin="6s"
           fill="freeze"
  />
</circle>
```

В рассмотренном примере анимации цвета промежуточные цвета нами не контролировались. Однако посредством атрибута `values` их можно задать.

При этом атрибуты `for` и `to` уже не понадобятся. Код листинга 13.32 изменяет цвет круга от красного к синему через желтый и зеленый.

**Листинг 13.32. Анимация с использованием промежуточных цветов**

```
<circle cx="100" cy="70" r="50" fill="red">
  <animate attributeName="fill"
           values="red; yellow; green; #0000ff"
           dur="15s"
           fill="freeze"
  />
</circle>
```

Как уже отмечалось, тег с описанием анимации не обязательно вставлять в элемент, свойство которого требуется анимировать. Можно еще просто сослаться на этот элемент по значению его атрибута `id`, как это делается в листинге 13.33.

**Листинг 13.33. Использование атрибута `id` при анимации**

```
<circle id="myelement" r="50"  cx="100" cy="70" fill="red"/>
<animate xlink:href="#myelement" attributeName="fill"
         from="red" to="blue"
         dur="5s"
/>
```

Элементы и описания их анимаций можно сгруппировать и поместить в контейнер `<defs>` определений, а затем клонировать группу для отображения, как в листинге 13.34.

**Листинг 13.34. Группировка элементов при анимации**

```
<defs>
  <g id="mygroup">
  <!-- Группа, содержащая круг и описание анимации его цвета -->
    <circle id="myelement" r="50"  cx="100" cy="70" fill="red"/>
    <animate xlink:href="#myelement" attributeName="fill"
         from="red" to="blue"
         dur="5s"
    />
  </g>
</defs>
<!-- Отображение анимированного круга -->
<use xlink:href="#mygroup" x="10" y="20"/>
```

Рассмотрим в качестве примера создание эффекта постепенного появления графического растрового изображения. Для этого достаточно анимировать CSS-свойство `opacity` (уровень непрозрачности) от 0 до 1 (листинг 13.35).

**Листинг 13.35. Пример эффекта постепенного появления изображения**

```
<defs>
  <g id="myimg">
    <image  id="img" xlink:href="picture.jpg" width="300" height="200" />
    <animate  xlink:href="#img"
              attributeName="opacity" attributeType="CSS"
              from="0" to="1"
              dur="5s"
    />
  </g>
</defs>
<use xlink:href="#myimg" x="10" y="20"/>
```

Данная конструкция хорошо воспринимается Intenet Explorer с плагином Adobe SVGViewer, Opera и Safari, но оказалась проблематичной для браузера Chrome 4.1. Листинг 13.36 иллюстрирует еще два варианта решения этой же задачи, выполняемые всеми указанными браузерами.

**Листинг 13.36. Варианты создания эффекта постепенного оявления изображения**

```
<!-- первый вариант -->
<image  id="myimg" xlink:href="picture.jpg" width="300" height="200" />
<animate xlink:href="#myimg"  attributeName="opacity" atributeType="CSS"
         from="0" to="1"
         dur="5s"
/>
<!-- второй вариант -->
<image id="img2" xlink:href="picture1.jpg" width="300" height="200" />
<rect id="myrect" width="300" height="200"
   fill="white" fill-opacity="0" />
<animate xlink:href="#myrect" attributeName="fill-opacity"
     from="1" to="0"
     dur="5s"
/>
```

Во втором варианте графическое изображение накрыто прямоугольником и его атрибут `fill-opacity` изменяется от 1 до 0.

**Рис. 13.48.** Превращение графического изображения путем анимации параметра `opacity` (листинг 13.37)

Рассмотрим пример создания плавного превращения одного графического изображения в другое (рис. 13.48). Для этого достаточно наложить одно изображение на другое и анимировать стилевой параметр `opacity` (уровень непрозрачности) верхнего изображения (листинг 13.37).

**Листинг 13.37. Превращение одного графического изображения в другое**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="500" height="300">
<title>Превращение изображения</title>
<image  id="img1" xlink:href="picture1.jpg" width="300" height="200" />
<image  id="img2" xlink:href="picture2.jpg" width="300" height="200" />
<animate  xlink:href="#img2" attributeName="opacity" attributeType="CSS"
         from="0" to="1"
```

```
        dur="5s"
/>
</svg>
```

Интересные эффекты получаются при анимации градиентов. В листинге 13.38 приведен пример, в котором в двух радиальных и одном линейном градиентах анимируется цветовой переход. Для этого следует анимировать атрибут `offset` тега `<stop>`. Чтобы наблюдать эффект анимации градиента, последний необходимо применить к объекту в качестве заливки. В рассматриваемом примере градиенты применяются к кругам и прямоугольнику. На рис. 13.49 показаны два кадра данной анимации.

**Листинг 13.38. Пример анимации градиентов**

```
 <?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="500" height="300">
<title>Анимация градиентов</title>
<defs>
<!-- Градиенты с анимацией -->
  <radialGradient id="MyGradient1"
       fx="20%" fy="25%" >
    <stop offset="0" stop-color="#f0ffff"/>
    <stop offset="100%" stop-color="#00a0a0">
      <animate attributeName="offset"
         from="0%" to="100%" dur="10s"
         repeatCount="indefinite"/>
    </stop>
  </radialGradient>

  <radialGradient id="MyGradient2"
       fx="25%" fy="50%" >
    <stop offset="0" stop-color="#ffffff"/>
    <stop offset="100%" stop-color="#ff00ff">
      <animate attributeName="offset"
         from="0%" to="100%" dur="5s"
         repeatCount="indefinite"/>
    </stop>
  </radialGradient>

  <linearGradient id="MyGradient3"
```

```
       x1="0%" y1="0%" x2="100%" y2="0%">
    <stop offset="0" style="stop-color:#ffffff"/>
    <stop offset="100%" style="stop-color:#008000">
      <animate attributeName="offset"
         from="0%" to="100%" dur="3s"
         repeatCount="indefinite"/>
    </stop>
  </linearGradient>
</defs>
<!-- Объекты с градиентной заливкой -->
<circle cx="60" cy="60" r="50"
    fill="url(#MyGradient1)" fill-opacity="0.5"/>
<circle cx="100" cy="100" r="35"
    fill="url(#MyGradient2)" fill-opacity="0.5"/>
<rect x="10" y="150" width="200" height="60"
    fill="url(#MyGradient3)" />
</svg>
```

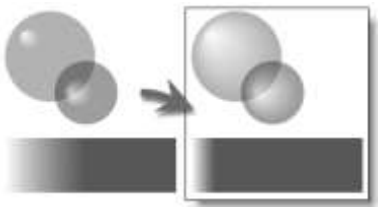


**Рис. 13.49.** Пример анимации градиентов (листинг 13.38)

Для анимации трансформаций объекта необходимо изменять значения атрибута `transform`. Однако для этой цели применяется не `<animate>`, а специальный тег `<animateTransform>`, который будет рассмотрен в следующем подразделе.

Анимацию изменения положения можно создать посредством тега

`<animate transform="translate(x, y")…/>`.

Вместе с тем, при более или менее сложной траектории движения проще и удобнее использовать тег `<animateMotion>` (см. *разд. 13.12.3*).

## 13.12.2. Тег *<animateTransform>*

Тег `<animateTransform>` предназначен для создания анимации трансформаций следующих типов: `translate` (перенос), `rotate` (поворот), `skewX` и `skewY` (наклон), `scale` (масштабирование) (см. *разд. 13.11*).

В теге <animateTransform> указывают атрибуты attributName="transform", type="*операция*" и другие, описывающие детали трансформации. Чтобы явно указать тип анимируемого свойства transform, следует записать attributType="XML". В атрибутах from и to задают начальные и конечные значения параметров операций трансформации соответственно. Если параметров более одного, то они указываются через запятую.

Пример анимации масштабирования прямоугольника иллюстрирует листинг 13.39.

**Листинг 13.39. Пример анимации прямоугольника**

```
<rect x="5" y="10" width="50" height="25" fill="grey" >
<animateTransform attributeName="transform" attributeType="XML"
                  type="scale" from="1" to="2" dur="5s"
                  fill="freeze"
/>
</rect>
```

Здесь прямоугольник увеличивается по ширине и высоте в два раза в течение 5 с. Операция scale имеет, как известно, два параметра и при их использовании можно писать, например, так:

```
from="1, 2" to="3, 0.5"
```

К одному и тому же объекту можно применить несколько операций трансформации. Аналогично, можно применить и несколько анимаций таких операций. Однако, чтобы возникло сложное движение, необходимо в каждом теге <animateTransform> задать атрибут additive="sum".

В листинге 13.40 определена группа, содержащая прямоугольник и текст (слово "Привет!"). К данной группе применяются анимации двух операций — поворота (rotate) и масштабирования (scale). Для создания эффекта суммирования двух анимаций в каждой из них указан атрибут additive="sum". На рис. 13.50 показаны начальное и конечное состояния изображения элементов группы.

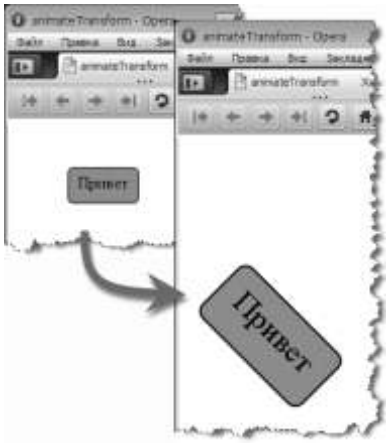



**Рис. 13.50.** Начальное и конечное состояния анимации поворота и масштабирования (листинг 13.40)

**Листинг 13.40. Пример двух анимаций**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="500" height="300">
<title>animateTransform</title>
<g id="mygroup">
  <rect  x="60" y="50" width="70" height="35"
         rx="5" fill="#a0a0a0" stroke="black"/>
  <text  x="70" y="70" >Привет</text>
</g>
<animateTransform xlink:href="#mygroup"
    attributeName="transform" attributeType="XML"
    type="rotate" from="0, 60 50" to="45,60,50" dur="5s"
    additive="sum"
    fill="freeze"
/>
<animateTransform xlink:href="#mygroup"
    attributeName="transform" attributeType="XML"
    type="scale" from="1" to="2" dur="5s"
    additive="sum"
    fill="freeze"
/>
```

```
</svg>
```

Конечное состояние можно было бы получить с помощью следующих статических операций трансформации:

```
<g transform="rotate(45,60,50) scale(2)">…</g>
```

Если опустить атрибут `additive="sum"` или указать `additive="replace"`, то возникнет эффект замещения предыдущей анимации последующей. В рассматриваемом примере можно будет наблюдать только анимацию масштабирования.

***Примечание***

Интересные визуальные эффекты можно создать путем анимации форм масок.

## 13.12.3. Тег *<animateMotion>*

Тег `<animateMotion>` предназначен для анимации изменения положения вдоль заданной траектории.

Сначала рассмотрим самый простой вариант движения вдоль ломаной прямой, заданной с помощью атрибута `values`. В теге `<animateMotion>` данный атрибут принимает в качестве значения список пар координат опорных точек (возможно указание размерности). Пары координат разделяются точкой с запятой, а сами координаты — запятой.

Фрагмент кода листинга 13.41 перемещает прямоугольник вдоль некоторой ломаной прямой в течение 5 с.

**Листинг 13.41. Перемещение вдоль траектории (вариант 1)**

```
<rect id="myrect" width="50" height="25"
      fill="#a0a0a0" stroke="black"/>
<animateMotion xlink:href="#myrect"  attributeType="XML"
               values="60,60;80,90;150,20;300,200"
               dur="5s"
 />
```

На рис. 13.51 показан один кадр движения прямоугольника. Для наглядности траектория сделана видимой.

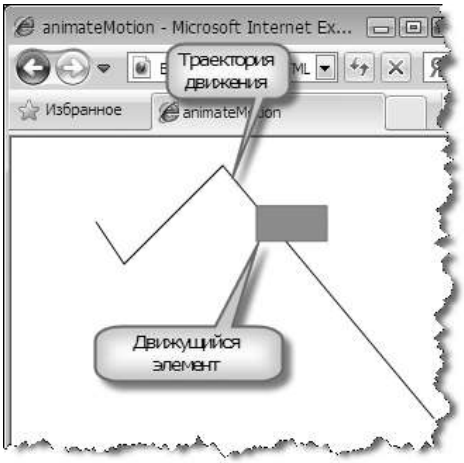



**Рис. 13.51.** Перемещение прямоугольника по траектории в виде ломаной прямой

Траекторию можно задать с помощью атрибута `path`, значения которого такие же, что и для атрибута `d` тега `<path>` (см. *разд. 13.3*). Посредством данного атрибута можно определить довольно сложные и гладкие кривые (листинг 13.42).

**Листинг 13.42. Перемещение вдоль траектории (вариант 2)**

```
<rect id="myrect" width="50" height="25"
      fill="#a0a0a0" stroke="black"/>
<animateMotion xlink:href="#myrect"  attributeType="XML"
               path="M60,60 L80,90 150,20 300,200"
               dur="5s"
 />
```

Траекторию можно определить и посредством тега `<path>`, а затем использовать при анимации. Чтобы она не была видна, элемент `<path>` следует заключить в контейнер `<defs>`. Ссылка на этот `<path>` должна содержаться в теге `<mpath>` в контейнере `<animateMotion>` (листинг 13.43).

**Листинг 13.43. Перемещение вдоль траектории (вариант 3)**

```
<defs>
<!-- Определение траектории движения -->
    <path id="path1" d="M60,60 L80,90 150,20 300,200"/>
```

```
</defs>
<rect id="myrect" width="50" height="25"
      fill="#a0a0a0" stroke="black"/>
<animateMotion xlink:href="#myrect"  attributeType="XML" dur="5s">
    <mpath xlink:href="#path1"/>
</animateMotion>
```

К траектории привязывается верхний левый угол подложки вместе с объектом. Если исходные координаты объекта равны нулю, то тогда к траектории привязывается верхний левый угол объекта, как это было в листингах 13.40—13.43. Если, например, требуется привязать к траектории центр прямоугольника, то достаточно установить координаты, равные половинам его ширины и высоты с отрицательным знаком.

Иногда требуется, чтобы движущийся объект поворачивался сообразно с направлением текущего участка траектории, т. е. ориентировался вдоль траектории (рис. 13.52). Для этой цели в теге `<animateMotion>` используется атрибут `rotate="auto"`. Значение `auto-reverse` этого атрибута делает то же самое, но с добавлением к вычисляемому углу 180°, а число задает угол поворота объекта, отсчитываемый от горизонтали. Однако последний вариант работает пока только в Internet Explorer и Opera.

Код листинга 13.44 аналогичен предыдущему, но он ориентирует прямоугольник вдоль траектории и привязывает к ней его центр (рис. 13.52).

**Листинг 13.44. Перемещение вдоль траектории (вариант 4)**

```
<defs>
<!-- Определение траектории движения -->
    <path id="path1" d="M60,60 L80,90 150,20 300,200"/>
</defs>
<rect id="myrect" width="50" height="25" x="-25" y="-12.5"
      fill="#a0a0a0" stroke="black"/>
<animateMotion xlink:href="#myrect"  attributeType="XML" dur="5s"
               rotate="auto" >
    <mpath xlink:href="#path1"/>
</animateMotion>
```

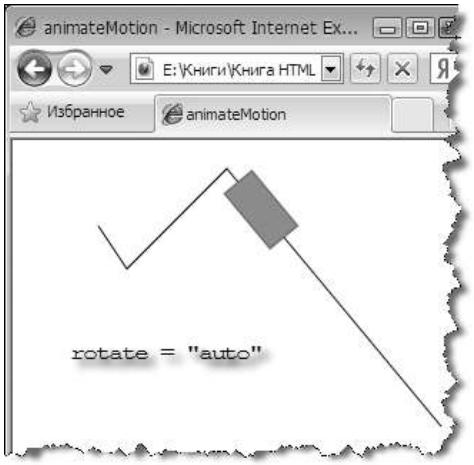



**Рис. 13.52.** Ориентация объекта вдоль траектории движения

Создание сложной анимации рассмотрим на примере построения модели "солнечной системы". Пусть вокруг солнца по эллипсу движется планета, а вокруг последней вращается ее спутник (планета поменьше). Солнце и планеты представим в виде кругов, причем солнце зальем анимированным радиальным градиентом, чтобы оно пульсировало. Схема данной солнечной системы показана на рис. 13.53. Таким образом, спутник движется вокруг планеты, а планета движется вокруг солнца вместе с вращающимся спутником. Реализация решения данной задачи представлена в листинге 13.45.

Чтобы орбиты не отображались, теги `<path>` вставлены в контейнер `<defs>`. Малая планета вместе с движением по малой орбите объединены в группу. Малая орбита позиционирована так, чтобы большая планета оказалась приблизительно в ее центре. Затем отдельно задаются движения группы и большой планеты по большой орбите.

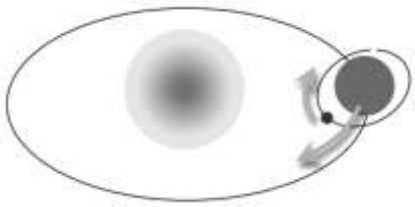


**Рис. 13.53.** Схема сложной анимации

**Листинг 13.45. Сложное движение**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="800" height="400">
<title>Сложное движение</title>
<defs>
  <!-- Заливка солнца анимированным градиентом -->
  <radialGradient id="MyGradient"
       cx="50%" cy="50%" >
    <stop offset="10%" stop-color="#ff0000" stop-opacity="0.8"/>
    <stop offset="100%" stop-color="#ff8080" stop-opacity="0.2">
      <animate attributeName="offset"
         from="0%" to="100%" dur="10s"
         repeatCount="indefinite"/>
    </stop>
  </radialGradient>
  <!-- Большая орбита -->
  <path id="path1" d="M50,180 A150,80 0 1,1 50.01,180.01" />
  <!-- Малая орбита -->
  <path id="path2" d="M10,-30 A40,30 -25 1,1 5,-30" />
</defs>
<!-- Большая планета -->
<circle id="circ1" cx="0" cy="0" r="25"  fill="grey"/>
<g id="subsystem">
<!-- Малая планета с движением по малой орбите -->
  <circle id="circ2" cx="0" cy="0" r="5"  fill="blue"/>
  <animateMotion xlink:href="#circ2"  attributeType="XML"
                 dur="5s" repeatCount="indefinite"  >
      <mpath xlink:href="#path2" />
  </animateMotion>
</g>
<!-- Солнце -->
<circle cx="180" cy="130" r="50"  fill="url(#MyGradient)" />
<!-- Движение большой планеты по большой орбите -->
<animateMotion xlink:href="#circ1"  attributeType="XML"
               dur="15s" repeatCount="indefinite" >
  <mpath xlink:href="#path1" />
</animateMotion>
```

```
<!-- Движение вращающейся малой планеты по большой орбите -->
<animateMotion xlink:href="#subsystem"  attributeType="XML"
               dur="15s" repeatCount="indefinite"  >
  <mpath xlink:href="#path1" />
</animateMotion>
</svg>
```

## 13.12.4. Тег *<animateColor>*

Тег `<animateColor>` предназначен для изменения во времени цвета. Анимируемыми свойствами объекта могут быть `fill` (заливка) и `stroke` (обводка). Рассмотренный в *разд. 13.12.1* тег `<animate>` имеет такие же возможности по изменению цвета.

Листинг 13.46 иллюстрирует пример использования для анимации цвета атрибута `values` тега `<animateColor>`.

**Листинг 13.46. Применение тега <animateColor>**

```
<rect id="myrect" width="50" height="25"
      fill="#a0a0a0" stroke="black"/>
<animateColor xlink:href="#myrect"  attributeName="fill"
              values="yellow; green; #00ffff; blue"

               dur="5s" fill="freeze"
 />
```

Возможно также применение атрибутов `from` и `to` со значениями начального и конечного цветов. В листинге 13.47 применяются несколько анимаций цвета с заданием времени начала последующей анимации так, чтобы оно совпадало с моментом окончания предыдущей.

**Листинг 13.47. Последовательность анимаций**

```
<rect width="50" height="25"
      fill="#a0a0a0" >
   <animateColor  attributeName="fill"
         from="red" to="yellow" begin="2s" dur="3s"/>
   <animateColor  attributeName="fill"
         from="yellow" to="green" begin="5s" dur="3s"/>
   <animateColor  attributeName="fill"
         from="green" to="blue" begin="8s" dur="3s"
         fill="freeze"/>
</rect>
```

### 13.12.5. Тег <*set*>

Тег <set> предназначен для установки значения атрибута. Изменение значения атрибута происходит не плавно, а мгновенно. Имя изменяемого свойства указывается как значение атрибута attributeName, а устанавливаемое значение — как значение атрибута to:

```
<set attributeName="имя_свойства" to="значение"/>
```

С помощью атрибута dur устанавливается продолжительность действия нового значения свойства, а не длительность перехода к нему. По истечении заданного интервала времени свойство приобретает свое исходное значение. Если атрибут dur не указывать, то установленное значение остается постоянным пока не будет изменено принудительно.

В листинге 13.48 серый прямоугольник сразу же после загрузки документа становится желтым и остается таковым в течение 10 с. Кроме того, через 5 с после загрузки у него увеличивается до 200 px ширина, но спустя 5 с она принимает исходное значение.

**Листинг 13.48. Применение тега <set> при анимации**

```
<rect id="myrect" width="50" height="25"
      fill="grey" />
<set  xlink:href="#myrect" attributeName="width" to ="200"
      begin="5s" dur="5s"/>
<set  xlink:href="#myrect" attributeName="fill" to ="yellow"
      dur="10s"/>
```

## 13.13. Интерактивность

Интерактивность в SVG-документах поддерживается в основном применением гиперссылок и обработкой событий, сгенерированных в результате манипуляций мышью, клавиатурой и др. Обработка событий возможна как с применением JavaScript, так и без него.

### 13.13.1. Гиперссылки

Гиперссылка (далее — ссылка) в SVG, как и в (X)HTML, задается контейнерным тегом <a>, но с обязательным атрибутом xlink:href= "*URL-адрес*", а не href. Все, что находится в контейнере <a> (указатель ссылки), чувствительно к щелчку и приводит к загрузке в браузер указанного в xlink:href документа. В качестве указателя ссылки могут быть тексты или графические элементы, в том числе и анимированные. К содержимому элемента <a>,

а также к самому контейнеру <a> можно применять операции преобразования, такие как трансформации и анимации. Дополнительный атрибут target позволяет указать, в какое окно браузера или фрейм следует загрузить запрашиваемый документ (см. *главу 6*).

В листинге 13.49 в качестве примера заданы три ссылки. Первая ссылка в качестве указателя содержит просто текст, вторая — прямоугольник и текст, третья — эллипс и текст, причем к самому элементу <a> применена операция поворота на 20° против часовой стрелки. Внешний вид данного документа в браузере показан на рис. 13.54. Обратите внимание на указание в теге <svg> пространства имен

```
xmlns:xlink="http://www.w3.org/1999/xlink"
```

необходимое при использовании в документе ссылок, причем не только на внешние документы, но и на внутренние элементы.


**Рис. 13.54.** Примеры ссылок (листинг 13.49)

**Листинг 13.49. Примеры ссылок**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="800" height="400">
<title>Гиперссылки</title>
<!-- Первая ссылка -->
<a xlink:href="http://www.yandex.ru">
```

```
  <text x="10" y="20" style="font:bold 14pt Arial; fill:red">
   Яндекс
  </text>
</a>
<!-- Вторая ссылка -->
<a xlink:href="http://www.rambler.ru" target="_blank">
  <rect x="10" y="50" width="100" height="30" rx="3"
        fill="cyan" stroke="blue"/>
  <text x="35" y="70" fill="red">
   Rambler
  </text>
</a>
<!-- Третья ссылка -->
<a xlink:href="http://www.google.ru" transform="rotate(-20)">
  <ellipse cx="70" cy="150" rx="60" ry="25"
           fill="yellow" stroke="black"/>
  <text x="35" y="155" style="font:bold 16pt Arial">
   Google
  </text>
</a>
</svg>
```

Ссылки, как и другие элементы, можно анимировать, например, выделять цветом, изменять шрифт и другие параметры при наведении и уходе указателя мыши. Однако это уже по существу обработка событий, которую мы рассмотрим в следующем разделе.

## 13.13.2. Обработка событий

В данном разделе мы рассмотрим на примерах некоторые возможности обработки событий, вызванных манипуляциями мышью, оставляя без внимания события, обусловленные нажатиями на кнопки клавиатуры, и некоторые другие. События можно обрабатывать двумя способами:

- посредством тегов и атрибутов SVG;
- с помощью скриптов на языке JavaScript.

### Применение тегов и атрибутов SVG

Манипуляции мышью генерируют следующие события:

- `mouseover` — при наведении указателя мыши на элемент;
- `mouseout` — при уходе указателя мыши с элемента;
- `mousemove` — при перемещении указателя мыши над элементом;

- `mousedown` — при нажатии на левую кнопку мыши, когда ее указатель находится над элементом;
- `mouseup` — при отпускании ранее нажатой левой кнопки мыши, когда ее указатель находится над элементом;
- `click` — при щелчке левой кнопкой мыши, когда ее указатель находится над элементом.

Возникшее событие можно как-то обработать, т. е. выполнить некоторые операции над объектом, с которым данное событие связано. Для обработки событий мыши можно воспользоваться тегами анимации с атрибутами `begin` и `end`, задающими моменты начала и окончания изменений, которые требуется применить к объекту. При этом моменты определяются не значениями времени, а событиями. Например, атрибут `begin="click"` указывает, что некоторое изменение объекта следует начать тогда, когда пользователь щелкнет на нем левой кнопкой мыши.

Рассмотрим в качестве примера создание графической кнопки с типичным визуальным эффектом, возникающим при нажатии и отпускании кнопки мыши. Графическую кнопку представим в виде прямоугольника, толщина обводки которого немного увеличивается при нажатии (событие `mousedown`) и возвращается в исходное состояние при отпускании (событие `mouseup`) кнопки мыши (рис. 13.55).

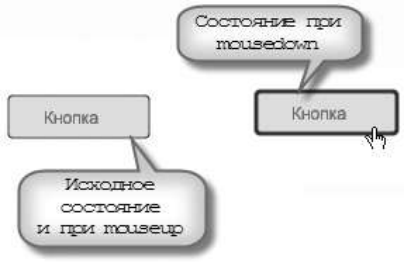



**Рис. 13.55.** Пример кнопки

Листинг 13.50 содержит код, посредством которого все это можно сделать.

**Листинг 13.50. Графическая кнопка (вариант 1)**

```
<rect x="10" y="50" width="100" height="30" rx="3"
    fill="cyan" stroke="blue" stroke-width="1">
    <set attributeName="stroke-width" to="3"
       begin="mousedown" end="mouseup"/>
</rect>
```

```
<text x="35" y="70" fill="red">
   Кнопка
</text>
```

Здесь тег <set> находится в контейнере <rect> и поэтому ясно, что установка свойства по событиям мыши относится к данному прямоугольнику. Вместе с тем, тег <set> можно расположить в другом месте кода и сослаться на прямоугольник по значению его id (листинг 13.51).

**Листинг 13.51. Графическая кнопка (вариант 2)**

```
<rect id="myrect" x="10" y="50" width="100" height="30" rx="3"
    fill="cyan" stroke="blue" stroke-width="1">
</rect>
<text id="mytext" x="35" y="70" fill="red">
   Кнопка
</text>
<set xlink:href="#myrect" attributeName="stroke-width" to="3"
       begin="mousedown" end="mouseup"/>
```

Код листинга 13.52 является модификацией предыдущего, в которой добавляется изменение цвета текста при наведении указателя мыши на прямоугольник.

**Листинг 13.52. Графическая кнопка (вариант 3)**

```
<rect id="myrect" x="10" y="50" width="100" height="30" rx="3"
    fill="cyan" stroke="blue" stroke-width="1">
</rect>
<text id="mytext" x="35" y="70" fill="red">
   Кнопка
</text>
<set xlink:href="#myrect" attributeName="stroke-width" to="3"
       begin="mousedown" end="mouseup"/>
<set xlink:href="#mytext" attributeName="fill" to="blue"
       begin="myrect.mouseover" end="myrect.mouseout"/>
```

***Внимание***

Обратите внимание на то, как заданы значения атрибутов begin и end во втором теге <set> для изменения цвета текста: события мыши mouseover и mouseout указываются с префиксом myrect, который, в свою очередь, указывает на элемент, принимающий событие (это значение его id). Без данного префикса изменение цвета текста происходило бы при наведении указателя мыши на сам текст.

Итак, мы рассмотрели создание кнопки, которая изменяет свои параметры при некоторых событиях мыши. Если такую кнопку заключить в контейнерный тег <a>, определяющий ссылку, то щелчок на кнопке инициирует загрузку соответствующего документа.

Описание внешнего вида одной кнопки можно поместить в контейнер <defs>, а затем использовать ее клоны для создания нескольких однотипно оформленных ссылок. Однако этот прием годится для неанимированных кнопок. Если же кнопка анимирована, то эффект анимации будет проявляться на всех ее клонах.

Разумеется, тег <set> не единственно возможный для задания обработчика событий. Вы можете применять и другие теги анимации. Например, следующий код (листинг 13.53) плавно увеличивает размеры графического изображения в течение 15 с, если пользователь навел на него указатель мыши.

**Листинг 13.53. Пример анимации при наведении указателя мыши**

```
<!-- Плавное масштабирование при наведении указателя мыши -->
<image id="myimg" xlink:href="img/picture.jpg"
      x="0" y="0" width="100" height="75"
<animateTransform xlink:href="#img1"
       attributeName="transform" attributeType="XML"
       type="scale" from="1" to="2" dur="15s"
       begin="onmouseover"
       fill="freeze"
/>
```

К данному коду можно добавить обработчик события clik, чтобы по щелчку на изображении оно плавно приняло исходные размеры (листинг 13.54).

**Листинг 13.54. Пример плавного масштабирования**

```
<!-- Плавное масштабирование при наведении указателя мыши и щелчке -->
<image id="myimg" xlink:href="img/picture.jpg"
      x="0" y="0" width="100" height="75"
<animateTransform xlink:href="#img1"
       attributeName="transform" attributeType="XML"
       type="scale" from="1" to="2" dur="15s"
       begin="onmouseover"
       fill="freeze"
/>
<animateTransform xlink:href="#img1"
        attributeName="transform" attributeType="XML"
```

```
        type="scale" from="2" to="1" dur="15s"
        begin="click"
        fill="freeze"
/>
```

В листинге 13.55 приведен код документа с тремя графическими изображениями, каждое из которых при наведении указателя мыши масштабируется и перемещается (рис. 13.56). Щелчком можно вернуть изображение в исходное состояние. Это весьма грубый эскиз фотогалереи.

**Листинг 13.55. Фотогалерея**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="800" height="400">
<title>События мыши</title>
<image id="img1" xlink:href="picture1.jpg"
       x="10" y="10" width="100" height="75"
       transform="matrix(1 0 0 1 0 0)"/>
  <image id="img2" xlink:href="picture2.jpg"
       x="120" y="10" width="100" height="75"
       transform="matrix(1 0 0 1 0 0)"/>
<image id="img3" xlink:href="picture3.jpg"
       x="230" y="10" width="100" height="75"
       transform="matrix(1 0 0 1 0 0)"/>

<set xlink:href="#img1" attributeName="transform"
     to="matrix(2 0 0 2 -10 80)"
     begin="img1.mouseover" end="click"/>
<set xlink:href="#img2" attributeName="transform"
     to="matrix(2 0 0 2 -10 80)"
     begin="mouseover" end="click"/>
<set xlink:href="#img3" attributeName="transform"
     to="matrix(2 0 0 2 -10 80)"
     begin="mouseover" end="click"/>
</svg>
```


**Рис. 13.56.** Эскиз фотогалереи

Код листинга 13.55 работает в Opera, Safari и Chrome. В Firefox 3.6 он не действует, поскольку данная версия браузера вообще не поддерживает анимацию своими средствами. В Internet Explorer с плагином Adobe SVGViewer приведенный код не работает, видимо, из-за особенностей плагина.

Обеспечить надежную и совместимую со всеми основными браузерами обработку событий в SVG-документах можно с помощью скриптов на языке JavaScript, которую вкратце мы рассмотрим далее.

## Применение JavaScript

JavaScript в SVG, как и вообще в XML-документах, применяется так же, как и в (X)HTML-документах. Здесь мы рассмотрим, причем весьма бегло, его применение в SVG-документах с целью обработки событий, вызванных манипуляциями мышью. Обработку событий клавиатуры и некоторых других мы здесь рассматривать не будем.

Для обработки события необходимо организовать трехместное отношение между именем события, элементом документа и обработчиком события. Например, щелчок левой кнопкой мыши (событие) может произойти, когда указатель мыши находится на том или ином элементе документа, и тогда должен выполниться тот или иной скрипт (обработчик данного события); для данного элемента мы можем указать, на какое событие и каким обработчиком он должен среагировать.

Обработчик события представляет собой код на языке JavaScript. Обычно это функция, код определения которой размещается в контейнере `<script type="text/javascript">` внутри дескриптора `<![CDATA[ … ]]>`, подобно тому, как вставляется каскадная таблица стилей.

Привязка обработчика события к SVG-элементу и собственно событию может осуществляться посредством атрибута-события тега данного элемента. Имя атрибута-события образуется из названия события с добавлением префикса on, например, `onclick` (при щелчке), `onmouseover` (при наведении указателя мыши) и т. п. Значением атрибута-события является одно или несколько выражений JavaScript, разделенных точкой с запятой, но обычно это вызов функции. Листинг 13.56 иллюстрирует типичную схему применения обработки события `onclick`, возникающего при щелчке на изображении прямоугольника.

**Листинг 13.56. Обработка события `onclick`**

```
<script type="text/javascript">
<![CDATA[
    function myfunc(){
    /* здесь код тела функции*/
    }
 ]]>
</script>
<rect width="100" height="50" fill="red"
onclick="myfunc()"/>
```

Здесь при щелчке (атрибут `onclick`) на прямоугольнике (тег `<rect>`) вызывается некоторая функция `myfunc()` — обработчик события, код которого определен в контейнере `<script>`.

Обработчик события выполняет некоторые действия. Если это действия над объектами документа, то применяются методы объектной модели документа (DOM — Document Object Model). Так, доступ к объекту по значению его атрибута `id` обеспечивает метод `getElementById("`*значение id*`")`, а чтение и установку значений атрибутов — методы `getAttribute("`*имя атрибута*`")` и `setAttribute("`*имя атрибута*`", "`*значение*`")`.

Например, установить значение атрибута `fill` элемента `<rect id="myrect" …/>` можно с помощью такого выражения на языке JavaScript:

```
document.getElementById("myrect").setAttribute("fill", "blue")
```

В листинге 13.57 приведен код документа с тремя графическими изображениями, каждое из которых при наведении указателя мыши масштабируется и перемещается (см. рис. 13.56). Щелчком мышью можно вернуть изображение в исходное состояние. Данный эскиз фотогалереи уже рассматривался

ранее (листинг 13.55), но без применения JavaScript. Новый вариант, в отличие от прежнего, работает во всех основных браузерах.

В данном примере определены функции `mover(who)` и `mclick(who)` в качестве обработчиков событий `onmouseover` и `onclick` соответственно. Они принимают в качестве параметра `who` значение `id` элемента, в котором событие возникло, и устанавливают для данного элемента некоторое значение атрибута `transform`.

**Листинг 13.57. Фотогалерея с применением JavaScript**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="800" height="400">
<title>События мыши</title>
<script type="text/javascript">
<![CDATA[
  function mover(who){ /* обработка наведения указателя*/
    document.getElementById(who).setAttribute(
          "transform","matrix(2 0 0 2 -10 80)"
    )
  }
  function mclick(who){ /* обработка щелчка */
  document.getElementById(who).setAttribute(
          "transform","matrix(1 0 0 1 0 0)")
  }
]]>
</script>

  <image id="img1" xlink:href="picture1.jpg"
      onmouseover="mover('img1')"
      onclick="mclick('img1')"
      x="10" y="10" width="100" height="75"
  />
  <image id="img2" xlink:href="picture2.jpg"
      onmouseover="mover('img2')"
      onclick="mclick('img2')"
      x="120" y="10" width="100" height="75"
  />
  <image id="img3" xlink:href="picture3.jpg"
```

```
      onmouseover="mover('img3')"
      onclick="mclick('img3')"
      x="230" y="10" width="100" height="75"
  />
</svg>
```

Обратите внимание, что по щелчку на увеличенной картинке устанавливается атрибут `transform="matrix(1 0 0 1 0 0)"`, который ничего не трансформирует (это тождественное преобразование), но картинка возвращается в исходное состояние. Дело в том, что все преобразования выполняются над исходным элементом, а не над ранее измененным. Поэтому в рассматриваемом примере функция `mclick(who)` могла бы установить для атрибута `transform` пустое значение:

```
document.getElementById(who).setAttribute("transform","")
```

Чтобы выполнить преобразование над результатом предыдущего преобразования, следует в качестве значения атрибута `transform` использовать последовательность всех требуемых частных преобразований. Например, следующий, более громоздкий, вариант кода функции `mclick()` делает то же самое:

```
function mclick(who){ /* обработка щелчка */
  var x=document.getElementById(who).getAttribute("transform")
  document.getElementById(who).setAttribute(
          "transform",x+"matrix(0.5 0 0 0.5 5 -40)")
  }
```

Функция сохраняет в переменной `x` текущее значение атрибута `transform` (в увеличенном состоянии картинки это `"matrix(2 0 0 2 -10 80)"` ), а затем к нему дописывает обратное преобразование (`"matrix(0.5 0 0 0.5 5 -40)"`, т. е. уменьшение в два раза). В результате картинка возвращается в исходное состояние. Но пара описанных преобразований эквивалентна одному тождественному преобразованию `"matrix(1 0 0 1 0 0)"` (см. *разд. 13.11.6*), которое и применяется к исходному изображению. Таким образом, последний вариант функции `mclick()` эквивалентен первому, но менее эффективен. Он был приведен лишь для иллюстрации применения методов `getAttribute()` и `setAttribute()` для изменения значений атрибутов.

Рассмотрим задачу однотипной трансформации элемента по щелчку на нем. Например, пусть требуется поворачивать какой-нибудь элемент на заданный угол при щелчке на нем, причем поворот должен происходить относительно текущего состояния элемента. Для решения как частной задачи (поворот), так и более общей (произвольная трансформация) нам понадобится функция, выполняющая произведение двух матриц, представляющих две трансформации. Эта функция (назовем ее `pmatrix()`) принимает два параметра вида

`"matrix(a1 b1 c1 d1 e1 f1)"` и `"matrix(a2 b2 c2 d2 e2 f2)"`

и возвращает строку вида

`"matrix(a3 b3 c3 d3 e3 f3)"`

Как параметры функции, так и возвращаемое ею значение имеют матричную форму значения атрибута `transform` (см. *разд. 13.11.6*). По существу, функция `pmatrix()` выполняет произведение двух матриц преобразований и возвращает результат в удобном для последующего использования виде. Код данной функции представлен в листинге 13.58.

**Листинг 13.58. Произведение матриц трансформаций**

```
function pmatrix(s1,s2){
/*Параметры: строки вида "matrix(a,b,c,d,e,g)"
  Возвращает строку вида "matrix(a,b,c,d,e,g)",
  являющуюся результатом произведения двух матриц
*/
var sep;// разделитель может быть запятой и пробелом
if (s1.indexOf(",")==-1) {sep=' '} else {sep="," };
var l=s1.length
var s1=s1.substring(7,s1.length-1); // содержимое скобок 1-го парам.
s1=s1.split(",");                   // делаем массив
var s2=s2.substring(7,s2.length-1); // содержимое скобок 2-го парам.
if (s2.indexOf(",")==-1) {sep=" "} else {sep=","};
s2=s2.split(",");                   // делаем массив

return "matrix("+(s1[0]*s2[0]+s1[2]*s2[1])+","+
      (s1[1]*s2[0]+s1[3]*s2[1])+","+
      (s1[0]*s2[2]+s1[2]*s2[3])+","+
      (s1[1]*s2[2]+s1[3]*s2[3])+","+
      (s1[0]*s2[4]+s1[2]*s2[5]+s1[4]*1)+","+
      (s1[1]*s2[4]+s1[3]*s2[5]+s1[5]*1)+")";
}
```

Листинг 13.59 содержит пример, в котором каждый щелчок на графическом растровом изображении поворачивает его на 30° относительно предыдущего положения.

**Листинг 13.59. Поворот графического изображения по щелчку**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

```
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="720"
   xmlns:xlink="http://www.w3.org/1999/xlink"
>
<title>Поворот изображения по щелчку</title>
<script type="text/javascript">
<![CDATA[
 function pmatrix(s1,s2){
 // код (см. листинг 13.33)
}

 function imgrotate(){ // поворачиваем картинку на 30 град.
  var x=document.getElementById("myimg"); // объект картинки
/* текущее значение атрибута transform: */
  curattr=x.getAttribute("transform");
/* к текущему состоянию применяем поворот: */
  x.setAttribute("transform",
    pmatrix(curattr,"matrix(0.867,0.5,-0.5,0.867,0,0)"))
 }
]]>
</script>
<image id="myimg"
    transform="matrix(1 0 0 1 120 120)"
    x="0" y="0" width="300" height="200" xlink:href="picture.jpg"
    onclick="imgrotate()"
/>
</svg>
```

Если в листинге 13.59 вместо обработчика `onclick="imgrotate()"` установить `onload="setInterval('imgrotate()', 500)"`, то картинка будет поворачиваться самостоятельно каждые 0,5 с. Здесь по окончании загрузки элемента вызывается метод `setInterval()`, который, в свою очередь, с периодом 500 мс вызывает функцию, указанную в качестве первого параметра.

Разумеется, рассмотренная задача допускает и иные решения. Например, можно не использовать функцию `pmatrix()`, а функцию `imgrotate()` определить так:

```
function imgrotate(){ // поворачиваем картинку на 30 град.
  var x=document.getElementById("myimg"); // объект картинки
/* к текущему состоянию применяем поворот: */
  x.setAttribute("transform",
      x.getAttribute("transform") + " matrix(0.867,0.5,-0.5,0.867,0,0)")
 }
```

Здесь новое значение атрибута `transform` устанавливается приписыванием к текущему значению через пробел строки, отвечающей за поворот на 30°. Однако при этом увеличивается объем данных, сохраняемых в атрибуте `transform`.


**Рис. 13.57.** Состояния пары панелей (листинг 13.60)

Для экономии места на Web-странице панели с гиперссылками можно расположить в стопке и перелистывать их с помощью мыши. В листинге 13.60 приведен пример создания двух панелей, одна из которых частично перекрывает другую. При наведении указателя мыши на панель заднего плана передняя панель поворачивается, открывая заднюю. При наведении указателя на переднюю панель она занимает исходное положение. На рис. 13.57 показаны различные состояния данной пары панелей.

Несмотря на относительно большой объем, код имеет довольно простую структуру. В контейнере `<defs>` определены два прямоугольника: один для отображения тени панели, а другой для ее лицевой стороны. Для лицевых сторон панелей определены заливки линейными градиентами. Вывод на экран прямоугольников с градиентной заливкой производится с помощью тегов `<use>`. Данные заготовки группируются вместе с текстами, ссылками и графическим растровым изображением. Обработчик наведения указателя мыши на заднюю панель выполняет наклон, масштабирование и перенос группы элементов, образующих переднюю панель. Вся совокупность перечисленных преобразований компактно записана в матричном виде. Обработчик наведения указателя мыши на переднюю панель задает пустое значение для свойства `transform`. В результате передняя панель либо остается в исходном положении, либо возвращается в него.

**Листинг 13.60. Пара панелей**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="720"
   xmlns:xlink="http://www.w3.org/1999/xlink"
>
<title>Панели меню</title>
<style type="text/css">
<![CDATA[
      .head { /* заголовок панели */
         font-size:20px;font-weight:bold;
         fill:#CC3300
      }
    text {font-size:14px; }
    a {font-family:sans-serif}
    a:hover {fill:darkred}
]]>
</style>
<script type="text/javascript">
<![CDATA[
/* Обработчики события onmouseover */
 function rot1(){  // приводим переднюю панель в исходное состояние
  var x=document.getElementById("mypane1");
  x.setAttribute("transform","")
 }
function rot2(){  // поворачиваем переднюю панель на 30 град.
  var x=document.getElementById("mypane1");
  x.setAttribute("transform",
    "matrix(0.4, -0.268, 0, 1, 0, 40)")
}
]]>
</script>

<defs>
    <!-- Тень -->
    <rect id="shadow"  width="180" height="210"  rx="10"
          fill="black" fill-opacity="0.4"/>
    <!-- Лицевая часть -->
    <rect id="face"    width="180" height="210"  rx="10"
          stroke="#00f0f0"  fill-opacity="0.9"/>

    <linearGradient id="MyGradient1"
```

```
          x1="0%" y1="0%" x2="20%" y2="100%">
        <stop offset="0" style="stop-color:#f0ffff;"/>
        <stop offset="100%" style="stop-color:#00a0a0;"/>
    </linearGradient>

    <linearGradient id="MyGradient2"
          x1="0%" y1="0%" x2="25%" y2="100%">
        <stop offset="0" style="stop-color:#f8f8f8;"/>
        <stop offset="100%" style="stop-color:#0080ff;"/>
    </linearGradient>
</defs>

<g onmouseover="rot2()">
<!-- Панель 'Разное' -->
  <!-- Тень -->
  <use xlink:href="#shadow" x="103" y="9"/>
  <!-- Лицевая часть -->
  <use xlink:href="#face" x="95" y="3" fill="url(#MyGradient1)"/>
  <!-- Содержимое панели -->
  <text x="150" y="30" class="head" >Разное</text>
  <a xlink:href="http://dunaevv1.narod.ru">
    <text x="105" y="57">Сам себе Web-дизайнер</text>
  </a>
  <a xlink:href="http://www.w3.org/TR/SVG11">
    <text x="105" y="77">Спецификация SVG</text>
  </a>
  <a xlink:href="http://ru.wikipedia.org/wiki/">
    <text x="105" y="117">Википедия</text>
  </a>
  <image x="190" y="100" width="79" height="76" xlink:href="book.gif"/>
  <!-- Здесь могут быть еще элементы -->
</g>

<g  id="mypane1" onmouseover="rot1()">
<!-- Панель 'Поисковики' -->
  <!-- Тень -->
  <use xlink:href="#shadow" x="48" y="46"/>
  <!-- Лицевая часть -->
  <use xlink:href="#face" x="40" y="40" fill="url(#MyGradient2)"/>
  <!-- Содержимое панели -->
  <text x="70" y="63" class="head">Поисковики</text>
  <a xlink:href="http://www.yandex.ru">
    <text x="85" y="90">Яндекс</text>
  </a>
```

```
  <a xlink:href="http://www.rambler.ru">
    <text x="85" y="110">Rambler</text>
  </a>
  <a xlink:href="http://www.google.ru">
    <text x="85" y="130">Google</text>
  </a>
  <a xlink:href="http://www.yahoo.com">
    <text x="85" y="150">Yacho</text>
  </a>
  <a xlink:href="http://www.altavista.com">
    <text x="85" y="170">AltaVista</text>
  </a>
  <a xlink:href="http://www.aport.ru">
    <text x="85" y="190">Апорт</text>
  </a>
  <image x="160" y="190" width="48" height="48" xlink:href="ask.png"/>
  <!-- Здесь могут быть еще элементы -->
</g>
</svg>
```

## 13.14. Вставка в SVG-документ XHTML-кода

В SVG-документы можно включать внешние элементы, имеющие иное пространство имен. В частности, есть возможность вставки XHTML-кода, которая оказывается весьма кстати при форматировании текстов, создании элементов форм, таблиц, внедрении видео, Web-страниц и т. п. внутри SVG-документа.

Для вставки внешних объектов применяется контейнерный тег `<foreignObject>`. Прямоугольная область, в которой отображается содержимое внешнего объекта, задается атрибутами:

- `x`, `y` — горизонтальная и вертикальная координаты верхнего левого угла;
- `width`, `height` — ширина и высота.

В данном разделе мы вкратце рассмотрим вставку в SVG-документ фрагментов XHTML-кода. Более подробную информацию по данной теме можно получить в спецификации (**http://www.w3.org/TR/SVG11/extend.html**).

Шаблон для вставки (X)HTML-кода:

```
<foreignObject x="x" y="y" width="w" height="h">
   <html xmlns="http://www.w3.org/1999/xhtml">
   <!-- Здесь располагается XHTML-код -->
   </html>
</foreignObject>
```

Здесь <html> можно заменить на <body>. Данные контейнеры удобны при вставке нескольких XHTML-элементов. Тогда атрибут xmlns указывают только в теге самого верхнего уровня, <html> или <body>. При вставке одного элемента указывают только соответствующий ему тег, но с атрибутом xmlns.

В листинге 13.61 приведен пример вставки нескольких элементов XHTML. При этом SVG-элементы <rect> и <foreignObject> сгруппированы, а группа повернута на 5°. Это сделано для демонстрации, что над вставленным фрагментом можно выполнять такие же операции, как и над SVG-элементами. Вид данного документа в браузере показан на рис. 13.58.

**Листинг 13.61. Вставка XHTML-кода в SVG-документ**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="400" height="400">
<title>Вставка XHTML</title>

<g transform="rotate(5)">
 <rect x="20" y="10" width="250" height="200" rx="15"
       fill="#00ffff" stroke="blue"/>
 <foreignObject x="30" y="10" width="230" height="180">
   <!-- Здесь XHTML-код -->
   <body xmlns="http://www.w3.org/1999/xhtml">
     <style type="text/css">
        h1 {color:red}
     </style>
     <h1>Заголовок</h1>
     <p>Здесь расположены элементы,<br/>
        заданные XHTML-кодом.
     </p>
     Введите что-нибудь:
     <input type="text" />
   </body>
 </foreignObject>
</g>
</svg>
```

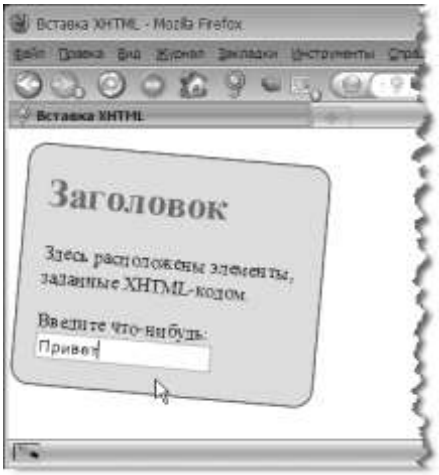



**Рис. 13.58.** Вставка XHTML-кода в SVG-документ

В SVG-документ можно вставить XHTML-документ из внешнего файла и при желании трансформировать его (масштабировать, повернуть и т. п.). В листинге 13.62 приведен пример вставки в SVG-документ двух плавающих фреймов с загрузкой в них Web-страниц. При этом фреймы со своим содержимым поворачиваются на 5° и масштабируются с коэффициентом 0,3, сохраняя свою функциональность (гиперссылки, поля ввода, Flash-ролики и т. п.). Вид данного документа в браузере Firefox показан на рис. 13.59.

**Листинг 13.62. Вставка Web-страниц**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.0"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="1000" height="800">
<title>Вставка XHTML</title>
 <foreignObject x="200" y="10" width="1000" height="800"
                transform="scale(0.3) rotate(5)">
   <iframe xmlns="http://www.w3.org/1999/xhtml"
           src="http://www.rambler.ru"
           width="1000" height="800">
   </iframe>
 </foreignObject>
 <foreignObject x="100" y="400" width="1000" height="800"
```

```
                transform="scale(0.3) rotate(-5)">
   <iframe xmlns="http://www.w3.org/1999/xhtml"
           src="http://dunaevv1.narod.ru"
           width="1000" height="800">
   </iframe>
 </foreignObject>
</svg>
```

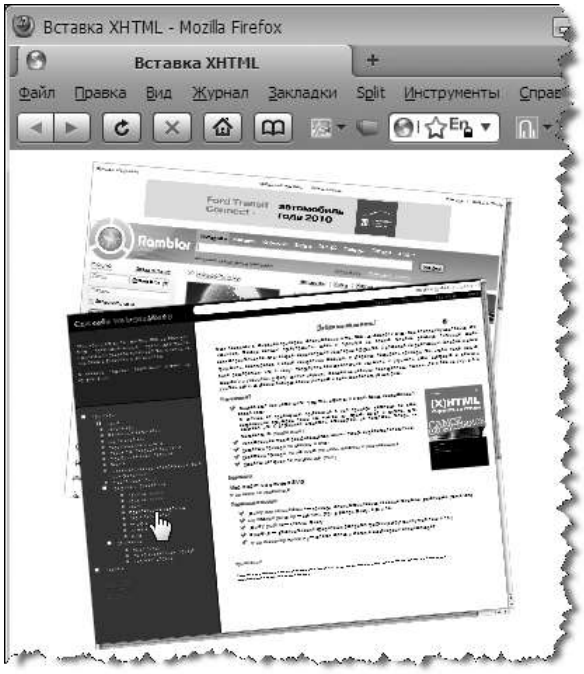



**Рис. 13.59.** Вставка и трансформация Web-страниц

Аналогичным образом в SVG-документ можно вставить и другие внешние объекты. В листинге 13.63 приведен пример внедрения FLV-проигрывателя с загружаемым в него видео формата FLV (см. *разд. 11.2.4*). При этом проигрыватель поворачивается на 10°.

**Листинг 13.63. Вставка видео**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
     xmlns="http://www.w3.org/2000/svg" version="1.0"
```

```
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="1000" height="800">
<title>Вставка XHTML</title>
<foreignObject x="50" y="20" width="600" height="500"
               transform="rotate(10)">
<body xmlns="http://www.w3.org/1999/xhtml" >
<embed src="flvplayer.swf" type="application/x-shockwave-flash"
       allowscriptaccess="always" allowfullscreen="true" wmode="opaque"
       flashVars='st=styles/mystyle.txt&amp;fyle=video/masterbook.flv'
       width="320" height="240"/>
</body>
</foreignObject>
</svg>
```