Глава 1

Сглаживание (Anti-Aliasing) и субпиксельная точность (**Subpixel Accuracy**).

Сглаживание — это хорошо известная техника для улучшения визуального качества изображения при отображении его на устройствах с низкой разрешительной способностью. Оно базируется на свойствах человеческого зрения. Взгляните на приведенную картинку и попытайтесь угадать, что означает указанная надпись (рис. 1.1).

1.1.

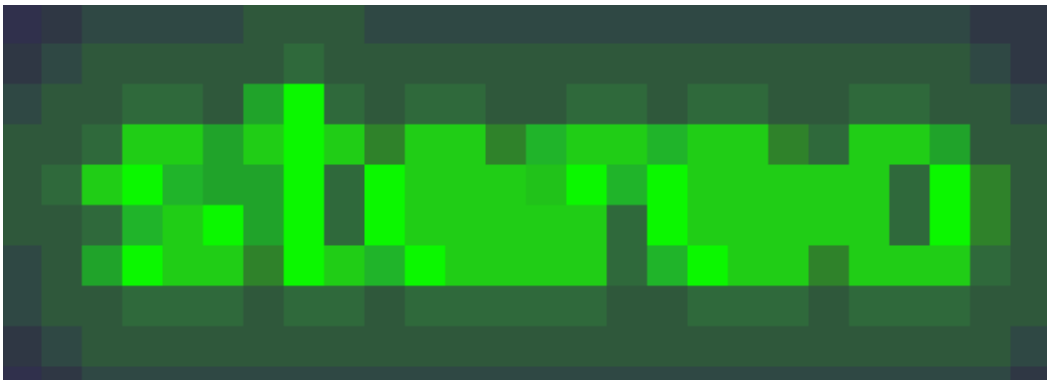


Рис.1.1. Увеличенное изображение

Разобрать надпись на приведенном рисунке достаточно сложно. Это слово написано с использование **сглаживания** (Anti-Aliasing).

Теперь взгляните на ту же картинку, имеющую нормальный размер и отображенную вместе с контекстом (рис. 1.2).

1.2.

Теперь Вы очень легко узнаете слово “stereo”. Хотя, картинка, отображенная на предыдущем (рис. 1.1) и данном рисунке (рис. 1.2), одна и та же.

Рис.1.2. Исходный размер изображения

Первая картинка (рис. 1.1) — это увеличенная версия второй картинки (рис. 1.2). Свойство сглаживания позволяет нам реконструировать потерянную информацию, базируясь на накопленном опыте. Результат превосходный!

Но суть не только в самом сглаживании. Суть в том, что мы можем рисовать примитивы с **субпиксельной точностью** (Subpixel Accuracy). Это особенно важно для визуальной толщины линий.

Но сначала давайте посмотрим, что даже при использовании интерполятора линий Брезенхема мы можем достичь лучших результатов по сравнению с использованием субпиксельной точности (**Subpixel Accuracy**). Следующий рисунок показывает увеличенные результаты простого интерполятора Брезенхема.

1.3.

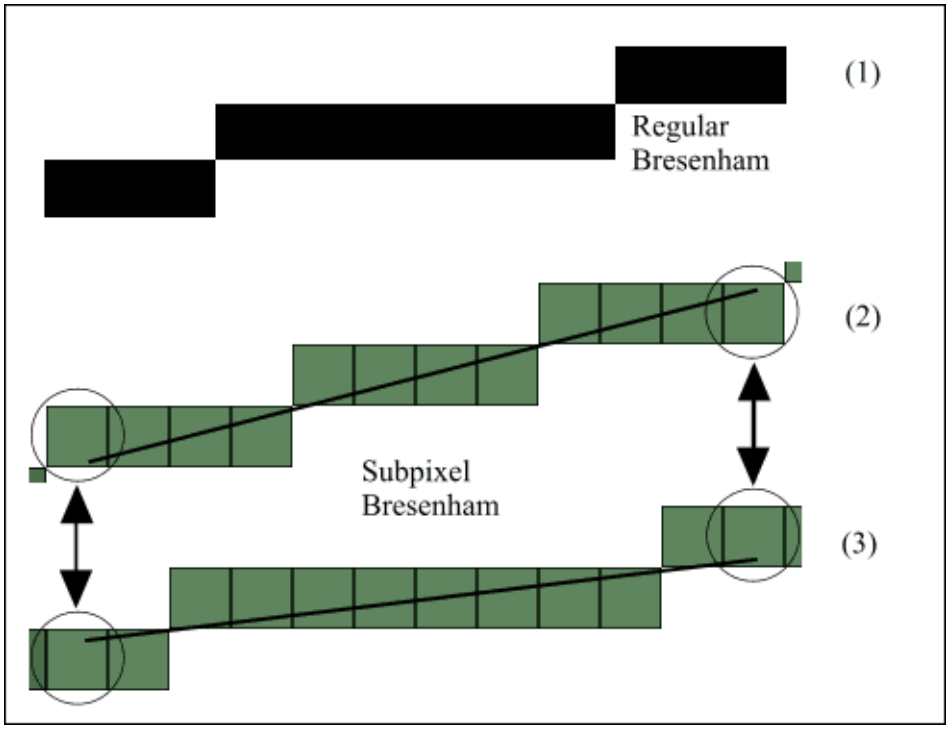


Рис.1.3. Интерполятор Брезенхема

Сравните случаи (2) и (3) (рис. 1.3). Тонкие черные линии — это то, что мы должны проинтерполировать. При использовании субпиксельной точности (Subpixel Accuracy) мы действительно получим два разных набора отображенных пикселей, несмотря на тот факт, что начало и конец обеих линий находятся в одних и тех же пикселях. Также линии имеют разные тангенсы, что является очень важным. При использовании классического алгоритма Брезенхема не учитывая Subpixel Accuracy мы каждый раз будем получать результат, отображенный на рисунке (рис. 1.3) в пункте (1). Это особенно важно при аппроксимации кривых короткими прямыми сегментами. Однако, при использовании свойства сглаживания (Anti-Aliasing) в комбинации со субпиксельной точностью (Subpixel Accuracy) мы можем получить гораздо лучший результат.

Разницу Вы можете оценить, взглянув на картинки:

1.4.

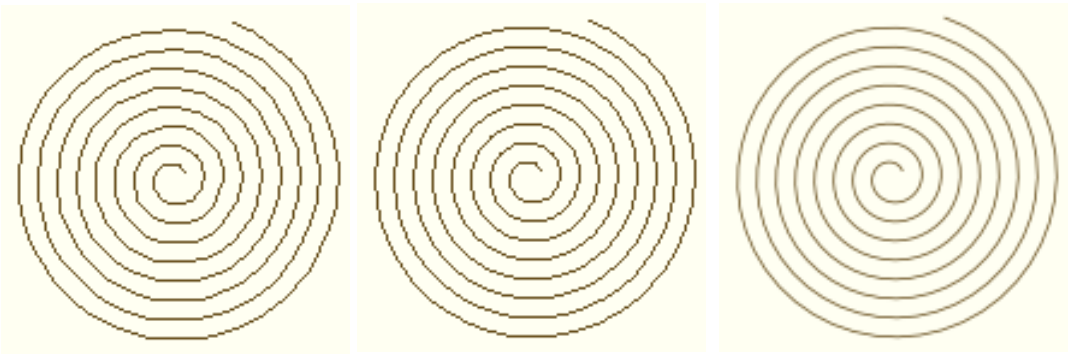


Рис.1.4. Аппроксимация

Все три спирали (рис. 1.4) аппроксимируются короткими прямыми сегментами линии.

Крайняя левая спираль отрисована с помощью правильного целочисленного алгоритма Брезенхема, при использовании которого координаты обращаются в пиксели (Вы получите похожий результат при использовании, например, функций Windows GDI MoveTo/ LineTo).

Средняя спираль отрисована при использовании модифицированного целочисленного алгоритма Брезенхема с точность 1/256 пикселя.

Крайняя правая спираль использует ту же точность (1/256 пикселя), но только со сглаживанием.

При использовании правильного целочисленного алгоритма аппроксимации мы получим более приемлемые результаты.

Обратите внимание, что очень важно иметь возможность реального позиционирования подпикселя сегмента строки. При использовании правильных координат пикселя со сглаживанием (Anti-Aliasing) спираль будет выглядеть гладкой, но, тем не менее, столь же неприемлемо, как крайняя левая.

Субпиксельная точность более важна для контроля визуальной толщины линии. Это возможно только в том случае, если будет использоваться хороший алгоритм сглаживания. С другой стороны, в сглаживании нет смысла, если можно установить ширину линии с дискретностью только в один пиксель. Сглаживание (Anti-Aliasing) и субпиксельная точноть (Subpixel Accuracy) всегда используются вместе.

Современные дисплеи имеют разрешающую способность до 120 DPI, в то время, как субпиксельная точность (Subpixel Accuracy) — до 300 DPI. Следующая картинка показывает линию толщиной начиная с 0,3 пикселей и увеличивая каждый раз на 0,3 пикселя.

1.5.

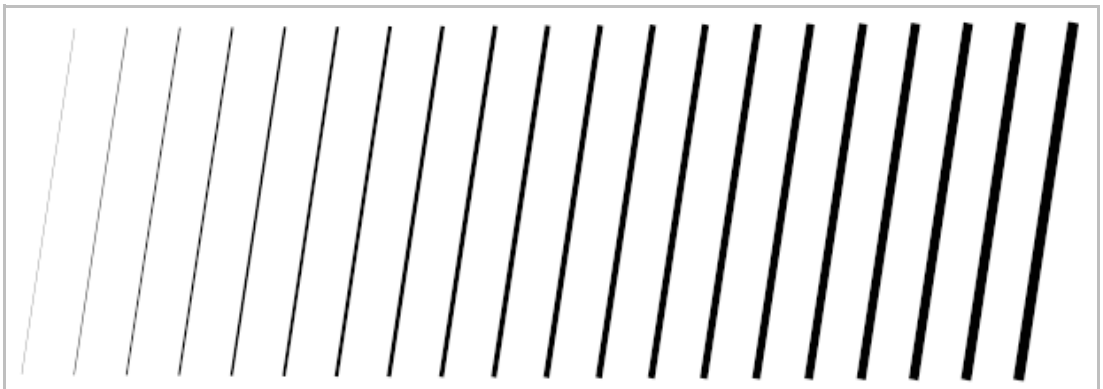


Рис.1.5. Линия

Глава 2

Библиотека символов представляет собой набор векторных элементов, которые в последствии могут использоваться для отображения векторных объектов:

- рисования точеных символов;
- рисования нестандартных линий (заполненных символами вдоль оси линии);
- заполнения полигонов нестандартной заливкой (символами);
- оконтуривания полигонов различными типами линий.

Для описания структуры символов применяется стандартный язык XML, что значительно облегчает редактирование и создание библиотеки. При загрузке в **XXXX** происходит интерпретация XML-описания и построение бинарного представления библиотеки, используемого в дальнейшем для отображения векторных объектов (“кэширование”). В том случае, если исходная библиотека изменилась — автоматически пересоздается бинарная версия. Структура хранения данных в библиотеке символов приведена ниже (рис. 2.1).

Библиотека состоит из:

- **раздел палитры цветов** (palette) — содержит перечень цветов (в RGB формате), применяемый для окраски символов и фигур;
- **раздела глифоф** (vector_glyph) — содержит элементарные примитивы (Glyph) из которых в дальнейшем и строится отображение конкретных векторных символов;
- **раздела фигур** (vector_shape) — содержит описание фигур, доступных для построения векторных символов;
- **символьных разделов** (vector_point, vector_line, vector_poly) — содержат описание символов, применяемых для отображения фигур в соотвтетствии с их типом (точечные, линейные или полигональные).

В каждом разделе содержатся полностью законченные **элементы** — глифы, фигуры или символы.

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

<symbols>
  <!-- раздел палитры цветов -->
  <palette name="palette">[1]
        .....
  </palette>

  <!-- раздел глифов -->
  <vector_glyph name="glyphs">
        .....
  </vector_glyph>

  <!-- раздел фигур -->
  <vector_shape name="shapes">
```

1. В соответствии со стандартом оформления значений переменных в XML необходимо всегда обрамлять переменную кавычками. Пример: `name="shapes"`

```
        .....
</vector_shape>

<!-- раздел, содержащий символы для отображения точечных
примитивов -->
<vector_point name="point_symbols">
        .....
</vector_point>

<!-- раздел, содержащий символы для отображения линейных
примитивов -->
<vector_line name="line_symbols">
        .....
 </vector_line>

<!-- раздел, содержащий символы для
отображения полигональных примитивов -->
<vector_poly name="poly_symbols">
        .....
</vector_poly>
</symbols>
```

*В описании разделов указывается параметр **name**, значение которого является зарезервированым и используется для служебных целей. Не меняйте указаное значение!*

Прорисовка каждого элемента (точнее его составляющих) выполняется последовательно и схожа с интерпретацией программного кода. Все “операторы” выполняют определенные действия, исходя из указанных параметров.

К примеру, указав оператор **move** и параметры x=2 и y=3, мы переместим текущую позицию курсора в точку с координатами [2,3]. Оператор **line** с соответствующими параметрами 4,7 отобразит линию из точки x=2,y=3 в точку x=4,y=7.

При создании символов необходимо помнить, что описание символа интерпретируется и выполняется (отображается) **последовательно** и порядок записи существенным образом влияет на то, каким образом будет отображаться результирующий элемент.

Название елементов

Элементы различних разделов связаны друг с другом. К примеру, в состав фигур входят глипсы, а глипсы - основа символов (рис. 2.1.). Для “связывания” между собой элементов используются символические названия, указываемые в параметре **caption**.

Пример:

```
<glyph caption="ElipsOid">
 <both x="0" y="0">
   <elipse x="0" y="0" rx="5" ry="5" angle="0"/>
 </both>
</glyph>
```

Для включения (связывания) елементов используется параметр **bind_name**. Вы просто указываете нужный оператор и его название.

Пример:

Фигура содержит глипс **ElipsOid**

```
<shape caption="20" brushindex="0">
  <offset x="-4" y="0"/>
  <glyph bind_name="ElipsOid"/>
</shape>
```

2.1.

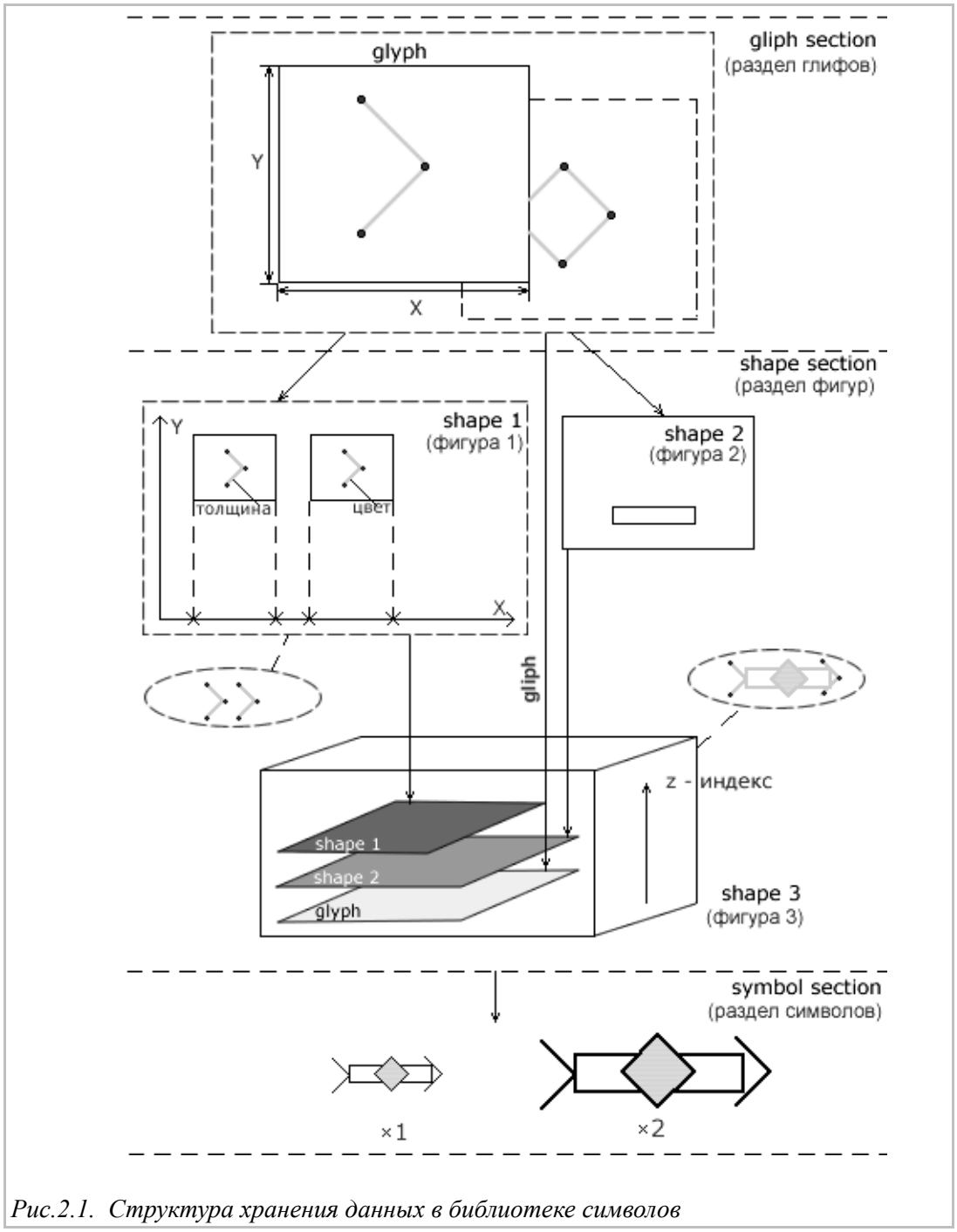


Рис.2.1. Структура хранения данных в библиотеке символов

Система координат и единицы измерения

Выберем на плоскости произвольную точку (“центр координат”) и проведем от нее 2-ве перпендикулярных оси — X и Y. Так мы получим систему координат, в которой будут указаны координаты графических примитивов (линий, кривых и прочих элементов, входящих в состав глифа). В качестве единиц измерения выберем некоторые “относительные единицы”.

Перед отображением графического примитива на экране монитора (или любом другом устройстве) необходимо определить реальное расположение (в пикселах и т.п.) “центра координат”. Например, при отображении точечного символа “центр координат” просто переместится в позицию на экране с указанными пиксельными координатами (рис. 2.2).

2.2.

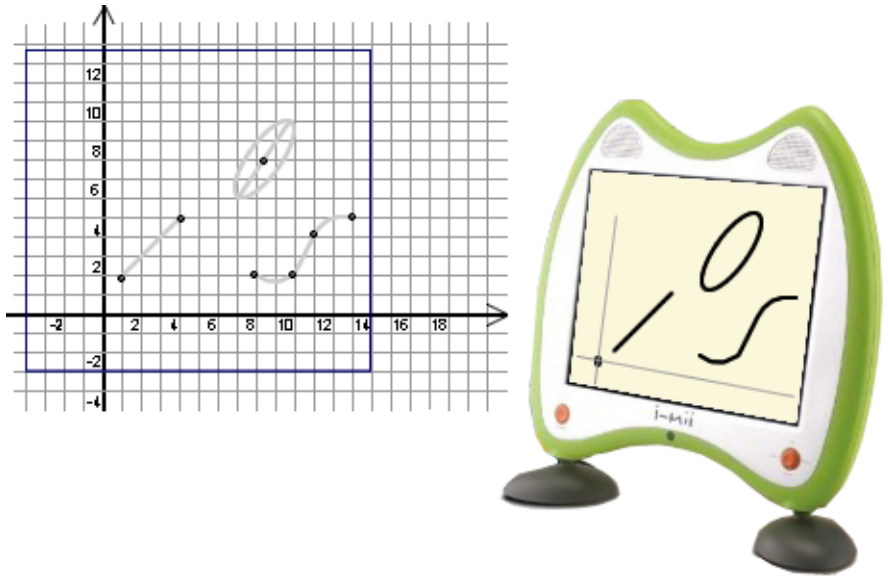


Рис.2.2. Перемещение центра координат

При отрисовке конкретного примитива “относительные единицы” будут пересчитаны в абсолютные (пикселы, сантиметры, дюймы и т.п.) в зависимости от коэффициента пересчета. Безусловно при пересчете будет соблюдаться пропорциональность.

Пример:

Укажем для эллипса следующие параметры в “относительных единицах”:

Координаты центра: **X=2 Y=2**
Радиус по оси: **RX=1 RY=2**
Угол наклона: **Angle=45**

При коэффициенте пересчета относительных единиц в пикселы, равным 3, получим следующие координаты в пикселах (рис. 2.3):

Координаты центра: **X=2 Y=2**
Радиус по оси: **RX=3 RY=6**
Угол наклона: **Angle=45**

2.3.

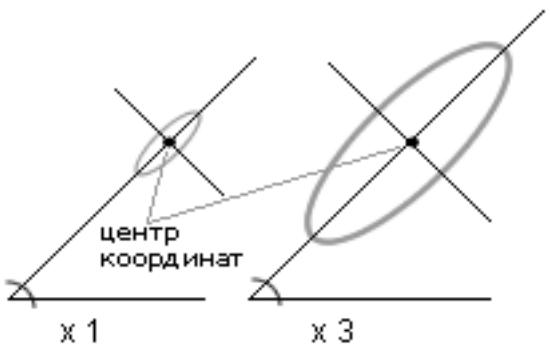


Рис.2.3. Масштабирование графического примитива

Масштабирование графического примитива выполняется относительно существующего центра координат и не приводит изменению его положения.

Подобная система “относительных единиц измерения” позволяет проводить различные операции с графическими примитивами (масштабирование, поворот и т.д.), не привязываясь к конкретным физическим единицам. Это позволяет задавать “относительные координаты” как положительными, так и отрицательными значениями (к примеру -20), не ограничивать степень “точности” задания координат и т.д.

Все преобразования и смещения, применяемые к графическим примитивам, выполняются относительно “центра координат”. К примеру, при повороте примитива (в процессе построения глифа) на 30°, осью будет выступать именно “центр координат”.

Глифы

Глиф (Glyph, рельеф) — это объект, содержащий информацию о форме символа. Основное предназначение глифа — описание некоторого графического примитива произвольной сложности, содержащего как линейные элементы (пути) так и полигональные (с заполнением или без).

Путь (path) — это цепочка (последовательность, вереница) последовательно отображаемых графических примитивов, оформленных в группу с общими параметрами отображения (линейный контур и т.д.) В составе одного глифа может находиться несколько “путей”, состоящих из различных графических примитивов.

Логически глиф похож на одноименную конструкцию, в описании структуры TTF-шрифта. Но, по сравнению с глифом шрифта, значительно расширено количество примитивов. Также в “пути” (path) допускается применение как линейных, так и полигональных объектов с различными параметрами их отображения (см. раздел “Путь” и визуализации графических примитивов на стр. 14).

Описание глифа может включать следующие операторы:

1. Графические примитивы
2. Способ обрисовки пути:
   - линейный контур (оператор outline) (рис. 2.12)
   - замкнутый контур с заполнением (оператор scanline) (рис. 2.12)
   - замкнутый контур с заполнением и окантовкой (оператор both) (рис. 2.12)
3. Позиция их прорисовки относительно друг друга (оператор move)

Все XML-описания глифов должны располагатся в разделе **vector_glyph** библиотеки символов.

Позиция курсора (cursor position) определяет координаты точки (в “относительных единицах”) с которой начнется отрисовка графического примитива. Понятие “позиции курсора” крайне важно для отрисовки пути, как единого целого. Т.е., при отображении пути, состоящего из линий, нет необходимости указывать координаты начальной и конечной точки т.к. координаты конечной точки предыдущей линии будут координатами начальной точки для следующей линии (в составе пути).

После отрисовки некоторых типов примитивов позиция курсора остается “неопределенной” (это справедливо, к примеру, для **эллипса**). Подобные фигуры также могут участвовать в построении единого пути. После отрисовки такой фигуры в составе пути необходимо явно указать позицию курсора (рис. 2.4), т.е. **переместить курсор** в указанную точку, используя оператор **move** (см. раздел Глифы на стр. 9).

2.4.

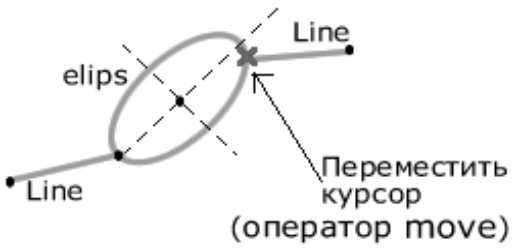


Рис.2.4. Перемещение курсора при построении разомкнутого пути (оператор outline)

Подобное отображение верно только для разомкнутого пути. При построении замкнутого контура, использование оператора **move** приведет к “замыканию” уже отрисованной части пути, а следующая часть пути будет отображаться как новый замкнутый контур.

Графические примитивы

Линии (оператор LINE)

2.5.

Рис.2.5. Линия

Проводит линию из текущей позиции курсора в точку с координатами (X,Y).

Пример:

Провести линию из (текущей позиции) в (1,-0.5)

```
<line x="1" y="-0.5"/>
```

Кривые Безье 2-го порядка (оператор BEZIER3)

2.6.

Рис.2.6. Кривая Безье 2-го порядка

Проводит кривую Безье, начиная из текущего положения курсора, через контрольную точку (Xb1,Yb1) в точку (X,Y).

Пример:

Провести кривую Безье 2-го порядка из (текущей позиции) через контрольную точку (23,12) в точку (12,1)

```
<bezier3 xb1="23" yb1="12" x="12" y="1"/>
```

Кривые Безье 3-го порядка (оператор BEZIER4)

2.7.

Рис.2.7. Кривая Безье 3-го порядка

Проводит кривую Безье, начиная из текущего положения курсора, через контрольные точки (Xb1,Yb1) и (Xb2,Yb2) в точку (X,Y).

Пример:

Провести кривую Безье 3-го порядка из текущей позиции курсора через 2-ве контрольные точки (4,-5) и (8,5) в точку (12,0)

```
<bezier4 xb1="4" yb1="-5" xb2="8" yb2="5" x="12" y="0"/>
```

Эллипс (оператор ELIPSE)

Отрисовывается как элипс с центром в точке `(x,y)`, радиусом (rx,ry) ось которого наклонена на угол `elipse_angle`.

Все углы наклона для фигур, основаных на елипсе (elipse, arc, sector) откладываются ***всегда*** *по часовой стрелке. Для отсчета угла наклона против часовой стрелки указывайте отрицательное значение.*

2.8.

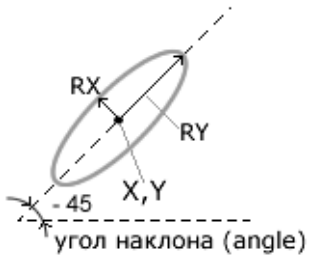


Рис.2.8. Эллипс

Для отрисовки эллипса необходимо указать:

- координаты центра x,y
- радиус по оси X и Y (rx,ry)
- угол поворота эллипса относительно центра — angle.

Положение курсора после отрисовки эллипса не определено! Поэтому (если есть необходимость продолжить отрисовку пути) следует вызвать функцию move, которая установит новое положение курсора.

Пример:

Эллипс с центром в точке (10,23), радиусом по координате X=25, Y=23
и углом поворота эллипса относительно центра -45° (против часовой стрелки).

```
<elipse x="10" y="23" rx="25" ry="23" angle="-45"/>
```

Дуга (оператор ARC)

Дуга проводится из текущего положения курсора, как часть эллипса с центром в (x,y) и радиусом (rx,ry) повернутого на угол elipse_angle. Начало дуги относительно нулевого угла эллипса повернуто по часовой стрелке на угол begin_angle, а длинна дуги (в градусах) определяется параметром end_angle.
Полная длинна дуги равна 360° и является положительным числом при откладывании угла по часовой стрелке.

2.9.

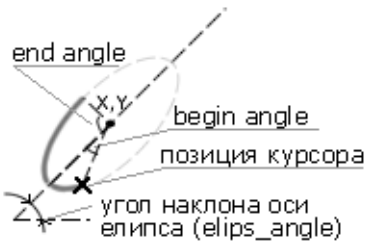


Рис.2.9. Дуга

Если begin_angle=0 и end_angle=90, тогда длина дуги равна четверти эллипса.

На рисунке угол elipse_angle откладывается против часовой стрелки.

Курсор после выполнения оператора **ARC** находится в конце дуги. Кроме того помимо дуги проводиться прямая линия из предыдущего положения курсора к начальной точке дуги.

Пример:

Отобразить дугу из точки “текущей позиции курсора” по эллипсу с параметрами:
- центр в точке (34,12);
- угол наклона оси — $-34°$ относительно центра (против часовой стрелки);
- начало дуги — $10°$;
- окончание дуги — $90°$.

```
<arc x="34" y="12" rx="25" ry="23" elipse_angle="-34"
 begin_angle="-10" end_angle="90"/>
```

Сектор (оператор SECTOR)

2.10.

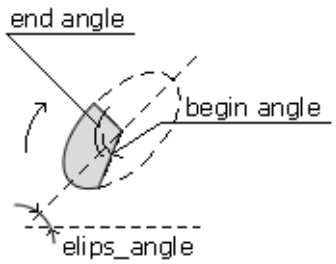


Рис.2.10. Сектор

Сектор отрисовуется по эллипсу с центром в x,y и радиусом rx,ry. Элипс повернут относительно центра на угол elips_angle. Начало дуги относительно нулевого угла эллипса повернуто по часовой стрелке на угол start_angle, окончание — на угол end_angle.
Отрисовка дуги выполняется по часовой стрелке.

Положение курсора после отрисовки эллипса не определено! Поэтому (если есть необходимость продолжить отрисовку пути) следует вызвать функцию move, которая установит новое положение курсора.

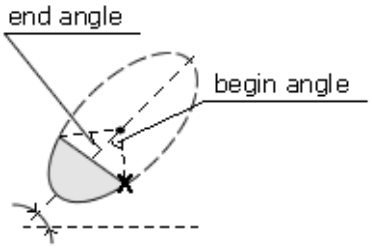
Пример:

Отобразим сектор эллипса (параметры эллипса: центр в точке (23,12), радиус (19,25), ось повернута на $35°$) начиная с $(-10)°$ относительно начала эллипса и заканчивая $45°$ относительно начала эллипса по часовой стрелке.

```
<sector x="23" y="12" rx="19" ry="25" elipse_angle="35"
 begin_angle="-10" end_angle="45"/>
```

Хорда (оператор CHORD)

2.11.



Хорда отрисовуется по эллипсу с центром в x,y и радиусом rx,ry. Эллипс повернут относительно центра на угол angle. Начало дуги относительно нулевого угла эллипса повернуто по часовой стрелке на угол start_angle, окончание на угол end_angle.
Отрисовка дуги выполняется по часовой стрелке.

Рис.2.11. Хорда

Положение курсора после отрисовки эллипса не определено! Поэтому (если есть необходимость продолжить отрисовку пути) следует вызвать функцию move, которая установит новое положение курсора.

Пример:

Отобразить сектор эллипса (параметры эллипса: центр в точке (23,12), радиус (19,15), ось повернута на $34°$) начиная с $(-10)°$ относительно начала эллипса по часовой стрелке и заканчивая $45°$ относительно начала эллипса по часовой стрелке.

```
<chord x="23" y="12" rx="19" ry="25" elipse_angle="34" begin_angle="-10"
  end_angle="-45"/>
```

“Путь” и визуализации графических примитивов

Как ранее упоминалось, один глиф может содержать произвольное количество путей. В свою очередь путь определяет как именно будет отображаться геометрическая фигура, построенная из последовательно отрисованных примитивов.

Для определения пути используются операторы:

OUTLINE — отображать как разомкнутый контур
SCANLINE — отображать как замкнутый контур с заполнением
BOTH — отображать как замкнутый контур с заполнением и окантовкой

Взглянем на пример описания “стрелки”, состоящей из двух линий (рис. 2.12):

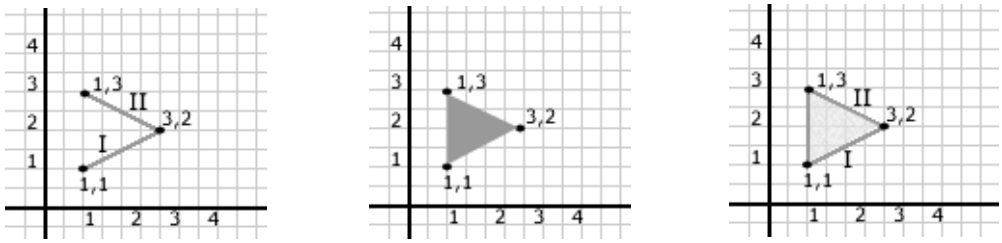
```
<outline x="1" y="1">
  <line x="3" y="2"/>
  <line x="1" y="3"/>
</outline>
```

Параметры X и Y определяют позицию курсора, с которой начинается отрисовка “пути”.

Если в описании стрелки заменить OUTLINE на SCANLINE то контур замкнется (рис. 2.12) и мы получим стрелку с заполнением (треугольник).

Для отображения окантовки вокруг стрелки с заполнением используйте оператор BOTH (рис. 2.12).

2.12.



Оператор **OUTLINE** Оператор **SCANLINE** Оператор **BOTH**

Рис.2.12. Операторы отображения контуров

При формировании пути необходимо помнить:

1. Существуют примитивы (эллипс, хорда и сектор), которые не начинаются с предыдущей точки, и их конечная точка не определена. Для них необходимо указывать все координаты и размеры.

2. Оператор **move** перемещает положение конечной точки в указанное место. Этот оператор необходимо использовать каждый раз после отображения эллипса, хорды или сектора.

Пример пути с использованием эллипса:

```
<outline x="1 y="1">
  <line x="3 y="2"/>
  <line x="1" y="3"/>
  <elipse x="7" y="7" rx="4" ry="4" angle="45"/>
  <move x="4" y="5"/>
  <line x="5" y="7"/>
</outline>
```

3. Операторы **move**, **elipse**, **chord**, **sector** при прорисовке пути вызывают замыкание предыдущей области в режимах **scanline**, **both**.

   К примеру, если мы провели две лини (смотри пред. пример), а потом указали оператор **elipse** — мы получим закрашенный треугольник, где две линии составляют ребра. Третье ребро образуется проведением линии из конца второй в начало первой.

4. Различные отрезки в рамках одного пути, наложенные друг на друга, вызовут создание “дырки” при отображении в режимах **both** или **scanline** (отрезок это часть пути до следующего **move**, **elipse**, **chord**, **sector**).

2.13.

Рис.2.13. Пример пути, отображающего кольцо

Пример пути, отображающего кольцо с внешним радиусом 20, внутренним радиусом 10:

```
<scanline x="100" y="100">
  <elipse x="0" y="0" rx="20" ry="20" angle="0"/>
  <elipse x="0" y="0" rx="10" ry="10" angle="0"/>
</scanline>
```

Заметим, что координаты начала пути в данном случае указаны лишь для корректности. Они не оказывают никакого влияния на отображение пути.

Пример-описание сложного глифа:

```
<glyph caption="glTest">
  <outline x="0" y="0">
    <bezier4 xb1="4" yb1="-5" xb2="8" yb2="5" x="12" y="0">
  </outline>
  <scanline x="10" y="10.5">
    <line x="23" y="20.5">
    <line x="24" y="24">
    <elipse x="10" y="23" rx="25" ry="23" angle="56">
    <move x="0" y="-2">
    <line x="12" y="10">
  </scanline>
</glyph>
```

Палитра цветов

Отдельный раздел библиотеки (оператор **palette**), содержащий внутреннюю палитру цветов — индексированный перечень цветов в RGB формате (оператор **palette_item**). Далее, в фигуре (shape) можно указать не RGB-описание цвета, а его индекс. Это очень удобно для формального определения цветов — “цвет фона по умолчанию”, “не активный” и т.д., а в фигуре или символе указывается только индекс.

Параметры:

- **index -** индекс цвета в палитре.
  Произвольный числовой номер, уникальный в пределах пилитры.

- **color -** текстовое представление цвета в RGB формате, принятое в стандарте HTML.

Правило формирования: #[прозрачность][R][G][B]
К примеру **#FFFF0000** указывает на абсолютно “не прозрачный“ красный цвет.

* **caption** - текстовый коментарий.

Пример:

Раздел палитры цветов, содержащий три индексных цвета (с номерами 1,2 и 5)

```
<palette name="palette">
  <palette_item index="1" color="#FFFF0000"
   caption="Красный"/>
  <palette_item index="2" color="#FF00FF00"
   caption="Зеленый"/>
  <palette_item index="5" color="#FF0000FF"
   caption="Синий"/>
</palette>
```

Фигура

Фигура (shape) — состоит из глифов (или других шейпов) и дополнительных параметров, характеризующих расположение каждого глифа, масштабирование, угол поворота и т.д.

Параметры отображения фигуры

В пределах одного шейпа для отрисовки глифов **всегда** используются **общие** параметры отображения (операторы **pencolor**, **brushcolor** и т.д.)
Если для нескольких глифоф необходимо задать другие параметры отрисовки — включите их в отдельную фигуру и добавьте ее в состав общей фигуры. При этом параметры отображения общей фигуры будут применяться только к глифам, непосредственно включенным в ее состав.

Парамет	Описание	Значение по умолчанию	Пример
pencolor	цвет контура в RGB-формате	#000000	pencolor="#FF0000"
brushcolor	цвет заливки в RGB-формате	#FFFFFF	brushcolor="#FFFFFF"
penwidth	толщина контура	1.0	penwidth="1.0"
penscale	масштаб толщины кисти	0.0	penscale="1.0"
penpattern	индекс внутренней палитры для контура	-1	penpattern="3"
brushpattern	индекс внутренней палитры для заливки	-1	brushpattern="14"

penindex	индекс внешней палитры для контура	0	penindex="0"
brushindex	индекс внешней палитры для заливки	1	brushindex="1"

Перо (pen) — определяет параметры виртуального “пера”, используемого для отрисовки произвольных контуров фигур.

Кисть (brush) — определяет параметры виртуальной “кисти”, применяемой при заполнении (“заливке“) фигур и произвольных площадей.

Рассмотрим более детально некоторые параметры:

pencolor, penpalette, penindex — определяют цвет пера.
Одновременно можно указывать три параметра, но при визуализации будет использован только один. Выбор применяемого цвета выполняется в порядке замещения:

pencolor → penpalette → penindex

penscale — определяет относительное изменение размера кисти.
К примеру, толщина линии (в относительных единицах) равна X (параметр **penwidth**). Ее толщина будет равна при отрисовке: толщина = X * (**penscale** * W),
где W — толщина, указанная пользователем.

penscale определяет не только масштаб толщины кисти, но и масштабируемость кисти вообще. Если параметр **penscale** не указан — кисть не будет масштабироваться, независимо от того изменил пользователь толщину (W) или нет. Ее толщина будет равна при отрисовке всегда : толщина = X.

Если какой либо из параметров не указан — будет использоваться значение “по умолчанию”

Пример простой фигуры, содержащей один глиф:

```
<shape caption="TestGly" pencolor="#dd648e"
 brushcolor="#d2c59d" penwidth="1" penindex="0" brushindex="1">
  <glyph bind_name="RorOid"/>
</shape>
```

Так-же необходимо заметить, что **penindex, brushindex** указывают на индекс в списке цветов, формируемом прикладной програмой.

Рекомендуем, при формировании описания фигуры, ожидать что:
0-й индекс — цвет пера (penindex=0)
1-й индекс — цвет кисти (brushindex=1)

Преобразование глифов и фигур

При добавлении глифа (или фигуры) его центр координат совмещается с центром координат фигуры, в которую он добавляется. Так же сохраняются все его относительные

размеры (рис. 2.14). Если необходимо переместить глиф или изменить масштаб — используйте операторы преобразования.

Оператор	Описание	Пример
offset	Сместить центр координат глифа (или фигуры). Указываются смещения по оси X и Y.	<offset x="1.5" y="3"\>
rotate	Повернуть (относительно центра координат) глиф на указанный угол (по часовой стрелке).	<rotate angle="15"\>
scale	Масштабировать. Указывается отдельно коэффициент масштабирования для оси X и Y.	<scale x="5" y="3"\>

2.14.

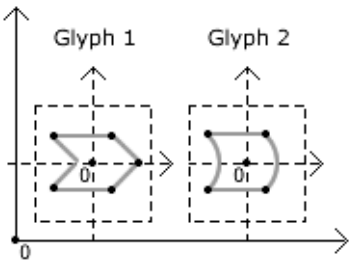


Рис.2.14. Фигура, содержащая два глифа

Центр координат каждой из фигур (Gliph1 и Gliph2) перемещен относительно центра координат результирующей фигуры.

Для применения операторов нужно сгруппировать глифы (оператор **shape_group**) и указать (для группы) все необходимые преобразования. Возможно выполнение произвольного количества последовательных преобразований. Преобразования выполняются в порядке их перечисления в составе группы. Весь перечень преобразований выполняется последовательно над каждым глифом, включенным в состав группы.
Если в выполнении преобразований нет необходимости — группировку выполнять не обязательно.
Допускается формирование вложеных груп, содержащих в себе другую групу. При этом строго соблюдается очередность выполнения преобразрваний.

Пример:

```
<shape caption="TestGly" pencolor="#dd648e"
 brushcolor="#d2c59d" penindex="0" brushindex="1">
  <glyph bind_name="GryWoTo"\>
  <shape_group>
    <offset x="1.5" y="3"\>
    <rotate angle="15"\>
    <scale x="5 y="3"\>
    <offset x="0.5" y="0"\>
    <glyph bind_name="GlyZX"\>
  <shape_group\>
  <shape bind_name="GlyWW">
```

```
  <shape_group>
    <offset x="10" y="5"\>
    <glyph bind_name="GlyROR"\>
    <shape_group>
      <offset x="15" y="5"\>
      <glyph bind_name="GlyROR"\>
    <shape_group\>
    <glyph bind_name="GlyQOT"\>
  <shape_group\>
  <glyph bind_name="GlyELOY"\>
</shape>
```

Символы

Символы (symbol) — это сложная структура, состоящая из фигуры (или фигур) и описания особенностей ее визуализации в зависимости от типов графических объектов (точки, линии, замкнутые области).

Символы разделяются на:

* точечные — применяются при отображении точечных объектов;
* линейные — применяются при отображении произвольных линейных объектов (линий, полилиний) и оконтуривания площадных объектов;
* площадные — применяются при заливке произвольных площадных объектов (полигонов, эллипсов, сложных контуров замкнутых).

При создании символа ему указывается только один тип. Он не может быть применен для отображения объектов другого типа.

Каждый тип символа (его XML-описание) хранится в отдельном разделе библиотеки:

* **vector_point** — точечные символы
* **vector_line** — линейные символы
* **vector_poly** — площадные символы

В библиотеке может быть только по одному разделу каждого типа.

Точечный символ

Содержит в себе только одну фигуру. Дополнительные параметры отсутствуют.
При визуализации, центр координат фигуры, составляющей символ, помещается в точку с указанными координатами.

Пример раздела библиотеки точечных символов:

```
<vector_point name="point_symbols">
  <!-- точечный символ PointTest -->
  <point caption="PointTest" bind_name="shape1"/>
        .....
</vector_point>
```

Линейный символ

Самый сложный из символов, как по построению, так и по отрисовке.

В параметрах указывается:

* идентификатор фигуры, используемой для отрисовки “начала линии” (параметр **start_bind_name**)
* идентификатор фигуры, используемой для отрисовки “окончания линии” (параметр **end_bind_name**)
* размер типового участка линии (параметры **interval_x** и **interval_y**)

Типовым участком (“кистью”) будем считать некоторую область, в результате последовательного повторения которой вдоль оси линии будет получено необходимое визуальное представление линии (рис. 2.15).

2.15.

Рис.2.15. Типовый участок “кисть”

Заполнение линии выполняется последовательно . Ось линии совпадает с осью “типового участка”.

Линейный символ (оператор **line**) может содержать несколько фигур, с разными правилами визуализации:

* растровая матрица (оператор **bitmap**)
  Перед визуализацией фигура (shape) будет пересчитана в растровое представление (“кисть”) и далее уже именно “матрицей” будет производиться заполнение (заливка, отрисовка) конкретного геометрического объекта.

* векторный знак (оператор **vector**)
  Фигура, указанная как векторный знак, будет полностью отрисовываться при визуализации очередного типового участка (как векторные элементы) при отображении геометрического объекта.
  Процедура отрисовки векторных знаков довольно ресурсоемкая. Ее следует избегать. Но если необходимо отобразить элементы символа (фигуры), деформирующиеся при операциях с растровой матрицей (повороты при отображении вдоль линии) — укажите их как векторный знак.

Пример раздела библиотеки линейных символов:

```
<vector_line name="line_symbols">
  <!-- линейный символ WotCot -->
  <line caption="WotCot" start_bind_name="shape1"
  end_bind_name="shape1" interval_x="7" interval_y="3">
    <bitmap bind_name="osnova"/>
    <vector bind_name="rectang" offset="0"/>
    <vector bind_name="elips" offset="5"/>
  </line>
        .....
```

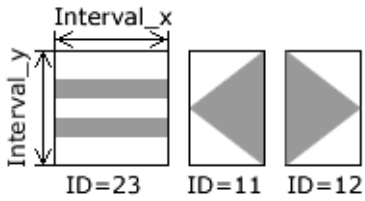
```
</vector_line>
```

Рассмотрим последовательность заполнения линии символом, описание которого приведено в примере.

Вначале определим:

* пусть фигуры с номерами 23, 11 и 12 выглядят, как представлено на рисунке 2.16;
* последовательность отрисовки будет соответствовать схеме, изображенной на рисунке 2.17;
* вид линии представлен на рисунке 2.18.

2.16.



ID = 23 — растровая матрица;
ID = 11 — векторный знак;
ID = 12 — векторный знак.

Рис.2.16. Отображение фигур

2.17.

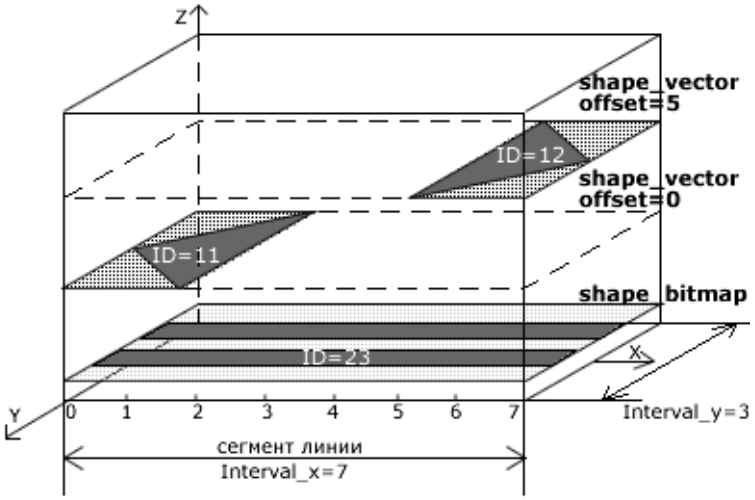


Рис.2.17. Порядок отрисовки фигур (визуализация “типового участка”)

2.18.

Если типовый участок находится на повороте оси линии — то выполняется его изгиб:

Рис.2.18. Отображение линии

Последовательность отрисовки

1. Масштабируем все входящие в состав линии фигуры (начальный и конечный маркеры, все фигуры заливки).

К примеру, предположим, что при отображении линии в API **XXXX** указано — толщина линии равна 2. Это означает, что необходимо увеличить начальный (параметр **start_bind_name**) и конечный маркеры (параметр **end_bind_name**), все фигуры заливки (операторы **bitmap** и **vector**) и интервал (параметры **interval_x** и **interval_y**) в 2 раза;

2. В составе символа первым указана растровая матрица (оператор **bitmap**). Создаем “битмап” (растровую матрицу) размером **interval_x** на **interval_y** (размер типового участка линии). В него отображаем фигуру (ID=23);

3. Фигуры 11 и 12 — векторные знаки. У первой фигуры (ID=11) смещение относительно начала типового участка равно нулю (offset=0, рис. 2.17). У второй с ID=12 — равно 5 (offset=5 рис. 2.17);

4. Начинаем заполнение линии “типовыми участками”. При изгибе оси линии поворачивается и соответствующая часть типового участка. При этом для векторных знаков пересчитываются координаты опорной точки, а для растровой матрицы — производится операция отсечения и поворота соответствующего участка (рис. 2.18);

5. Если длина линии не кратна длине типового участка, то последний (“не полный” участок) будет отрисовываться по следующим правилам:
   - векторный знак отрисовывается, только если его опорная точка не выходит за пределы линии;
   - растровый знак “обрезается” по границе линии.

Площадной символ

Содержит в себе только одну фигуру и параметры **interval_x** и **interval_y**, определяющие размер типового участка (“кисти”), используемого для заполнения указанной области.

Пример раздела библиотеки площадных символов:

```
<vector_poly name="poly_symbols">
  <!-- линейный символ PolyRolTol -->
  <poly caption="PolyRolTol" interval_x="1"
  interval_y="1" bind_name="shape1"/>
        .....
</vector_poly>
```

Особенности построения сложных символов

Каждый символ включает в себя как минимум одну фигуру. В свою очередь, фигура может содержать произвольное количество, как фигур, так и глифов. Подобная иерархическая структура позволяет создавать символы **произвольной сложности.**
Необходимо с вниманием относиться к построению и использованию глифов и фигур. Не желательно создавать сложные иерархические структуры (рис. 2.19).
Так же (если это возможно), необходимо избегать использования лишних векторных составляющих символа. Это позволит ускорить визуализацию.

2.19.

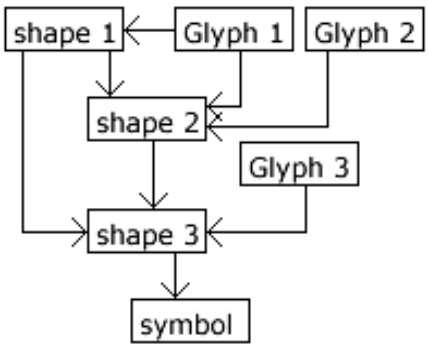


Рис.2.19. Сложная иерархическая структура

Излишне сложная структура символа не удобна как при отображении, так и при последующем редактировании.