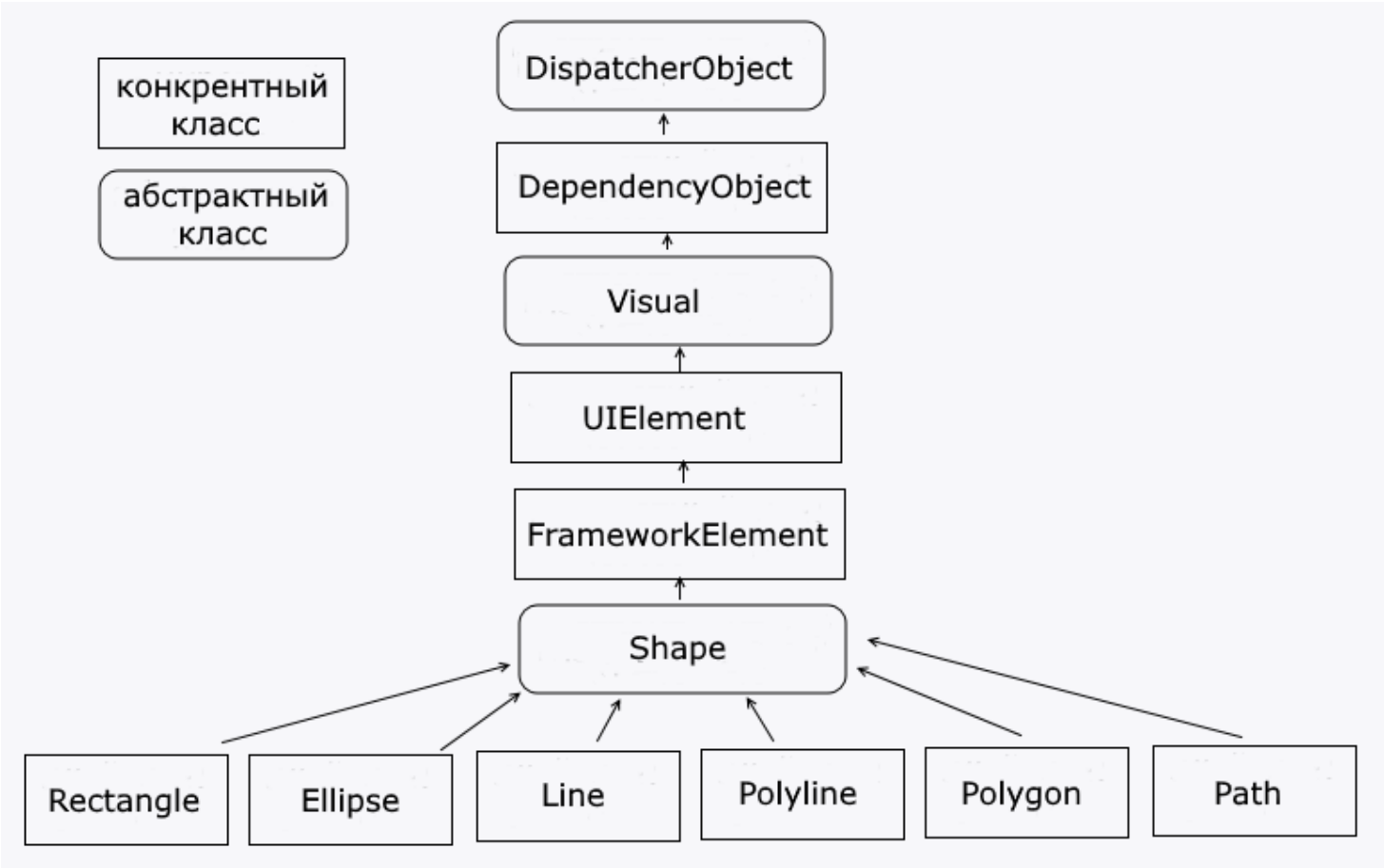
Работа с графикой

Фигуры

Одним из способов построения двухмерной графики в окне - это использование фигур. Фигуры фактически являются обычными элементами как например кнопка или текстовое поле. К фигурам относят такие элементы как **Polygon** (Многоугольник), **Ellipse** (овал), **Rectangle** (прямоугольник), **Line** (обычная линия), **Polyline** (несколько связанных линий). Все они наследуются от абстрактного базового класса System.Windows.Shapes.Shape:



От базового класса они наследуют ряд общих свойств:

- **Fill** заполняет фон фигуры с помощью кисти - аналогичен свойству Background у прочих элементов
- **Stroke** задает кисть, которая отрисовывает границу фигуры - аналогичен свойству BorderBrush у прочих элементов

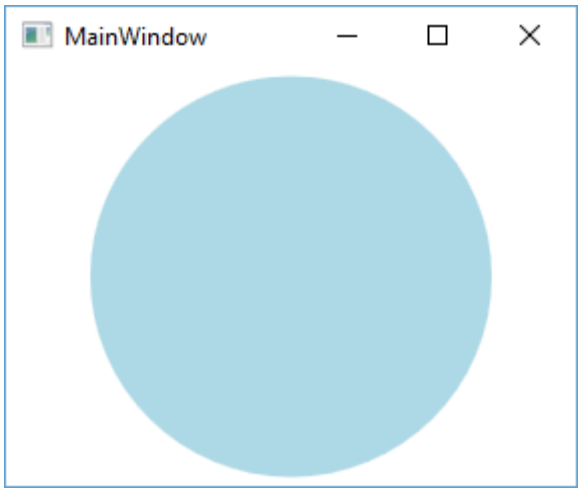
- **StrokeThikness** задает толщину границы фигуры - аналогичен свойству BorderThikness у прочих элементов
- **StrokeStartLineCap** и StrokeEndLineCap задают для незамкнутых фигур (Line) контур в начале и в конце линии соответственно
- **StrokeDashArray** задает границу фигуры в виде штриховки, создавая эффект пунктира
- **StrokeDashOffset** задает расстояние до начала штриха
- **StrokeDashCap** задает форму штрихов

Ellipse

Ellipse представляет овал:

```
<Ellipse Fill="LightBlue" Width="200" Height="200" />
```

При одинаковой ширине и высоту получается круг:

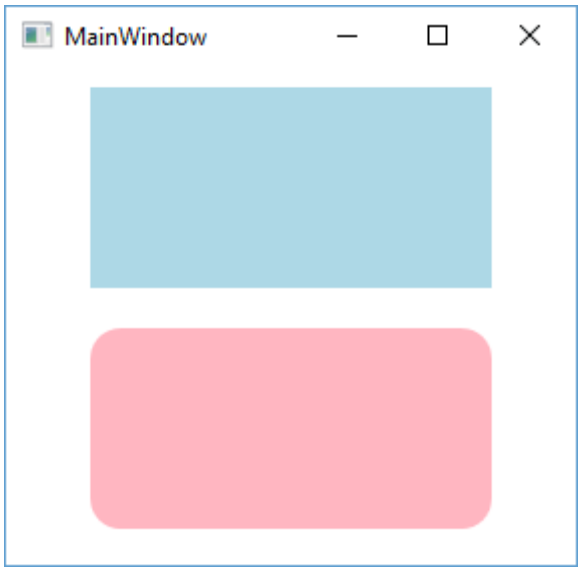


Rectangle

Rectangle представляет прямоугольник:

```
<StackPanel Background="White">
    <Rectangle Fill="LightBlue" Width="200" Height="100" Margin="10" />
    <Rectangle Fill="LightPink" Width="200" Height="100" RadiusX="15" RadiusY="15" Margin="10" /
</StackPanel>
```

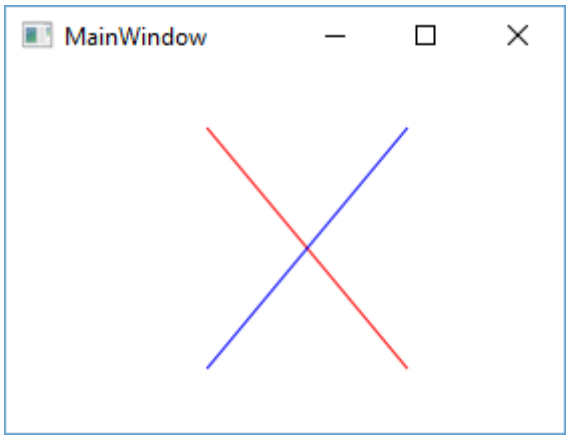
С помощью свойств RadiusX и RadiusY можно округлить углы прямоугольника:



Line

Line представляет простую линию. Для создания линии надо указать координаты в ее свойствах X1, Y1, X2 и Y2. При этом надо учитывать, что началом координатной системы является верхний левый угол:

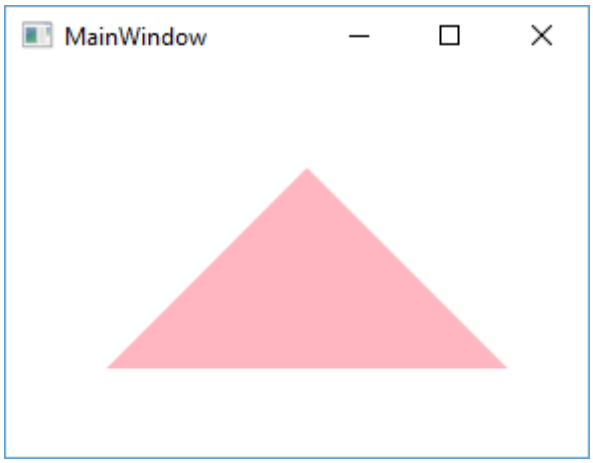
```
<Line X1="100" Y1="30" X2="200" Y2="150" Stroke="Red" />
<Line X1="100" Y1="150" X2="200" Y2="30" Stroke="Blue" />
```



Polygon

Polygon представляет многоугольник. С помощью коллекции Points элемент устанавливает набор точек - объектов типа Point, которые последовательно соединяются линиями, причем последня точка соединяется с первой:

```
<Polygon Fill="LightPink" Points="50, 150, 150, 50, 250, 150" />
```

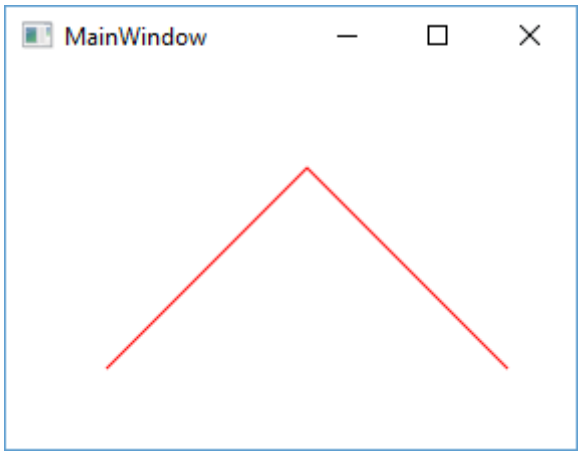


В данном случае у нас три точки (50, 150), (150, 50) и (250, 150), которые образуют треугольник.

Polyline

Polyline представляет набор точек, соединенных линиями. В этом плане данный элемент похож на Polygon за тем исключением, что первая и последняя точка не соединяются:

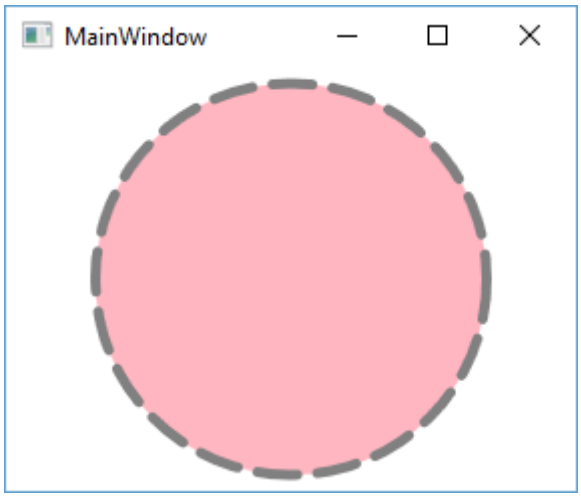
```
<Polyline Stroke="Red" Points="50, 150, 150, 50, 250, 150" />
```



Настройка контура

С помощью ряда свойств мы можем настроить отображение контура. Например:

```
<Ellipse Width="200" Height="200" Fill="LightPink"
        StrokeThickness="5" StrokeDashArray="4 2"
        Stroke="Gray" StrokeDashCap="Round" />
```



Цвет самого контура определяется с помощью свойства Stroke, а его толщина - с помощью StrokeThickness.

StrokeDashArray устанавливает длину штрихов вместе с отступами. Например, StrokeDashArray="4 2" устанавливает длину штриха в 4 единицы, а последующего отступа в 2 единицы. И эти значения будут повторяться по всему контуру. При другой установке, например, StrokeDashArray="1 2 3" уже задается два штриха. Первый штрих имеет длину в 1 единицу, а второй - в 3 единицы и между ними расстояие в 2 единицы. И так вы можем настроить количество штрихов и расстояний между ними.

StrokeDashCap задает форму на концах штрихов и может принимать следующие значения:

- Flat: стандартные штрихи с плоскими окончаниями
- Square: штрихи с прямоугольными окончаниями
- Round: штрихи с округлыми окончаниями
- Triangle: штрихи с окончаниями в виде треугольников

Программное рисование

Создание фигур программным образом осуществляется так же, как и создаются и добавляются все остальные элементы:

```
Ellipse el = new Ellipse();
el.Width = 50;
el.Height = 50;
el.VerticalAlignment = VerticalAlignment.Top;
el.Fill = Brushes.Green;
el.Stroke = Brushes.Red;
el.StrokeThickness = 3;
grid1.Children.Add(el);
```

Нарисуем, к примеру, координатную плоскость:

```
Line vertL =new Line();
vertL.X1 = 10;
vertL.Y1 = 150;
vertL.X2 = 10;
vertL.Y2 = 10;
vertL.Stroke = Brushes.Black;
grid1.Children.Add(vertL);
Line horL =new Line();
horL.X1 = 10;
horL.X2 = 150;
horL.Y1 = 150;
horL.Y2 = 150;
horL.Stroke = Brushes.Black;
grid1.Children.Add(horL);
for(byte i = 2;i< 14;i++)
{
    Line a =new Line();
    a.X1 = i * 10;
    a.X2 = i * 10;
    a.Y1 = 155;
    a.Y2 = 145;
    a.Stroke = Brushes.Black;
    grid1.Children.Add(a);
}
for(byte i = 2;i< 14;i++)
{
    Line a =new Line();
    a.X1 = 5;
    a.X2 = 15;
    a.Y1 = i * 10;
    a.Y2 = i * 10;
    a.Stroke = Brushes.Black;
    grid1.Children.Add(a);
}
Polyline vertArr =new Polyline();
vertArr.Points = new PointCollection();
vertArr.Points.Add(new Point(5, 15));
vertArr.Points.Add(new Point(10, 10));
vertArr.Points.Add(new Point(15, 15));
vertArr.Stroke = Brushes.Black;
grid1.Children.Add(vertArr);
Polyline horArr =new Polyline();
horArr.Points = new PointCollection();
horArr.Points.Add(new Point(145, 145));
horArr.Points.Add(new Point(150, 150));
horArr.Points.Add(new Point(145, 155));
horArr.Stroke = Brushes.Black;
grid1.Children.Add(horArr);
```

Пути и геометрии

Фигуры удобны для создания самых простейших рисунков, дизайна, однако что-то более сложное и комплексное с их помощью сделать труднее. Поэтому для этих целей применяется класс **Path**, который представляет геометрический путь. Он также, как и фигуры, наследуется от класса Shape, но может заключать в себе совокупность объединенных фигур. Класс Path имеет свойство **Data**, которое определяет объект Geometry - геометрический объект для отрисовки. Этот объект задает фигуру или совокупность фигур для отрисовки.

Класс Geometry - абстрактный, поэтому в качестве объекта используется один из производных классов:

- **LineGeometry** представляет линию, эквивалент фигуры Line
- **RectangleGeometry** представляет прямоугольник, эквивалент фигуры Rectangle
- **EllipseGeometry** представляет эллипс, эквивалент фигуры Ellipse
- **PathGeometry** представляет путь, образующий сложную геометрическую фигуру из простейших фигур
- **GeometryGroup** создает фигуру, состоящую из нескольких объектов Geometry
- **CombinedGeometry** создает фигуру, состоящую из двух объектов Geometry
- **StreamGeometry** - специальный объект Geometry, предназначенный для сохранения всего геометрического пути в памяти

Например, использование LineGeometry:

```
<Path Stroke="Blue">
    <Path.Data>
        <LineGeometry StartPoint="100,30" EndPoint="200,130" />
    </Path.Data>
</Path>
```

будет аналогично следующему объекту Line:

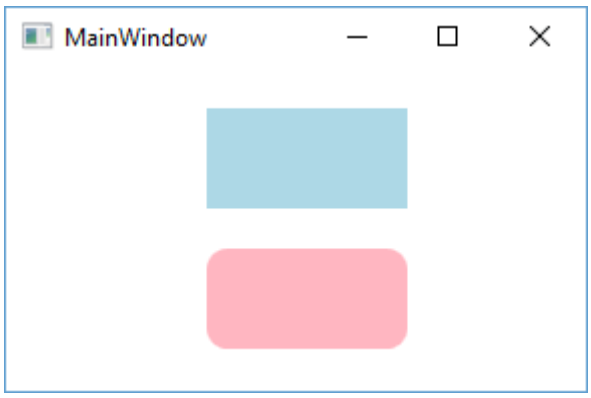
```
<Line X1="100" Y1="30" X2="200" Y2="130" Stroke="Blue" />
```

Свойства StartPoint и EndPoint задают начальную и конечную точки линии.

RectangleGeometry:

```
<StackPanel>
    <Path Fill="LightBlue">
        <Path.Data>
            <RectangleGeometry Rect="100,20 100,50" />
        </Path.Data>
    </Path>
    <Path Fill="LightPink">
        <Path.Data>
            <RectangleGeometry Rect="100,20 100,50" RadiusX="10" RadiusY="10" />
        </Path.Data>
    </Path>
</StackPanel>
```

Свойство Rect задает параметры прямоугольника в формате "координата X, координата Y ширина, высота". Также с помощью свойств RadiusX и RadiusY можно задать радиус скругления углов прямоугольника.



EllipseGeometry:

```
<Path Fill="LightPink" Stroke="LightBlue">
    <Path.Data>
        <EllipseGeometry RadiusX="50" RadiusY="25" Center="120,70" />
    </Path.Data>
</Path>
```

Свойство Center устанавливает цетр овала, а свойста RadiusX и RadiusY - радиусы.

GeometryGroup объединяет несколько геометрий:

```
<Path Fill="LightPink" Stroke="LightBlue">
    <Path.Data>
        <GeometryGroup  FillRule="Nonzero">
            <LineGeometry StartPoint="10,10" EndPoint="220,10" />
            <EllipseGeometry Center="100,100" RadiusX="50" RadiusY="40" />
            <RectangleGeometry Rect="120,100 80,20" RadiusX="5" RadiusY="5" />
        </GeometryGroup>
    </Path.Data>
</Path>
```

Объект GeometryGroup устанавливает свойство **FillRule**. Если оно равно **EvenOdd** (значение по умолчанию), то перекрывающиеся поверхности двух геометрий являются прозрачными. А при значении **FillRule="Nonzero"** (как в данном случае), перекрывающиеся поверхности геометрий будут окрашены также, как и остальные части пути.

CombinedGeometry

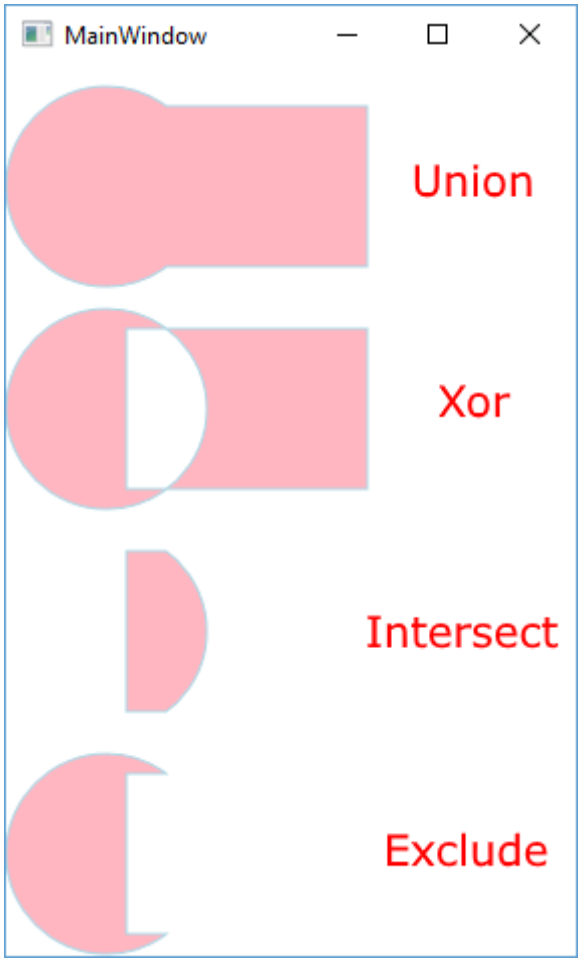
CombinedGeometry состоит из двух геометрий. В этом он похож на GeometryGroup, который также может объединять две геометрии. Однако между ними есть различия. Отличие состоит в том, что объект CombinedGeometry имеет свойство GeometryCombinedMode, которое указывает модель перекрытия двух геометрий:

- **Union**: фигура включает обе геометрии
- **Intersect**: фигура включает область, которая одновременно принадлежит обеим геометриям
- **Xor**: фигура включает только непересекающие области геометрий
- **Exclude**: фигура включает первую геометрию с исключением тех областей, которые принадлежат также и второй геометрии

Применим все способы:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Path Fill="LightPink" Stroke="LightBlue">
        <Path.Data>
            <CombinedGeometry GeometryCombineMode="Union">
                <CombinedGeometry.Geometry1>
                    <EllipseGeometry Center="50,60" RadiusX="50" RadiusY="50" />
                </CombinedGeometry.Geometry1>
                <CombinedGeometry.Geometry2>
                    <RectangleGeometry Rect="60, 20 120,80" />
                </CombinedGeometry.Geometry2>
            </CombinedGeometry>
        </Path.Data>
    </Path>
    <Path Grid.Row="1" Fill="LightPink" Stroke="LightBlue">
        <Path.Data>
            <CombinedGeometry GeometryCombineMode="Xor">
                <CombinedGeometry.Geometry1>
                    <EllipseGeometry Center="50,60" RadiusX="50" RadiusY="50" />
                </CombinedGeometry.Geometry1>
                <CombinedGeometry.Geometry2>
                    <RectangleGeometry Rect="60, 20 120,80" />
                </CombinedGeometry.Geometry2>
            </CombinedGeometry>
        </Path.Data>
    </Path>
    <Path Grid.Row="2" Fill="LightPink" Stroke="LightBlue">
        <Path.Data>
            <CombinedGeometry GeometryCombineMode="Intersect">
                <CombinedGeometry.Geometry1>
                    <EllipseGeometry Center="50,60" RadiusX="50" RadiusY="50" />
                </CombinedGeometry.Geometry1>
                <CombinedGeometry.Geometry2>
                    <RectangleGeometry Rect="60, 20 120,80" />
                </CombinedGeometry.Geometry2>
            </CombinedGeometry>
        </Path.Data>
    </Path>
    <Path Grid.Row="3" Fill="LightPink" Stroke="LightBlue">
        <Path.Data>
            <CombinedGeometry GeometryCombineMode="Exclude">
                <CombinedGeometry.Geometry1>
                    <EllipseGeometry Center="50,60" RadiusX="50" RadiusY="50" />
                </CombinedGeometry.Geometry1>
                <CombinedGeometry.Geometry2>
```

```
                    <RectangleGeometry Rect="60, 20 120,80" />
                </CombinedGeometry.Geometry2>
            </CombinedGeometry>
        </Path.Data>
    </Path>
</Grid>
```



PathGeometry

PathGeometry позволяет создавать более сложные по характеру геометрии. **PathGeometry** содержит один или несколько компонентов **PathFigure**. Объект **PathFigure** в свою очередь формируется из сегментов. Все сегменты наследуются от класса **PathSegment** и бывают нескольких видов:

- **LineSegment** задает отрезок прямой линии между двумя точками
- **ArcSegment** задает дугу
- **BezierSegment** задает кривую Безье
- **QuadraticBezierSegment** задает квадратичную кривую Безье
- **PolyLineSegment** задает сегмент из нескольких линий
- **PolyBezierSegment** задает сегмент из нескольких кривых Безье

- **PolyQuadraticBezierSegment** задает сегмент из нескольких квадратичных кривых Безье

Эти сегменты составляют свойство **Segment** объекта PathFigure. Кроме того, PathFigure имеет еще несколько важных свойств:

- **StartPoint** - точка начала первой фигуры
- **IsClosed** - если значение равно true, то первая и последняя точки (если они не совпадают) соединяются
- **IsFilled** - если значение равно true, то площадь внутри пути окрашивается кистью, заданной свойством Fill у объекта Path

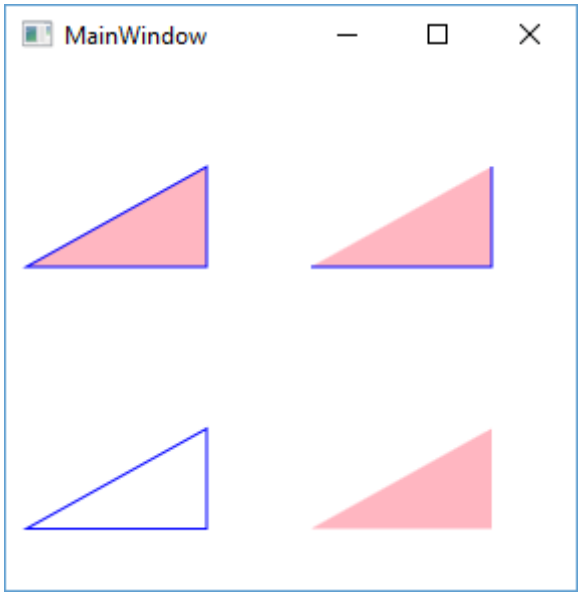
Линии

Создание линий с помощью PathGeometry:

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>

    <Path Fill="LightPink" Stroke="Blue">
        <Path.Data>
            <PathGeometry>
                <PathFigure IsClosed="True" StartPoint="10,100">
                    <LineSegment Point="100,100" />
                    <LineSegment Point="100,50" />
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
    <Path Grid.Column="1" Fill="LightPink" Stroke="Blue">
        <Path.Data>
            <PathGeometry>
                <PathFigure IsClosed="False" StartPoint="10,100">
                    <LineSegment Point="100,100" />
                    <LineSegment Point="100,50" />
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
    <Path Grid.Row="1"  Stroke="Blue">
        <Path.Data>
            <PathGeometry>
                <PathFigure IsClosed="True" StartPoint="10,100">
                    <LineSegment Point="100,100" />
                    <LineSegment Point="100,50" />
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
    <Path Grid.Row="1" Grid.Column="1"  Fill="LightPink">
        <Path.Data>
            <PathGeometry>
                <PathFigure StartPoint="10,100">
                    <LineSegment Point="100,100" />
                    <LineSegment Point="100,50" />
                </PathFigure>
            </PathGeometry>
        </Path.Data>
```

```
    </Path>
</Grid>
```

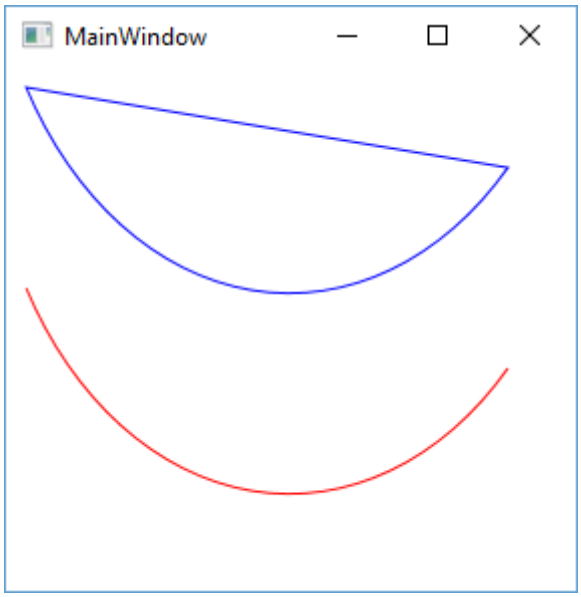


Управляя свойством IsClosed мы можем получить как замкнутый, так и незамкнутый контур. А при применении свойства Fill в элементе Path содержимое контура заполняется цветом. Таким образом, мы можем получить не просто линию, а целостные фигуры, заполненные цветом.

Дуга

```
<Grid>
    <Path Stroke="Blue">
        <Path.Data>
            <PathGeometry>
                <PathFigure IsClosed="True" StartPoint="10,10">
                    <ArcSegment Point="250,50" Size="150,200" />
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
    <Path Stroke="Red">
        <Path.Data>
            <PathGeometry>
                <PathFigure IsClosed="False" StartPoint="10,110">
                    <ArcSegment Point="250,150" Size="150,200" />
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
</Grid>
```

Для создания дуги у ArcSegment задается свойство Point, которое указывает на конечную точку дуги (начальная точка задается через свойство StartPoint элемента PathFigure), а свойство Size устанавливает размер окружностей, по которым строится дуга.



Кривые Безье

Кривые Безье представляют линии, для построения которых применяются сложные математические преобразования. В WPF кривые Безье представлены различными типами: простые и квадратичные кривые. Для построения кривых Безье используются начальная и конечная точки, а также ряд промежуточных точек:

```
<Grid>
    <Path Stroke="Blue">
        <Path.Data>
            <PathGeometry>
                <PathFigure StartPoint="60,10">
                    <BezierSegment Point1="180,30" Point2="100,140" Point3="210,150" />
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
    <Path Stroke="Red">
        <Path.Data>
            <PathGeometry>
                <PathFigure IsClosed="False" StartPoint="10,110">
                    <QuadraticBezierSegment Point1="130,130" Point2="40,240" />
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
</Grid>
```

Начальная точка кривых устанавливается с помощью свойства StartPoint элемента PathFigure. Для простой кривой Безье свойства Point1 и Point2 задают промежуточные точки, а Point3 является конечной точкой. Для квадратичной кривой Point2 - конечная точка, а Point1 - промежуточная.

Сокращенная запись пути

Также принят упрощенный вариант записи фигур. Например, следующее описание фигуры

```
<Path Stroke="Red">
    <Path.Data>
        <PathGeometry>
            <PathFigure StartPoint="20,170">
                <LineSegment Point="50,170" />
            </PathFigure>
        </PathGeometry>
    </Path.Data>
</Path>
```

можно написать следующим образом

```
можно написать следующим образом
```

или

```
<Path Stroke="Red">
    <Path.Data>
        <PathGeometry Figures="M 20,170 L 50,170 Z" />
    </Path.Data>
</Path>
```

Что в данном случае означате сокращенная запись?

M x,y	Создает новый объект PathFigure и указывает на его начальную точку
Z	Завершает фигуру и устанавливает свойство IsClosed в true. Если же не требуется завершать фигуру, то вместо Z употребляется буква M
L x,y	Создает объект LineSegment до указанной точки
A raduisX, radiusY, degrees, isLargArc, IaClockwize, x,y	Создает новый объект ArcSegment с соответствующими параметрами
C x1,y1,x2,y2,x,y	Создает новый объект BezierSegment по указанным точкам
Q x1,y1, x,y	Создает новый объект QuadraticBezierSegment по указанным точкам
S x1,y1, x,y	Создает новый объект BezierSegment по указанным точкам

Программное создание сегментов

Создадим программно координатную плоскость с использованием объекта PathGeometry:

```
PathGeometry pathGeom =new PathGeometry();
Path p =new Path();

LineSegment vertLS =new LineSegment();
PathFigure vertPF =new PathFigure();
vertPF.StartPoint = new Point(10, 150);
vertLS.Point = new Point(10, 10);
vertPF.Segments.Add(vertLS);
pathGeom.Figures.Add(vertPF);

LineSegment horLS =new LineSegment();
PathFigure horPF =new PathFigure();
horPF.StartPoint = new Point(10, 150);
horLS.Point = new Point(150, 150);
horPF.Segments.Add(horLS);
pathGeom.Figures.Add(horPF);

for(byte i = 2;i< 14;i++)
{
    PathFigure pf = new PathFigure();
    pf.StartPoint = new Point(i * 10, 155);
    LineSegment a = new LineSegment();
    a.Point = new Point(i * 10, 145);
    pf.Segments.Add(a);
    pathGeom.Figures.Add(pf);
}

for(byte i = 3;i< 15;i++)
{
    PathFigure pf =new PathFigure();
    pf.StartPoint = new Point(5, i * 10);
    LineSegment a =new LineSegment();
    a.Point = new Point(15, i * 10);
    pf.Segments.Add(a);
    pathGeom.Figures.Add(pf);
}

PolyLineSegment vertArr =new PolyLineSegment();
vertArr.Points = new PointCollection();
vertArr.Points.Add(new Point(10, 10));
vertArr.Points.Add(new Point(15, 15));
PathFigure vertArrF =new PathFigure();
vertArrF.StartPoint = new Point(5, 15);
vertArrF.Segments.Add(vertArr);
pathGeom.Figures.Add(vertArrF);

PolyLineSegment horArr = new PolyLineSegment();
horArr.Points = new PointCollection();
horArr.Points.Add(new Point(150, 150));
horArr.Points.Add(new Point(145, 155));
```

```
PathFigure horArrF = new PathFigure();
horArrF.StartPoint = new Point(145, 145);
horArrF.Segments.Add(horArr);
pathGeom.Figures.Add(horArrF);

p.Data = pathGeom;
p.Stroke = Brushes.Black;

grid1.Children.Add(p);
```

Трансформации

Трансформации представляют инструмент изменения положения или размера элементов WPF. Трансформации могут быть полезны в тех ситуациях, когда надо изменить положение элемента, либо анимировать. Все трансформации наследуются от абстрактного базового класса **System.Windows.Media**.Transform и представляют следующие классы:

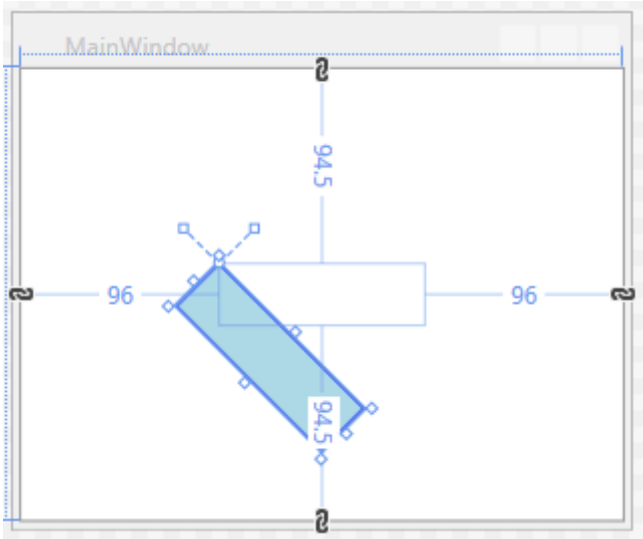
- **TranslateTransform**: сдвигает элементы по горизонтали и вертикали
- **RotateTransform**: вращает элемент
- **ScaleTransform**: выполняет операции масштабирования
- **SkewTransform**: изменяет позицию элемента путем наклона на определенное количество градусов
- **MatrixTransform**: изменяет координатную систему в соответствии с определенной матрицей
- **TransformGroup**: представляет группу трансформаций

RotateTransform

RotateTransform поворачивает элемент вокруг оси на определенное количество градусов. Данный объект принимает три основых параметра:

- Angle: угол поворота
- CenterX: устанавливает центр вращения по оси X
- CenterY: устанавливает центр вращения по оси Y

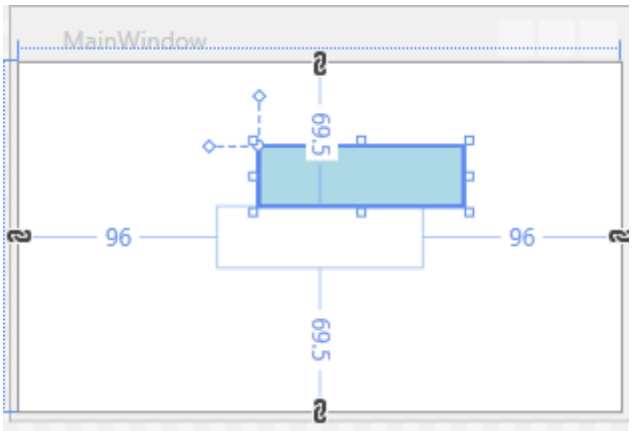
```
<Rectangle Width="100" Height="30" Stroke="Blue" Fill="LightBlue">
    <Rectangle.RenderTransform>
        <RotateTransform Angle="45" />
    </Rectangle.RenderTransform>
</Rectangle>
```



TranslateTransform

TranslateTransform позволяет сместить положение элемента по оси X, с помощью свойства X, и по оси Y - с помощью свойства Y.

```
<Rectangle Width="100" Height="30" Stroke="Blue" Fill="LightBlue">
    <Rectangle.RenderTransform>
        <TranslateTransform X="20" Y="-30" />
    </Rectangle.RenderTransform>
</Rectangle>
```

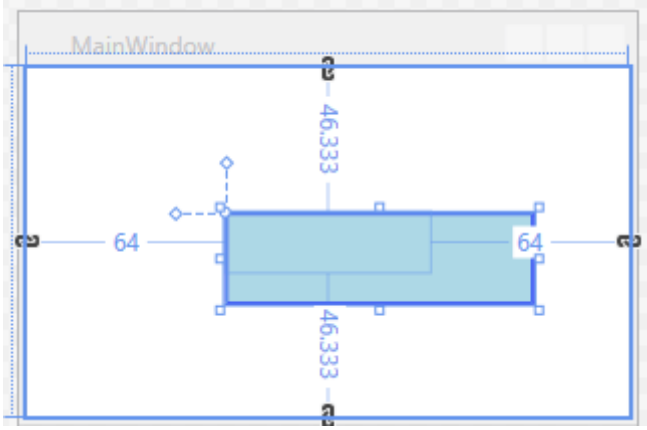


ScaleTransform

Обеспечивает масштабирование элемента на определенную величину. Для изменения ширины надо задать свойство **ScaleX**, а для изменения длины - свойство **ScaleY**. Кроме того, также имеются свойства **CenterX** и **CenterY**, позволяющие позиционировать элемент.

Например, увеличение прямоугольника в полтора раза:

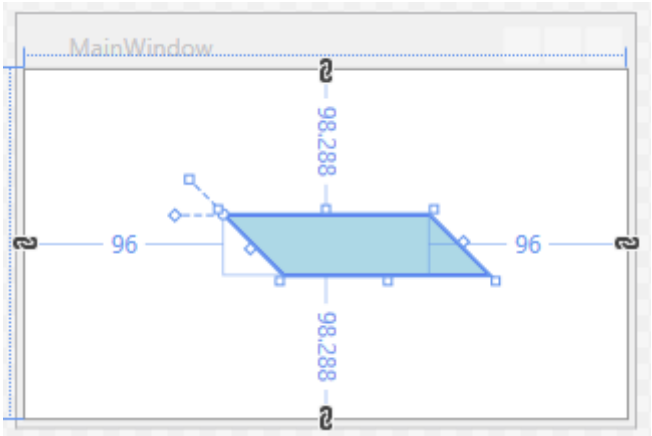
```
<Rectangle Width="100" Height="30" Stroke="Blue" Fill="LightBlue">
    <Rectangle.RenderTransform>
        <ScaleTransform ScaleX="1.5" ScaleY="1.5" />
    </Rectangle.RenderTransform>
</Rectangle>
```



SkewTransform

SkewTransform позволяет задать наклон элемента вдоль оси X с помощью свойства **AngleX**, и по оси Y - с помощью свойства **AngleY**. А с помощью свойств **CenterX** и **CenterY** можно изменить положение элемента относительно осей X и Y:

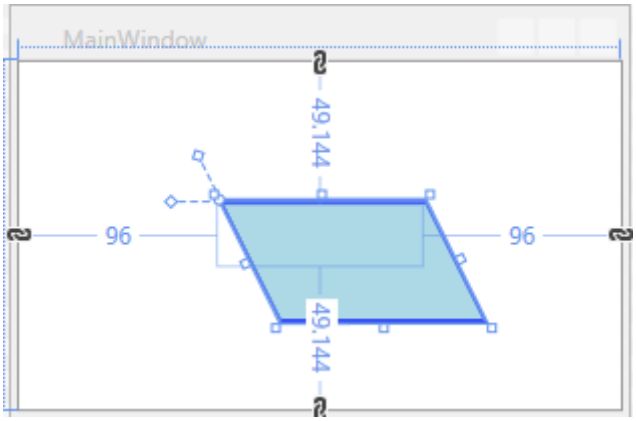
```
<Rectangle Width="100" Height="30" Stroke="Blue" Fill="LightBlue">
    <Rectangle.RenderTransform>
        <SkewTransform AngleX="45"  />
    </Rectangle.RenderTransform>
</Rectangle>
```



MatrixTransform

Осуществляет матричное преобразование элемента. В свойстве **Matrix** мы задаем первые два столбца, которые применяются при преобразовании. Последний столбец по умолчанию имеет значения {0 0 1}.

```
<Rectangle Width="100" Height="30" Stroke="Blue" Fill="LightBlue">
    <Rectangle.RenderTransform>
        <MatrixTransform Matrix="1 0 1 2 1 -3" />
    </Rectangle.RenderTransform>
</Rectangle>
```



TransformGroup

TransformGroup позволяет комбинировать различные трансформации вместе:

```
<Rectangle Width="100" Height="30" Stroke="Blue" Fill="LightBlue">
    <Rectangle.RenderTransform>
        <TransformGroup>
            <RotateTransform Angle="45" />
            <TranslateTransform Y="-40" X="30" />
        </TransformGroup>
    </Rectangle.RenderTransform>
</Rectangle>
```

RenderTransform и LayoutTransform

Для применения трансформаций у фигур и стандартных элементов управления WPF используются свойства **RenderTransform** и **LayoutTransform**. Несмотря на то, что для обоих свойств трансформации задаются одинаково, их действие различается. Так, свойство LayoutTransform применяется до компоновки элемента, а RenderTransform - после, поэтому одинаковые трансформации для этих свойств могут давать немного разные результаты:

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Button Width="80" Height="30" Background="LightBlue" Content="Hello">
        <Button.RenderTransform>
            <RotateTransform Angle="-45" />
        </Button.RenderTransform>
    </Button>
    <Button Grid.Column="1" Width="80" Height="30" Background="LightBlue" Content="Hello">
        <Button.LayoutTransform>
            <RotateTransform Angle="-45" />
        </Button.LayoutTransform>
    </Button>
</Grid>
```