XAML Do-It-Yourself シリーズ

第 11 回 2D グラフィックス

Microsoft®

XAML Do-It-Yourself
第 11 回 2D グラフィックス

XAML Do-It-Yourself 第 11 回は、2D グラフィックスについて学習します。

XAML を使って作成する WPF アプリケーションでは、従来の Windows フォーム アプリケーションと比較して、2D および 3D グラフィックスが非常に容易に扱えるようになっています。

ここでは 2D グラフィックスついて、次のことを学習します。

・Shape 派生クラスによる基本図形の描画
・PathGeometry を使った複雑な図形の描画
・ビットマップ画像の表示とエフェクト

■Shape 派生クラスによる描画

XAML で 2D グラフィックスを描画する方法はいくつかありますが、Shape クラスの派生クラスを使った場合と、Geometry クラスの派生クラスを使った場合の 2 つに大きく分けることができます。まず Shape クラスの派生クラスを用いるグラフィックスについて紹介します。

Shape クラスの派生クラスには、次に示すような基本的な図形が含まれています。

Shape クラスの派生クラス

クラス	概要
Ellipse	楕円
Rectangle	矩形
Line	直線
Polyline	連続した直線
Path	直線や円弧などを複合した図形
Polygon	多角形

この中で Path は直線や円弧を連続して使用し図形を描画するというものですが、それ以外の図形は非常に基本的なものです。なお、Polygon は Polyline と似ていますが、これは連続する直線の始点と終点を結んだ、閉じた多角形を描画します。

これらの図形を XAML で記述するのは非常に簡単です。図形を描画したい領域の左上隅を原点として、図形の座標を指定します。なお座標は "デバイス独立ピクセル" と呼ばれるピクセル単位で指定を行います。精度は倍精度浮動小数点 (double) です。

例えばパネルの原点から (100,100) の位置に直線を描画する場合は、次のように、(X1,Y1) = (0,0) と (X2,Y2) = (100,100) を属性で指定します。ここでは色や線の太さも指定しています。

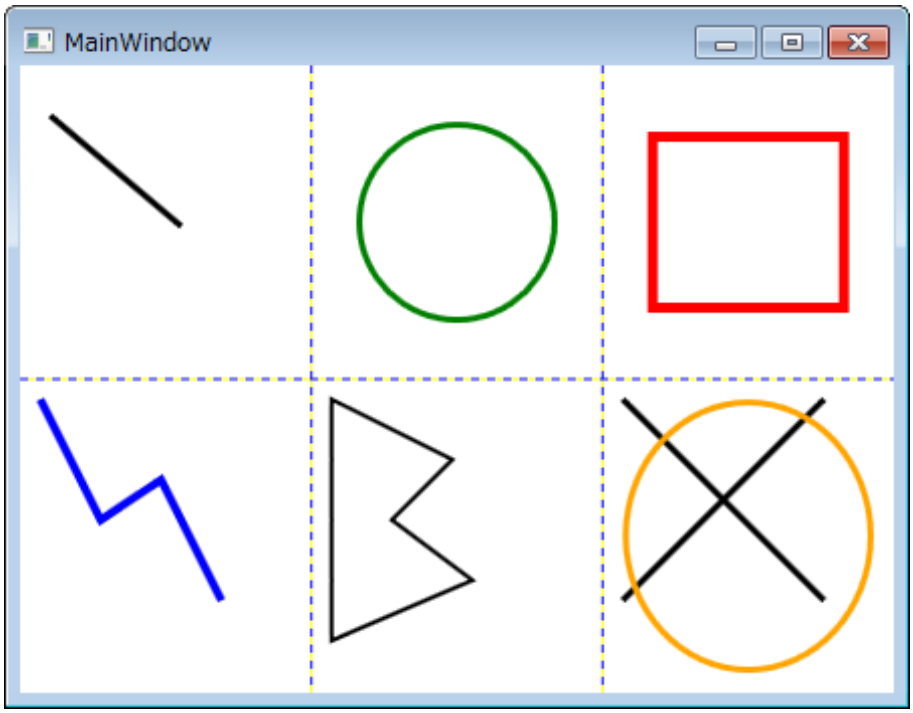
```
<Grid>
  <Line X1="0" Y1="0"
    X2="100" Y2="100"
    Stroke="Black"
    StrokeThickness="3"/>
</Grid>
```

ほかの図形でも同様に、必要な座標を指定することで簡単に描画できます。それぞれの図形の例として次のコードを紹介します。

```
<Window x:Class="WpfApplication11.MainWindow"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="MainWindow" Height="350" Width="450">
  <Grid ShowGridLines="True">
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <!-- 1 行 1 列目 直線-->
    <Line Grid.Row="0" Grid.Column="0"
      X1="15" Y1="25" X2="80" Y2="80"
      Stroke="Black" StrokeThickness="3"/>

    <!-- 1 行 2 列目 楕円-->
    <Ellipse Grid.Row="0" Grid.Column="1" Width="100" Height="100"
      Stroke="Green" StrokeThickness="3"/>

    <!-- 1 行 3 列目 矩形-->
    <Rectangle Grid.Row="0" Grid.Column="2" Stroke="Red"
      StrokeThickness="5" Width="100" Height="90"/>

    <!-- 2 行 1 列目 連続した直線 -->
    <Polyline Grid.Row="1" Grid.Column="0" Margin="10,10,10,10"
      Stroke="Blue" StrokeThickness="4"
      Points="0,0 30,60 60,40 90,100" />

    <!-- 2 行 2 列目 多角形 -->
    <Polygon Grid.Row="1" Grid.Column="1"
      Stroke="Black" StrokeThickness="2" Margin="10">
      <Polygon.Points>
        <Point X="0" Y="0"/>
        <Point X="60" Y="30"/>
        <Point X="30" Y="60" />
        <Point X="70" Y="90" />
        <Point X="0" Y="120" />
```

```
      </Polygon.Points>
    </Polygon>

    <!-- 2 行 3 列目 複数の図形 -->
    <Line Grid.Row="1" Grid.Column="2" Margin="10"
      X1="0" Y1="0" X2="100" Y2="100"
      Stroke="Black" StrokeThickness="3"/>
    <Line Grid.Row="1" Grid.Column="2" Margin="10"
      X1="100" Y1="0" X2="0" Y2="100"
      Stroke="Black" StrokeThickness="3"/>
    <Ellipse Grid.Row="1" Grid.Column="2" Margin="10"
      Stroke="Orange" StrokeThickness="3"/>
  </Grid>
</Window>
```

このコードでは、Grid パネルを 6 分割して、それぞれのセルにさまざまな図形を描画します。実行すると次のような表示になります。



<Polyline> 要素では、Margin 属性を指定していますが、これを指定すると、指定した分だけ原点が移動します。

<Polyline> 要素や <Polygone> 要素では座標を複数個指定します。この場合の座標の指定方法は、上記の <Polyline> 要素のように Points 属性に座標をスペースで区切って記述することも、<Polygone> 要素のように、<Polygone.Points> 要素を記述して、<Point> 要素で 1 つずつ座標を指定することも可能です。

なお Path については、後ほど説明します。

■Geometry 派生クラスを使った描画

Shape の派生クラスによる描画は非常にシンプルですが、複雑な図形を描画する場合は記述が大変です。より複雑な図形を描画する場合は、Geometry クラスの派生クラスを使って図形を描画します。Geometry クラスの主な派生クラスを示します。

Geometry の派生クラス

クラス	概要
LineGeometry	直線
RectangleGeometry	矩形
EllipseGeometry	楕円
PathGeometry	複合した図形の記述
StreamGeometry	短縮型のパスの記述
GeometryGroup	複数の Geometry をグループ化
CombineGeometry	Geometry の結合

この中で実際に図形の定義が行えるのは、LineGeometry、RectangleGeometry、EllipseGeometry の 3 つです。これらの図形を組み合わせて複雑な図形を作成していきます。ただし、Geometry の派生クラスは、これらの Geometry クラスを使って図形を定義するものの、実際の描画は行いません。描画を行うためには、Shape クラスの派生クラスに含まれる Path を利用します。

この方法で、たとえば (0,0) から (100,100) に直線を描画する場合は、次のように記述します。

```
<Grid>
  <Path Stroke="Black" StrokeThickness="3">
    <Path.Data>
      <LineGeometry StartPoint="0,0" EndPoint="100,100" />
    </Path.Data>
  </Path>
</Grid>
```

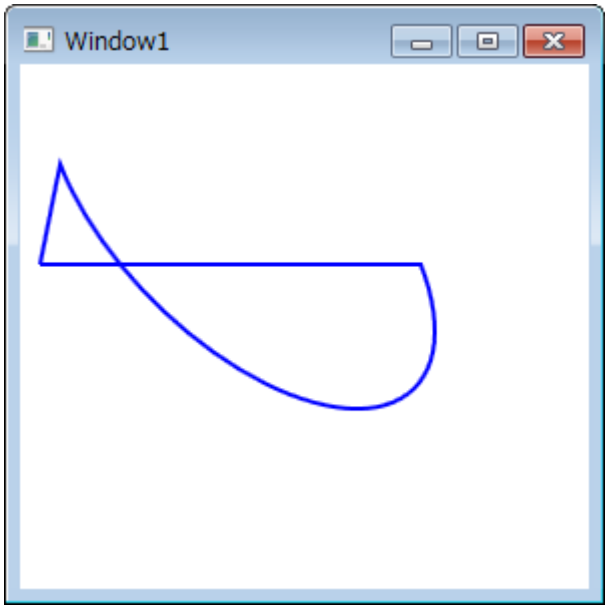
また、PathGeometry を使って複数の基本的な図形を組み合わせた図形を作成することもできます。PathGeometry ではさらに、LineSegment (直線)、PolylineSegment (連続する直線)、ArcSegment (円弧)、BezierSegment (ベジエ曲線)、PolyBezierSegment (連続するベジエ曲線) などを使って、図形を記述できます。この場合も、実際に描画を行うには <Path> 要素を使用します。

```
<Window x:Class="WpfApplication11.Window1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="Window1" Height="300" Width="300">
 <Grid>
   <Path Stroke="Blue" StrokeThickness="2">
     <Path.Data>
```

```
      <PathGeometry>
        <PathGeometry.Figures>
          <PathFigureCollection>
            <PathFigure StartPoint="10,100"> <!-- 開始座標 -->
              <PathFigure.Segments>
                <PathSegmentCollection>
                  <!-- 図形の指定  -->
                  <LineSegment Point="20,50" />
                  <ArcSegment Size="100,50" RotationAngle="45"
                      IsLargeArc="True" SweepDirection="CounterClockwise"
                      Point="200,100" />
                  <LineSegment Point="10,100" />
                </PathSegmentCollection>
              </PathFigure.Segments>
            </PathFigure>
          </PathFigureCollection>
        </PathGeometry.Figures>
      </PathGeometry>
    </Path.Data>
  </Path>
 </Grid>
</Window>
```

この記述では、直線と円弧を連続して描き、さらに始点までを直線で閉じた図形を描画します。

StreamGeometry を使った描画



上記のような XAML の記述では、要素の階層が深く、分かりにくくなっています。そこで同様の図形を、今度は StreamGeometry を使って描画してみます。これは次のようになります。

```
<Window x:Class="WpfApplication11.Window2"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="Window2" Height="300" Width="300">
  <Grid>
    <Path Stroke="Red" StrokeThickness="2"
      Data="M 10,100 L 20,50 A 100,50 45 1 0 200,100 L 10,100 Z" />
  </Grid>
</Window>
```

StreamGeometry では、描画の指示をコマンドとパラメーターで指定できます。上記の例では、次のような 4 つのコマンドが Data 属性に含まれています。

```
M 10,100
L 20,50
A 100,50 45 1 0 200,100
L 10,100
Z
```

"M" は描画開始座標の移動を表します。"L" は直線を、"A" は円弧を描画します。そして、"Z" がコマンドの終了を表します。StreamGeometry を利用すると、コンパクトに図形を表現できます。

また、今回は紹介しませんが、GeometryGroup を使って複数の図形をグループにまとめたり、CombineGeometry を使って、図形を結合したりすることも可能です。これらを利用することで、さらに複雑な図形を効率よく描画できます。

■ビットマップ画像とエフェクト

2D グラフィックスで重要なビットマップ画像の描画についても確認しておきましょう。これまでも ListBox による一覧表示で画像を使用しました。

画像を描画するには <Image> 要素を用います。

```
  <Grid>
    <Image Source="image/Tree.jpg" Width="200" Height="150"
      Stretch="Uniform"/>
  </Grid>
```

画像は Width 属性と Height 属性で指定した大きさで描画されますが、Stretch 属性により描画内容は変化します。Stretch 属性には、"None"、"Fill"、"Uniform"、"UniformToFill" の 4 種類が指定できます。

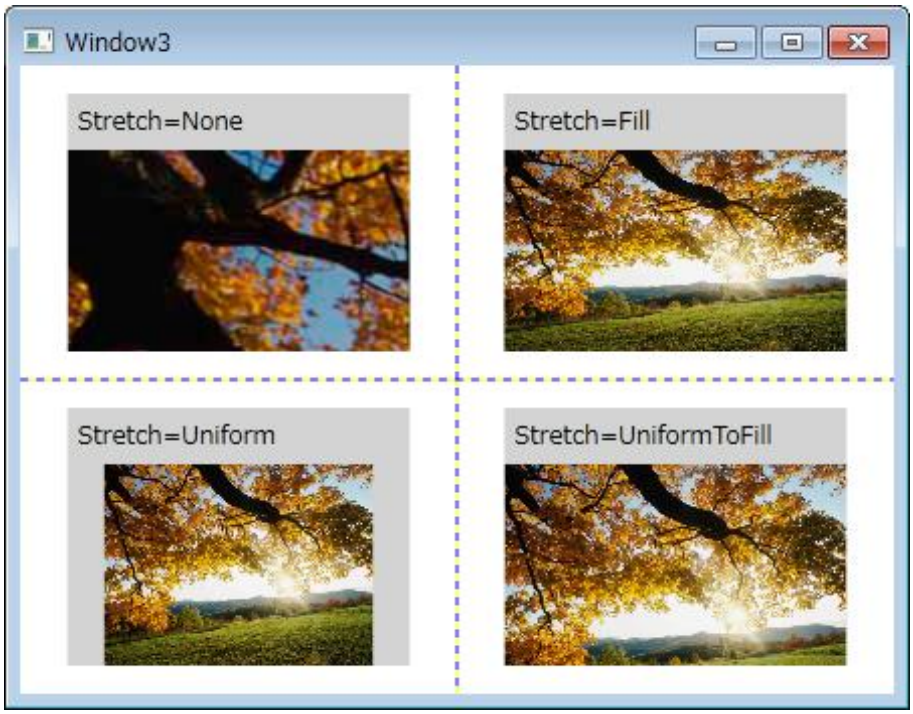
"None" は画像をそのまま描画します。描画領域よりも画像が大きい場合は、画像の左上部分のみが描画されます。"Fill" は指定した領域に収まるように、画像が縮小 (拡大) されます。場合によっては画像の縦横の比率が変化します。"Uniform" は画像の縦横比を変えないで、指定された領域に収まるサイズで描画されます。縦横比によっては、描画されない領域が発生します。最後の "UniformToFill" は、縦横比を変化させないで、指定された領域いっぱいに画像が描画されます。この場合は画像の一部が描画されない場

合があります。

以下に実際の描画例を示します。なおここでは、プロジェクト内の image フォルダーに、"Autumn Leaves.jpg" という画像ファイルがあるものとします。

```
<Window x:Class="WpfApplication11.Window3"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window3" Height="350" Width="450">
  <Window.Resources>
    <Style TargetType="Image">
      <Setter Property="Width" Value="170" />
      <Setter Property="Height" Value="100" />
      <Setter Property="Source" Value="image¥Autumn Leaves.jpg"/>
    </Style>
  </Window.Resources>
  <Grid ShowGridLines="True">
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
       <StackPanel Grid.Row="0" Grid.Column="0" HorizontalAlignment="Center"
VerticalAlignment="Center" Background="LightGray">
      <Label>Stretch=None</Label>
      <Image Stretch="None" />
    </StackPanel>
    <StackPanel Grid.Row="0" Grid.Column="1" HorizontalAlignment="Center"
VerticalAlignment="Center" Background="LightGray">
      <Label>Stretch=Fill</Label>
      <Image Stretch="Fill"/>
    </StackPanel>
    <StackPanel Grid.Row="1" Grid.Column="0" HorizontalAlignment="Center"
VerticalAlignment="Center" Background="LightGray">
      <Label>Stretch=Uniform</Label>
      <Image Stretch="Uniform"/>
    </StackPanel>
    <StackPanel Grid.Row="1" Grid.Column="1" HorizontalAlignment="Center"
VerticalAlignment="Center" Background="LightGray">
      <Label>Stretch=UniformToFill</Label>
      <Image Stretch="UniformToFill"/>
    </StackPanel>
  </Grid>
</Window>
```

このコードを実行すると、4 つの Stretch 属性の指定結果を確認できます。



●画像のエフェクト

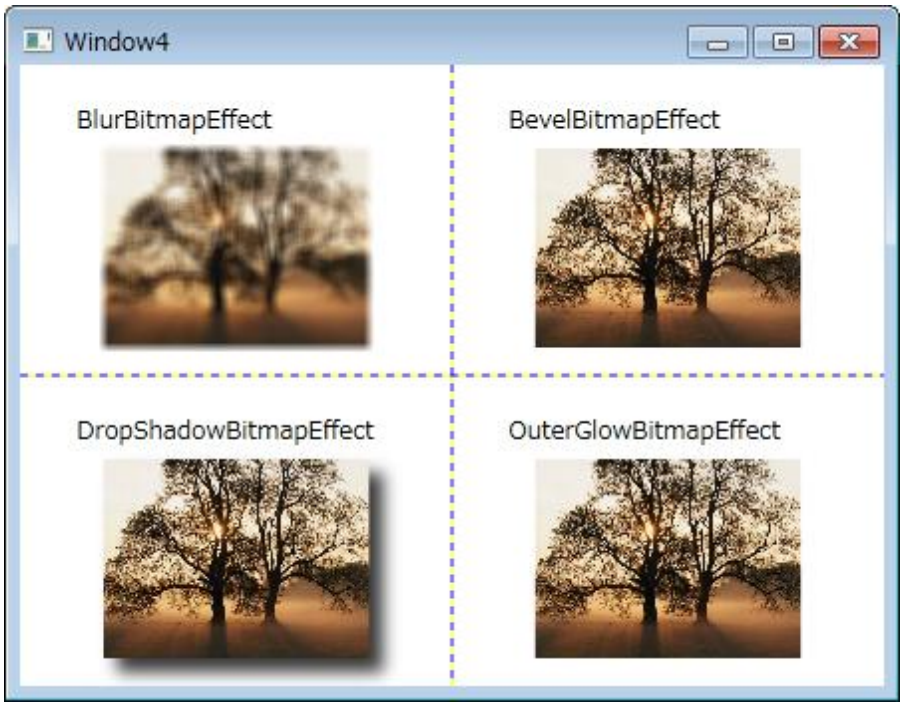
また、画像にさまざまなエフェクトをかけて描画することもできます。画像のエフェクトには、ぼかしを行う BlurBitmapEffect、立体的なボタンのように描画する BevelBitmapEffect 、影を付ける DropShadowBitmapEffect、画像の回りをぼかす OuterGlowBitmapEffect などが用意されています。

これらの画像のエフェクトは次のコードで確認できます。実行には画像ファイルが必要です。

```
<Window x:Class="WpfApplication11.Window4"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window4" Height="350" Width="450">
  <Window.Resources>
    <Style TargetType="Image">
      <Setter Property="Width" Value="170" />
      <Setter Property="Height" Value="100" />
      <Setter Property="Source" Value="image¥Tree.jpg"/>
      <Setter Property="Stretch" Value="Uniform"/>
    </Style>
  </Window.Resources>
   <Grid ShowGridLines="True">
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
```

```
    </Grid.ColumnDefinitions>
    <StackPanel Grid.Row="0" Grid.Column="0" HorizontalAlignment="Center"
VerticalAlignment="Center" >
      <Label>BlurBitmapEffect</Label>
      <Image >
        <Image.BitmapEffect>
          <BlurBitmapEffect Radius="2" KernelType="Box" />
        </Image.BitmapEffect>
      </Image>
    </StackPanel>
    <StackPanel Grid.Row="0" Grid.Column="1" HorizontalAlignment="Center"
VerticalAlignment="Center" >
      <Label>BevelBitmapEffect</Label>
      <Image >
        <Image.BitmapEffect>
          <BevelBitmapEffect BevelWidth="10" />
        </Image.BitmapEffect>
      </Image>
    </StackPanel>
    <StackPanel Grid.Row="1" Grid.Column="0" HorizontalAlignment="Center"
VerticalAlignment="Center" >
      <Label>DropShadowBitmapEffect</Label>
      <Image >
        <Image.BitmapEffect>
          <DropShadowBitmapEffect Color="Gray" ShadowDepth="10"/>
        </Image.BitmapEffect>
      </Image>
    </StackPanel>
    <StackPanel Grid.Row="1" Grid.Column="1" HorizontalAlignment="Center"
VerticalAlignment="Center" >
      <Label>OuterGlowBitmapEffect</Label>
      <Image >
        <Image.BitmapEffect>
          <OuterGlowBitmapEffect GlowColor="Black" GlowSize="10" />
        </Image.BitmapEffect>
      </Image>
    </StackPanel>
  </Grid>
</Window>
```

このコードを実行すると、ビットマップ画像に設定された 4 種類のエフェクトが確認できます。



■まとめ

今回は、2D グラフィックスの基本的な操作と、ビットマップ画像の表示について学習しました。複雑な図形を描画する場合は、StreamGeometry を利用することで XAML の要素の記述を省略することができます。

次回は 3D グラフィックスについて学習します。