

## Bài 8

# Đồ họa hai chiều trong WPF (2D-Graphics)

Trước đây, để xây dựng một ứng dụng đồ họa đẹp, hiển thị các đối tượng đồ họa với những hiệu ứng và chuyển động người lập trình phải mất nhiều công sức. Với WPF các công việc trên trở nên đơn giản hơn nhiều, bởi vì WPF đã tích hợp sẵn đồ họa vector, đa phương tiện, hình ảnh động (animation) và các đối tượng đồ họa phức hợp. Các đối tượng đồ họa trong WPF không chỉ để hiển thị một cách đơn thuần, chúng còn có khả năng phát sinh các sự kiện mà thông thường chỉ có trong các điều khiển thông dụng của Window. Lập trình viên có thể xây dựng các ứng dụng đồ họa đẹp, sinh động và thú vị với Microsoft Visual Studio .NET hay thậm chí chỉ cần sử dụng NotePad.

Bài này giới thiệu về cách xây dựng các đối tượng đồ họa như đoạn thẳng, chuỗi đoạn thẳng, đa giác,... với các cách thức tô vẽ phong phú, đẹp mắt cũng như các hiệu ứng dịch chuyển bằng mã lệnh XAML.

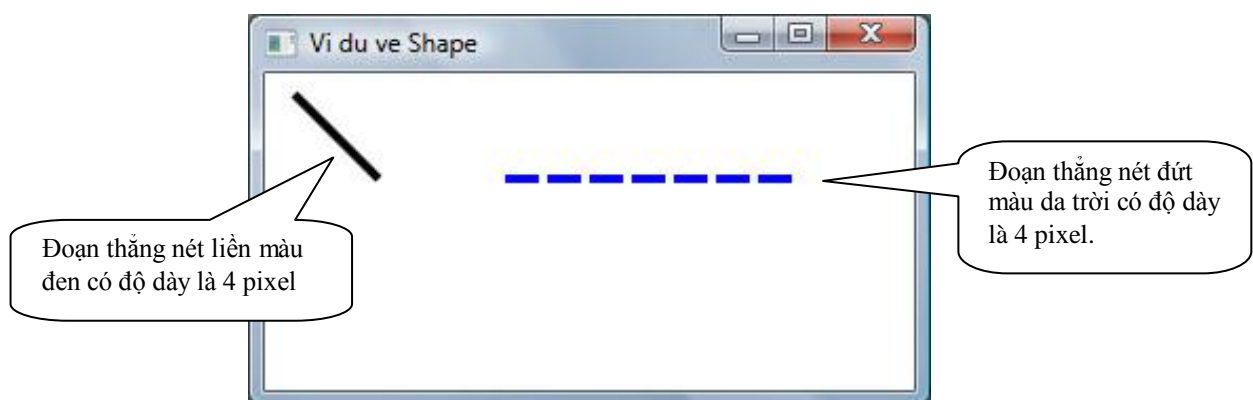
### 1. Các đối tượng đồ họa cơ bản - Shape

Để bắt đầu, chúng ta sẽ tìm hiểu các mã lệnh XAML để hiển thị các đối tượng đồ họa cơ bản như **Line** (đoạn thẳng), **Ellipse** (hình elip), **Polygon** (đa giác), **Polyline** (chuỗi đoạn thẳng), **Rectangle** (chữ nhật) và **Path** (hình phức hợp). Các đối tượng này được kế thừa từ đối tượng cơ sở Shape. Các đối tượng kế thừa từ Shape có chung một số thuộc tính như:

- **Stroke**: Mô tả màu sắc đường viền của một hình hoặc màu của một đoạn thẳng.
- **StrokeThickness**: Độ dày của đường viền.
- **Fill**: Cách tô phần bên trong của một hình.
- **Data**: Mô tả các tọa độ, các đỉnh của một hình, đơn vị đo là pixel.

#### 1.1 Đoạn thẳng (Line)

Đoạn thẳng là một đối tượng được định nghĩa dựa trên hai đầu mút là hai điểm. Chúng ta có thể định nghĩa độ dày của đoạn thẳng, màu sắc hay cách vẽ đoạn thẳng (nét liền, nét đứt...). Hình dưới đây minh họa một số ví dụ về đoạn thẳng



Hình 8.1. Ví dụ về đoạn thẳng

Mã lệnh XAML của ví dụ trên như sau.

#### Đoạn mã trình của hai đoạn thẳng trên bằng XAML:

```
<Window x:Class="Lesson08_Graphics.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Vi du ve Shape" Height="338" Width="324">
```

```

<Canvas Height="300" Width="300">
  <!-- Vẽ một đoạn thẳng nằm xiên từ tọa độ (10,10) tới (50,50).
        Độ dày đoạn thẳng là 4 pixel mà có màu đen
  -->
  <Line
    X1="10" Y1="10"
    X2="50" Y2="50"
    Stroke="Black"
    StrokeThickness="4" />

  <!-- Vẽ một đoạn thẳng nằm ngang từ tọa độ (10,50) to (150,50).
        nằm cách lề trái của canvas 100 pixel
        Đoạn thẳng màu xanh da trời, độ dày 4 pixel,
        nét đứt xen kẽ cứ mỗi đoạn màu xanh là 4 thì lại xen khoảng trắng là 1.
  -->
  <Line
    X1="10" Y1="50"
    X2="150" Y2="50"
    Canvas.Left="100"
    Stroke="Blue"
    StrokeThickness="4"
    StrokeDashArray="4 1" />

</Canvas>
</Window>

```

Thông thường ta hay chọn layout là Canvas để chứa các đối tượng đồ họa bởi vì Canvas cho phép đặt các đối tượng bên trong theo vị trí tuyệt đối.

Trong ví dụ trên thẻ `<Line/>` dùng để định nghĩa một đoạn thẳng. Thẻ này có một số thuộc tính cơ bản:

- `X1="10" Y1="10"` : Tọa độ đỉnh thứ nhất là  $X=10$  và  $Y = 10$
- `StrokeThickness="4"` : Độ dày của đoạn thẳng là 4 pixel
- `X2="50" Y2="50"` : Tọa độ đỉnh thứ hai là  $X=50$  và  $Y = 50$
- `Stroke="Black"` : Màu của đoạn thẳng là màu đen
- `StrokeThickness="4"` : Độ dày của đoạn thẳng là 4 pixel
- `StrokeDashArray="4 1"` : Đoạn thẳng được tô theo nét đứt, cứ 4 pixel có màu thì 1 pixel là khoảng trắng.

### Đoạn mã trình C# vẽ đoạn thẳng.

```

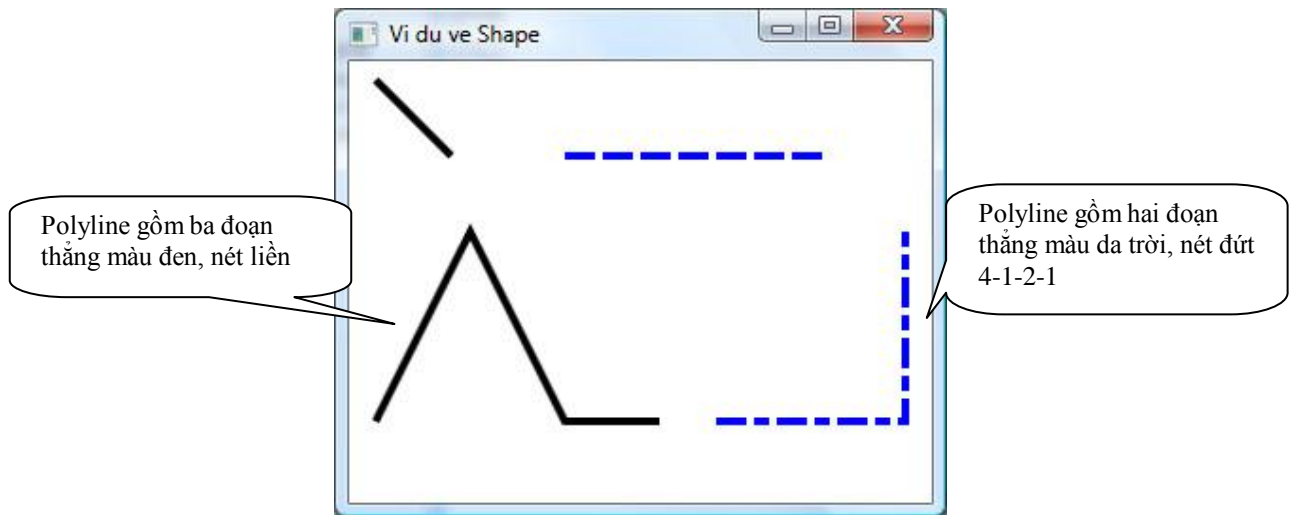
// Add a Line Element
myLine = new Line();
myLine.Stroke = System.Windows.Media.Brushes.LightSteelBlue;
myLine.X1 = 1;
myLine.X2 = 50;
myLine.Y1 = 1;
myLine.Y2 = 50;
myLine.HorizontalAlignment = HorizontalAlignment.Left;
myLine.VerticalAlignment = VerticalAlignment.Center;
myLine.StrokeThickness = 2;
myGrid.Children.Add(myLine);

```

## 1.2 Chuỗi đoạn thẳng (Polyline)

Polyline là đối tượng bao gồm nhiều đoạn thẳng liên tiếp nối với nhau. Một Polyline gồm  $N$  đoạn thẳng thì được định nghĩa bởi  $N+1$  điểm.

Hình dưới minh họa hai Polyline, một Polyline gồm ba đoạn và một Polyline gồm hai đoạn.



Hình 8.2. Ví dụ về Polyline

Để hiển thị một Polyline bằng mã lệnh, tạo một đối tượng **Polyline** và sử dụng thuộc tính **Points** của nó để khai báo tọa độ của các đỉnh. Tiếp đến, có thể sử dụng các thuộc tính **Stroke** và **StrokeThickness** để mô tả màu sắc và độ dày của Polyline.

Đối với mã XAML, cú pháp khai báo dãy các điểm là: mỗi cặp tọa độ X,Y phân biệt với nhau bởi khoảng trống và giữa X với Y phân biệt bởi dấu phẩy.

Chú ý rằng, đối tượng Polyline cũng có thuộc tính **Fill** để tô màu bên trong, nhưng thuộc tính này không có tác dụng. Nếu muốn tô màu cho vùng bên trong của tập hợp các điểm thì sử dụng đối tượng Polygon.

**Đoạn mã sau minh họa mã lệnh XAML của ví dụ này.**

```
<Window x:Class="Lesson08_Graphics.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Vi du ve Shape" Height="338" Width="324">
  <Canvas Height="300" Width="300">

    <!--Vẽ một chuỗi đoạn thẳng gồm ba đoạn nối tiếp nhau
      được nối bởi bốn đỉnh (X,Y) = (10,110) (60,10) (110,110) và (160,110) -->
    <Polyline
      Points="10,110 60,10 110,110 160,110"
      Stroke="Black"
      StrokeThickness="4" Canvas.Left="0" Canvas.Top="80" />
    <!--Vẽ một chuỗi đoạn thẳng gồm hai đoạn nối tiếp nhau với nét đứt
      được nối bởi ba đỉnh (X,Y) = (10,110) (110,110) và (110,10) -->
    <Polyline
      Points="10,110 110,110 110,10"
      Stroke="Blue"
      StrokeThickness="4"
      StrokeDashArray="4 1 2 1"
      Canvas.Left="180" Canvas.Top="80" />
  </Canvas>
</Window>
```

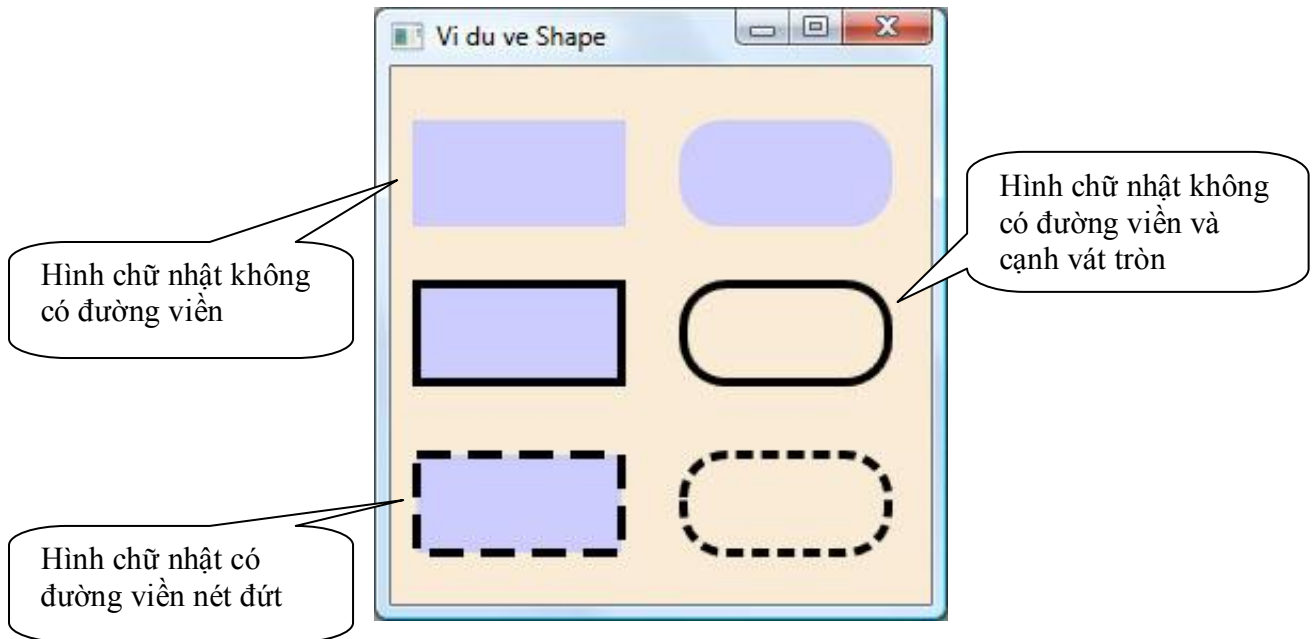
Thẻ `<Polyline/>` được sử dụng để tạo Polyline.

Thuộc tính `Points="X1,Y1 X2,Y2 X3,Y3 X4,Y4"` khai báo tập hợp các điểm tạo nên Polyline.

Thuộc tính `StrokeDashArray="4 1 2 1"` có nghĩa là Polyline được vẽ bằng nét đứt theo thứ tự 4 nét màu 1 nét trắng tiếp đến là 2 nét màu mà 1 nét trắng, và tiếp tục lặp lại...

### 1.3 Hình chữ nhật (Rectangle)

Đối tượng **Rectangle** được xác định bởi tọa độ của góc trên trái và độ rộng, độ cao của hình chữ nhật cần hiển thị. Ngoài ra, ta có thể thiết lập các thuộc tính cho đường viền (màu sắc, độ dày, kiểu dáng) và tô phần bên trong của hình.



Hình 8.3. Ví dụ về hình chữ nhật

Đoạn mã sau minh họa mã lệnh XAML của ví dụ này.

```
<Window x:Class="Lesson08_Graphics.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Vi du ve Shape" Height="338" Width="324">
  <Canvas Height="300" Width="300" Background="AntiqueWhite">
    <!-- Vẽ hình chữ nhật không có đường viền, được tô màu #CCCCFF-->
    <Rectangle
      Width="100"
      Height="50"
      Fill="#CCCCFF"
      Canvas.Left="10"
      Canvas.Top="25" />
    <!-- Vẽ hình chữ nhật có đường viền màu đen độ dày 4 pixel,
      được tô màu #CCCCFF-->
    <Rectangle
      Width="100"
      Height="50"
      Fill="#CCCCFF"
      Stroke="Black"
      StrokeThickness="4"
      Canvas.Left="10"
      Canvas.Top="100" />
    <!-- Vẽ hình chữ nhật các góc vát tròn không có đường viền,
      được tô màu #CCCCFF-->
    <Rectangle
      Width="100"
      Height="50"
      RadiusX="20"
      RadiusY="20"
      Fill="#CCCCFF"
      Canvas.Left="135"
      Canvas.Top="25" />
    <!-- Vẽ hình chữ nhật các góc vát tròn có đường viền màu đen
      độ dày 4 pixel, không được tô màu-->
```

```

<Rectangle
  Width="100"
  Height="50"
  RadiusX="20"
  RadiusY="20"
  Stroke="Black"
  StrokeThickness="4"
  Canvas.Left="135"
  Canvas.Top="100" />
<!-- Vẽ hình chữ nhật có đường viền nét đứt màu đen độ dày 4 pixel,
      được tô màu #CCCCFF-->
<Rectangle
  Width="100"
  Height="50"
  Fill="#CCCCFF"
  Stroke="Black"
  StrokeThickness="4"
  StrokeDashArray="4 2"
  Canvas.Left="10"
  Canvas.Top="180" />
<!-- Vẽ hình chữ nhật các góc vát tròn có đường viền nét đứt màu đen
      độ dày 4 pixel, không được tô màu-->
<Rectangle
  Width="100"
  Height="50"
  RadiusX="20"
  RadiusY="20"
  Stroke="Black"
  StrokeThickness="4"
  StrokeDashArray="2 1"
  Canvas.Left="135"
  Canvas.Top="180" />
</Canvas>
</Window>

```

Thẻ `<Rectangle/>` dùng để vẽ một hình chữ nhật.

Các thuộc tính `Canvas.Left`, `Canvas.Top`, `Width`, `Height` chỉ định tọa độ góc trên trái và độ rộng, độ cao của hình chữ nhật.

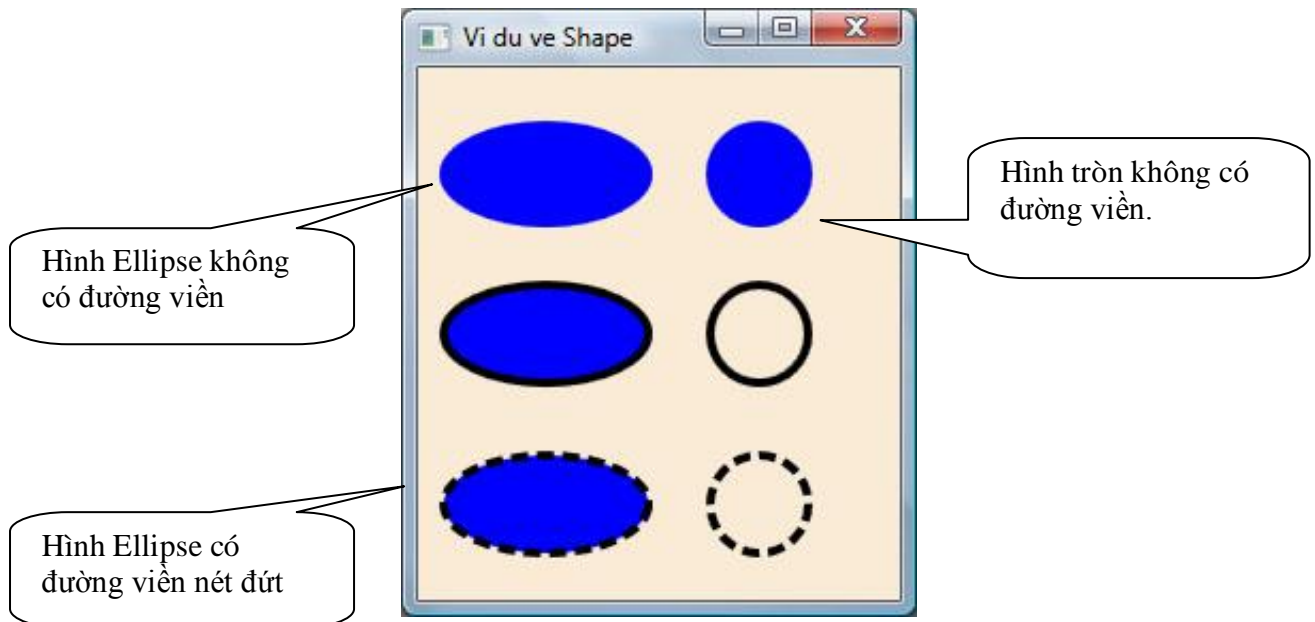
Thuộc tính `Fill` chỉ định màu tô bên trong hình chữ nhật. nếu bỏ qua thuộc tính này thì hình chữ nhật sẽ là trong suốt.

Các thuộc tính `Stroke`, `StrokeThickness`, `StrokeDashArray` chỉ định kiểu đường viền của hình chữ nhật. Nếu không chỉ định giá trị cho các thuộc tính này thì hình chữ nhật sẽ không có đường viền.

Các thuộc tính `RadiusX`, `RadiusY` là bán kính của hình ellipse để tạo ra các góc tròn của hình chữ nhật.

## 1.4 Hình elip (Ellipse) và hình tròn (Circle)

Hình **Ellipse** được xác định bởi tọa độ của góc trên trái và độ rộng, độ cao của hình chữ nhật ngoại tiếp của **Ellipse** cần hiển thị. Hình tròn là hình **Ellipse** với chiều rộng và chiều cao bằng nhau. **Ellipse** cũng có các thuộc tính cho đường viền (màu sắc, độ dày, kiểu dáng) và tô phần bên trong của hình tương tự như hình chữ nhật.



Hình 8.4. Ví dụ về hình Ellipse

**Đoạn mã sau minh họa mã lệnh XAML của ví dụ này.**

```
<Window x:Class="Lesson08_Graphics.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Ví dụ vẽ Shape" Height="338" Width="324">
  <Canvas Height="300" Width="300" Background="AntiqueWhite">
    <!-- Vẽ hình Ellipse được tô màu Blue. -->
    <Ellipse
      Width="100"
      Height="50"
      Fill="Blue"
      Canvas.Left="10"
      Canvas.Top="25" />

    <!-- Vẽ hình Ellipse được tô màu Blue và viền đen. -->
    <Ellipse
      Width="100"
      Height="50"
      Fill="Blue"
      Stroke="Black"
      StrokeThickness="4"
      Canvas.Left="10"
      Canvas.Top="100"/>

    <!-- Vẽ hình Ellipse được tô màu Blue và viền đen nét đứt. -->
    <Ellipse
      Width="100"
      Height="50"
      Fill="Blue"
      Stroke="Black"
      StrokeThickness="4"
      StrokeDashArray="2 1"
      Canvas.Left="10"
      Canvas.Top="180"/>

    <!-- Vẽ hình tròn được tô màu Blue. -->
    <Ellipse
      Width="50"
      Height="50"
      Fill="Blue"
      Canvas.Left="135"
      Canvas.Top="25"/>
```

```

<!-- Vẽ hình tròn rỗng có viền đen. -->
<Ellipse
Width="50"
Height="50"
Stroke="Black"
StrokeThickness="4"
Canvas.Left="135"
Canvas.Top="100" />

<!-- Vẽ hình tròn rỗng có viền đen nét đứt. -->
<Ellipse
Width="50"
Height="50"
Stroke="Black"
StrokeThickness="4"
StrokeDashArray="2 1"
Canvas.Left="135"
Canvas.Top="180" />
</Canvas>
</Window>

```

Thẻ `<Ellipse />` dùng để vẽ một hình Ellipse hay hình tròn.

Các thuộc tính `Canvas.Left`, `Canvas.Top`, `Width`, `Height` chỉ định tọa độ góc trên trái và độ rộng, độ cao của hình chữ nhật ngoại tiếp của Ellipse. Nếu `Width` và `Height` bằng nhau thì ta sẽ có hình tròn.

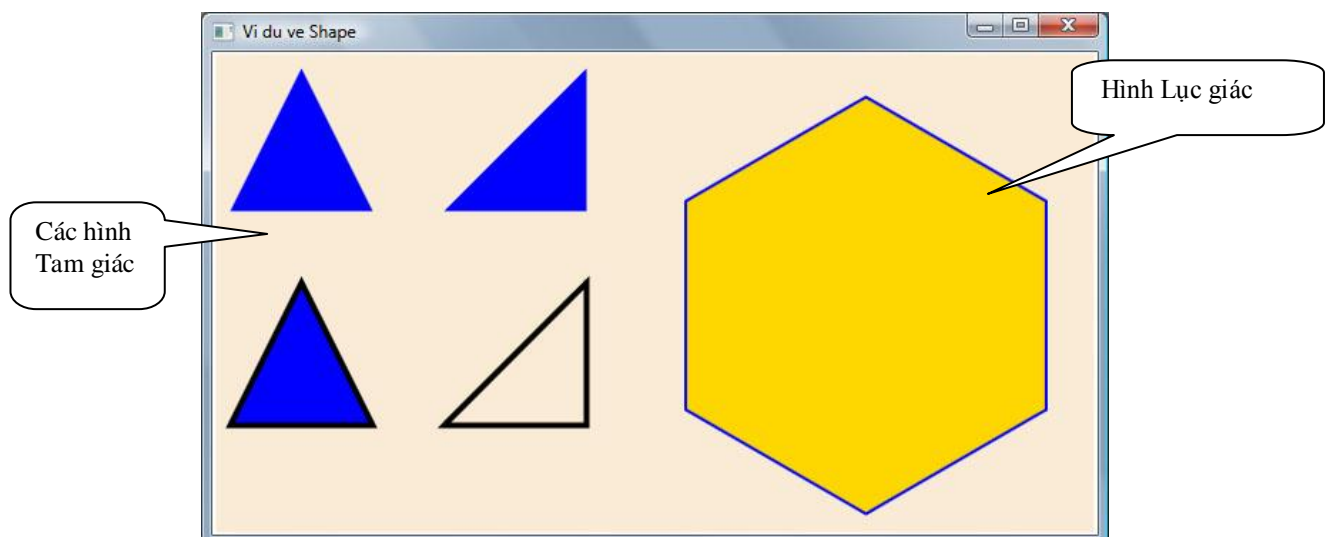
Thuộc tính `Fill` chỉ định màu tô bên trong hình Ellipse. nếu bỏ qua thuộc tính này thì hình Ellipse sẽ là trong suốt.

Các thuộc tính `Stroke`, `StrokeThickness`, `StrokeDashArray` chỉ định kiểu đường viền của hình Ellipse. Nếu không chỉ định giá trị cho các thuộc tính này thì hình Ellipse sẽ không có đường viền.

## 1.5 Đa giác (Polygon)

**Polygon** là đối tượng dùng để trình diễn các hình dạng phức tạp, gồm đoạn thẳng nối tiếp khép kín. Một **Polygon** N đỉnh được định nghĩa bởi một tập hợp N cặp tọa độ tương ứng với mỗi đỉnh của nó.

Hình dưới minh họa một số Polygon dạng tam giác và lục giác.



Hình 8.5. Ví dụ vẽ Polygon

**Polygon** cũng có các thuộc tính tương tự như **Polyline** như: thuộc tính **Points** của nó để khai báo tọa độ của các đỉnh, các thuộc tính **Stroke** và **StrokeThickness** để mô tả màu sắc và độ dày đường viền Polyline. Tuy nhiên, đối tượng Polygon còn có tính **Fill** để tô màu bên trong đa giác.

Đối với mã XAML, cú pháp khai báo dãy các điểm là: mỗi cặp tọa độ X,Y phân biệt với nhau bởi khoảng trống và giữa X với Y phân biệt bởi dấu phẩy.

**Đoạn mã sau minh họa mã lệnh XAML của ví dụ này.**

```
<Window x:Class="Lesson08_Graphics.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Vi du ve Shape" Height="374" Width="637">
  <Canvas Height="335" Width="616" Background="AntiqueWhite">
    <!-- Vẽ một hình tam giác tô màu Blue và không có đường viền -->
    <Polygon Points="10,110 60,10 110,110"
      Fill="Blue" />

    <!-- Vẽ một hình tam giác tô màu Blue, có đường viền màu đen dày 4 pixel
      Thuộc tính Canvas.Top dùng để dịch tam giác xuống 150 pixels so với
      đỉnh Canvas. -->
    <Polygon Points="10,110 60,10 110,110"
      Fill="Blue"
      Stroke="Black" StrokeThickness="4"
      Canvas.Top="150" />

    <!-- Vẽ một hình tam giác tô màu Blue và không có đường viền.
      Thuộc tính Canvas.Left dùng để dịch tam giác sang phải 150 pixels. -->
    <Polygon Points="10,110 110,110 110,10"
      Fill="Blue"
      Canvas.Left="150" />

    <!-- Vẽ một hình tam giác có đường viền màu đen không tô màu.
      vị trí của tam giác cách cạnh trái và đỉnh của Canvas 150 pixel.-->
    <Polygon Points="10,110 110,110 110,10"
      Stroke="Black" StrokeThickness="4"
      Canvas.Left="150" Canvas.Top="150" />

    <!-- Vẽ một hình lục giác màu nền là Gold, đường viền màu đen.-->
    <Polygon Name="hexagon"
      Stroke="Blue"
      StrokeThickness="2.0"
      Fill="Gold"
      Points="176,30 302.44,103 302.44,249 176,322 49.5603,249 49.5603,103"
      Canvas.Left="280" Canvas.Top="0" />

  </Canvas>
</Window>
```

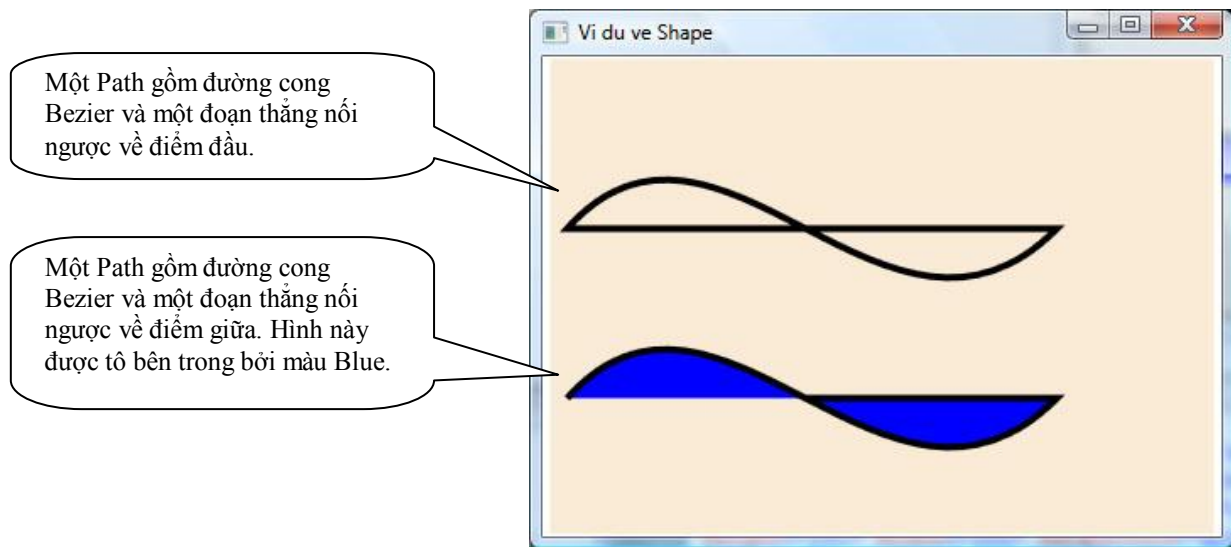
Thẻ `<Polygon />` được sử dụng để tạo đa giác.

Thuộc tính `Points="X1,Y1 X2,Y2 X3,Y3 X4,Y4"` khai báo tập hợp các đỉnh của đa giác.

## 1.6 Đường cong Bezier bằng đối tượng Path

Đối tượng **Path** được sử dụng để tạo nên những hình phức tạp, gồm nhiều phần nối với nhau. Ví dụ dưới đây minh họa sử dụng đối tượng **Path** tạo nên một hình gồm một đường cong Bezier, cuối đường cong là một đoạn thẳng nối ngược trở lại điểm đầu.





Hình 8.6. Ví dụ về Path

**Đoạn mã sau minh họa mã lệnh XAML của ví dụ này.**

```
<Window x:Class="Lesson08_Graphics.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Vi du ve Shape" Height="319" Width="418">
  <Canvas Height="280" Width="393" Background="AntiqueWhite">
    <!-- Vẽ một Path gồm hai phần:
      - Đường cong Bezier từ tọa độ (10,100) tới (300,100)
      tọa độ hai điểm điều khiển là (100,0) và (200,200)
      - Tại điểm kết thúc (300,100) vẽ một đường nằm ngang
      ngược về điểm khởi đầu.
    -->
    <Path
      Data="M 10,100 C 100,0 200,200 300,100 z"
      Stroke="Black"
      StrokeThickness="4" />
    <!-- Vẽ một Path các đỉnh Canvas 100 pixel gồm hai phần:
      - Đường cong Bezier từ tọa độ (10,100) tới (300,100)
      tọa độ hai điểm điều khiển là (100,0) và (200,200).
      - Tại điểm kết thúc (300,100) vẽ một đường nằm ngang
      ngược về điểm có tọa độ X = 150.
    -->
    <Path
      Data="M 10,100 C 100,0 200,200 300,100 H 150"
      Stroke="Black"
      StrokeThickness="4"
      Fill="Blue"
      Canvas.Top="100"/>
  </Canvas>
</Window>
```

Thẻ `<Path />` được sử dụng để tạo đường cong Bezier.

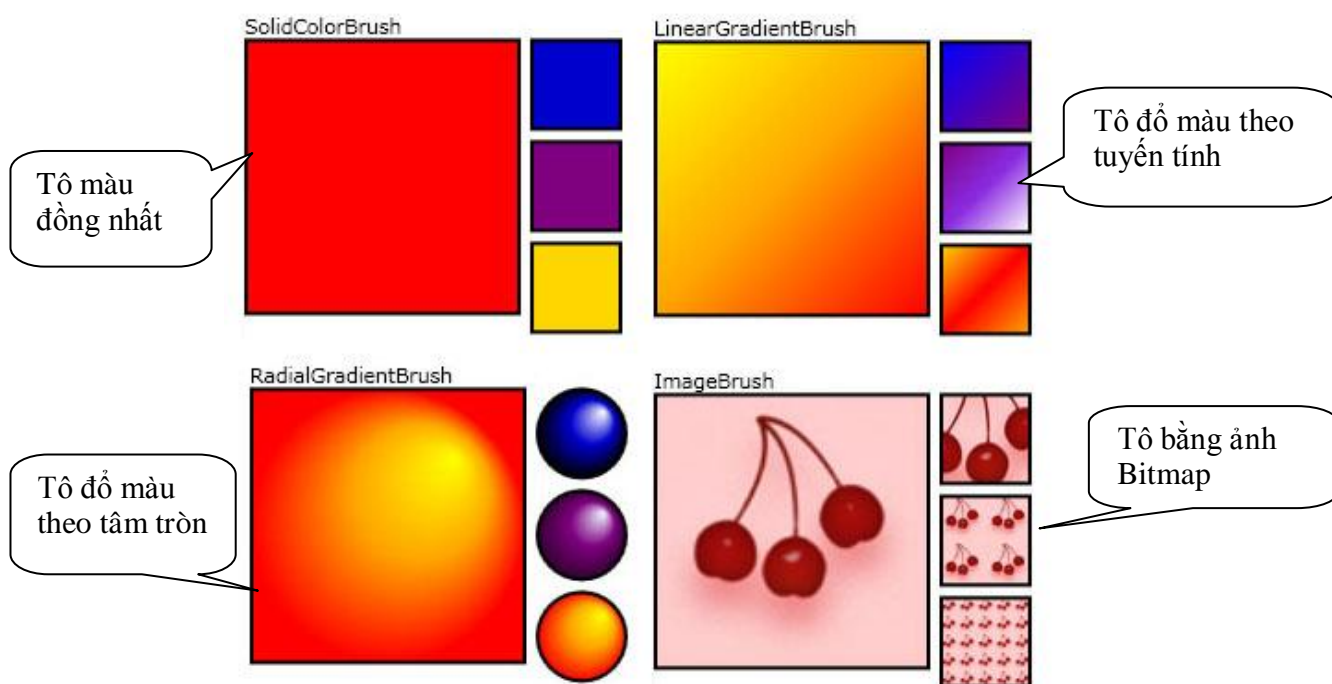
Thuộc tính `Data="M 10,100 C 100,0 200,200 300,100 z"` khai báo các thông số tạo nên một Path. Trong đó `M 10,100` nghĩa là đường cong bắt đầu từ điểm có tọa độ (10,100) tính theo hệ tọa độ của Canvas chứa Path này. Các thông số của thuộc tính **Data** có phân biệt chữ hoa, chữ thường. Nếu là chữ hoa thì tọa độ điểm được tính theo vị trí tuyệt đối, chữ thường thì tọa độ được tính theo vị trí tương đối. Ví dụ, **M** khai báo tọa độ điểm bắt đầu của Path tính theo vị trí tuyệt đối, còn nếu thay bằng **m** thì sẽ hiểu là vị trí tương đối. Ký tự **C** dùng để khai báo hai điểm điều khiển (Control Point) của đường cong. Ví dụ với `C 100,0 200,200` thì hai điểm điều khiển sẽ có tọa độ là (100,0) và (200,200). Sau hai điểm điều khiển là điểm kết thúc của đường cong, trong ví dụ trên, tọa độ điểm kết thúc là (300,100).

Đoạn thứ hai của Path trong ví dụ này là một đường kẻ ngang nối từ điểm kết thúc tới điểm khởi đầu của đường cong nhờ tham số **z**. Nếu muốn đặt đường kẻ này tới một điểm nào đó trên đường nằm ngang thì ta thay tham số

**z** bằng tham số **H**. Trong ví dụ ở hình thứ 2 ta có tham số **H 150** nghĩa là đường kẻ ngang bắt đầu từ điểm cuối của đường cong và kết thúc ở điểm có tọa độ  $X = 150$  tính theo vị trí tuyệt đối (nếu là chữ thường **h** thì nghĩa là vị trí tương đối).

## 2. Sử dụng chổi tô - Brush

Tất cả những gì chúng nhìn thấy trên màn hình, chúng hiển thị được là nhờ được tô bởi chổi tô (**Brush**). Chổi tô có thể sử dụng để tô nền của một nút bấm (Button), tô các nét chữ (Text) hay tô màu bên trong cho một đối tượng hình học như hình chữ nhật, đa giác,... Trong phần này chúng ta sẽ tìm hiểu cách sử dụng chổi tô trong WPF bằng mã lệnh XAML để tô các đối tượng hình học theo nhiều cách khác nhau như tô màu đồng nhất (**Solid Color**), tô kiểu đồ màu theo tuyến tính (**Linear Gradient Color**), tô đồ màu dọc theo bán kính hình tròn (**Radial Gradient Color**) và sử dụng ảnh bitmap để tô.



Hình 8.7. Minh họa một số kiểu tô hình ảnh

Thao tác cơ bản để tô vẽ cho một đối tượng trước hết là tạo một đối tượng chổi tô (Brush) tùy theo từng loại như trên, sau đó gắn chổi tô với thuộc tính có liên quan của đối tượng cần sử dụng chổi tô này. Mỗi loại đối tượng có một số thuộc tính khác nhau để chỉ định tô màu cho phần bên trong của nó. Bảng sau đây liệt kê một số loại đối tượng và thuộc tính được dùng để gắn với chổi tô.

| Loại đối tượng (Class) | Thuộc tính tô vẽ (Brush properties) |
|------------------------|-------------------------------------|
| Border                 | BorderBrush, Background             |
| Control                | Background, Foreground              |
| Panel                  | Background                          |
| Pen                    | Brush                               |
| Shape                  | Fill, Stroke                        |
| TextBlock              | Background                          |

### 2.1. Tô màu đồng nhất – Solid Color

Có nhiều cách để tô màu đồng nhất cho một đối tượng, ta có thể sử dụng trực tiếp thuộc tính tô màu của đối tượng (đối tượng hình học sử dụng thuộc tính **Fill**, đối tượng Button sử dụng thuộc tính **Background**,...) hoặc tạo

đối tượng tên là **SolidColorBrush** để định nghĩa màu sắc cần tô và gắn cho đối tượng cần sử dụng chổi tô đồng nhất này.

Đối tượng **SolidColorBrush** sử dụng thuộc tính **Color** để chỉ định màu cần tô. Giá trị gắn cho Color có thể là tên của màu đã được định nghĩa sẵn (là các thuộc tính tĩnh của đối tượng Brush) như *Red*, *MediumBlue*,... hoặc sử dụng công thức biểu diễn màu theo dạng mẫu 32 bit “#RRGGBB”, trong đó RR (giá trị của màu đỏ - Red), GG (giá trị của màu lá cây - Green), BB (giá trị của màu da trời - Blue) là các số Hexa (hệ đếm cơ số 16) có giá trị nằm trong khoảng từ 00 đến FF ( tương đương với giá trị từ 0 đến 255) . Ngoài ra, có thể sử dụng mẫu “#AARRGGBB”, AA là độ Alpha dùng để chỉ độ trong suốt của màu. Độ Alpha càng gần với 0 (00 hệ 16) thì màu càng trở nên trong suốt và càng gần với 255 (FF hệ 16) thì màu càng rõ nét.

Sau đây là ví dụ hiển thị một hình chữ nhật được tô đồng nhất màu đỏ.



Đoạn mã XAML của hình chữ nhật này như sau:

```
<!--Sử dụng mã định nghĩa sẵn -->
<Rectangle Width="75" Height="75">
    <Rectangle.Fill>
        <SolidColorBrush Color="Red" />
    </Rectangle.Fill>
</Rectangle>

<!--Sử dụng mã màu dạng #RRGGBB -->
<Rectangle Width="75" Height="75">
    <Rectangle.Fill>
        <SolidColorBrush Color="#FF0000" />
    </Rectangle.Fill>
</Rectangle>

<!-- Sử dụng trực tiếp thuộc tính Fill -->
<Rectangle Width="75" Height="75" Fill="#FF0000"/>
```

Đoạn mã lệnh C# tương đương như sau:

```
// Tạo đối tượng Rectangle để vẽ hình chữ nhật.
Rectangle exampleRectangle = new Rectangle();
exampleRectangle.Width = 75;
exampleRectangle.Height = 75;

// Tạo đối tượng SolidColorBrush dùng để tô màu cho hình chữ nhật.
SolidColorBrush myBrush = new SolidColorBrush(Colors.Red);
exampleRectangle.Fill = myBrush;
```

Ta cũng có thể tô màu cho các điều khiển nhờ thuộc tính Background. Ví dụ, tô màu cho nút bấm

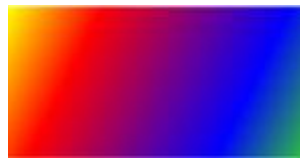
```
<Button Background="#FFFF0000">Nút bấm</Button>
<!-- Hoặc -->
<Button> Nút bấm
    <Button.Background>
        <SolidColorBrush Color="Red" />
    </Button.Background>
</Button>
```

## 2.2. Tô đổ màu theo định hướng tuyến tính – Linear Gradient Color

Tô đổ màu (hay tản màu) là kỹ thuật tô một vùng bằng nhiều màu trộn với nhau dọc theo một trục định hướng (Gradient axis). Ta có thể sử dụng kỹ thuật này để tạo nên các hình ảnh ấn tượng với độ sáng hay độ bóng của màu tạo cảm giác ba chiều (3D) hoặc cũng có thể dùng kỹ thuật này để tạo ra các giả lập bề mặt kính, nhôm, nước hay các bề mặt vật liệu mềm khác. WPF cung cấp hai loại đổ màu là đổ màu theo hướng tuyến tính (đối tượng **LinearGradientBrush**) hay đổ màu dọc theo bán kính hình tròn (đối tượng **RadialGradientBrush**).

Đối tượng **LinearGradientBrush** dùng để tô một vùng theo kỹ thuật tản màu dựa trên kỹ thuật nội suy các màu nằm giữa các cặp màu chỉ định dọc theo một trục (Gradient axis) dạng đường thẳng. Hướng của đường thẳng chính là hướng đổ màu, ta phải chỉ định những màu cần xuất hiện tại các điểm nằm trên đường thẳng này nhờ đối tượng **GradientStop**. Ta có thể chỉ định hướng đổ màu nằm ngang từ trái qua phải, nằm dọc từ trên xuống, hay theo đường chéo bất kỳ. Mặc định thì hướng đổ màu sẽ theo đường chéo từ góc trên bên trái tới góc dưới bên phải.

Dưới đây là một ví dụ vẽ một hình chữ nhật gồm bốn màu khác nhau được tô theo chế độ trộn màu.



Mã lệnh XAML của hình trên như sau:

```
<!-- Hình chữ nhật được tô tản màu theo đường chéo với bốn màu. -->
<Rectangle Width="200" Height="100">
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
            <GradientStop Color="Yellow" Offset="0.0" />
            <GradientStop Color="Red" Offset="0.25" />
            <GradientStop Color="Blue" Offset="0.75" />
            <GradientStop Color="LimeGreen" Offset="1.0" />
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

Mã lệnh C# tương ứng:

```
Rectangle diagonalFillRectangle = new Rectangle();
diagonalFillRectangle.Width = 200;
diagonalFillRectangle.Height = 100;

// Hình chữ nhật được tô tản màu theo đường chéo với bốn màu.
LinearGradientBrush myLinearGradientBrush =
    new LinearGradientBrush();
myLinearGradientBrush.StartPoint = new Point(0, 0);
myLinearGradientBrush.EndPoint = new Point(1, 1);
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Yellow, 0.0));
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Red, 0.25));
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Blue, 0.75));
myLinearGradientBrush.GradientStops.Add(
    new GradientStop(Colors.LimeGreen, 1.0));

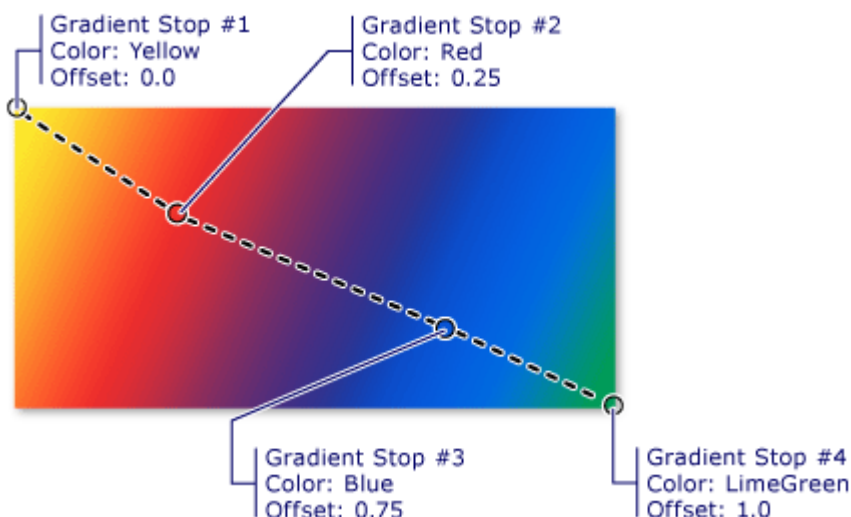
// Sử dụng chỗi tô để tô hình chữ nhật.
diagonalFillRectangle.Fill = myLinearGradientBrush;
```

Thuộc tính **StartPoint** và **EndPoint** của **LinearGradientBrush** để xác định điểm đầu và điểm cuối của trục tô mỗi thuộc tính này được xác định bởi một cặp giá trị "x,y". Trong đó, x và y là các số thực (double) có giá trị

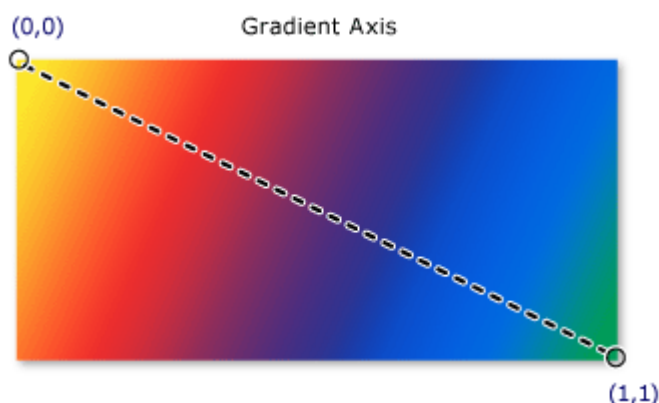
từ 0 đến 1, tương ứng với 0% và 100% tính tương đối so với đỉnh trên trái của hình chữ nhật cần vẽ (hay đỉnh của hình chữ nhật bao đối với các hình khác như Ellipse, Polygon,...).

Thẻ `<GradientStop Color="Color value" Offset="m.n" />` để chỉ định các điểm chốt nằm dọc theo trục tô. Thuộc tính `Color` xác định màu cần tô tại điểm chốt. Thuộc tính `Offset` nhằm xác định vị trí của điểm chốt, giá trị này là một số thực nằm trong khoảng từ 0 đến 1, giá trị càng gần 0 thì càng gần với điểm xuất phát `StartPoint`, giá trị gần với 1 thì càng gần với điểm kết thúc `EndPoint` của trục tô. Hệ thống sẽ tự động nội suy các màu nằm giữa các cặp điểm chốt.

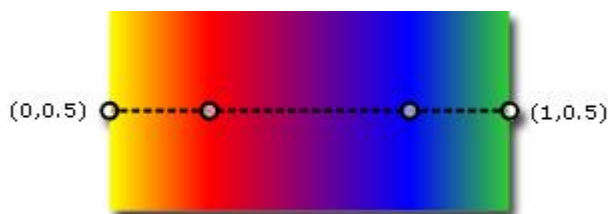
hình dưới đây minh họa các điểm chốt của ví dụ trên. Đường nét đứt chính là trục tô (Gradient Axis) còn vòng tròn là vị trí của điểm chốt (GradientStop).



Tọa độ các điểm `StartPoint` và `EndPoint` của trục tô ảnh hưởng đến hướng tô màu. Mặc định thì giá trị của `StartPoint` là (0,0) và `EndPoint` là (1,1), nghĩa là trục tô là đường chéo của hình chữ nhật bao.



Hình dưới đây minh họa các tô đồ màu theo chiều ngang gồm bốn màu.



Mã lệnh XAML của hình trên như sau:

```
<!-- Tô đồ màu hình chữ nhật theo hướng nằm ngang từ trái sang phải với bốn màu -->
<Rectangle Width="200" Height="100">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,0.5">
      <GradientStop Color="Yellow" Offset="0.0" />
      <GradientStop Color="Red" Offset="0.25" />
      <GradientStop Color="Blue" Offset="0.75" />
      <GradientStop Color="LimeGreen" Offset="1.0" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

</Rectangle>

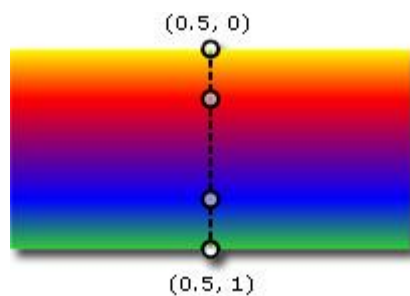
Mã lệnh C# tương ứng:

```
Rectangle horizontalFillRectangle = new Rectangle();
horizontalFillRectangle.Width = 200;
horizontalFillRectangle.Height = 100;

// Tạo chổi tô theo chiều ngang với 4 điểm chốt.
LinearGradientBrush myHorizontalGradient =
    new LinearGradientBrush();
myHorizontalGradient.StartPoint = new Point(0, 0.5);
myHorizontalGradient.EndPoint = new Point(1, 0.5);
myHorizontalGradient.GradientStops.Add(
    new GradientStop(Colors.Yellow, 0.0));
myHorizontalGradient.GradientStops.Add(
    new GradientStop(Colors.Red, 0.25));
myHorizontalGradient.GradientStops.Add(
    new GradientStop(Colors.Blue, 0.75));
myHorizontalGradient.GradientStops.Add(
    new GradientStop(Colors.LimeGreen, 1.0));

// Dùng chổi tô để tô hình chữ nhật.
horizontalFillRectangle.Fill = myHorizontalGradient;
```

Tô hình chữ nhật theo chiều đứng



Mã lệnh XAML của hình trên như sau:

```
<!-- Tô đổ màu hình chữ nhật theo hướng từ trên xuống với bốn màu. -->
<Rectangle Width="200" Height="100">
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
            <GradientStop Color="Yellow" Offset="0.0" />
            <GradientStop Color="Red" Offset="0.25" />
            <GradientStop Color="Blue" Offset="0.75" />
            <GradientStop Color="LimeGreen" Offset="1.0" />
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

Mã lệnh C# tương ứng:

```
Rectangle verticalFillRectangle = new Rectangle();
verticalFillRectangle.Width = 200;
verticalFillRectangle.Height = 100;

// Tạo chổi tô theo chiều đứng với 4 điểm chốt.
LinearGradientBrush myVerticalGradient =
    new LinearGradientBrush();
myVerticalGradient.StartPoint = new Point(0.5, 0);
myVerticalGradient.EndPoint = new Point(0.5, 1);
myVerticalGradient.GradientStops.Add(
    new GradientStop(Colors.Yellow, 0.0));
myVerticalGradient.GradientStops.Add(
```



```

    new GradientStop(Colors.Red, 0.25));
myVerticalGradient.GradientStops.Add(
    new GradientStop(Colors.Blue, 0.75));
myVerticalGradient.GradientStops.Add(
    new GradientStop(Colors.LimeGreen, 1.0));

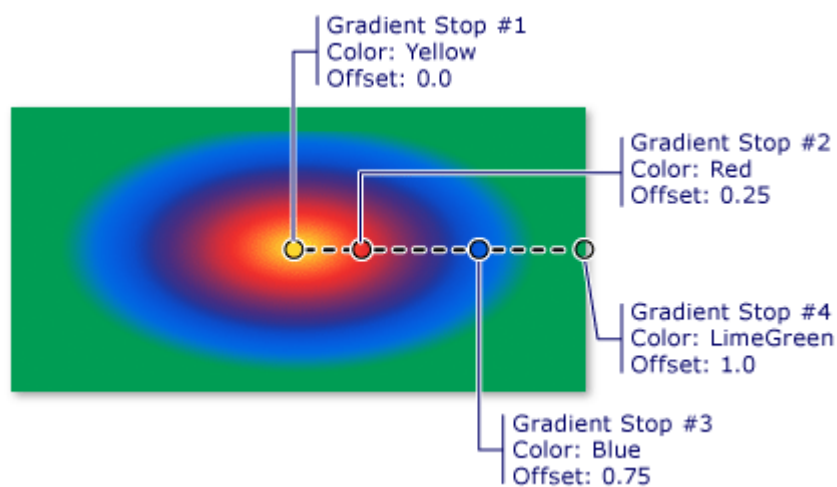
// Dùng chổi tô để tô hình chữ nhật.
verticalFillRectangle.Fill = myVerticalGradient;

```

### 2.3. Tô đổ màu theo bán kính hình tròn – Radial Gradient

Kỹ thuật tô đổ màu theo bán kính hình tròn tương tự như kỹ thuật đổ màu tuyến tính, nhưng điểm xuất phát bắt đầu từ tâm đường tròn và màu được lan dần ra ngoài cho tới biên của đường tròn, sử dụng đối tượng tên là **RadialGradientBrush**. Các điểm chốt vẫn sử dụng đối tượng **GradientStop** như phần trên.

Dưới đây là ví dụ minh họa cách tô đổ màu theo bán kính hình tròn



Mã lệnh minh họa thành trạng thái trên như sau:

```

<!-- Hình chữ nhật được tô bốn màu đổ từ trong ra ngoài -->
<Rectangle Width="200" Height="100">
    <Rectangle.Fill>
        <RadialGradientBrush
            GradientOrigin="0.5,0.5" Center="0.5,0.5"
            RadiusX="0.5" RadiusY="0.5">
            <GradientStop Color="Yellow" Offset="0" />
            <GradientStop Color="Red" Offset="0.25" />
            <GradientStop Color="Blue" Offset="0.75" />
            <GradientStop Color="LimeGreen" Offset="1" />
        </RadialGradientBrush>
    </Rectangle.Fill>
</Rectangle>

```

Mã lệnh C# của ví dụ trên như sau:

```

RadialGradientBrush myRadialGradientBrush = new RadialGradientBrush();
myRadialGradientBrush.GradientOrigin = new Point(0.5, 0.5);
myRadialGradientBrush.Center = new Point(0.5, 0.5);
myRadialGradientBrush.RadiusX = 0.5;
myRadialGradientBrush.RadiusY = 0.5;
myRadialGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Yellow, 0.0));
myRadialGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Red, 0.25));
myRadialGradientBrush.GradientStops.Add(
    new GradientStop(Colors.Blue, 0.75));

```

```
myRadialGradientBrush.GradientStops.Add(
    new GradientStop(Colors.LimeGreen, 1.0));

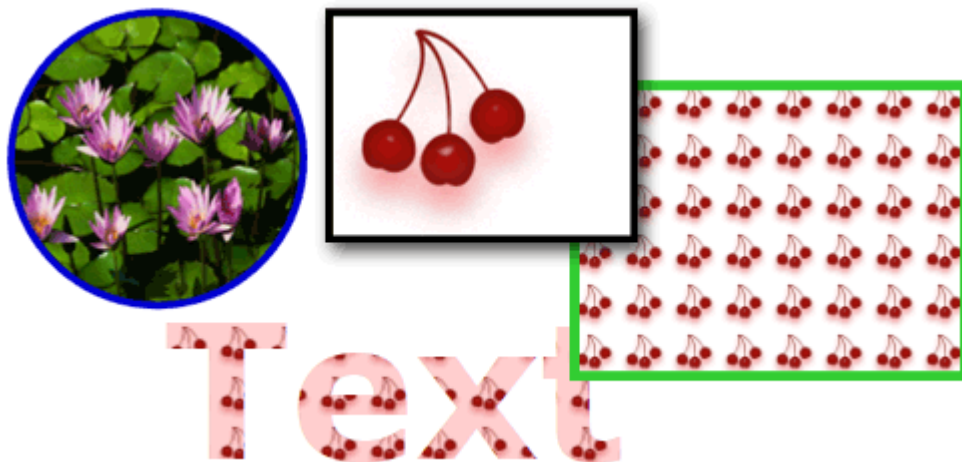
Rectangle myRectangle = new Rectangle();
myRectangle.Width = 200;
myRectangle.Height = 100;
myRectangle.Fill = myRadialGradientBrush;
```

Các thông số **Center** - Tọa độ tâm đường tròn, **RadiusX** - Bán kính ngang, **RadiusY** - Bán kính dọc được nhập các giá trị số thực trong khoảng từ 0 đến 1 tương ứng với tỷ lệ khoảng cách tới đỉnh trên trái của hình chữ nhật.

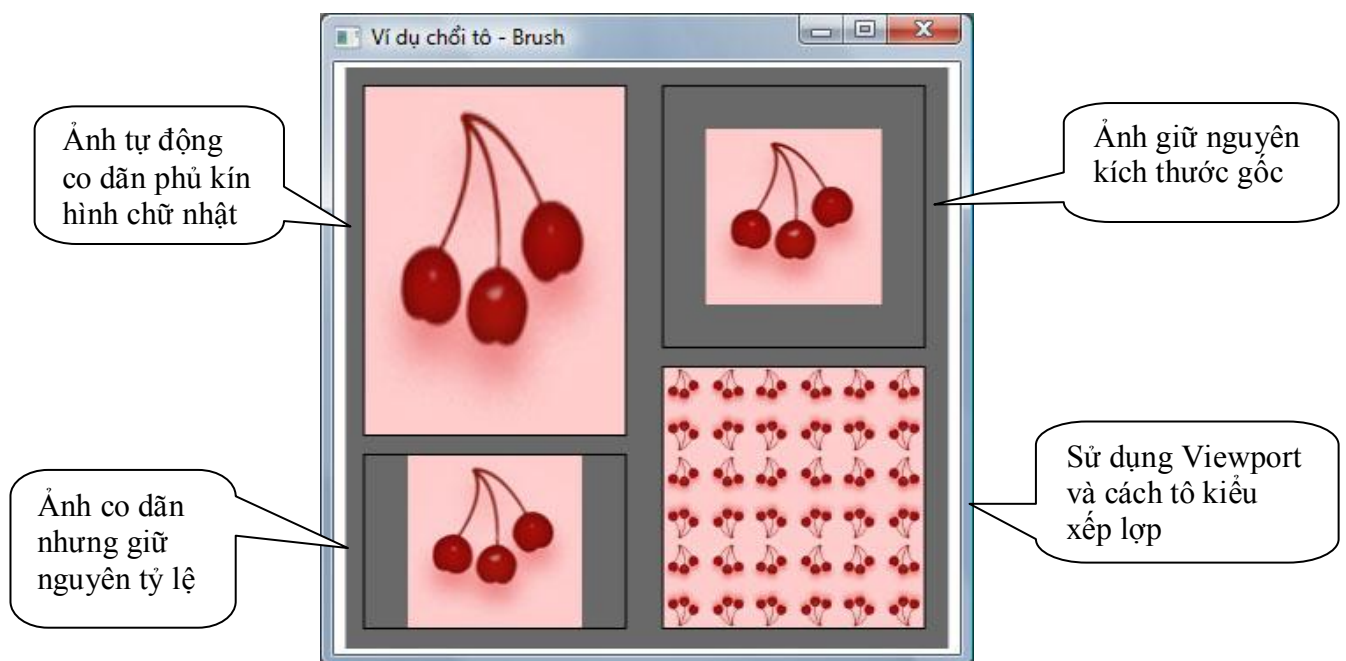
Thông số **GradientOrigin** là tọa độ của điểm xuất phát. Các điểm chốt sẽ chạy từ điểm **GradientOrigin** này, dọc theo đường kính hình tròn.

## 2.4. Tô bằng ảnh Bitmap – Radial Gradient

Ngoài các kỹ thuật tô màu cho các đối tượng hình học như trên, WPF còn hỗ trợ tô một vùng bằng những hình ảnh có sẵn (ảnh Bitmap, JPG,...) một cách dễ dàng nhờ đối tượng **ImageBrush**. Ta chỉ cần đưa tệp ảnh vào tài nguyên của Project và gắn đường dẫn ảnh cho thuộc tính **ImageSource** của đối tượng **ImageBrush**, sau đó dùng chổi tô **ImageBrush** để tô các đối tượng hình học (Shape), các điều khiển (Control), Panel hay Text,....



Dưới đây là ví dụ minh họa tô màu cho hình chữ nhật bằng hình ảnh.



Hình 8.8. Minh họa tô một vùng bằng hình ảnh



Mã lệnh minh họa thành trạng thái trên như sau:

```
<Window x:Class="Lesson08_Graphics.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Ví dụ chỗi tô - Brush" Height="373" Width="372">
  <Canvas Height="331" Width="343" Background="DimGray">
    <!-- Tô hình chữ nhật bằng hình ảnh
      File ảnh pinkcherries.jpg có kích thước 100x100 -->
    <!-- Mặc định ảnh tự động co giãn trùng khít với hình
      Do tỷ lệ chiều rộng và cao của vùng tô khác với tỷ lệ
      của ảnh nên sẽ bị méo hình -->
    <Rectangle Canvas.Left="10" Canvas.Top="10" Height="200" Width="150"
      Stroke="Black" >
      <Rectangle.Fill>
        <ImageBrush ImageSource="sampleImages\pinkcherries.jpg" />
      </Rectangle.Fill>
    </Rectangle>

    <!-- Thuộc tính Stretch="None" để hiển thị ảnh với kích thước gốc,
      không tự co giãn -->
    <Rectangle Canvas.Left="180" Canvas.Top="10" Height="150" Width="150"
      Stroke="Black" >
      <Rectangle.Fill>
        <ImageBrush ImageSource="sampleImages\pinkcherries.jpg"
          Stretch="None" />
      </Rectangle.Fill>
    </Rectangle>

    <!-- Thuộc tính Stretch="Uniform" để co giãn ảnh khớp với vùng tô
      nhưng giữ tỷ lệ ảnh gốc-->
    <Rectangle Canvas.Left="10" Canvas.Top="220" Height="100" Width="150"
      Stroke="Black" >
      <Rectangle.Fill>
        <ImageBrush ImageSource="sampleImages\pinkcherries.jpg"
          Stretch="Uniform"/>
      </Rectangle.Fill>
    </Rectangle>

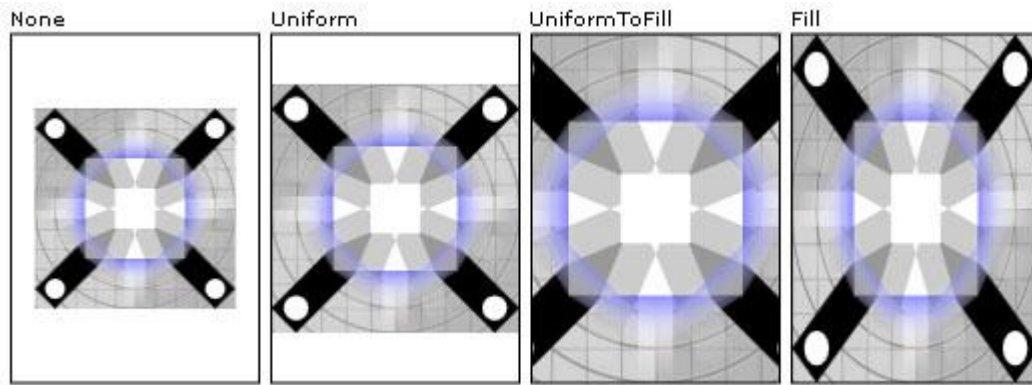
    <!-- Sử dụng thuộc tính Viewport để chỉ định khung nhìn cần tô
      Thuộc tính TileMode="FlipXY" để làm cho các khung nhìn
      tô kiểu xếp lợp đối xứng theo cả trục X và trục Y -->
    <Rectangle Canvas.Left="180" Canvas.Top="170" Height="150" Width="150"
      Stroke="Black" >
      <Rectangle.Fill>
        <ImageBrush ImageSource="sampleImages\pinkcherries.jpg"
          TileMode="FlipXY"
          Viewport="0,0,25,25" ViewportUnits="Absolute" />
      </Rectangle.Fill>
    </Rectangle>
  </Canvas>
</Window>
```

Thuộc tính **ImageSource** của **ImageBrush** để chỉ định đường dẫn đến tệp hình ảnh.

Thuộc tính **Stretch** để chỉ định các co giãn hình khi tô vùng, giá trị mặc định của **Stretch** là **Fill**. Thuộc tính này có các giá trị:

- **None**: Chỗi tô không tự động co giãn hình.
- **Uniform**: Chỗi tô co giãn hình trùng khít với một chiều của vùng tô nhưng giữ nguyên tỷ lệ của ảnh gốc.
- **UniformToFill**: Chỗi tô co giãn hình phủ kín vùng tô nhưng giữ nguyên tỷ lệ của ảnh gốc.
- **Fill**: Chỗi tô co giãn hình phủ kín vùng tô, không giữ tỷ lệ ảnh. Nếu tỷ lệ hai chiều của vùng tô khác với tỷ lệ hai chiều của ảnh thì ảnh tô sẽ bị méo.

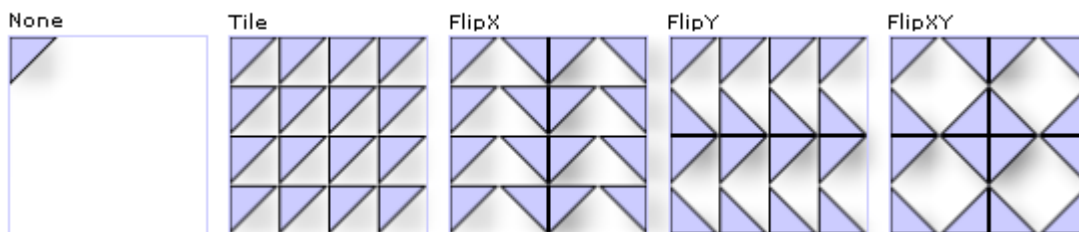
Các giá trị thuộc tính **Stretch** được minh họa như hình dưới đây.



Thuộc tính **TileMode** chỉ định kiểu xếp lặp, gồm các giá trị sau:

- **None**: Không xếp lặp.
- **Tile**: Xếp lặp để phủ kín vùng cần tô.
- **FlipX**: Xếp lặp để phủ kín vùng cần tô, hình được lật theo chiều ngang.
- **FlipY**: Xếp lặp để phủ kín vùng cần tô, hình được lật theo chiều dọc.
- **FlipXY**: Xếp lặp để phủ kín vùng cần tô, hình được lật theo cả hai chiều.

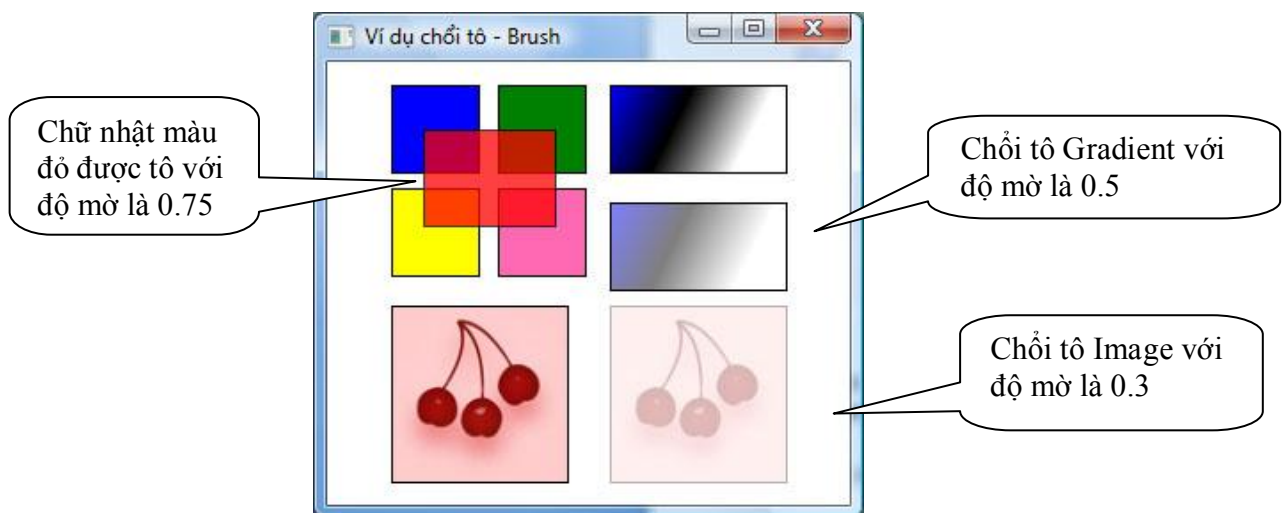
Các thuộc tính trên được minh họa bởi hình dưới.



## 2.5. Thiết lập độ mờ của chổi tô - Opacity

Để tạo độ mờ (**Opacity**), WPF cung cấp thuộc tính **Opacity** áp dụng cho các đối tượng hình học (Shape), hình ảnh hay RadientBrush,...

Ví dụ sau đây minh họa thiết lập độ mờ cho chổi tô.



Hình 8.9. Minh họa chổi tô với độ mờ

Mã XAML của ví dụ trên như sau:

```
<Window x:Class="Lesson08_Graphics.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Ví dụ chỗi tô - Brush" Height="285" Width="310">
  <Canvas Height="243" Width="242">

    <Rectangle Canvas.Left="10" Canvas.Top="10" Height="50" Width="50"
      Stroke="Black" Fill="Blue"/>
    <Rectangle Canvas.Left="70" Canvas.Top="10" Height="50" Width="50"
      Stroke="Black" Fill="Green"/>
    <Rectangle Canvas.Left="10" Canvas.Top="68" Height="50" Width="50"
      Stroke="Black" Fill="Yellow"/>
    <Rectangle Canvas.Left="70" Canvas.Top="68" Height="50" Width="50"
      Stroke="Black" Fill="HotPink"/>
    <!-- Hình chữ nhật được tô màu đỏ với độ Opacity = 0.75-->
    <Rectangle Canvas.Left="28" Canvas.Top="35" Height="55" Width="75"
      Stroke="Black" Fill="Red" Opacity="0.75"/>

    <!-- Hình chữ nhật tô đỏ màu kiểu linear gradient. -->
    <Rectangle Width="100" Height="50" Canvas.Left="133" Canvas.Top="10"
      Stroke="Black" StrokeThickness="1">
      <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0.75,0.75"
          Opacity="1">
          <GradientStop Color="Blue" Offset="0.0" />
          <GradientStop Color="Black" Offset="0.5" />
          <GradientStop Color="White" Offset="1" />
        </LinearGradientBrush>
      </Rectangle.Fill>
    </Rectangle>

    <Rectangle Canvas.Left="133" Canvas.Top="76" Height="50" Width="100"
      Stroke="Black" StrokeThickness="1" >
      <Rectangle.Fill>
        <LinearGradientBrush EndPoint="0.75,0.75" StartPoint="0,0"
          Opacity="0.5">
          <GradientStop Color="Blue" Offset="0.0" />
          <GradientStop Color="Black" Offset="0.5" />
          <GradientStop Color="White" Offset="1" />
        </LinearGradientBrush>
      </Rectangle.Fill>
    </Rectangle>

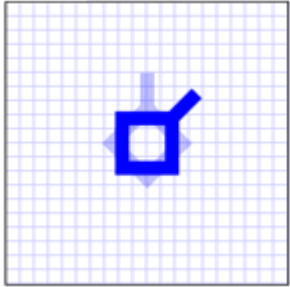
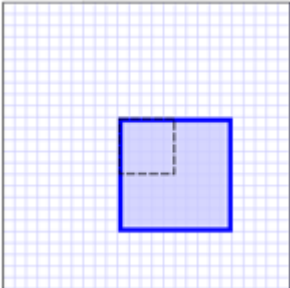
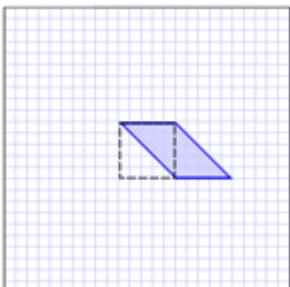
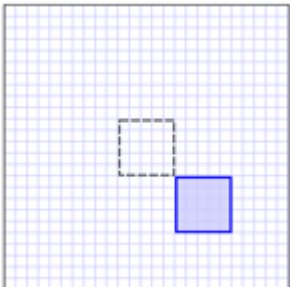
    <Rectangle Canvas.Left="10" Canvas.Top="134" Height="100" Width="100"
      Stroke="Black" >
      <Rectangle.Fill>
        <ImageBrush ImageSource="sampleImages\pinkcherries.jpg" />
      </Rectangle.Fill>
    </Rectangle>
    <!-- ảnh được tô với độ mờ 0.3 -->
    <Rectangle Canvas.Left="133" Canvas.Top="134" Height="100" Width="100"
      Stroke="Black" Opacity="0.3" >
      <Rectangle.Fill>
        <ImageBrush ImageSource="sampleImages\pinkcherries.jpg" />
      </Rectangle.Fill>
    </Rectangle>
  </Canvas>
</Window>
```

Giá trị của **Opacity** là một số thực từ 0 đến 1. Nếu **Opacity** càng gần bằng 0 thì chỗi tô là trong suốt, giá trị càng gần một thì chỗi tô càng rõ nét.

### 3. Biến đổi hình học – Transform

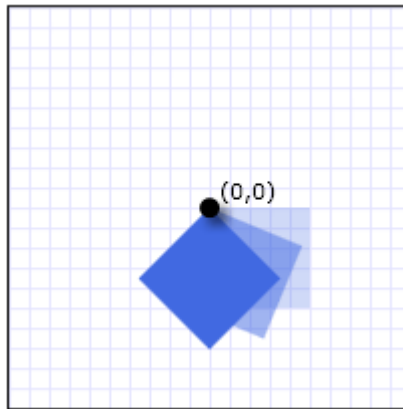
Trong phần này, chúng ta tìm hiểu cơ bản về các kỹ thuật biến đổi hình học với WPF như các phép dịch chuyển (translate), phép co giãn (scale), phép lệch hình (skew), phép quay (rotate)...

WPF cung cấp một số lớp (class) để hỗ trợ cho công việc biến đổi hình học

| Lớp (Class)               | Mô tả   | Minh họa  |
|---------------------------|---|---|
| <b>RotateTransform</b>    | Quay đối tượng theo góc chỉ định bởi thuộc tính <b>Angle</b> .                                |    |
| <b>ScaleTransform</b>     | Co giãn đối tượng theo chiều ngang và dọc với các thuộc tính <b>ScaleX</b> và <b>ScaleY</b> . |   |
| <b>SkewTransform</b>      | Làm lệch hình theo các góc chỉ định bởi <b>AngleX</b> và <b>AngleY</b> .                      |  |
| <b>TranslateTransform</b> | Dịch chuyển đối tượng tới vị trí chỉ định bởi <b>X</b> và <b>Y</b> .                          |  |

Khi biến đổi một đối tượng, ta không chỉ biến đổi bản thân nó mà còn biến đổi cả không gian tọa độ chứa đối tượng đó. Mặc định, tâm của phép biến đổi thực hiện tại gốc của hệ tọa độ của đối tượng.

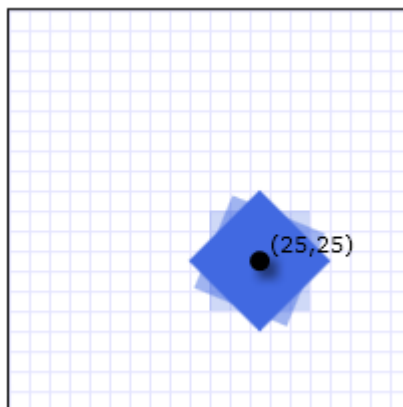
Ví dụ sau minh họa sử dụng phép quay để xoay một hình chữ nhật một góc 45 độ xung quanh tâm (0,0) của trục tọa độ của đối tượng (chính là góc trên trái của hình chữ nhật)



Mã lệnh XAML như sau:

```
<Canvas Width="200" Height="200">
  <Rectangle
    Canvas.Left="100" Canvas.Top="100"
    Width="50" Height="50"
    Fill="RoyalBlue" Opacity="1.0">
    <Rectangle.RenderTransform>
      <RotateTransform Angle="45" />
    </Rectangle.RenderTransform>
  </Rectangle>
</Canvas>
```

Ví dụ tiếp theo minh họa xoay hình chữ nhật một góc 45 độ quanh tâm của chính nó, có tọa độ là (25,25).



Mã lệnh XAML như sau:

```
<Canvas Width="200" Height="200">
  <Rectangle
    Canvas.Left="100" Canvas.Top="100"
    Width="50" Height="50"
    Fill="RoyalBlue" Opacity="1.0">
    <Rectangle.RenderTransform>
      <RotateTransform Angle="45" CenterX="25" CenterY="25"/>
    </Rectangle.RenderTransform>
  </Rectangle>
</Canvas>
```

Không chỉ dừng ở việc biến đổi các đối tượng hình học, WPF còn cho phép biến đổi các điều khiển (Control).

Ví dụ sau minh họa xoay một nút bấm một góc 45 độ theo chiều kim đồng hồ, tâm góc quay là tâm của nút bấm. Mặc định thì tâm góc quay là góc trên trái của nút bấm. Sử dụng thuộc tính `RenderTransformOrigin` của `Button` và thiết lập vị trí tâm quay. Cặp giá trị của `RenderTransformOrigin` là số thực từ 0 đến 1, được tính theo tỷ lệ chiều rộng và cao của đối tượng tính từ góc trên trái.

Sử dụng thuộc tính `RenderTransform` của `Button` và để thực hiện biến đổi hình học, góc quay được thiết lập nhờ thuộc tính `Angle` của đối tượng `RotateTransform`.



Mã lệnh XAML của ví dụ trên:

```
<Border Margin="30"
    HorizontalAlignment="Left" VerticalAlignment="Top"
    BorderBrush="Black" BorderThickness="1">
    <StackPanel Orientation="Vertical">
        <Button Content="A Button" Opacity="1" />
        <Button Content="Rotated Button"
            RenderTransformOrigin="0.5,0.5">
            <Button.RenderTransform>
                <RotateTransform Angle="45" />
            </Button.RenderTransform>
        </Button>
        <Button Content="A Button" Opacity="1" />
    </StackPanel>
</Border>
```

## 4. Câu hỏi ôn tập

### 1. Các đối tượng hình học có thể được xây dựng bởi?

- A. Mã lệnh XAML
- B. Mã lệnh C#
- C. Cả hai

Câu trả lời: C

### 2. Đối tượng *Polyline* gồm *N* cạnh được định nghĩa bởi bao nhiêu cặp tọa độ?

- A. *N* cặp tọa độ
- B. *N*+1 cặp tọa độ
- C. *N*-1 cặp tọa độ

Câu trả lời: B

### 3. Đối tượng *Rectangle* được định nghĩa bởi?

- A. Các thuộc tính Left, Top, Right, Bottom
- B. Các thuộc tính Left, Top, Width, Height

Câu trả lời: B

### 4. Đối tượng *Ellipse* được định nghĩa bởi?

- A. Các thuộc tính Tọa độ tâm của Ellipse và các bán kính của nó
- B. Hình chữ nhật ngoại tiếp của Ellipse.

Câu trả lời: B

### 5. Đối tượng *Polygon* gồm *N* đỉnh được định nghĩa bởi bao nhiêu cặp tọa độ?

- A. *N* cặp tọa độ
- B. *N*+1 cặp tọa độ

C. N-1 cặp tọa độ

*Câu trả lời: A*

**6. Các đối tượng hình học có thể được tô bên trong bởi:**

- A. Màu đồng nhất
- B. Tô đổ màu theo tuyến tính hoặc theo bán kính hình tròn
- C. Tô màu bởi hình ảnh có sẵn
- D. Đáp án A,B,C
- E. Đáp án A và B

*Câu trả lời: D*

**7. Khi sử dụng ảnh để tô màu cho đối tượng thì mặc định ảnh sẽ**

- A. Giữ nguyên kích thước gốc.
- B. Tự co giãn để phủ kín đối tượng.
- C. Giữ nguyên kích thước và tự động xếp lớp phủ kín đối tượng.
- D. Không đáp án nào đúng.

*Câu trả lời: B*

## 5. Tài liệu tham khảo

1. WPF Graphics, Animation, and Media Overview; <http://msdn.microsoft.com/en-us/library/ms742562.aspx> .
2. Shapes and Basic Drawing in WPF Overview; URL: <http://msdn.microsoft.com/en-us/library/ms747393.aspx> .
3. WPF Brushes Overview; URL: <http://msdn.microsoft.com/en-us/library/aa970904.aspx> .
4. Transforms Overview; URL <http://msdn.microsoft.com/en-us/library/ms750596.aspx> .