Transforms

Estimated time for completion: 45 minutes

Goals:

- Examine each of the transforms.
- Experience the difference between the Layout Transform and the Render Transform
- Apply Transforms in a group

Overview:

This lab will introduce the student to transformations which affect layout. Any element in WPF may have various transforms applied to it to affect how it is positioned and/or rendered. Throughout this lab you will apply various transforms to a shape to see how it impacts the rendering and layout.

Part 1 – The TranslateTransform

In this part you will play with the TranslateTransform.

Steps:

1. Open a new .XAML file with your favorite XML editor.
2. Add a new `Page` or `Window` element as your root object. The lab solution will use a `Page`.
    a. Make sure to include the appropriate WPF namespaces so you can create elements.
3. Add a `Canvas` as the root panel.
    a. Set the `Width` and `Height` to "300".
    b. Set the `Background` to "DarkGray".
4. Add a `Rectangle` to the `Canvas`.
    a. Set the `Width` and `Height` to "50".
    b. Set the `Fill` to "Yellow".
    c. Set the `Stroke` to "Black".
    d. Set the `StrokeThickness` to "3".

5. Your markup should look something like:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

   <Canvas Width="300" Height="300" Background="DarkGray">
      <Rectangle Width="50" Height="50" Fill="Yellow"
                 Stroke="Black" StrokeThickness="3" />
   </Canvas>

</Page>
```

6. Next, add a `RenderTransform` to the `Rectangle`.
7. In the `RenderTransform`, add a `TranslateTransform` and set the `X` and `Y` properties to "25".

```
   <Rectangle Width="50" Height="50" Fill="Yellow"
              Stroke="Black" StrokeThickness="3">
     <Rectangle.RenderTransform>
        <TranslateTransform X="25" Y="25" />
     </Rectangle.RenderTransform>
   </Rectangle>
```

8. The `Rectangle` should shift within the `Canvas` and now have the top/left corner positioned at (25,25).
9. Change the `X` and `Y` properties to be "-25".
10. Note how the rectangle moves above and to the left of the canvas position.
11. Finally, just to show the coordinate system in play here, change the `X`/`Y` properties to ".25" and "1.5" respectively. This will place a miniscule change in the position of the rectangle.

Part 2 – The RotateTransform

In this part you will play with the RotateTransform to create a "sun-like" shape.

Steps:

1. Open a new .XAML file with your favorite XML editor.
2. Add a new `Page` or `Window` element as your root object. The lab solution will use a `Page`.
    a. Make sure to include the appropriate WPF namespaces so you can create elements.
3. Add a `Grid` as the root panel element.
    a. We only want a single cell, so do not define any columns or rows.

4. Add a `Rectangle` to the Grid.
    a. Set the `Width` and `Height` to "40".
    b. Set the `Fill` to "Yellow".
    c. Set the `Stroke` to "Red".
    d. Set the `StrokeThickness` to "4".
5. Copy the `Rectangle` definition and add it a second time to the `Grid`. Now the second `Rectangle` is overlaying the first – it is the only visible element at this point.
6. Apply a `RenderTransform` the Rectangle.
7. Add a `RotateTransform` to the Rectangle – set the `Angle` to "20" to rotate it by 20 degrees.
8. Notice how the rectangle rotated, but didn't stay centered on the original rectangle? This is because the center point for the rotation is set to the top/left corner (0,0) by default. In order to keep the center point stationary, we need to adjust the rotational center.
    a. Set the "`CenterX`" and "`CenterY`" properties to "20". The rectangle should shift over and now be overlaying the original rectangle, but rotated.
9. Create 3 more rotated rectangles – at **40**, **60** and **80** degrees respectively.
10. Finally, add an `Ellipse` as the final element.
    a. Set the `Width` and `Height` to be "40".
    b. Set the `Fill` to "Orange".
    c. Set the `Stroke` to "Red".
    d. Set the `StrokeThickness` to "4".
11. Sit back and bask in the sunshine!

Part 3 – The SkewTransform

In this part you will play with the SkewTransform. The SkewTransform allows the X and Y positions to be skewed independently.

Steps:

1. Open a new .XAML file with your favorite XML editor.
2. Add a new `Page` or `Window` element as your root object. The lab solution will use a `Page`.
3. Add a `Canvas` as the root tag.
4. Add a `Button` and position it at (100,150) using the `Canvas.Left` and `Canvas.Top` attached properties.
    a. Set the `Content` of the button to some text – the lab will use "This is a button".

b. Set the `FontSize` to "36pt" to make the button larger.

5. Apply a Render Transform to the `Button` and inside that, add a `SkewTransform`.
   a. Set the `AngleY` to "-20" to rotate the `Y` angle by 20 degrees counter-clockwise.
6. Notice that the button continues to function properly – you can still click on it even though it is now a parallelogram in shape.
7. Play with the `AngleX` and `AngleY` properties to see what can be done to the shape.

```
<Page xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
xmlns:x='http://schemas.microsoft.com/winfx/2006/xaml'>

  <Canvas>
    <Button FontSize="36pt" Canvas.Top="150" Canvas.Left="100"
            Content="This is a button">
       <Button.RenderTransform>
          <SkewTransform AngleY="-20" />
       </Button.RenderTransform>
    </Button>
  </Canvas>

</Page>
```

Part 4 – The ScaleTransform

In this part you will play with the ScaleTransform.

Steps:

1. Open a new .XAML file with your favorite XML editor.
2. Add a new `Page` or `Window` element as your root object. The lab solution will use a `Page`.
3. Add a `WrapPanel` as the root panel.
   a. Set the `Orientation` to "Horizontal".
4. Add a `TextBlock`
   a. Set the `Margin` to be "5" and the `Text` to be "Enter some text" or whatever you like.
5. Add a `TextBox` to the `WrapPanel`.
   a. Set the `Margin` to be "5" and the `MinWidth` to be "100".
6. Add a `Button` to the `WrapPanel`
   a. Set the `Margin` to be "5" and the `Content` to be "OK".
7. Your markup should look something like:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

   <WrapPanel Orientation="Horizontal">
      <TextBlock Margin="5" Text="Enter some text" />
      <TextBox Margin="5" MinWidth="100" />
      <Button Margin="5" >OK</Button>
   </WrapPanel>

</Page>
```

8. Next, apply a `RenderTransform` to the `WrapPanel`.
9. In the `RenderTransform`, add a `ScaleTransform` setting the `ScaleX` and `ScaleY` properties to "5".

```
<WrapPanel Orientation="Horizontal">
   ...
   <WrapPanel.RenderTransform>
      <ScaleTransform ScaleX="5" ScaleY="5" />
   </WrapPanel.RenderTransform>

</WrapPanel>
```

10. Notice how everything is scaled smoothly – no jagged edges here!
11. Notice also, how everything still works properly – you can type in the TextBox and click the button.
12. Next, change the `ScaleY` to be "-5". Add a `CenterY` property and set it to "50" so that you can see the effect here.
13. Now, the entire panel is flipped on the Y axis – and yet still works properly.

Part 5 – Grouping Transforms

In this part you will add multiple transforms to the scene to see how the impact each other.

Steps:

1. Using the bit of XAML you created in Part 4, remove the `CenterY` property from the `ScaleTransform`.
    a. The panel might not be visible now because it is flipped "up". Can you think of a way to "shift" the panel in the Y direction?
2. Let's move the panel through a `TranslateTransform`. Add the `TranslateTransform` directly below the `ScaleTransform`.
    a. Set the `Y` property to "200".

3. The XAML is no longer valid because you cannot add more than one transform to an element. In order to group transforms, we need an additional class involved – the `TransformGroup`.
4. Add a `TransformGroup` between the `RenderTransform` tag and the `ScaleTransform`. The `TransformGroup` should be the direct parent for both of the transforms.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

   <WrapPanel Orientation="Horizontal">
      <TextBlock Margin="5" Text="Enter some text" />
      <TextBox Margin="5" MinWidth="100" />
      <Button Margin="5" >OK</Button>

      <WrapPanel.RenderTransform>
         <TransformGroup>
            <ScaleTransform ScaleX="5" ScaleY="-5" />
            <TranslateTransform Y="200" />
         </TransformGroup>
      </WrapPanel.RenderTransform>
   </WrapPanel>
</Page>
```

5. Now the panel should be visible.
6. Add additional transforms and play with it to get a sense of what you can do with this mechanism.

Part 6 – Layout vs. Render Transforms

Up to this point, you have been using a render transform to affect the visual appearance of the output. In this part you will experiment with layout vs. render transforms to discover when it is appropriate to use a specific style of transform.

Steps:

1. Open a new .XAML file with your favorite XML editor.
2. Add a new `Page` or `Window` element as your root object. The lab solution will use a `Page`.
    a. Make sure to include the appropriate WPF namespaces so you can create elements.
3. Add a `StackPanel` as the root panel.
4. Add an Image to the `StackPanel`.
    a. Set the `Source` to be any image on your machine – the lab here will use "C:\Windows\web\wallpaper\img1.jpg" but any image will do. If you are using

Internet Explorer to run your XAML (instead of something like XAMLPad) then you will need to point it to an image in the same directory due to security restrictions.

b. Set the `Width` of the image to be "200".

5. Add a `Slider` directly below the image.
   a. Give the slider a name, the lab will use the name "slider1".
   b. Set the `Width` to be "200"
   c. Set the `Minimum` property to be "0".
   d. Set the `Maximum` property to be "360".
   e. Set the `TickFrequency` property to be "10".
   f. Set the `TickPlacement` property to be "BottomRight".

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

   <StackPanel>

     <Image Source="c:\windows\web\wallpaper\img1.jpg" Width="200" />
     <Slider Name="slider1" Minimum="0" Maximum="360"
             Width="200" TickFrequency="50" TickPlacement="BottomRight" />

   </StackPanel>

</Page>
```

6. Next, add a `RenderTransform` to the image.
7. Inside that, add a `RotateTransform`.
   a. Set the `CenterX` and `CenterY` properties to "100". This will cause the image to rotate about the center approximately.
   b. Now we're going to give you a peek at an upcoming session – data binding. The goal is to bind the rotate transform's `Angle` property to the slider value. To accomplish this, use the following `Angle` definition for the `RotateTransform`:

```
<Image.RenderTransform>
   <RotateTransform Angle="{Binding ElementName=slider1, Path=Value}"
                    CenterX="100" CenterY="100"/>
</Image.RenderTransform>
```

8. Move the slider to the right. What happens when the image hits the slider?

9. Now change the `RenderTransform` to a `LayoutTransform` – everything else remains the same, simply change the start and end tags.
10. Perform the same test – what does the slider do now? Can you explain why?