Creating a 360 Degree Panorama in Silverlight

360 Degree Panorama photos are often done using QuickTimes VR or Flash to give a 360 perspective. However they are just as easy to create in Silverlight meaning you can embed them in to your application to enhance maps and create virtual tours.

The first step is to create you cylindrical panorama image. This is a single image that represents a 360 view. These can be created with a standard digital camera, a number of photos and some stitching software to put them all together. Hugin Panorama photo stitcher is a cross platform open source photo stitching application that will allow you to easily stitch together your final image although there are many others available.

Once you have your finished image it should look something like this:


Full 360 Degree Cylindrical Panorama of London

* Note: This image is a creative commons image I am using for the sample and not taken by myself.

Next split the image into two separate images of equal size you using a photo editing software and add to your silverlight project. This is essential as the code behind will scroll the two images to the left. Then as one of the images falls off the left hand side of the canvas it is placed to the far right ready to scroll into view again. This action creates a seamless scrolling 360 degree view.

In the XAML place the two images on a canvas side by side. For this to work you must ensure that the canvas is no wider than one of the image halves. You must also clip the canvas to the same size to prevent the user from seeing the images as they scroll off the edge.

```
&lt;Canvas x:Name=&quot;Frame&quot; Width=&quot;400&quot; Height=&quot;200&quot;&gt;

     &lt;Canvas.Clip&gt;
              &lt;RectangleGeometry Rect=&quot;0, 0, 400, 200&quot;/&gt;
      &lt;/Canvas.Clip&gt;

        &lt;Image x:Name=&quot;tile2&quot; Source=&quot;Images/LeftImage.jpg&quot;
 Stretch=&quot;None&quot; Canvas.Left=&quot;0&quot; Canvas.Top=&quot;0&quot;  /&gt;

        &lt;Image x:Name=&quot;tile1&quot; Source=&quot;Images/RightImage.jpg&quot;
Stretch=&quot;None&quot; Canvas.Left=&quot;400&quot; Canvas.Top=&quot;0&quot;/&gt;

      &lt;/Canvas&gt;
```

The animation and calculations to move the images into a loop is done using the CompositionTarget.Rendering event. This is fired each time the screen updates (so speed will depend on the fps). This method is used instead of storyboard animations so that we can reset the images positions when necessary.

```
public MainPage()
        {
            InitializeComponent();
            CompositionTarget.Rendering += new EventHandler(CompositionTarget_Rendering);
        }

private void CompositionTarget_Rendering(object sender, EventArgs e) {

            if (_direction== Directions.Left)
            {
                if ((Canvas.GetLeft(tile1) + tile1.ActualWidth) &lt; 0)
                {
                   //If tile1 fallen off the left
                    Canvas.SetLeft(tile1, Canvas.GetLeft(tile2) + tile2.ActualWidth);
                }
                else if ((Canvas.GetLeft(tile2) + tile2.ActualWidth) &lt; 0)
                {
                    //If tile2 fallen off the left
                    Canvas.SetLeft(tile2, Canvas.GetLeft(tile1) + tile1.ActualWidth);
                }

                //Move the tiles along left
                Canvas.SetLeft(tile1, Canvas.GetLeft(tile1) - _speed);
                Canvas.SetLeft(tile2, Canvas.GetLeft(tile2) - _speed);

            } else if (_direction== Directions.Right) {

                if (Canvas.GetLeft(tile1) &gt; Frame.ActualWidth )
                {
                    //If tile1 fallen off the right
                    Canvas.SetLeft(tile1, Canvas.GetLeft(tile2) - tile1.ActualWidth);
                }
                else if (Canvas.GetLeft(tile2) &gt; Frame.ActualWidth)
                {
                    //If tile2 fallen off the right
                    Canvas.SetLeft(tile2, Canvas.GetLeft(tile1) - tile2.ActualWidth);
                }

                //Move the tiles along right
                Canvas.SetLeft(tile1, Canvas.GetLeft(tile1) + _speed);
                Canvas.SetLeft(tile2, Canvas.GetLeft(tile2) + _speed);
            }
        }
```

Finally three buttons set the _direction value which sets how the animation is shown giving us our finished sample. Note that as the image scrolls left the user is given the impression of looking right. Therefore the button with the left arrow should set the _direction to Right and visa versa.