

Graphics and Multimedia

Overview

- Benefits of WPF graphics:
 - Resolution and device-independent
 - □ Improved precision
 - Advanced graphics and animation support
 - □ Hardware acceleration
- Available graphics elements:
 - □ 2-D shapes
 - □ 2-D geometries
 - □ 2-D effects
 - □ 3-D rendering
 - Animation
 - Media: images, video, and audio

Graphics Rendering

- Visual the basic abstraction from which every
 FrameworkElement object derives
 - □ It is a core WPF object, whose primary role is to provide rendering support
 - □ It provides support for: output display, transformations, clipping, hit testing, and bounding box calculations
 - □ It does not provide support for: event handling, layout, styles, data binding, and globalization
 - A Visual object stores its render data as a vector graphics instruction list
 - there are 4 types of render data: vector graphics, image, glyph, and video

Graphics Rendering cont.

- DrawingVisual used to render shapes, images, or text
 It can be used to create a custom visual object
- Viewport3DVisual a bridge between 2D Visual and Visual3D objects
 - □ It requires to define a **Camera** value and a **Viewport** value
- ContainerVisual a container for a collection of Visual objects
- **Drawing** an abstract class describing a 2-D drawing
 - A base class for: DrawingGroup, GeometryDrawing, GlyphRunDrawing, ImageDrawing, VideoDrawing

Graphics Rendering cont.

- DrawingGroup describes operations that are applied to its contents in the following order:
 - **1. OpacityMask**
 - 2. Opacity
 - 3. BitmapEffect
 - 4. ClipGeometry
 - 5. GuidelineSet
 - 6. Transform
- DrawingContext populates a Visual with visual content acquired from certain methods, e.g. DrawingGroup.Open and DrawingVisual.RenderOpen

Visual Rendering Behaviour

- Application objects that have a visual appearance define a set of serialized drawing data
 - The system is responsible for responding to all repaint requests for rendering the application objects
 - WPF can efficiently optimize what needs to be redrawn in the application
- WPF uses vector graphics as its rendering data format
- WPF supports automatic scaling by using the device independent pixel as its primary unit of measurement
 - Logical pixel equates to 1/96 of an inch (which is consistent with standard resolution of Windows)
 - Graphics and text scale properly without any extra work from the application developer

Transformations

• A **Transform** is a 3x3 matrix:

M11	M12	0.0
M21	M22	0.0
OffsetX	OffsetY	0.1

- RotateTransform, ScaleTransform, SkewTransform, TranslateTransform
- The Transform class inherits from the Animatable class, so it can be animated
- To apply multiple transforms to an UI element, use a TransformGroup

Krzysztof Mossakowski Faculty of Mathematics and Information Science <Button Content="A Button" RenderTransformOrigin="0.5,0.5"> <Button.RenderTransform> <RotateTransform</pre> x:Name="AnimatedRotateTransform" Angle="0" /> </Button.RenderTransform> <Button.Triggers> <EventTrigger</pre> RoutedEvent="Button.Click"> <BeginStoryboard> <Storyboard> **CoubleAnimation** Storyboard.TargetName= "AnimatedRotateTransform" Storyboard.TargetProperty= "Angle" To="360" Duration="0:0:1" FillBehavior="Stop" /> </storyboard> </BeginStoryboard> </EventTrigger> </Button.Triggers> </Button>

Bitmap Effects

- A bitmap effect takes a BitmapSource as an input and produces a new BitmapSource after applying the effect
- As a special case, in WPF, effects can be set as properties on live Visual objects
 - In this case, at the time of rendering, a Visual is automatically converted to its
 BitmapSource, so the output replaces the Visual object's default rendering
- Available effects (custom effects can also be created):
 - BlurBitmapEffect
 - OuterGlowBitmapEffect
 - DropShadowBitmapEffect
 - BevelBitmapEffect
 - EmbossBitmapEffect

Krzysztof Mossakowski Faculty of Mathematics and Information Science

Beveled Button

<BevelBitmapEffect BevelWidth="15" EdgeProfile="CurvedIn" LightAngle="320" Relief="0.4" Smoothness="0.4" />



Blurred Button

1		Supplier of the	

<OuterGlowBitmapEffect GlowColor="Blue"
Opacity="0.4" GlowSize="30" Noise="1" />
...



<DropShadowBitmapEffect Color="Black"
Direction="320" ShadowDepth="25" Softness="1"
Opacity="0.5" />

_				

http://www

Bitmap Effects Example

<TabControl> <Tabltem Header="None"> <Canvas> <Button>Button</Button> <TextBlock Canvas.Left="50"> Text in a textblock</TextBlock> <Image Canvas.Top="35"</pre> Source="/Images/Fiona 67x77.gif"/> </Canvas> </Tabltem> <Tabltem Header="Blur"> <Canvas> <Canvas.BitmapEffect> <BlurBitmapEffect/> </Canvas.BitmapEffect> <Button>Button</Button> <TextBlock Canvas.Left="50"> Text in a textblock</TextBlock> <Image Canvas.Top="35"</pre> Source="/Images/Fiona 67x77.gif"/> </Canvas> </Tabltem> [...] </TabControl>



Brushes

- Everything visible on a screen is visible because it was painted by a brush
- Available brushes:
 - SolidColorBrush
 - LinearGradientBrush
 - RadialGradientBrush
 - ImageBrush (uses a ImageSource)
 - DrawingBrush (uses a Drawing which can contain shapes, images, text, and media)
 - VisualBrush (uses a Visual object, e.g. Button)
- Common brush features:
 - Opacity
 - □ Transform, RelativeTransform
 - □ can be animated
 - □ can be frozen (i.e. cannot be modified)

Krzysztof Mossakowski

Faculty of Mathematics and Information Science

Gradient Brushes Examples







```
Windows Programming
```

```
<StackPanel Canvas.Top="300" Canvas.Left="50">
  <Border Name="ReflectedVisual">[...]</Border>
  <Rectangle Height="1" Fill="Gray" HorizontalAlignment="Stretch" />
  <Rectangle Height="{Binding Path=ActualHeight, ElementName=ReflectedVisual}"</pre>
               Width="{Binding Path=ActualWidth, ElementName=ReflectedVisual}">
    <Rectangle.Fill>
      <VisualBrush Opacity="0.75" Stretch="None"</pre>
                                                                      My text in a box
                                                                               My button
           Visual="{Binding ElementName=ReflectedVisual}">
        <VisualBrush.RelativeTransform>
                                                                      My text in a box My butto
          <TransformGroup>
            <ScaleTransform ScaleX="1" ScaleY="-1" />
            <TranslateTransform Y="1" />
          </TransformGroup>
                                                                      My text in a box
                                                                               My button
        </VisualBrush.RelativeTransform>
                                                                      My text in a box My butto
      </VisualBrush>
    </Rectangle.Fill>
    <Rectangle.OpacityMask>
      <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
        <GradientStop Color="#FF000000" Offset="0.0" />
        <GradientStop Color="#44000000" Offset="0.5" />
        <GradientStop Color="#00000000" Offset="0.9" />
      </LinearGradientBrush>
    </Rectangle.OpacityMask>
    <Rectangle.BitmapEffect>
      <BlurBitmapEffect Radius="1.5" />
    </Rectangle.BitmapEffect>
  </Rectangle>
</StackPanel>
```

Lecture 9 - 12

Windows Programming

TileBrush Examples

TileMode



Krzysztof Mossakowski Faculty of Mathematics and Information Science



Viewport, ViewportUnits



http://www.mini.pw.edu.pl/~mossakow

Windows Programming

Lecture 9 - 14

Opacity Masks

Without Opacity Mask



The Opacity Mask



With Opacity Mask





Without Opacity Mask

The Opacity Mask



With Opacity Mask



Without Opacity Mask



The Opacity Mask



With Opacity Mask





Krzysztof Mossakowski Faculty of Mathematics and Information Science

http://www.mini.pw.edu.pl/~mossakow

Brush Transformation

- Algorithm used for processing and transforming a brush:
 - Process the brush's contents 1
 - Project the brush's output 2. onto the 1 x 1 transformation rectangle
 - 3. Apply the brush's **RelativeTransform**, if it has one
 - Project the transformed 4. output onto the area to paint
 - 5. Apply the brush's Transform, if it has one

Krzysztof Mossakowski **Faculty of Mathematics and Information Science**

http://www.mini.pw.edu.pl/~mossakow

DrawingBrush (Tiled)





Shapes

Krz

Facu

The Shape class is a base for all available shape objects:
 Ellipse, Line, Path, Polygon, Polyline, and Rectangle

Lecture 9 - 16

- Common properties of shape objects:
 - □ Stroke how the shape's outline is painted
 - □ **StrokeThickness** thickness of the shape's outline
 - □ Fill how the interior of the shape is painted

```
□ Transform
                                      PointCollection myPointCollection =
                                          new PointCollection();
 □ Stretch
                                      myPointCollection.Add(new Point(0, 0));
                                      myPointCollection.Add(new Point(0, 1));
                                      myPointCollection.Add(new Point(1, 1));
<Polygon
                                      Polygon myPolygon = new Polygon();
   Points="0,0 0,1 1,1"
                                      myPolygon.Points = myPointCollection;
                                      myPolygon.Fill = Brushes.Blue;
    Fill="Blue"
                                      myPolygon.Width = 100;
    Width="100"
                                      myPolygon.Height = 100;
    Height="100"
                                      myPolygon.Stretch = Stretch.Fill;
    Stretch="Fill"
                                      myPolygon.Stroke = Brushes.Black;
    Stroke="Black"
    StrokeThickness="2" />
                                      myPolygon.StrokeThickness = 2;
```

Geometry

- The Geometry class and the classes which derive from it enable to describe the geometry of a 2-D shape
- Geometry objects are more versatile than Shape objects
 - □ A **Shape** object is used to render 2-D graphics
 - A Geometry object can be used to define the geometric region for 2-D graphics, define a region for clipping, or define a region for hit testing, for example
- The **Path** shape uses a **Geometry** to describe its contents
 - It can be used to render a Geometry (by setting the Data, Fill, and Stroke properties)

Available Geometries

- Simple geometry types:
 - LineGeometry, RectangleGeometry, EllipseGeometry
- A PathGeometry contains a collection of PathFigure objects. Each
 PathFigure object can contain segments:
 - LineSegment, PolyLineSegment, ArcSegment, BezierSegment, PolyBezierSegment, QuadraticBezierSegment, PolyQuadraticBezierSegment
- A StreamGeometry defines a complex geometric shape that may contain curves, arcs, and lines
 - The contents of a StreamGeometry do not support data binding, animation, or modification (unlike a PathGeometry)
- The CombinedGeometry object and the Combine method performs a boolean operation to combine the area defined by two geometries
- The GeometryGroup class creates an amalgamation of the Geometry objects it contains without combining their area

PathGeometry Example



Faculty of Mathematics and Information Science

Windows Programming

Path Markup Syntax

<Path Stroke="Black" Fill="Gray" Data="M 10,100 C 10,300 200,-200 200,100 V 200 H 10 Z" />

- M move
- L line
- H horizontal line
- V vertical line
- C cubic Bezier curve
- Q quadratic Bezier curve
- S smooth Bezier curve
- A elliptical arc
- Z close path



Imaging

- WPF Imaging includes a codec for BMP, JPEG, PNG, TIFF, Windows Media Photo, GIF, and ICON image formats
 ICON is the only format without encoding support
- A BitmapSource represents a single, constant set of pixels at a certain size and resolution
 - It can be an individual frame of a multiple frame image, or it can be the result of a transform performed on a BitmapSource
- A BitmapFrame is used to store the actual bitmap data of an image format
- Access to image metadata is provided through the Metadata property of a BitmapSource object

3-D Graphics

- 3-D graphics content in WPF is encapsulated in an element,
 Viewport3D, that can participate in the two-dimensional element structure
 - □ Viewport3D is a surface on which a 3-D scene is projected
 - 2-D and 3-D objects cannot be interpenetrated within a Viewport3D
 - Coordinate systems for 2-D and 3-D graphics differ:



Available 3-D transformations (inherited from the abstract Transform3D class): TranslateTransform3D, ScaleTransform3D, RotateTransform3D

3-D Graphics cont.

- Camera allows to specify the onlooker's point of view for a 3-D scene
 - ProjectionCamera allows to specify different projections and their properties to change how the onlooker sees 3-D models
 - PerspectiveCamera specifies a projection that foreshortens the scene
- Model3D is the abstract base class that represents a generic 3-D object. The objects that make up the scene graph derive from this class
 - GeometryModel3D creates a 3-D model comprised of a MeshGeometry3D and a Material
- Available lights: AmbientLight, DirectionalLight, PointLight, SpotLight

```
<!-- Viewport3D is the rendering surface. -->
<Viewport3D Name="myViewport" >
                                                              3DContent
  <!-- Add a camera. -->
  <Viewport3D.Camera>
    <PerspectiveCamera FarPlaneDistance="20"</pre>
          LookDirection="0,0,1" FieldOfView="45"
          UpDirection="0,1,0" NearPlaneDistance="1"
          Position="0,0,-3" />
  </Viewport3D.Camera>
  <!-- Add models. -->
  <Viewport3D.Children>
    <ModelVisual3D>
      <ModelVisual3D.Content>
        <Model3DGroup >
          <Model3DGroup.Children>
            <!-- Lights, MeshGeometry3D and DiffuseMaterial -->
            <DirectionalLight Color="#FFFFFFF" Direction="3,-4,5" />
            <!-- Define a red cone. -->
            <GeometryModel3D>
              <GeometryModel3D.Geometry>
                <MeshGeometry3D [...] />
              </GeometryModel3D.Geometry>
              <GeometryModel3D.Material>
                ConffuseMaterial
                  ConffuseMaterial.Brush
                    <SolidColorBrush Color="Red" Opacity="1.0"/>
                  </DiffuseMaterial.Brush>
                </DiffuseMaterial>
              </GeometryModel3D.Material>
```

[...]

Animation

- WPF includes an efficient timing system that is exposed through managed code and XAML and that is deeply integrated into the WPF framework
 - WPF animation makes it easy to animate controls and other graphical objects
- In WPF, object are animated by applying animation to their properties. A property can be animated if:
 - □ It is a dependency property
 - □ It belongs to a class that inherits from **DependencyObject** and implements **IAnimatable**
 - □ There is a compatible animation type available
- Animations can be used almost anywhere, which includes in styles and control templates
 - □ Animations do not have to be visual

Animation Types

<Type>Animation (e.g. DoubleAnimation or ColorAnimation)

□ It animate between a starting (**From**) and destination value (**To**), or by adding an offset value (**By**) to its starting value

<Type>AnimationUsingKeyFrames

Any number of target values and their interpolation method can be specified

<Type>AnimationUsingPath

Allows to use a geometric path in order to produce animated values

<Type>AnimationBase

□ Abstract class which can be used to create custom animations

Timeline

- All the animation types inherit from the **Timeline** class
- A Timeline defines a segment of time and allows to specify the following properties:
 - □ **Duration** (the default is 1 second)
 - AutoReverse whether a timeline playes backward after it reaches the end of its duration
 - RepeatBehavior specifies how many times a timeline plays (the default is 1.0)
 - FillBehavior specifies what should be done with the value of the animated property when the animation stops
- The real work of keeping time control is done by the timeline's Clock
 - A Clock provides information essential to the animation: CurrentTime, CurrentProgress, and CurrentState

Animation Example – Fade In and Out

- 1. Create a **DoubleAnimation** (it creates a transition between two double values)
- 2. Create a **Storyboard** (use the **TargetName** property to specify the object to animate and **TargetProperty** to specify the property to animate)
- 3. Associate the **Storyboard** with a trigger

```
<Rectangle Name="MyRectangle" Width="100" Height="100" Fill="Blue">
<Rectangle.Triggers>
<EventTrigger RoutedEvent="Rectangle.Loaded">
<BeginStoryboard>
<Storyboard>
<DoubleAnimation Storyboard.TargetName="MyRectangle"
<Storyboard.TargetProperty="Opacity" From="1.0" To="0.0"
<Duration="0:0:5" AutoReverse="True" RepeatBehavior="Forever" />
</Storyboard>
</BeginStoryboard>
</Rectangle.Trigger>
</Rectangle.Triggers>
</Rectangle>
```

Faculty of Mathematics and Information Science

Key-Frame Animation

- Available interpolation methods
 - Linear interpolation the animation progresses at a constant rate of the segment duration
 - Discrete interpolation the animation function jumps from one value to the next without interpolation
 - Splined interpolation the animation is based on a Bezier curve which describes the rate of change for that spline key frame
 - Two points of the curve are constant (0.0, 0.0), (1.0, 1.0), the other two can be set using the KeySpline property
- Key frames with different interpolation types can be used in a single key frame animation
- The KeyTime property of the animation key frame's specifies when that key frame ends
 - □ It is possible to specify the **KeyTime** property's value in percentage
 - The special value "Paced" can be used when a constant rate is desired

Key-Frame Animation Examples

```
<Storyboard>

<DoubleAnimationUsingKeyFrames

Storyboard.TargetName="MyAnimatedTranslateTransform"

Storyboard.TargetProperty="X"

Duration="0:0:10">

<LinearDoubleKeyFrame Value="0" KeyTime="0:0:0" />

<LinearDoubleKeyFrame Value="350" KeyTime="0:0:2" />

<LinearDoubleKeyFrame Value="50" KeyTime="0:0:7" />

<LinearDoubleKeyFrame Value="200" KeyTime="0:0:8" />

</DoubleAnimationUsingKeyFrames>

</Storyboard>
```

```
<Storyboard>

<DoubleAnimationUsingKeyFrames

Storyboard.TargetName="ComboAnimatedTranslateTransform"

Storyboard.TargetProperty="X"

Duration="0:0:15"

RepeatBehavior="Forever">

<DiscreteDoubleKeyFrame Value="500" KeyTime="0:0:7" />

<LinearDoubleKeyFrame Value="200" KeyTime="0:0:10" />

<SplineDoubleKeyFrame Value="350" KeyTime="0:0:15"

KeySpline="0.25,0.5 0.75,1" />

</DoubleAnimationUsingKeyFrames>

</Storyboard>
```

Path Animation

- A path animation uses a **PathGeometry** as its input
 - As the path animation progresses, it reads the x, y, and angle information from the path and uses that information to generate its output
- There are 3 typed of path animations:
 - A MatrixAnimationUsingPath generates Matrix values from its PathGeometry
 - one way to move an object along a path is to use a MatrixTransform and a MatrixAnimationUsingPath to transform an object along a complex path
 - the DoesRotateWithTangent property can be used to force the object to rotate along the tangent of the path
 - A PointAnimationUsingPath generates Point values from the xand y-coordinates of its PathGeometry

A DoubleAnimationUsingPath generates Double values from its PathGeometry

Path Animation Example

```
<Button MinWidth="100" Content="A Button">
  <Button.RenderTransform>
    <MatrixTransform x:Name="ButtonMatrixTransform"/>
  </Button.RenderTransform>
  <Button.Triggers>
    <EventTrigger RoutedEvent="Button.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <MatrixAnimationUsingPath</pre>
                 Storyboard.TargetName="ButtonMatrixTransform"
                 Storyboard.TargetProperty="Matrix"
                 DoesRotateWithTangent="True"
                 Duration="0:0:5"
                 RepeatBehavior="Forever" >
            <MatrixAnimationUsingPath.PathGeometry>
              <PathGeometry</pre>
                 Figures="M 10,100 C 35,0 135,0 160,100 180,190 285,200 310,100"
                 PresentationOptions:Freeze="True" />
            </MatrixAnimationUsingPath.PathGeometry>
          </MatrixAnimationUsingPath>
        </storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Button.Triggers>
</Button>
```

Property Animation Techniques

- There are 4 ways to animate a property of an object:
 - □ Storyboard animation
 - allows to define and apply animations in XAML, interactively control animations after they start, create a complex tree of animations, or animate in a Style, ControlTemplate or DataTemplate
 - □ Local animation
 - provide a convenient way to animate a dependency property of any **Animatable** object
 - □ Clock animation
 - allows to create complex timing trees or interactively control animations after they start
 - □ Per-frame animation
 - useful when there is a need to completely bypass the WPF animation system, e.g. physics animations

Krzysztof Mossakowski Faculty of Mathematics and Information Science

http://www.mini.pw.edu.pl/~mossakow

Storyboard

- A Storyboard is a type of container timeline that provides targeting information for the timelines it contains
- It can be used to animate dependency properties of animatable classes
- To apply animations to their targets, begin the Storyboard using a trigger action or a method
 - In XAML, use a BeginStoryboard object with an EventTrigger, Trigger, or DataTrigger

□ In code, use the **Begin** method

The Storyboard can be controlled using code or XAML
 Available operations: pause, resume, seek, skipToFill, stop,

remove

Lecture 9 - 35

Timing Events

There are 5 timing events provided by the Timeline and Clock classes:

Completed, CurrentGlobalSpeedInvalidated, CurrentStateInvalidated, CurrentTimeInvalidated, RemoveRequested

```
myStoryboard.Stop(myRectangle);
// This statement might execute before the storyboard has stopped.
myRectangle.Fill = Brushes.Blue;
```

```
// Register for the CurrentStateInvalidated timing event.
myStoryboard.CurrentStateInvalidated +=
    new EventHandler(myStoryboard_CurrentStateInvalidated);
// Change the rectangle's color after the storyboard stops.
void myStoryboard_CurrentStateInvalidated(object sender, EventArgs e) {
    Clock myStoryboardClock = (Clock)sender;
    if (myStoryboardClock.CurrentState == ClockState.Stopped) {
        myRectangle.Fill = Brushes.Blue;
    }
```

Freezable

A Freezable is a special type of object that has two states: unfrozen and frozen

□ When frozen, a **Freezable** can no longer be modified

- Freezing a Freezable can improve its performance, because it no longer needs to spend resources on change notifications
 - A frozen Freezable can also be shared across threads, while an unfrozen Freezable cannot
- To freeze the object, call the **Freeze** method
 - To avoid throwing an InvalidOperationException, check the value of the Freezable object's CanFreeze property before calling Freeze

□ Use the **IsFrozen** property can be used to check if the object

Multimedia

- There are 2 classes that can be used to present audio, video, and video with audio content; both rely on a minimum of the Microsoft Windows Media Player 10 OCX for media playback
 - MediaElement is a UIElement that can be consumed as the content of many controls
 - it is usable in XAML and code
 - □ **MediaPlayer** is designed for **Drawing** objects
 - media loaded using a MediaPlayer can only be presented using a VideoDrawing or by directly interacting with a DrawingContext
 - it cannot be used in XAML
- Two media modes are supported:
 - □ Independent mode media content drives media playback

Clock mode – a **MediaTimeline** drives media playback

Faculty of Mathematics and Information Science

http://www.mini.pw.edu.pl/~mossakow

MediaElement

- To play media in an application, just add a MediaElement control to the user interface and provide a Uri to the media
- The LoadedBehavior and UnloadedBehavior properties control the behaviour of the MediaElement when IsLoaded is true or false, respectively
- To display a MediaElement it must have content to render and it will have its ActualWidth and ActualHeight properties set to zero until content is loaded
 - For video content, once the MediaOpened event has been raised the ActualWidth and ActualHeight will report the size of the loaded media
 - Setting both the Width and Height properties will cause the media to stretch to fill the area provided for the MediaElement
 - To preserve the media's original aspect ratio, either the Width or Height property should be set but not both

```
<StackPanel Name="MainPanel" Background="Black">
  <MediaElement Name="myMediaElement" MediaOpened="Element MediaOpened" />
  <StackPanel HorizontalAlignment="Center" Width="260" Orientation="Horizontal">
    <Button Name="PlayButton" Margin="30,10,10,10">Play</Button>
    <Button Name="PauseButton" Margin="10">Pause</Button>
    <Button Name="ResumeButton" Margin="10">Resume</Button>
    <Button Name="StopButton" Margin="10">Stop</Button>
  </StackPanel>
  <Slider Name="timelineSlider" Margin="5" HorizontalAlignment="Center" />
  <StackPanel.Triggers>
    <EventTrigger RoutedEvent="Button.Click" SourceName="PlayButton">
      <EventTrigger.Actions>
        <BeginStoryboard Name= "myBegin">
          <Storyboard SlipBehavior="Slip">
            <MediaTimeline Source="Movies\TestMovie.wmv"</pre>
                           Storyboard.TargetName="myMediaElement"
                           CurrentTimeInvalidated="MediaTimeChanged" />
          </storyboard>
        </BeginStoryboard>
      </EventTrigger.Actions>
    </EventTrigger>
    <EventTrigger RoutedEvent="Button.Click" SourceName="PauseButton">
      <EventTrigger.Actions>
        <PauseStoryboard BeginStoryboardName="myBegin" />
      </EventTrigger.Actions>
    </EventTrigger>
    ſ...1
  </StackPanel.Triggers>
</StackPanel>
```

Faculty of Mathematics and Information Science

MediaPlayer

```
MediaTimeline timeline = new MediaTimeline(
    new Uri(@"Movies\TestMovie.wmv", UriKind.Relative));
timeline.RepeatBehavior = RepeatBehavior.Forever;
// Create the clock, which is shared with the MediaPlayer
MediaClock clock = timeline.CreateClock();
MediaPlayer player = new MediaPlayer();
player.Clock = clock;
// Create the VideoDrawing
VideoDrawing videoDrawing = new VideoDrawing();
videoDrawing.Rect = new Rect(0, 0, 1, 1);
videoDrawing.Player = player;
// Assign the DrawingBrush
DrawingBrush brush = new DrawingBrush(videoDrawing);
brush.Viewport = new Rect(0, 0, 0.5, 0.25);
```

brush.TileMode = TileMode.Tile;

```
// Start the timeline
clock.Controller.Begin();
```

```
MainCanvas.Background = brush;
```

<Canvas Name="MainCanvas" Background="Pink" />