11. Разработайте приложение, в котором будет несколько элементов для вывода видео. Все элементы повернуты под разными углами и частично перекрывают друг друга (например, используйте расположение «веером»). Реализуйте функцию проигрывания одного видео с текущей позиции при наведении курсора мышки на этот элемент.

12. Реализуйте игру «составь кадры в хронологическом порядке». Программа выбирает случайным образом кадры из видеопотока. Задача игрока отгадать последовательность воспроизведения кадров в видео.

13. Разработайте программу поиска «ключевых» кадров в видеопотоке. Ключевые кадры это кадры, при которых меняется полностью вся сцена. Для выявления таких кадров используйте признаки резкой смены яркости и контрастности.

14. Создайте приложение, которые отображает ряд кадров из указанного видео и при нажатии на полученный кадр начинает воспроизведение видео с этого кадра.

15. Разработайте приложение, воспроизводящее видео на странице книги. При нажатии на книгу страница переворачивается, а на новой странице воспроизводится видео через одну минуту от текущего. Тем самым реализуется возможность перемотки.

16. Разработайте программу поиска кадров в видеопотоке с какой-либо преобладающей цветовой компонентой. Например, кадров где много синего цвета.

Лабораторная работа №3. Анимация и видеоэффекты в WPF

Работа с областью просмотра

Одной из возможностей для создания видеоэффектов является воспроизведение видео в области, определённой по какой либо фигуре. Для этого необходимо задать свойство `Clipping` объекта `MediaElement`. Ниже следующий XAML код показывает возможность воспроизведения видео в эллипсе.

```
<MediaElement Name="media" Source="skazka.avi"
Height="200" Width="300">
    <MediaElement.Clip>
            <EllipseGeometry Center="150,100"
        RadiusX="120" RadiusY="80" />
    </MediaElement.Clip>
    </MediaElement>
```

Для задания более сложной области воспроизведения можно использовать свойство GeometryGroup, позволяющее объединить несколько фигур в одну область. Например, окружность вместе с эллипсом:

```
<MediaElement.Clip>
    <GeometryGroup FillRule="Nonzero">
        <EllipseGeometry Center="200,100"
        RadiusX="50" RadiusY="50" />
        <RectangleGeometry Rect="75,50,125,100"/>
    </GeometryGroup>
</MediaElement.Clip>
```

Областью, по которой происходит обрезание видео, можно управлять динамически в коде. Рассмотрим пример, когда окружность будет плавать слева направо по кадру видео. Опишем в программе булевскую переменную FRMove, которая будет отвечать за направление движения области (true – движение осуществляется вправо, false – влево) и точку Point – центр окружности.

```
private Boolean FRMove = true;
private Point ellipsePoint = new Point(0, 100);
```

В XAML код, в этом случае, не нужно добавлять элемент EllipseGeometry поскольку он будет порождаться в программе динамически на каждый тик таймера.

```
if (ellipsePoint.X == media.Width) FRMove = false;
if (ellipsePoint.X == 0) FRMove = true;
if (FRMove) ellipsePoint.X += 1;
    else ellipsePoint.X -= 1;
media.Clip = new EllipseGeometry(ellipsePoint, 50, 50);
```

Добавление эффектов

Кроме свойства Clipping объекта MediaElement можно использовать свойства Effect, или BitmapEffect которое позволяет накладывать различные видеоэффекты при воспроизведении. Нижеприведенный код, включаемый в MediaElement, позволяет создать эффект размытия.

```
<MediaElement.Effect>
    <BlurEffect Radius="5"/>
</MediaElement.Effect>
```

Следующий код демонстрирует использование свойства `BitmapEffect` для создания красной тени от всего `MediaElement`.

```
<MediaElement.BitmapEffect>
    <DropShadowBitmapEffect Color="Red"/>
</MediaElement.BitmapEffect>
```

Базовая анимация

WPF имеет огромные возможности по работе с графикой и макетом приложения, что позволяет создавать привлекательные пользовательские интерфейсы и документы. Применение анимации позволяет сделать пользовательский интерфейс еще более наглядным и удобным в использовании.

Анимация — это имитация изменений, которая обеспечивается быстрым показом серии слегка отличающихся друг от друга изображений. Мозг человека воспринимает группу изображений как одну непрерывно изменяющуюся картинку. В фильмах такой эффект достигается за счет применения камер, записывающих множество фотографий (кадров) в секунду. При воспроизведении кадров зрители видят движущееся изображение.

Анимация на компьютере выполняется по аналогичному принципу. Например, программа, в которой реализуется исчезновение прямоугольника, может работать следующим образом.

В программе создается таймер.

1. Через заданные временные интервалы проверяется значение таймера для определения истекшего времени.
2. При каждой проверке значения таймера вычисляется текущее значение непрозрачности для прямоугольника в зависимости от прошедшего времени.
3. Затем прямоугольник обновляется с использованием нового значения и перерисовывается.

До появления WPF, Windows разработчики были вынуждены создавать и поддерживать собственные системы управления временем либо использовать специальные пользовательские библиотеки. В WPF входит эффективная система контроля времени, которая доступна посредством управляемого кода и языка XAML.

Рассмотрим небольшой пример, анимирующий прямоугольник.

```
<StackPanel Margin="20">
    <Rectangle Name="MyRectangle"
               Width="100" Height="100">
        <Rectangle.Fill>
            <SolidColorBrush x:Name="MySolidBrush"
                             Color="Blue" />
        </Rectangle.Fill>
        <Rectangle.Triggers>
        <EventTrigger RoutedEvent="Rectangle.MouseEnter">
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation
                    Storyboard.TargetName="MyRectangle"
                    Storyboard.TargetProperty="Width"
                    From="100" To="200"
                    Duration="0:0:1" />

                    <ColorAnimation
                    Storyboard.TargetName="MySolidBrush"
                    Storyboard.TargetProperty="Color"
                    From="Blue" To="Red"
                    Duration="0:0:1" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
        </Rectangle.Triggers>
    </Rectangle>
</StackPanel>
```

В данном примере в контейнере `StackPanel` описан прямоугольник размером 100x100 пикселей и с именем `MyRectangle`. Далее среди триггеров для этого прямоугольника описывается *триггер события* (EventTrigger). Фактически триггер события позволяет описать обработчик какого либо события внутри XAML кода. В нашем примере описан обработчик события наведения мышки на прямоугольник. При создании триггера события нужно указать маршрутизируемое событие, которое запускает триггер, и действие (или действия), выполняемые триггером. В случае анимации наиболее часто используемым действием является `BeginStoryboard`, который фактически запускает саму анимацию.

Далее внутри триггера события описывается объект, который называется *раскадровка* (`Storyboard`). Раскадровка (`storyboard`) — это усовершенствованная временная шкала. Ее можно применять для группирования множества анимаций (в нашем примере две) и, кроме того, она позволяет управлять воспроизведением анимации — приостанавливать ее, прекращать и изменять текущую позицию. Однако самым базовым средством, предлагаемым классом `StoryBoard`, является способность указывать на определенное свойство и определенный элемент, используя свойства `TargetProperty` и `TargetName`. Другими словами, раскадровка заполняет пробел между анимацией и свойством, для которого должна осуществляться анимация. Свойство `Storyboard.TargetProperty` идентифицирует свойство, которое должно изменяться (в данном случае — `Width` и `Color`). Если имя класса не указано, то раскадровка использует родительский элемент, которым является расширяемый прямоугольник. Исходя из этого строка `Storyboard.TargetName= "MyRectangle"` может отсутствовать.

Внутри раскадровки используется анимация с помощью класса `DoubleAnimation`. Объект `DoubleAnimation` используется для создания перехода между двумя значениями типа `Double`. В нашем случае меняется ширина прямоугольника. Чтобы задать начальное значение, устанавливается свойство `From`. Чтобы задать конечное значение, устанавливается свойство `To`. Свойство `Duration` анимации определяет длительность перехода от начального к конечному значению.

Кроме анимации размера прямоугольника в раскадровке присутствует анимация цвета с помощью класса `ColorAnimation`. Приведенные анимации работают на основе линейной интерполяции, в ходе которой изменяются свойства последовательно от начального до конечного значения. Такая анимация называется *базовой* или *From/To/By*. WPF предоставляет ряд классов для *From/To/By* анимации. Имена этих классов формируются как `<Тип>Animation`. `<Тип>` — тип значения, для которого класс выполняет анимацию.

Следующий код демонстрирует, каким образом можно использовать базовую анимацию непосредственно в WPF коде.

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Shapes;
using System.Windows.Media.Animation;
```

```
namespace AnimationWpfApplication2
{
public partial class MainWindow : Window
    {
    private Storyboard myStoryboard;
    public MainWindow()
    {
        InitializeComponent();
        StackPanel myPanel = new StackPanel();
        myPanel.Margin = new Thickness(10);

        Rectangle myRectangle = new Rectangle();
        myRectangle.Name = "myRectangle";
        this.RegisterName(myRectangle.Name,
                          myRectangle);
        myRectangle.Width = 100;
        myRectangle.Height = 100;
        myRectangle.Fill = Brushes.Blue;

        DoubleAnimation myDoubleAnimation =
        new DoubleAnimation();
        myDoubleAnimation.From = 1.0;
        myDoubleAnimation.To = 0.0;
        myDoubleAnimation.Duration =
        new Duration(TimeSpan.FromSeconds(5));
        myDoubleAnimation.AutoReverse = true;
        myDoubleAnimation.RepeatBehavior =
        RepeatBehavior.Forever;

        myStoryboard = new Storyboard();
        myStoryboard.Children.Add(myDoubleAnimation);
        Storyboard.SetTargetName(myDoubleAnimation,
                                 myRectangle.Name);
        Storyboard.SetTargetProperty(myDoubleAnimation,
            new PropertyPath(Rectangle.OpacityProperty));

        myRectangle.Loaded += new
        RoutedEventHandler(myRectangleLoaded);
        myPanel.Children.Add(myRectangle);
        this.Content = myPanel;
        }
```

```
        private void myRectangleLoaded(object sender,
                                       RoutedEventArgs e)
        {
            myStoryboard.Begin(this);
        }
    }
}
```

Кроме базовой анимации в WPF возможно использовать анимацию по ключевым кадрам и анимацию с использованием пути.

Анимация по ключевым кадрам

Анимация с использованием ключевых кадров является более эффективным средством, чем анимация класса *From/To/By*, поскольку при ее использовании можно задать любое число конечных значений, а также управлять методами их интерполяции. Некоторые типы могут быть анимированы только с помощью анимации с полными кадрами.

Подобно анимации From/To/By, анимация по ключевым кадрам анимирует значение заданного свойства. Она создает переход между заданными значениями с заданным временем (`Duration`). Однако если анимация From/To/By создает переход между двумя значениями, то анимация по ключевым кадрам может создавать переходы между любым числом целевых значений. В отличие от анимации From/To/By, анимация по полным кадрам не содержит свойств `From`, `To` или `By`, с которыми требуется задать ее заданные значения. Анимация по ключевым кадрам задает значения, которые описаны с помощью объектов ключевых кадров. Чтобы задать целевые значения анимации, создаются объекты ключевых кадров и они добавляются в коллекцию элементов анимации `KeyFrames`. При запуске анимации она выполняет переход между указанными кадрами.

В дополнение к поддержке нескольких целевых значений, некоторые методы полных кадров поддерживают даже несколько методов интерполяции. Метод интерполяции анимации определяет, как она переходит от одного значения к следующему. Существуют три типа интерполяции: дискретная, линейная и сплайновая.

Чтобы создать анимацию по ключевым кадрам, выполните следующие действия.

1. Объявите анимацию и задайте ее `Duration`, так же как для анимации From/To/By.

2. Для каждого значения создайте ключевой кадр соответствующего типа, задайте его значение и `KeyTime` и добавьте в коллекцию `KeyFrames` анимации.
3. Свяжите анимацию со свойством так же, как в анимации From/To/By.

Рассмотрим пример, где квадрат будет двигаться по замкнутой траектории с поворотом вокруг своей оси:

```
<Border Width="400" BorderBrush="Black">
    <Rectangle  Fill="Blue"
                Width="50" Height="50"
                HorizontalAlignment="Left">
        <Rectangle.RenderTransform>
            <TransformGroup>
                <TranslateTransform x:Name="MyTranslate"
                                    X="0" Y="0" />
                <RotateTransform x:Name="MyRotate"
                                 Angle ="0"/>
            </TransformGroup>
        </Rectangle.RenderTransform>
        <Rectangle.Triggers>
            <EventTrigger
            RoutedEvent="Rectangle.MouseLeftButtonDown">
                <BeginStoryboard>
                    <Storyboard>
                        <DoubleAnimationUsingKeyFrames
                        Storyboard.TargetName="MyTranslate"
                        Storyboard.TargetProperty="X"
                        Duration="0:0:6"
                        RepeatBehavior="Forever">

                            <LinearDoubleKeyFrame Value="0"
                                   KeyTime="0:0:0" />
                            <LinearDoubleKeyFrame Value="350"
                                   KeyTime="0:0:2" />
                            <LinearDoubleKeyFrame Value="350"
                                   KeyTime="0:0:4" />
                            <LinearDoubleKeyFrame Value="0"
                                   KeyTime="0:0:6" />
                        </DoubleAnimationUsingKeyFrames>

                        <DoubleAnimationUsingKeyFrames
```

```
                    Storyboard.TargetName="MyTranslate"
                    Storyboard.TargetProperty="Y"
                    Duration="0:0:6"
                    RepeatBehavior="Forever">
                        <LinearDoubleKeyFrame Value="0"
                               KeyTime="0:0:0" />
                        <LinearDoubleKeyFrame Value="150"
                               KeyTime="0:0:2" />
                       <LinearDoubleKeyFrame Value="-150"
                               KeyTime="0:0:4" />
                        <LinearDoubleKeyFrame Value="0"
                               KeyTime="0:0:6" />
                    </DoubleAnimationUsingKeyFrames>

                    <DoubleAnimationUsingKeyFrames
                    Storyboard.TargetName="MyRotate"
                    Storyboard.TargetProperty="Angle"
                    Duration="0:0:6"
                    RepeatBehavior="Forever">
                        <LinearDoubleKeyFrame Value="0"
                               KeyTime="0:0:0" />
                        <LinearDoubleKeyFrame Value="120"
                               KeyTime="0:0:2" />
                        <LinearDoubleKeyFrame Value="240"
                               KeyTime="0:0:4" />
                        <LinearDoubleKeyFrame Value="360"
                               KeyTime="0:0:6" />
                    </DoubleAnimationUsingKeyFrames>
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Rectangle.Triggers>
    </Rectangle>
</Border>
```

В данном примере аналогично использованию классов `DoubleAnimation` и `ColorAnimation` используется класс `DoubleAnimationUsingKeyFrames`, внутри которого описываются объекты ключевых кадров. В нашем примере это `LinearDoubleKeyFrame`. Имя объекта ключевого кадра формируется в соответствии со следующим соглашением об именах:

```
<InterpolationMethod><Тип>KeyFrame
```

где <InterpolationMethod> — метод интерполяции, который использует полный кадр, а <Тип> — тип значения, которое класс анимирует.

Основной целью при описании ключевого кадра является задание KeyTime и Value. Свойство Value определяет значение для ключевого кадра. Свойство KeyTime определяет, когда (в пределах выполнения анимации (Duration)) достигается значение ключевого кадра Value.

Анимация на основе пути

Анимация на основе пути (path-based animation) модифицирует значение в соответствии с фигурой, описанной в объекте PathGeometry, и в первую очередь применяется для перемещения элемента по некоторому пути. Анимация с использованием пути очень полезна для анимации объекта по сложной траектории. Рассмотрим пример движения эллипса вдоль прямоугольника:

```
<Window.Resources>
    <PathGeometry x:Key="MyPath">
        <PathFigure IsClosed="True">
            <PolyLineSegment
                Points="0,0 220,0 220,175 0,175" />
        </PathFigure>
    </PathGeometry>
</Window.Resources>

<Canvas Width="300" Height="300">
    <Path Stroke="Red" Data="{StaticResource MyPath}" />
    <Ellipse Width="30" Height="30" Fill="Blue">
        <Ellipse.RenderTransform>
            <TranslateTransform
                x:Name="MyTransform"  />
        </Ellipse.RenderTransform>

        <Ellipse.Triggers>
            <EventTrigger RoutedEvent="Path.Loaded">
                <BeginStoryboard>
                    <Storyboard RepeatBehavior="Forever">

                    <DoubleAnimationUsingPath
                    Storyboard.TargetName="MyTransform"
                    Storyboard.TargetProperty="X"
```

```
                    PathGeometry="{StaticResource MyPath}"
                    Source="X"
                    Duration="0:0:5"  />

                    <DoubleAnimationUsingPath
                    Storyboard.TargetName="MyTransform"
                    Storyboard.TargetProperty="Y"
                    PathGeometry="{StaticResource MyPath}"
                    Source="Y"
                    Duration="0:0:5"  />

                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger>
        </Ellipse.Triggers>
    </Ellipse>
</Canvas>
```

Путь, по которому движется эллипс, описывается с помощью класса `PathGeometry`. С помощью `PathGeometry` можно описать сложную фигуру, состоящую из дуг, кривых и линий, которая может быть как разомкнутой, так и замкнутой.

Анимация на основе пути описывается аналогичным образом, что и базовая анимация или анимация с использованием ключевых кадров. При этом используется объект `DoubleAnimationUsingPath` со свойством `PathGeometry`, через которое и подключается путь.

Более подробно создание анимации рассматривается в MSDN (http://msdn.microsoft.com/ru-ru/library/ms752312(v=vs.110).aspx).

Задания к лабораторной работе

1. Разработайте программу, которая при воспроизведении видео, показывает случайным образом области воспроизведения в виде увеличивающихся прямоугольников. Количество одновременно показанных прямоугольников постепенно увеличивается, как и их размер, что приводит постепенно к тому, что видео воспроизводится во все окне.

2. Создайте приложение для отображения видео в окне. Областью воспроизведения является несколько звезд, плавно меняющих свое местоположение и размеры.

3. Реализуйте приложение, в котором пользователь может самостоятельно определить область воспроизведения видео, выбирая и располагая на окне прямоугольники и овалы произвольной величины.

4. Создайте приложение, воспроизводящее видео в окне. Область воспроизведения - круг, плавающий по окну. Реализуйте упругие столкновения с границами окна. Скорость движения и радиус круга может меняться динамически во время воспроизведения.

5. Создайте приложение для отображения видео в окне. Областью воспроизведения является пульсирующее сердце. При воспроизведении плавно меняется размытие.

6. Создайте анимацию на основе пути падения снежинок, листьев и пр. с помощью XAML разметки.

7. Разработайте дизайн приложения с динамически меняющимися элементами. При наведении на элемент может меняться цвет, форма элемента, текстовая надпись, появляться или скрываться тень и т.п.

8. Создайте видеопроигрыватель с вращающейся областью просмотра. Вращение реализуйте через анимацию в XAML разметке.

9. Реализуйте через анимацию в XAML разметке анимацию катящегося мяча по различным наклонным поверхностям. Мяч должен вращаться и двигаться с различной скоростью.

10. Разработайте приложение анимирующее движение видео по ряду кривых Безье внутри окна. Анимация должна реализовываться через XAML разметку.

11. Создайте приложение анимирующее движение вращающегося куба по заданной траектории. При этом используйте только XAML разметку.

12. Разработайте приложение на основе XAML, анимирующее движение корабля по волнам. В сцене должны присутствовать анимированные чайки, облака и солнце.

13. Разработайте программу на основе XAML, анимирующее движение планет в солнечной системе. В сцене должны присутствовать анимированное движение спутников вокруг планет.

14. Реализуйте анимацию батальной сцены с помощью XAML разметки. В сцене должны присутствовать все три типа анимации срабатывающие на различные события.