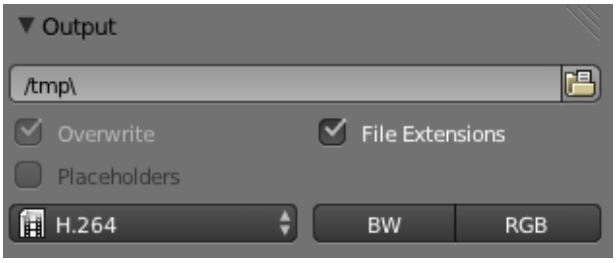

Рис. 1.30. Параметры импорта

Задания к лабораторной работе

Разработайте в Blender свою анимированную 3D сцену, состоящую из нескольких объектов. Используйте различные типы: 3D объектов, текстур, источников освещения. При анимации попробуйте изменять положение камеры.

Лабораторная работа №2. Создание видеоплеера средствами XAML и WPF

Воспроизведение медиаданных

Главным элементом управления в WPF для воспроизведения видео и звуковых данных является `MediaElement`, который фактически служит оболочкой функциональности класса `MediaPlayer`. Подобно всем элементам, `MediaElement` помещается непосредственно в пользовательский интерфейс. В случае применения `MediaElement` для воспроизведения аудио его расположение не имеет значения, но если воспроизводится видео, он должен быть размещен там, где планируется отображаться видео-окно.

Простейший дескриптор `MediaElement` – это все, что понадобится для воспроизведения звука. Например, если добавить следующую разметку к пользовательскому интерфейсу:

```
<MediaElement Source="test.mp3"></MediaElement>
```

то аудиофайл test.mp3 будет воспроизведен немедленно после загрузки.

Для управления воспроизведением из программы необходимо определить свойство `LoadedBehavior`. По умолчанию это свойство равно `Play`, но можно также использовать `Manual` – в этом случае файл загружается, а за запуск воспроизведения в нужный момент отвечает код.

Кроме того, понадобится выбрать имя через свойство Name, чтобы можно было взаимодействовать с медиа-элементом в коде. Обычно взаимодействие предусматривает вызов очевидных методов Play(), Pause() и Stop(). Также можно установить свойство Position, чтобы перемещаться по аудиозаписи.

Свойства MediaOpened и MediaEnded позволяют определить обработчики событий, которые вызываются по завершению загрузки медиаданных в память и при завершении воспроизведения соответственно.

Свойство Stretch позволяет определить способ заливки прямоугольной области MediaElement при воспроизведении видео.

Таким образом, XAML-разметка для воспроизведения видео без свойств выравнивания может выглядеть следующим образом:

```
<MediaElement Name="media" Source="sladkaya.skazka.avi"
    LoadedBehavior="Manual" Stretch="Fill"
    MediaOpened="MediaOpened" MediaEnded="MediaEnded"/>
```

Дополним эту разметку тремя слайдерами (Slider) для управления громкостью, скоростью воспроизведения и местом воспроизведения, тремя кнопками Play, Stop, Pause и меткой для указания времени от начала воспроизведения в секундах (рис. 1.1.).

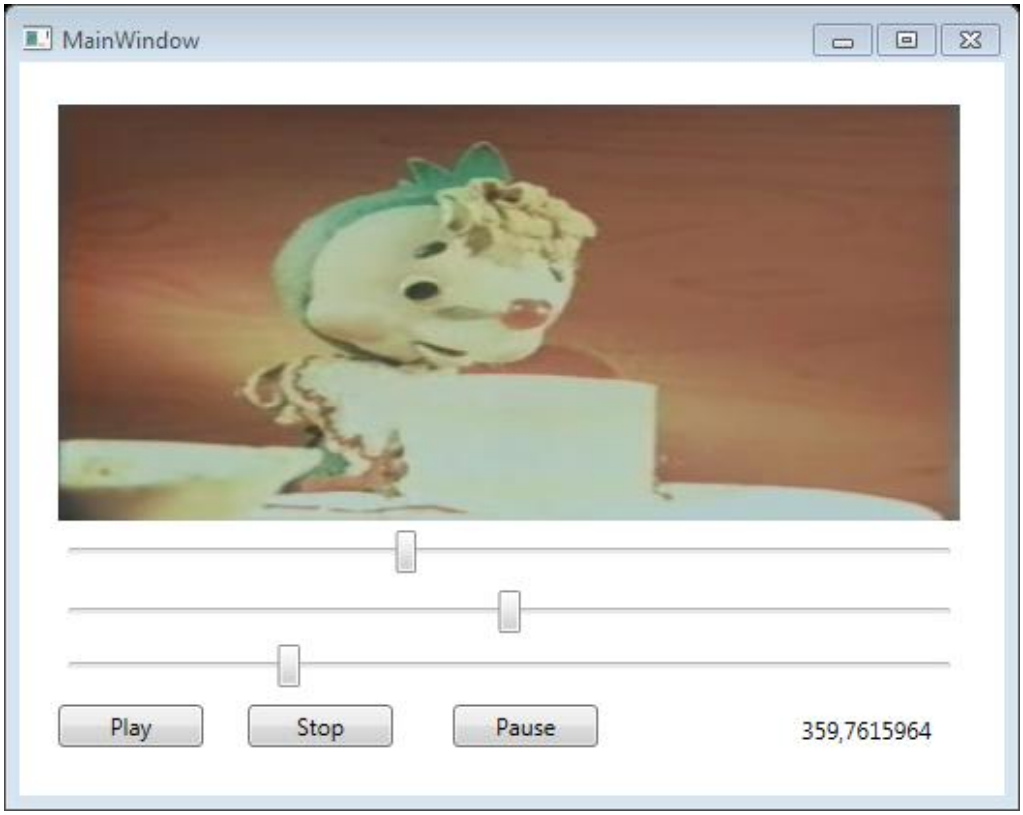

Рис. 1.1. Окно простейшего медиа-проигрывателя

Описание кнопок в XAML разметке без свойств выравнивания и задания размеров может выглядеть следующим образом:

```
<Button Content="Play" Click="Button_Click_1"/>
<Button Content="Stop" Click="Button_Click_2"/>
<Button Content="Pause" Click="Button_Click_3"/>
```

Описание слайдеров в XAML разметке без свойств выравнивания и задания размеров будет выглядеть следующим образом:

```
<Slider Name="timelineSlider"
    ValueChanged="ChangeMediaPosition"
    Minimum="0" />
<Slider Name="volumeSlider"
    ValueChanged="ChangeMediaVolume"
    Minimum="0"
    Maximum="1"
    Value="0.5" />
<Slider Name="speedSlider"
```

```
    ValueChanged="ChangeSpeedRatio"
    Minimum="0.5"
    Maximum="4"
    Value="1" />
```

Первый слайдер служит для представления позиции в видео, так называемая *тайм линия*. Минимальное значение определено как ноль, максимальное будет определяться в коде. Второй элемент служит для задания громкости в пределах от нуля (без звука) до единицы, с предустановленной громкостью 0,5. Третий элемент определяет скорость воспроизведения от 0,5 до четырех.

Свойство `ValueChanged` определяет обработчик события, которое происходит при изменении значения слайдера.

Теперь рассмотрим код реализующий логику медиапроигрывателя. Начнем с простых действий по нажатию кнопок, реализующих управление воспроизведением:

```
// Play
private void Button_Click_1(object sender,
    RoutedEventArgs e)
{
    media.Play();
    InitializePropertyValues();
}

// Stop
private void Button_Click_2(object sender,
    RoutedEventArgs e)
{
    media.Stop();
}

// Pause
private void Button_Click_3(object sender,
    RoutedEventArgs e)
{
    media.Pause();
}

void InitializePropertyValues()
{
```

```
        media.Volume = (double)volumeSlider.Value;
        media.SpeedRatio = (double)speedSlider.Value;
    }
```

Как мы видим в методе `InitializePropertyValues` считываются данные со слайдеров, отвечающих за громкость, скорость воспроизведения и устанавливаются соответствующие свойства `MediaElement`. Аналогичным образом реализованы обработчики событий изменения значений этих слайдеров:

```
    private void ChangeMediaVolume(object sender,
        RoutedPropertyChangedEventArgs<double> args)
    {
        media.Volume = (double)volumeSlider.Value;
    }

    private void ChangeSpeedRatio(object sender,
        RoutedPropertyChangedEventArgs<double> args)
    {
        media.SpeedRatio = (double)speedSlider.Value;
    }
```

При открытии медиаданных (событие `MediaOpened`), исходя из их длительности, настраиваем максимальное значение слайдера для управления позицией воспроизведения.

```
    private void MediaOpened(object sender, EventArgs e)
    {
        timelineSlider.Maximum =
            media.NaturalDuration.TimeSpan.TotalSeconds;
    }
```

При окончании воспроизведения (событие `MediaEnded`) останавливаем воспроизведение и устанавливаем позицию воспроизведения на начало.

```
    private void MediaEnded(object sender, EventArgs e)
    {
        media.Stop();
        flag = false;
        timelineSlider.Value = 0;
```

```
}
```

Для управления текущей позицией воспроизведения требуется более тонкая настройка. В первую очередь необходимо перемещать бегунок по слайдеру при проигрывании медиаданных. Это можно решить через создание таймера, на тик которого будет корректироваться позиция слайдера исходя из текущей позиции в видео:

```
// Подготавливаем таймер к работа
MyTimer = new DispatcherTimer();
MyTimer.Tick += new EventHandler(MyTimer_Tick);
MyTimer.Interval = new TimeSpan(100000);
MyTimer.Start();

private void MyTimer_Tick(object sender, EventArgs e)
{
    timelineSlider.Value = media.Position.TotalSeconds;
    Label1.Content =
        media.Position.TotalSeconds.ToString();
}
```

Попытка создания обработчика изменения значения слайдера таймлинии например таким образом:

```
private void ChangeMediaPosition(object sender,
    RoutedPropertyChangedEventArgs<double> args)
{
    media.Position =
        TimeSpan.FromSeconds(timelineSlider.Value);
}
```

приведет к невозможности воспроизведения. Тик таймера в этом случае будет изменять позицию бегунка, что вызовет обработчик `ChangeMediaPosition`, где изменится опять позиция внутри видео, что приведет к коллапсу. Эту ситуацию можно разрешить введением булевского флага, который будет временно запрещать изменение позиции внутри видео потока. В этом случае код будет выглядеть следующим образом.

```
private void ChangeMediaPosition(object sender,
    RoutedPropertyChangedEventArgs<double> args)
```

```
{
    if (!flag) media.Position =
        TimeSpan.FromSeconds(timelineSlider.Value);
}

private void MyTimer_Tick(object sender, EventArgs e)
{
    flag = true;
    timelineSlider.Value = media.Position.TotalSeconds;
    Label1.Content =
        media.Position.TotalSeconds.ToString();
    flag = false;
}

private void Window_Loaded_1(object sender,
    RoutedEventArgs e)
{
    // Подготоваливаем таймер к работа
    MyTimer = new DispatcherTimer();
    MyTimer.Tick += new EventHandler(MyTimer_Tick);
    MyTimer.Interval = new TimeSpan(100000);
    MyTimer.Start();
    timelineSlider.AddHandler(MouseLeftButtonUpEvent,
        new MouseButtonEventHandler(
        timeSlider_MouseLeftButtonUp),
        true);
}

private void timeSlider_MouseLeftButtonUp(object sender,
    MouseButtonEventArgs e)
{
    media.Play();
}
```

Обработчик события `MouseLeftButtonUp` необходим для возобновления воспроизведения после перетаскивания бегунка тайм линии.

Захват кадра

Видеоприложению может потребоваться извлечь снимок экрана из потока видео и визуализировать его в виде растрового рисунка. Один из таких примеров можно наблюдать в некоторых программах воспроизве-

дения видео, где пользователь в любой момент во время воспроизведения может нажать на значок камеры, чтобы получить кадр из видео. Другим вариантами использования может быть создание эскизов наборов файлов видео или тайм линии с возможностью визуализации видеопоследовательности в виде отдельных кадров. Примером последнего служат проприетарные видеоредакторы: *Adobe Premier*, *Sony Vegas*, *Movavi Video Editor* и т. д.

В WPF существует несколько способов реализации задач, и захват кадра из потока видео и визуализация в виде растрового рисунка не является исключением. Не имеет значения, какой используется метод, – основные функции визуализации растровых рисунков визуальных элементов в WPF обрабатываются классом `RenderTargetBitmap`, который является частью пространства имен `Windows.Media.Imaging`. Этот класс создает растровые рисунки из любого элемента видимого дерева приложения WPF.

Рассмотрим простейший способ захвата кадра из проигрываемого видео. Для этого в XAML разметку добавим кнопку `Grab` и элемент управления `Image` для представления изображения.

```
<Button Content="Grab" Click="Button_Click_5"/>
<Image Name ="thumb" />
```

В обработчике события нажатия кнопки `Grab` создадим объект класса `RenderTargetBitmap`, указав размер видео, стандартное разрешение для экранного изображения в 96 dpi, и формат пикселя. Для этого объекта вызовем метод `Render`, который позволяет получить растровое изображение любого визуального элемента окна. В нашем случае мы получим растровое изображение `MediaElement` и далее передадим его в `Image`.

```
private void Button_Click_5(object sender,
    RoutedEventArgs e)
{
    var imageSource = new RenderTargetBitmap(
        media.NaturalVideoWidth,
        media.NaturalVideoHeight,
        96, 96, PixelFormats.Default);
    imageSource.Render(media);
    thumb.Source = imageSource;
}
```

Для получения кадра из видео, без вывода кадра на экран в качестве изображения, необходимо выделить отдельный поток для функции, который будет реализовывать захват кадра. Пример реализации показан ниже.

```
ThreadPool.QueueUserWorkItem(delegate
            {
                setScreenCaptureWorker(controlName,
                    timeSpan, source);
            });
```

Функция `setScreenCaptureWorker(controlName,timeSpan, source)`, реализуется разработчиком, и осуществляет захват кадра. Элементу MediaPlayer необходимо некоторое время для того чтобы загрузить данные, поэтому захват кадра производится в отдельном потоке. Также, можно воспользоваться функцией Thread.Sleep(1000) после вызова метода MediaPlayer.Open(), что бы MediaPlayer успел загрузить все данные.

Использование MediaPlayer

Как уже упоминалось ранее элемент управления `MediaElement`, находящийся в пространстве имен `System.Windows.Media`, является визуальной оберткой класса `MediaPlayer`. Таким образом, `MediaPlayer` отсутствует в дереве видимых элементов WPF и может быть порожден только динамически в коде. Отсутствие в видимом дереве может быть преимуществом, если для вашего файла мультимедиа не требуется визуального представления самого видеопотока (например, визуализация каталога с видеофайлами в виде набора миниатюрных растровых изображений, без их воспроизведения). Вторым достоинством является то, `MediaPlayer` можно использовать для воспроизведения аудио, так как, очевидно, в этом случае не требуется никакого визуального представления.

Существуют также некоторые отличия в способе загрузки файла мультимедиа в приложение при использовании `MediaElement` и `MediaPlayer`. В элементе управления `MediaElement` указывается значение свойства `Source`, а при использовании класса `MediaPlayer` вызывается метод `Open` и передается допустимый объект `Uri`. Ниже приведен код, который создает объект `MediaPlayer`, загружает в него видеофайл и воспроизводит видео на самом окне.

```
// Создаем медиаплеер
private MediaPlayer mp = new MediaPlayer();
```

```
private void Button_Click_4(object sender,
    RoutedEventArgs e)
{
    mp.Open(new Uri("skazka.avi", UriKind.Relative));
    VideoDrawing drawing = new VideoDrawing();
    drawing.Rect = new Rect(0, 0, 380, 284);
    drawing.Player = mp;
    mp.Play();
    DrawingBrush brush = new DrawingBrush(drawing);
    this.Background = brush;
}
```

Как мы видим, после вызова метода `Open`, создается объект класса `VideoDrawing`, который необходим для проигрывания файла мультимедиа. Далее создается объект класса `DrawingBrush`, передав объект `VideoDrawing` в качестве параметра его конструктора. `DrawingBrush` предназначен для закрашивания области, которая может включать фигуры, текст, видеоматериалы, изображения и пр. И наконец, инициализируем фон окна с помощью объекта `DrawingBrush`.

С помощью `DrawingBrush` прорисовывать видео, поступающее от экземпляра `MediaPlayer`, на любой поверхности, поддерживающей кисть. Чтобы добиться этого, необходимо свойство `Drawing` класса `DrawingBrush` настроить на экземпляр класса `VideoDrawing`. Этот экземпляр класса `VideoDrawing` может отображать видео, настраивая свое свойство `Player` на экземпляр `MediaPlayer`, содержащий файл видео. Наконец, класс `DrawingBrush` можно применить к свойству `Brush` класса `DiffuseMaterial`, например, трехмерного объекта.

При использовании `MediaPlayer` необходимо обратить внимание на ряд важных деталей:

`MediaPlayer` рекомендуется создавать вне обработчика событий. В этом случае он будет существовать на протяжении жизненного цикла окна. Причина в том, что метод `MediaPlayer.Close()` вызывается тогда, когда объект `MediaPlayer` удаляется из памяти. Если создать объект `MediaPlayer` в обработчике событий, то он будет удален из памяти почти немедленно по завершению обработчика события и, вероятно, вскоре после этого будет удален сборщиком мусора, и тогда будет вызван метод `Close()` и воспроизведение прервется.

Однако в нашем примере возможно создание `MediaPlayer` и в самом обработчике события, поскольку он останется в памяти и не будет

удален сборщиком мусора по причине привязки его к свойству Background нашего окна.

Рекомендуется создавать обработчик события Window.Unloaded, в котором вызывается метод Close() для остановки любого воспроизводящегося в данный момент звука при закрытии окна. Это необходимо в первую очередь при воспроизведении мультимедиа в дочерних окнах. Если подобный обработчик с методом Close() не будет создан, то при закрытии дочернего окна воспроизведение звука из видео может продолжаться еще некоторое время.

Воспроизведение видео на трёхмерной поверхности

Используя WPF, можно строить сложные трёхмерные сцены на основе понятного кода разметки [1]. Каким образом строятся 3D сцены, было рассмотрено в курсе компьютерная графика и в [2, 3]. Напомним, что трехмерный объект группируется внутри XAML элемента ModelVisual3D. Материалы для 3D объекта создаются с помощью таких кистей, как SolidColorBrush, LinearGradientBrush, ImageBrush или VisualBrush. Прорисовка видео на трехмерной поверхности выполняется с помощью VisualBrush. В следующем фрагменте программы кисть VisualBrush применяется к материалу DiffuseMaterial трехмерного объекта:

```
<GeometryModel3D.Material>
    <DiffuseMaterial>
        <DiffuseMaterial.Brush>
            <VisualBrush>
                <VisualBrush.Visual>
                    <MediaElement
                        Source="file:///e:\skazka.avi" />
                </VisualBrush.Visual>
            </VisualBrush>
        </DiffuseMaterial.Brush>
    </DiffuseMaterial>
</GeometryModel3D.Material>
```

Если несколько граней сгруппированы в один 3D объект с помощью ModelVisual3D, то видео будет воспроизводиться соответственно на всех гранях (рис 1.2). С таким 3D объектом возможны любые трехмерные преобразования не прерывающих воспроизведение видео на его гранях.

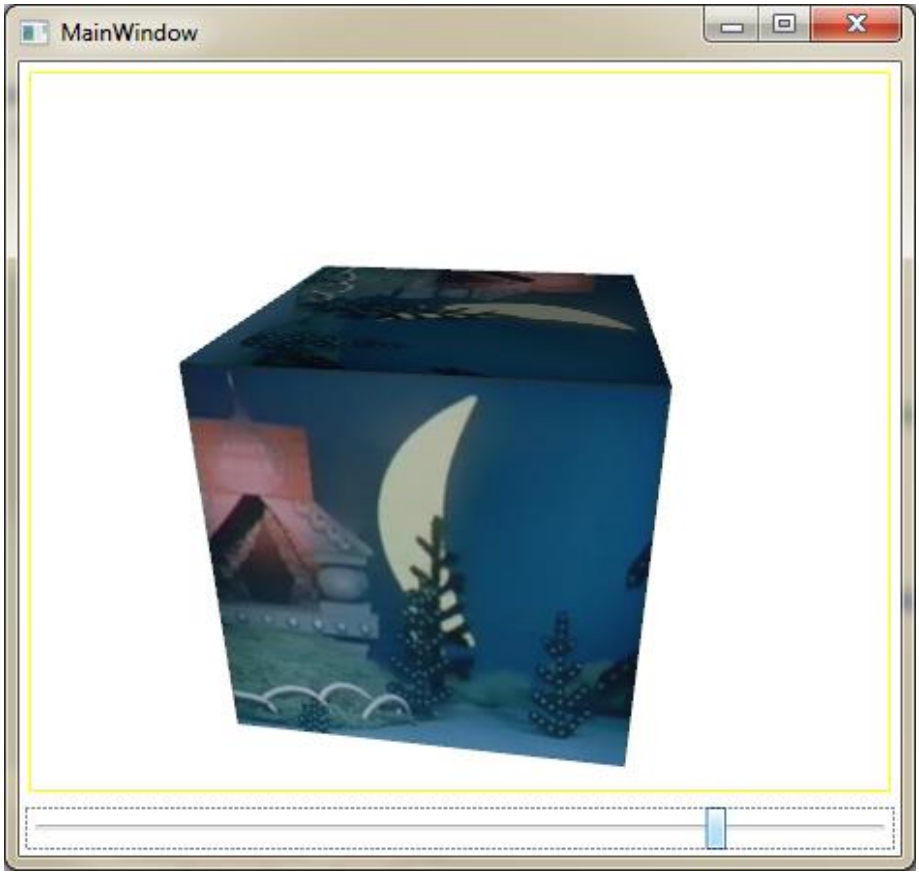

Рис. 1.2. Воспроизведение видео на кубе

Кроме воспроизведения видео на 3D поверхностях можно использовать всю мощь WPF: накладывать один элемент на другой с разной степенью прозрачности, реализую тем самым полупрозрачные или зеркальные элементы.

Рассмотрим пример, где с помощью `VisualBrush` на элементе `Rectangle` будет воспроизведено видео, при этом будет создано идеальное отражение видео. В среде дизайнеров этот эффект известен под названием «мокрый пол».

Простейший способ создания эффекта мокрого пола состоит в помещении элемента управления `MediaElement` в первое гнездо элемента управления `StackPanel`. Затем ниже создается элемент `Rectangle` с такими же размерами, как и видео. Далее воспользуйтесь `VisualBrush` для заполнения `Rectangle` и с помощью привязки данных свяжите его свойство `Visual` с элементом управления `MediaElement`. Наконец, отразим видео, чтобы создать зеркальное изображение, и применим `OpacityMask` для плавного уменьшения интенсивности отражения. Код XAML выглядит следующим образом:

```
<StackPanel VerticalAlignment="Center">
    <MediaElement Name="myVid" Source="skazka.avi"
        LoadedBehavior="Play" Width="320" Height="240" />
        <Rectangle Width="320" Height="240">
            <Rectangle.Fill>
                <VisualBrush Visual="{Binding
                    ElementName=myVid}" />
            </Rectangle.Fill>
            <Rectangle.OpacityMask>
                <LinearGradientBrush StartPoint="0.5,0"
                    EndPoint="0.5,1">
                    <GradientStop Color="#AA000000"
                        Offset="1" />
                    <GradientStop Color="#00000000"
                        Offset="0" />
                </LinearGradientBrush>
            </Rectangle.OpacityMask>
            <Rectangle.RenderTransform>
            <TransformGroup>
                <ScaleTransform ScaleY="-1" />
                <TranslateTransform Y="242" />
            </TransformGroup>
        </Rectangle.RenderTransform>
    </Rectangle>
</StackPanel>
```

На рис. 1.3 показан полученный эффект мокрого пола.

Рис. 1.3. Эффект «мокрого пола»

Ссылки

1. http://msdn.microsoft.com/ru-ru/magazine/cc163455.aspx#S3
2. http://www.codeproject.com/Articles/411827/Using-the-MediaPlayer-in-WPF
3. http://professorweb.ru/my/WPF/UI_WPF/level26/26_2.php
4. http://andrey.moveax.ru/post/wpf-rendertargetbitmap.aspx
5. http://blogs.msdn.com/b/delay/archive/2008/09/03/video-frame-grabbing-made-easy-how-to-quickly-capture-multiple-video-frames-with-wpf.aspx

Задания к лабораторной работе

1. Разработайте программу, отображающую в окне четыре области с возможностью воспроизведения различных медиапотоков (видео, звук). Предусмотреть возможность управления воспроизведением каждого потока в отдельности и возможность открытия любых медиа фалов. Исследовать качество воспроизведения фалов различных типов.
2. Создайте приложение, отображающее трехмерный вращающийся куб, на каждой грани которого воспроизводится видео. Реализуйте элементы управления видео (кнопки воспроизведения, остановки, паузы) для каждой грани в отдельности.
3. Создайте приложение, отображающее трехмерный вращающийся куб, на каждой грани которого воспроизводится видео. Реализуйте элементы управления видео (кнопки воспроизведения, остановки, паузы, слайдеры позиции, громкости и скорости воспроизведения) для одной текущей грани. Текущая грань – это грань, повернутая к пользователю и занимающая наибольшую площадь на в окне.
4. Разработайте приложение, собирающее из указанной папки и из всех ее подпапок медиаконтент с возможностью одновременного или последовательного воспроизведения.
5. Разработайте специализированный браузер, собирающей из указанной папки и из всех ее подпапок медиаконтент с возможностью проигрывания указанных элементов. Каждый элемент в браузере отображается в виде изображения. Для реализации этой функции из видео необходимо выбрать один значимый кадр.
6. Разработайте приложение, воспроизводящее последовательно и случайным образом короткие фрагменты видео из различных видео файлов.
7. Реализуйте игру «угадай кадр». Из набора видеофайлов выбирается случайный кадр и игрок должен угадать название фильма. Название фильма содержится в название файла.
8. Реализуйте тайм линию в медиапроигрывателе, над которой будут показываться миниатюрные изображения кадров видео для этой позиции там линии.
9. Создайте приложение видеопроигрыватель, с возможностью отбрасывания видеотени позади основного видео. При создании используйте преобразование скос.
10. Создайте приложение с возможностью наложения разных звуковых дорожек с разной громкостью на видео.

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>
```