

Интерфейс и функционал программных средств (ПС), описанных в лабораторных работах может отличаться от версий выбранного ПС для выполнения лабораторных работ.

Лабораторная работа №1.

Создание 3D видеосцен средствами Blender

В этом разделе описано как создать и анимировать 3d видеосцену в виде маленького "Хлебного Человечка". Запустите Blender дважды кликнув по его иконке. Blender выдаст сцену с кубом. Если куб будет отсутствовать, то его можно добавить, выбрав Add-Mesh-Cub. Задайте вид сверху, выбрав в меню, расположенном в левом нижнем углу, команду View-Top, либо кнопкой Numpad7 (рис 1.1.).

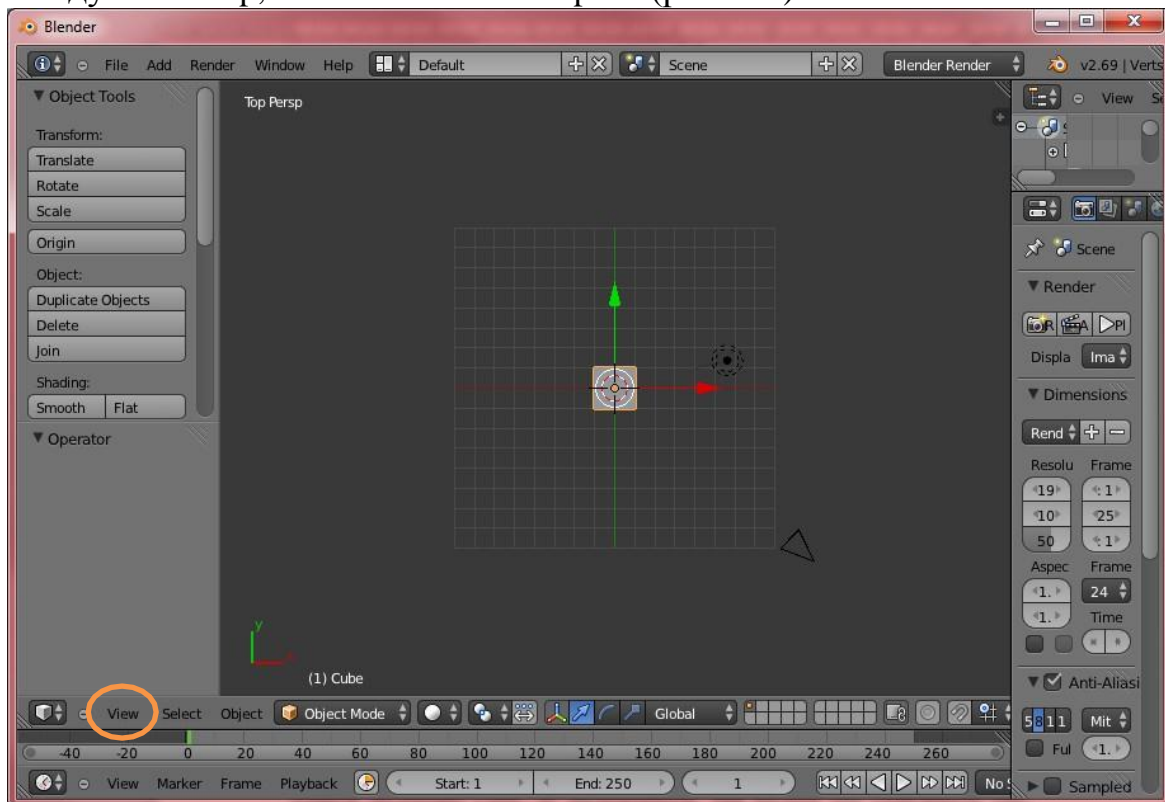


Рис 1.1. Окно Blender'a сразу после запуска

Выделите куб правой кнопкой мыши. Перейдем в режим редактирования объекта, выбрав Edit Mode(рис. 1.2.).

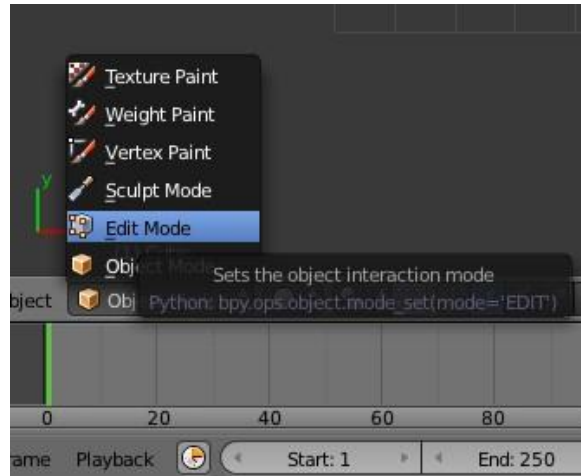


Рис. 1.2. Переход в режим редактирования объекта

Теперь меш-объект находится в режиме редактирования, в котором вы можете перемещать отдельно любую вершину, входящую в состав объекта. По умолчанию выбраны все вершины (желтые), все ребра (темно-желтый) и все грани (светло желтые) (рис. 1.3.).

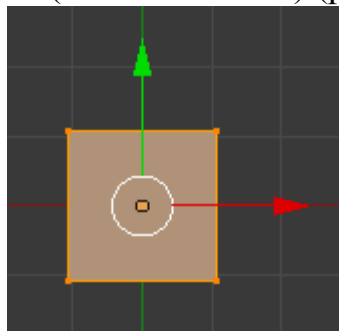


Рис 1.3. Куб в режиме редактирования и выбраны все вершины

Если грани у вашего куба прозрачные, а ребра просто черные, то для того, чтобы они отображались как на рис 1.3, нужно нажать на клавиатуре кнопку А, чтобы выделился весь объект.

Будем называть нашего Хлебного человечка "Гас". Первая задача - это создать туловище Гасу. Для этого мы поработаем с нашим кубом в режиме редактирования, используя инструменты которые нам предоставляет Blender.

Теперь на панели Mesh Tools найдите кнопку Subdivide (разделить) и один раз нажмите ее рис. 1.4. Это поделит каждую сторону куба на два, создав новые вершины и грани рис .1.5.



Рис 1.4. Кнопки для редактирования меш-объектов

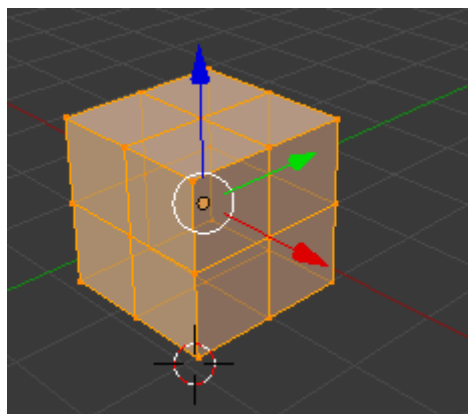


Рис 1.5. Куб разделенный один раз

Поместите курсор мыши в 3D-окно и нажмите А. Это снимет выделение со всех вершин. Теперь нажмите В, курсор изменится на пару ортогональных серых линий. Возьмите выше и левее от куба, нажмите левой кнопкой мыши и удерживая ее нажатой, тащите мышь вниз и вправо, так, чтобы появившаяся серая прямоугольная рамка покрыла все вершины слева. Эта последовательность, позволяет вам выбирать группу вершин, попавших в серую прямоугольную рамку рис 1.6.

Рис 1.6. Последовательность выбора группы вершин рамкой

Во многих ситуациях, вершины могут быть скрыты за другими вершинами. Например, как в нашем случае, наш подразделенный куб имеет 26 вершин, но вы можете видеть только девять, потому что другие скрыты.

Обычный клик правой кнопки мыши выберет только одну из этих вершин, скрытых друг за другом.

При выделении можно использовать кнопку Limit Selection to visible (рис. 1.7.), которая позволяет выделять только видимые вершины либо даже вершины скрытые под ними.

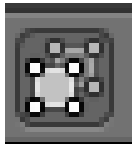


Рис. 1.7. Кнопка Limit Selection to visible

Наша задача выделить девять вершин на левой грани куба. Для контроля выделения можно переключатся в просмотр 3D сцены со стороны камеры (View-Camera, либо кнопкой Numpad0)

Теперь выберите Mesh-Delete либо нажмите X и из появившегося меню выберите Vertices чтобы стереть выбранные вершины (рис 1.8). Не беспокойтесь, что левая грань пропала.

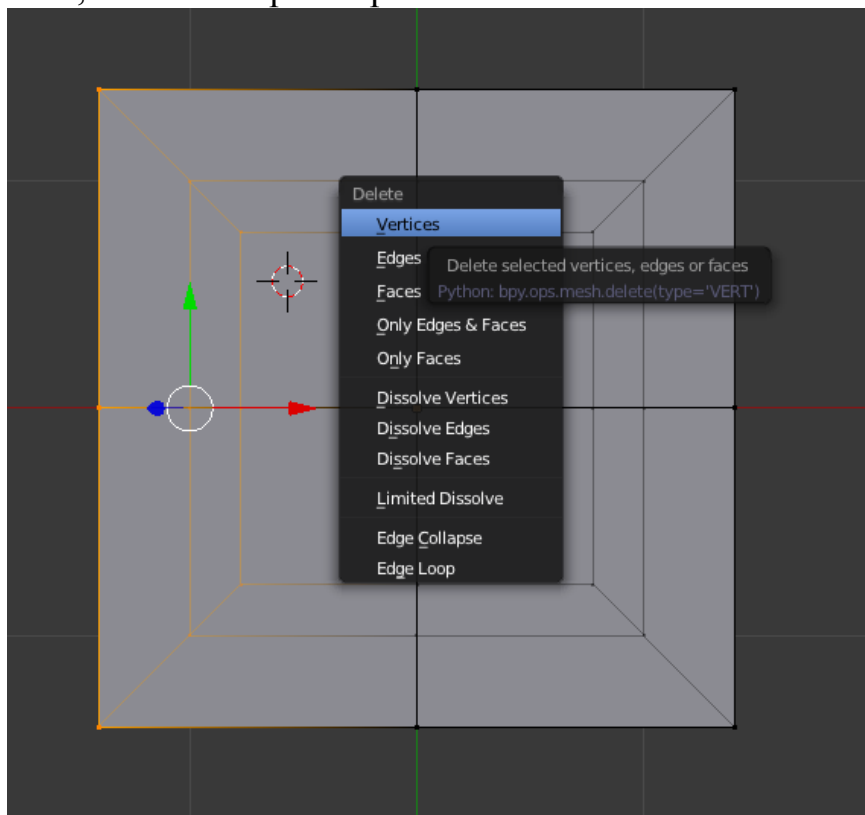


Рис 1.8. Меню удаления (клавиша X)

Теперь, с помощью рамки выделения, выберите ряд вершин сбоку, как на рис 1.9. Нажмите E (или e) и выберите из меню Extrude чтобы экструдировать («выдавить», «вытолкнуть») вершины. Это создаст новые вершины и грани. Созданные вершины готовы к перемещению и могут двигаться за мышью. Двигайте их вправо.

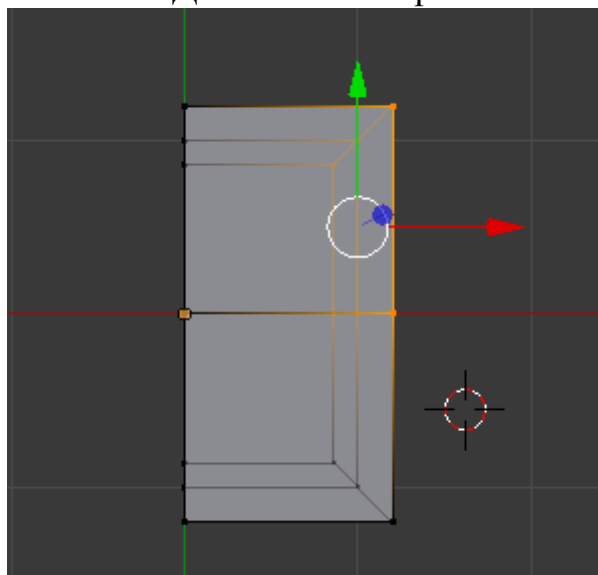


Рис 1.9. Выделение двух вершин

Переместите вершины на полтора квадрата сетки вправо, и кликните левой кнопкой мыши для фиксации положения. Выдавите (команда Extrude) снова, нажав E и переместив новые вершины вправо. Чуть поднимите выделенные вершины как показано на рис. 1.10. с помощью команды Mesh-Transform-Grab/Move или нажимая кнопку G.

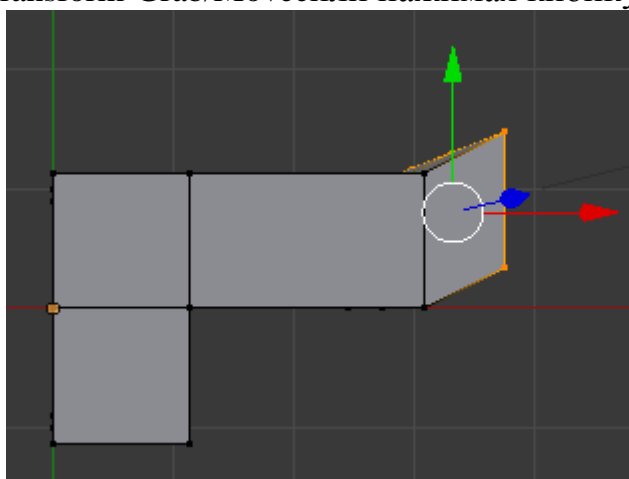


Рис 1.10. Выполнение команды Extrude

Теперь у Гаса есть левая рука (лицом он смотрит на нас). Мы создадим ему левую ногу, таким же способом, "выдавливая" нижние вершины. Попробуйте добиться того же результата, что и на рис 1.11.

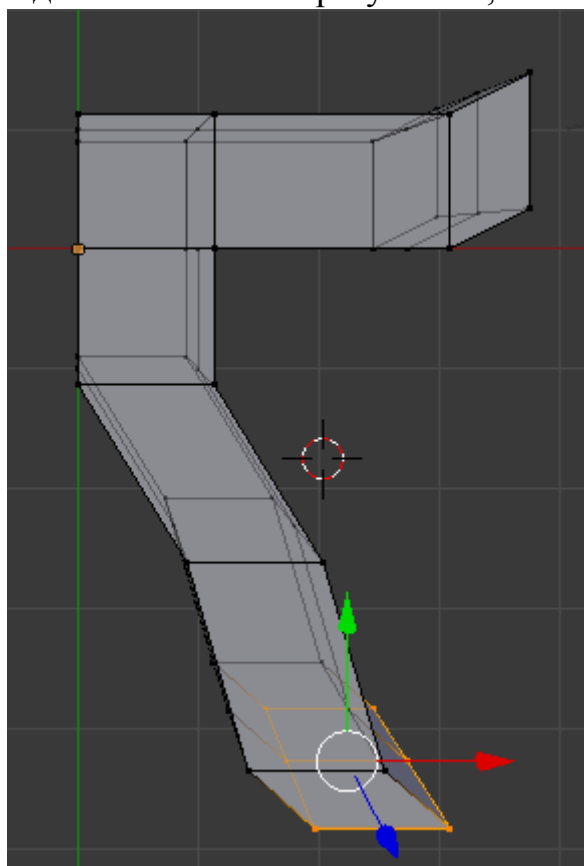



Рис 1.11. Половинка туловища

В результате выдавливания могут получиться дублирующиеся вершины, которые могут располагаться в одних и тех же позициях. Чтобы избавиться от таких вершин необходимо использовать кнопку Remove Doubles на панели MeshTools.

Теперь настало время создать другую половинку Гаса, выберите все вершины (A) и нажмите внизу 3D-окна кнопку Pivot center for rotation/scaling похожую на колечко со стрелками . Появится меню, из него выберите пункт 3D Cursor. Курсор устанавливаем посередине будущего человечка (рис. 1.12). Относительно этого курсора будет произведено отражение по оси X.

Теперь, не двигайте мышью. Выбираем Duplicate на панели Mesh Tools или нажимаем Shift-D чтобы продублировать все выбранные вершины, потом, не двигая мышью, нажмите левую кнопку мыши. Далее нажимаем Mesh и из появившегося меню Mirror, выбираем пункт

Local X, чтобы зеркально отразить дубликат. Результат представлен на рис. 1.12.

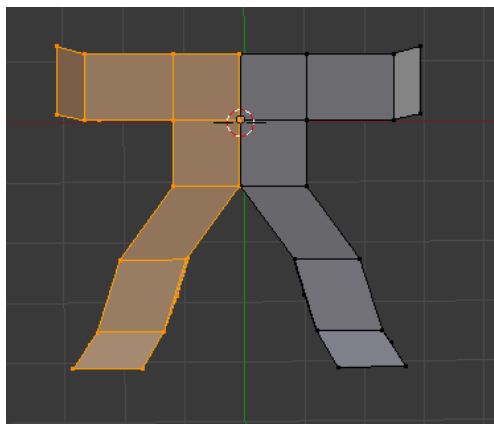


Рис. 1.12. Результат зеркального отражения

Снимите выделение с вершин и снова выберите все, нажав дважды А. Далее удалите совпадающие вершины, нажав кнопку Removedoubles.

Поместите курсор прямо над туловищем Гаса, на один квадратик решетки выше. Добавьте еще один куб (Add-Mesh-Cube). Далее используя разные точки зрения (кнопки Numpad) и режим перемещения (кнопка G) опустите голову чуть ниже в туловище и проследите, что бы голова была на одном уровне по оси zc самым телом (рис. 1.13).

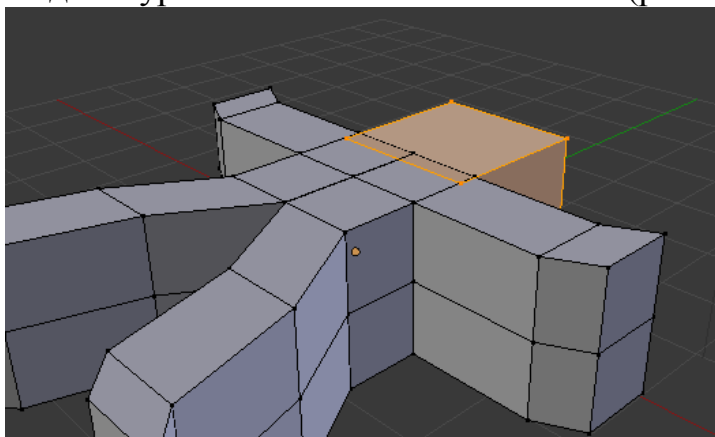


Рис 1.13. Добавления головы

Теперь сделаем всю форму более гладкой, нажав кнопку Subdivision Surface на панели добавления модификаторов (рис. 1.14.). Этот модификатор динамически вычисляет гладкий высокополигонный объект из грубого низкополигонного объекта.

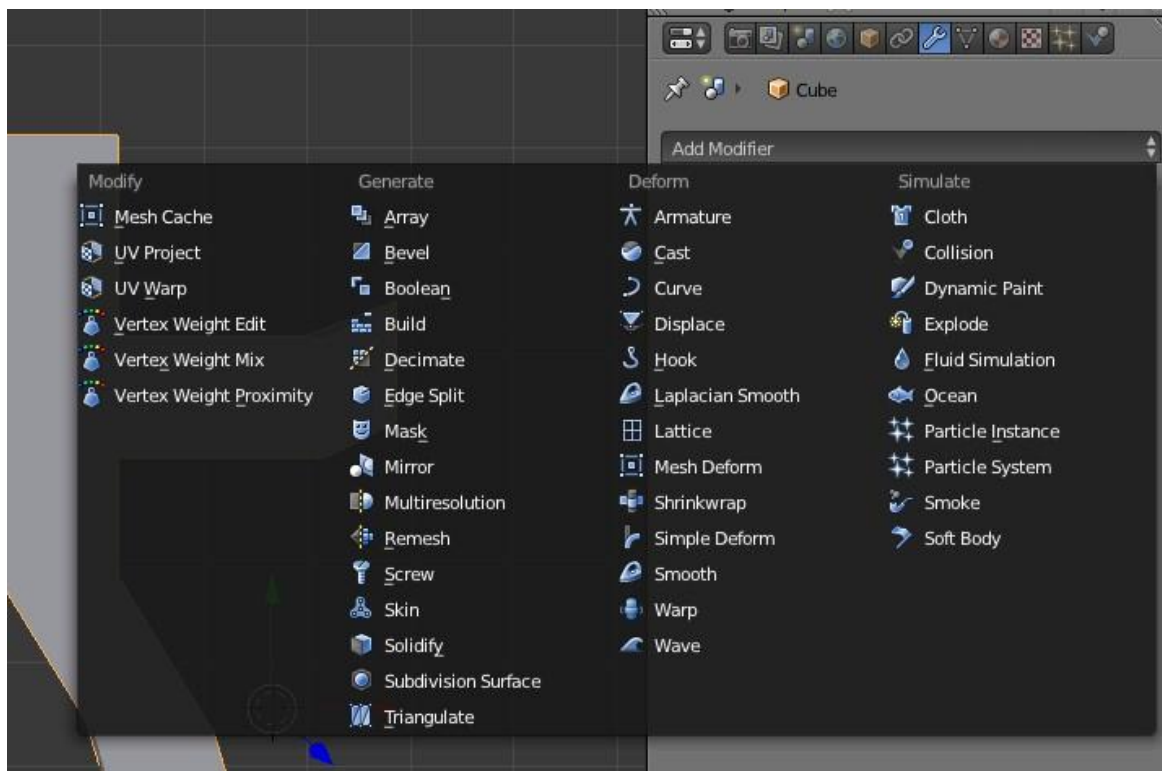


Рис 1.14. Панель добавления модификаторов

С помощью View выбираем количество граней на нашем человечке. Результат приведен на рис. 1.15.

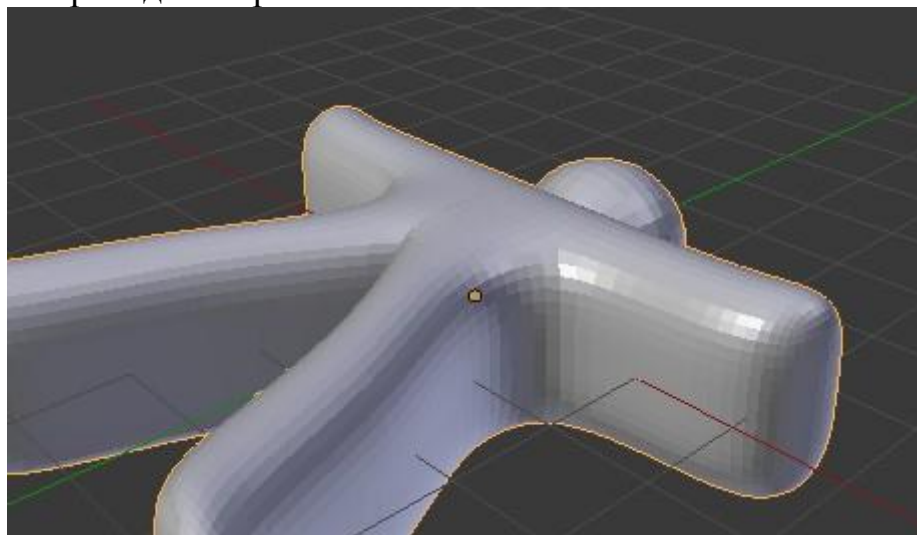


Рис. 1.15. Результат выполнения команды Subdivision Surface

Как мы видим, человечек получился сильно толстым по оси z. Отмасштабируем его по оси z. Для этого используем команду Object-Transform-Scale или кнопку S. После перехода в режим масштаби-

вания нажмите Z, что определит масштабирование только по оси z. Задайте толщину Гаса в 0,2-0,25 от его исходной толщины.

С помощью преобразования поворота и переноса повернем и поставим Гаса на плоскости как показано на рис. 1.16. Под ноги ему добавим еще один объект – плоскость (Add-Mesh-Plane). Отмасштабируем эту плоскость.

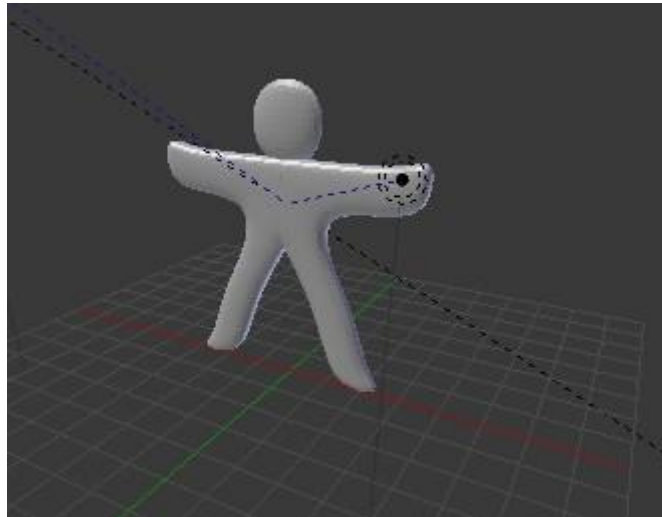


Рис. 1.16. Объект после поворота и переноса

Далее можно настроить положение камеры. Для этого существует множество способов. Один из них заключается в выделении камеры и задание ее положения и углов поворота. Выделение камеры происходит путем нажатие на нее правой кнопкой мыши. Далее можно в меню камеры (рис. 1.16.) задается местоположении камеры (Location) и в режиме просмотра от камеры (Numpad0) выбирается угол поворота (Rotation).

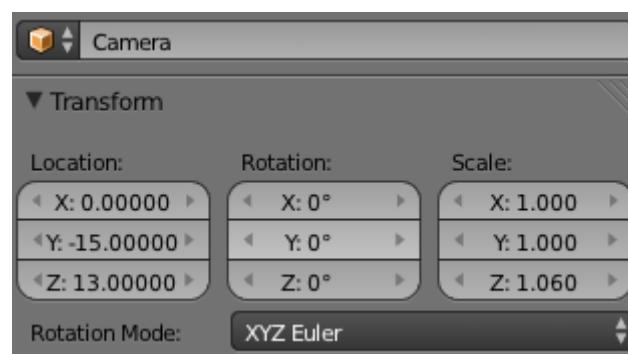


Рис. 1.16. Окно задание положение камеры

Второй способ задания направления камеры на объект заключается в том, что мы выделяем камеру и сам объект удерживая Shift, нажима-

ем Ctrl+T и далее в меню выбираем команду Damped Track Constraint. Это обеспечит направление камеры на наш объект.

После задания положения камеры добавим еще один источник освещения Add-Lamp-Hemi. Местоположение источника освещения и его направление определяется аналогично положению камеры. Для настройки самого источника освещения существует ряд настроек в специальной панели инструментов (рис. 1.17.).

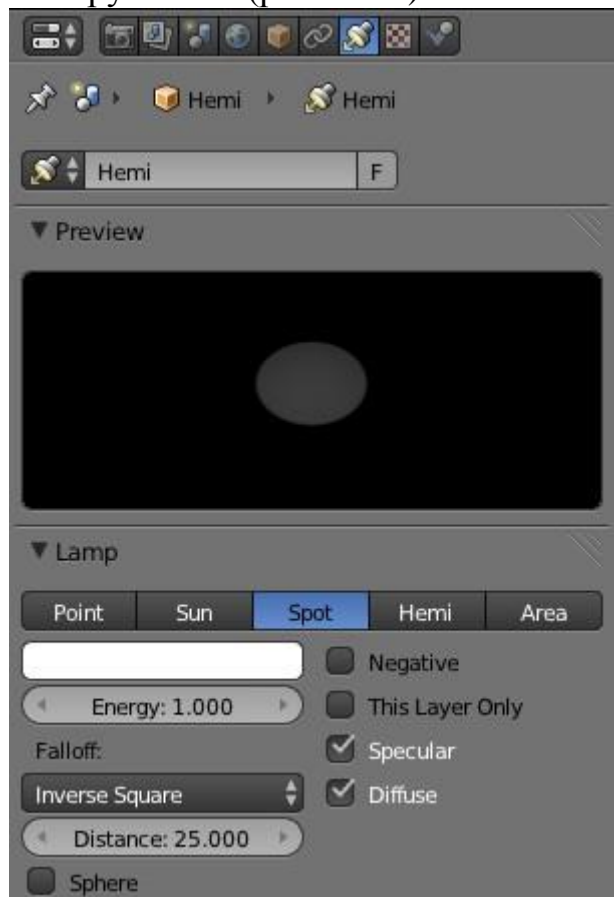


Рис. 1.18. Панель задания свойств источника света

Самое главное установить мощность источника освещения (Energy).

После настройки камеры и источников освещения можно посмотреть, что получается, если выполнить рендеринг (кнопка Render на панели Render(рис. 1.19.)).



Рис. 1.19. Кнопки для рендера и анимации

В итоге мы можем получить следующее (рис. 1.20).



Рис. 1.20. Результат рендеринга

Как мы видим, Гас получился серым и невзрачным. Поэтому на следующем этапе выберем для него материал. Для этого выделим его правой кнопкой и используем диалог Material(рис. 1.21.).

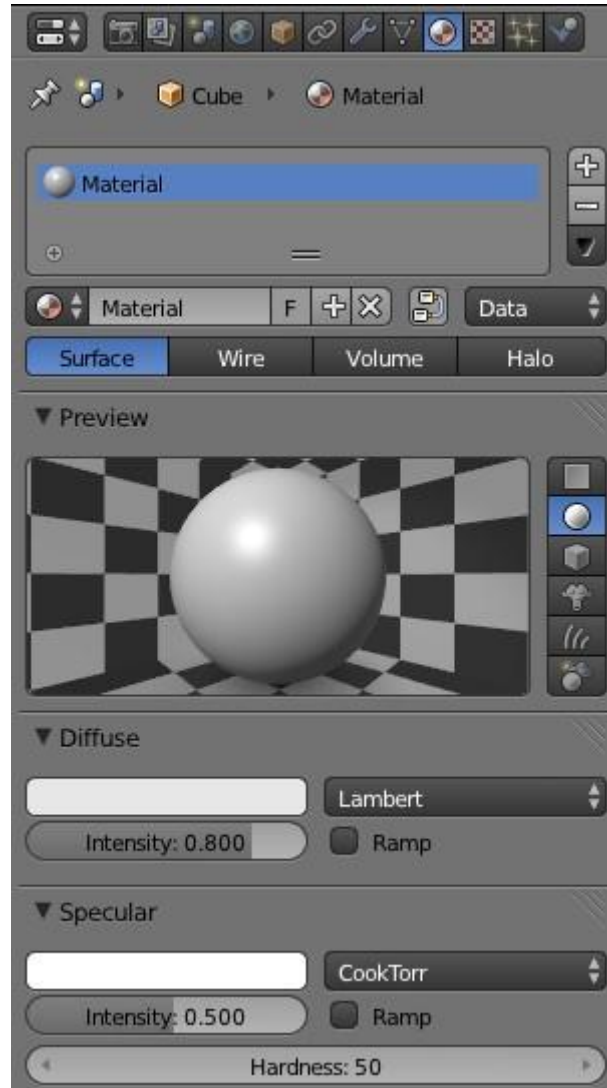


Рис. 1.21. Диалог задания материала

Для задания цвета материала достаточно нажать на белое поле в разделе Diffuse и выбрать цвет. Можно также сгенерировать или загрузить текстуру, используя при этом диалог Texture(рис. 1.22.).




Рис. 1.22. Диалог Texture

При создании новой текстуры выбирается тип текстуры (type). В качестве текстуры может выступать файл с изображением или видео. В таком случае тип текстуры указывается как Image or Movie.

Продолжим улучшать нашего человечка и добавим ему глаза и нос. При рендеринге может получиться следующее (рис. 1.23.).



Рис. 1.23. Объект после наложения материала и текстур

И так 3d модель готова. Следующий этап – это подготовка модели к анимации, а именно создание скелета внутри Гаса. Для создание первой кости в скелете разместим 3D курсор внутри тела Гаса и добавим кость Add-Armature-Single Bone. Далее отразим кость по вертикали командой Armature-Mirror-Z Local. Чтобы эту кость можно было увидеть переключимся в режим просмотра 3D модели в каркасном виде с помощью кнопки . Кость внутри Гаса будет выглядеть при этом следующим образом (рис. 1.24.)

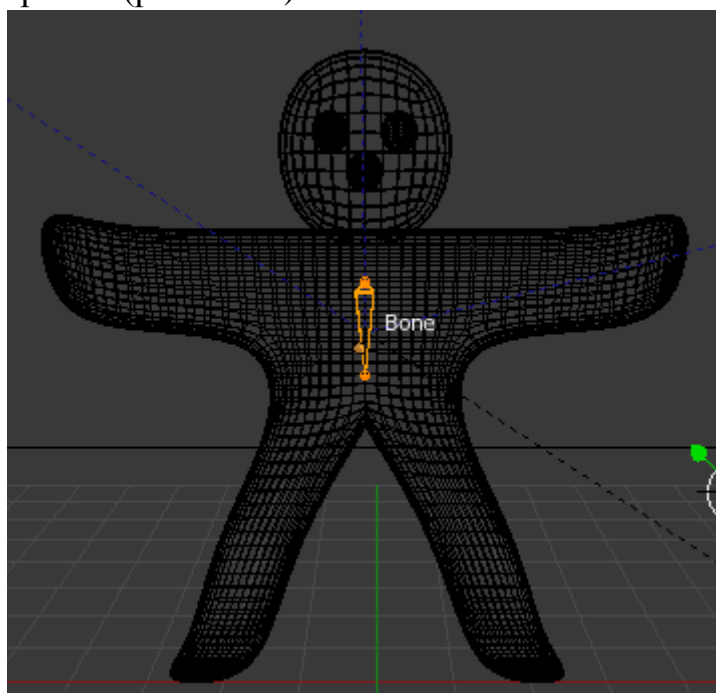


Рис. 1.24. Начало построение скелета 3D модели

На панели Object Data для вывода имени кости отметим два пункта Names и X-Ray (рис. 1.25.).



Рис. 1.25. Диалог для работы со скелетом

Выделим конец кости правой кнопкой мыши, и выполнив команду Extrude (кнопка E) добавим новую кость. Для начала составим основу скелета из трех костей, которые не будут практически двигаться в теле (рис. 1.26.).

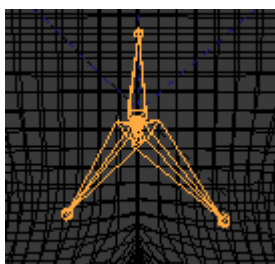


Рис. 1.26. Основа скелета

Незабываем контролировать, переключаясь в разные виды, что бы кости скелета находились внутри тела. Добавляя новые кости в скелет, мы должны получить следующее (рис. 1.27.).

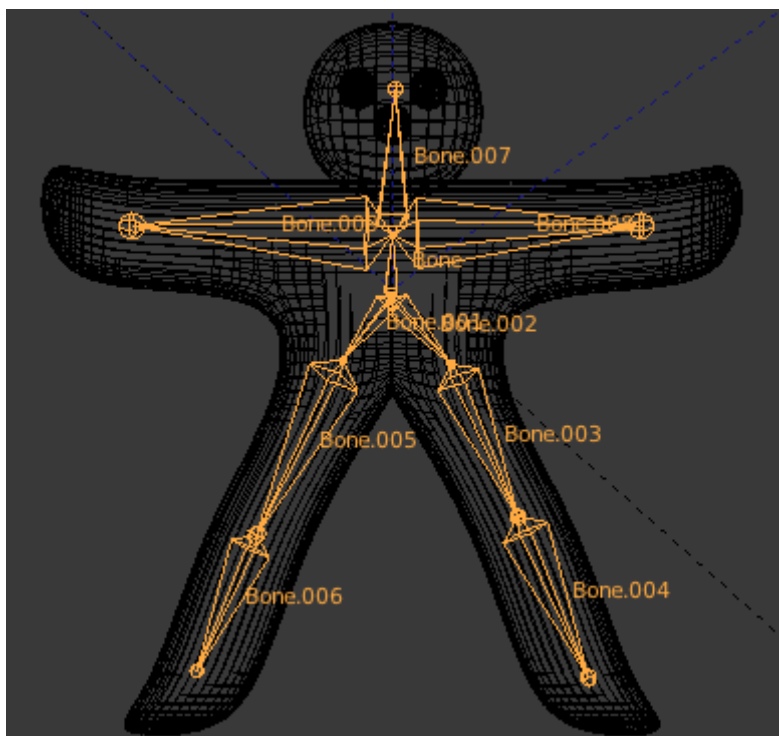



Рис. 1.27. Полный скелет 3D объекта

Теперь, нам нужно сделать так, чтобы деформация в арматуре вызывала соответствующую деформацию в теле Гаса. Выделим правой кнопкой тело Гаса а затем удерживая Shift арматуру (скелет). При этом сам Гас будет выделен оранжевым, а кости желтым. Далее выберем Object-Parent-With Automatic Weight (это же самое можно получить, нажимая Ctrl+P и выбирая соответствующий пункт). Это установит арматуру (скелет) в качестве родительского элемента по отношению к телу. Подтверждение этому – является изменение в дереве объектов в сцене (Scene) (Рис. 1.28.).



Рис. 1.28. Дерево объектов сцены

После этого в добавление к режимам Object Mode и Edit Mode добавился режим Pose Mode (, в котором можно подвигать кости скелета (R) и увидеть как меняется поза и самого Гаса. Кости в этом режиме отображаются синим цветом.

Перейдем к созданию анимации. Для ее создания нам потребуется таймлайн (рис. 1.29.).

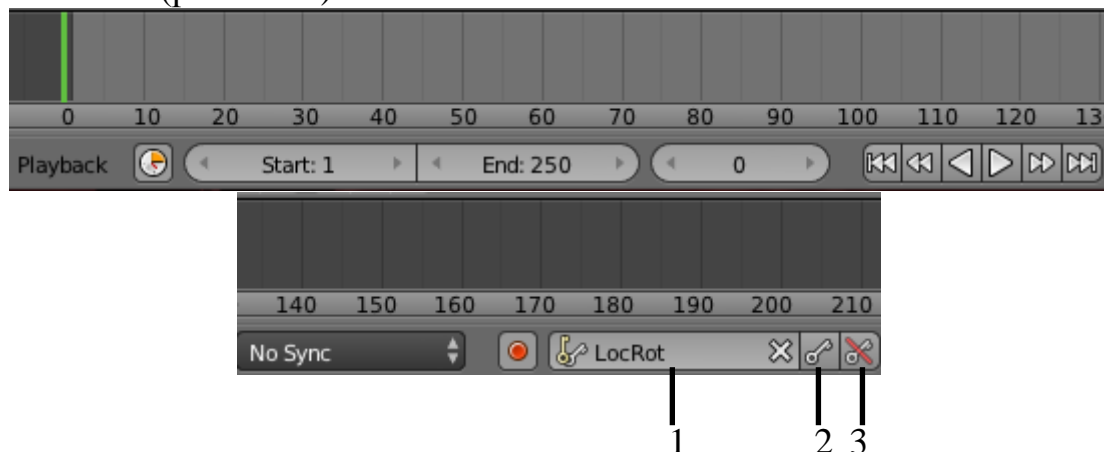


Рис. 1.29. Таймлайн для создания анимации

Сейчас мы находимся на нулевом кадре. Ограничим время анимации 50 кадрами. Для этого установим курсор на 50 кадр и выберем Frame-Set End Frame. Выберем режим для сохранения в ключевом кадре поворотов и местоположения (LocRot (рис. 1.29-1)). Находясь на нулевом кадре в режиме Pose Mode, нажмем кнопку создания и сохранения ключевого кадра с текущими настройками (рис. 1.29-2). Перейдем на 10 кадр в таймлайнии. Внесем изменения в скелет, вращая кости Гаса. Не выходя из режима Pose Mode, выделим все кости (A) и опять создадим и сохраним ключевой кадр. Промежуточные кадры будут интерполированы Blender-ом. Проведем эту процедуру для 30, 40 и 50 кадров. Полученную анимацию можно быстро просмотреть, используя кнопку Play под тайм линией (рис. 1.29.), либо выполнив рендеринг каждого кадра (Render-Render Animation, Render-Play Rendered Animation) увидеть полноценную анимированную сцену с положения камеры. Рендеринг займет определённое время, но даст возможность воспроизводить анимацию с наложением текстур, расчетом освещенности и пр.

Обратите внимание что при рендеринге происходит автоматический импорт анимации на диск. Путь и формат записи данных можно определить в пункте Output диалога рендеринга (рис. 1.30.).

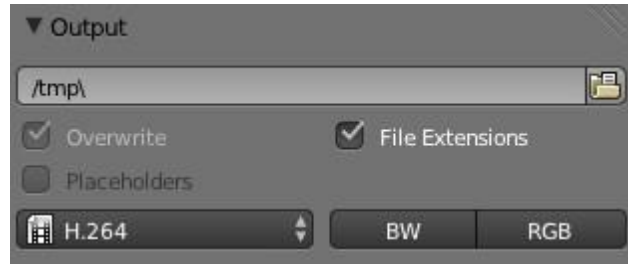


Рис. 1.30. Параметры импорта

Задания к лабораторной работе

1. 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.

2. Оформите отчет по ходу выполнения работы.

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 медиа-данных 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.).

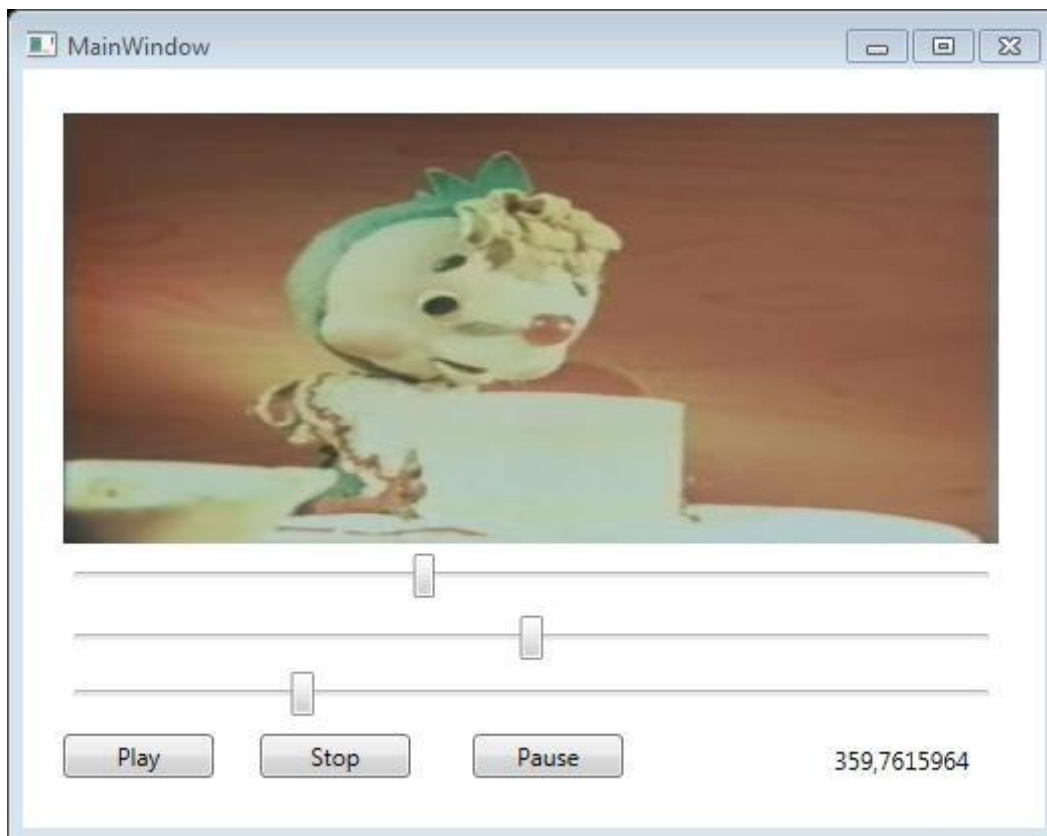


Рис. 1.1. Окно простейшего медиа-проигрывателя

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 тайм-линии 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 булевого 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)
{
    // Подготавливаем 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 воспроизводимого 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 трёхмерный объект группируется внутри XAML элемента ModelVisual3D. Материалы для 3D объекта создаются с помощью таких кистей, как SolidColorBrush, LinearGradientBrush, ImageBrush или VisualBrush. Прорисовка видео на трёхмерной поверхности выполняется с помощью VisualBrush. В следующем фрагменте программы кисть VisualBrush применяется к материалу DiffuseMaterial трёхмерного 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 трёхмерные преобразования не прерывающих воспроизведение видео на его гранях.

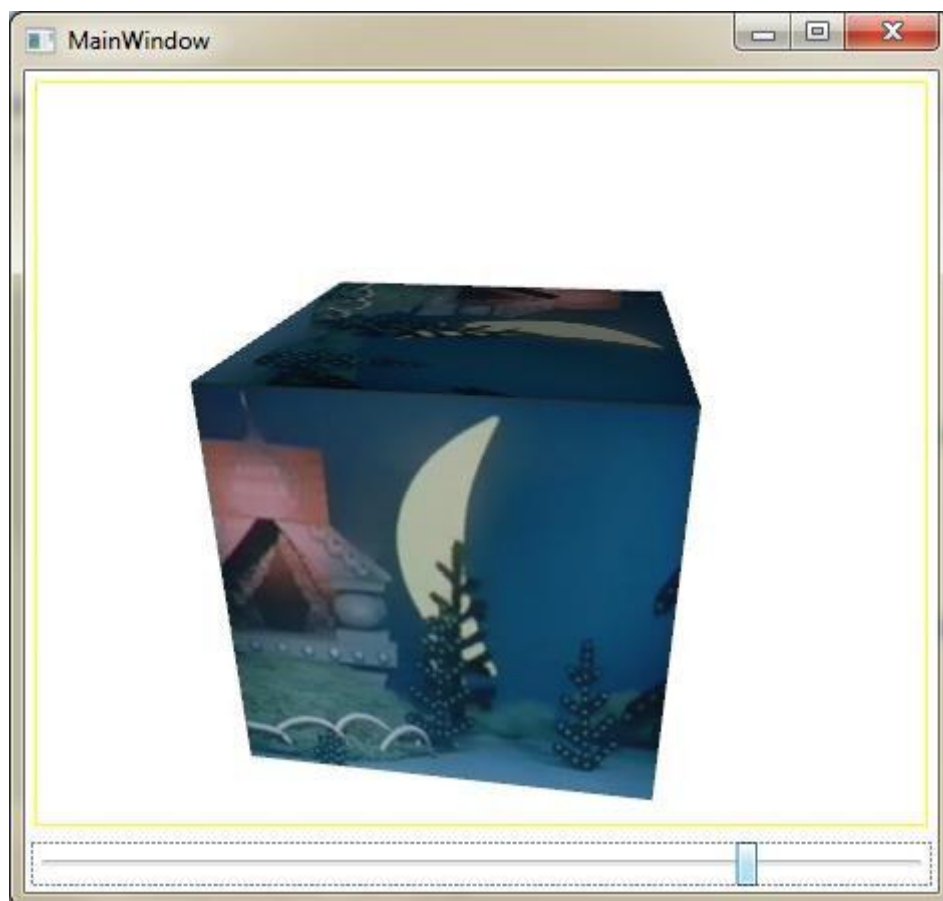


Рис. 1.2. Воспроизведение видео на кубе

Кроме воспроизведения видео на 3D поверхностях можно использовать всю мощь WPF: накладывать один элемент на другой с разной степенью прозрачности, реализуя 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 Visuals 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

Номер задания соответствует последней цифре номера зачетки

1. Разработайте программу, отображающую в окне четыре области с возможностью воспроизведения различных медиапоток (видео, звук). Предусмотреть возможность управления воспроизведением каждого потока в отдельности и возможность открытия любых медиа файлов. Исследовать качество воспроизведения файлов 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. Реализуйте тайм линию в медиапроигрывателе, над которой будут показываться миниатюрные изображения кадров видео для этой позиции там линии.
5. Создайте приложение видеопроигрыватель, с возможностью отбрасывания видеотени позади основного видео. При создании используйте преобразование скос.
6. Создайте приложение с возможностью наложения разных звуковых дорожек с разной громкостью на видео.

7. 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.

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

9. 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.

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

Оформите отчет.

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 KeyFramesанимации.
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.

Задания к лабораторной работе

Номер задания соответствует последней цифре номера зачетки

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.

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

Оформите отчет.

Разработка XAML кода в MS Expression Blend

Expression Blend представляет собой программный продукт, предназначенный для разработки приложений WPF, путем генерации и редактирования XAML кода. Возможности Expression Blend выходят далеко за рамки относительно простой поддержки редактирования кода XAML в Visual Studio. Expression Blend предоставляет развитые инструментальные средства для компоновки и настройки элементов управления, создания анимационных последовательностей, специальных стилей оформления и шаблонов, построения новых классов UserControl из имеющейся векторной графики, визуальной разработки шаблонов данных, назначения различных режимов работы и визуальных состояний для элементов пользовательского интерфейса и выполнения многих других полезных операций.

При создании нового проекта необходимо указать один из шаблонов, из которых мы рассмотрим только WPF Application. Данный шаблон предназначен для построения традиционных настольных приложений на платформе WPF в виде исполняемых файлов.

Интерфейс Expression Blend

Рассмотрим основные элементы Expression Blend. В центре окна располагается монтажный стол, который является визуальным конструктором и служит для создания внешнего вида любого окна в приложении WPF и элементов пользовательского интерфейса.

Слева внизу на монтажном столе расположен ряд элементов управления, которые, называются элементами управления монтажного стола (рис. 3.1.). С их помощью вы можете изменять общий вид поверхности этого визуального конструктора.



Рис. 3.1. Элементы управления монтажного стола

Первый элемент управления служит для задания масштаба, далее за ним следует элемент управления, обозначенный знаком математической функции ($f(x)$), и служащий для включения или отключения любых эффектов визуализации, накладываемых на элемент пользовательского интерфейса в визуальном конструкторе. В состав Expression Blend входит целый ряд предварительно заданных визуальных эффектов, в том числе эффект падающей тени. При построении пользовательских интерфейсов может периодически возникать потребность скрывать подобные визуальные эффекты на время конструирования, чтобы было удобнее настраивать основные элементы пользовательского интерфейса. В этом случае и используется данный элемент управления монтажного стола.

Далее следуют три элемента управления, с помощью которых можно задавать реакцию монтажного стола на расположение на нем элемента пользовательского интерфейса. Если щелкнуть на кнопке Show snap grid (Показать сетку для привязки), на поверхность визуального конструктора будет наложена сетка. После этого можно активизировать одну из двух других кнопок: Turn on snapping to gridlines (Включить привязку к линиям сетки) или Turn on snapping to snaplines (Включить привязку к линиям привязки).

Последняя кнопка в области элементов управления монтажного стола служит для просмотра или сокрытия аннотаций в Expression Blend. Подобные аннотации можно рассматривать как своего рода записки программирующего в среде Expression Blend.

В Expression Blend предоставляется довольно развитый редактор разметки в коде XAML. Но прежде чем просматривать и править подобную разметку, вам придется перейти от кнопки Design (Конструировать) к кнопке XAML или Split (Разделить). Все эти кнопки находятся в правом верхнем углу монтажного стола (рис. 3.2.).



Рис. 3.2. Кнопки переключения в различные режимы просмотра XAML кода

Следующим важным компонентом среды Expression Blend IDE является панель Objects and Timeline (Объекты и временная шкала) (рис. 3.3.), располагаемая по умолчанию с левой стороны в рабочем окне. На панели Objects and Timeline отображается иерархическое представление разметки XAML для данного конкретного окна, оформляемого в визуальном конструкторе.

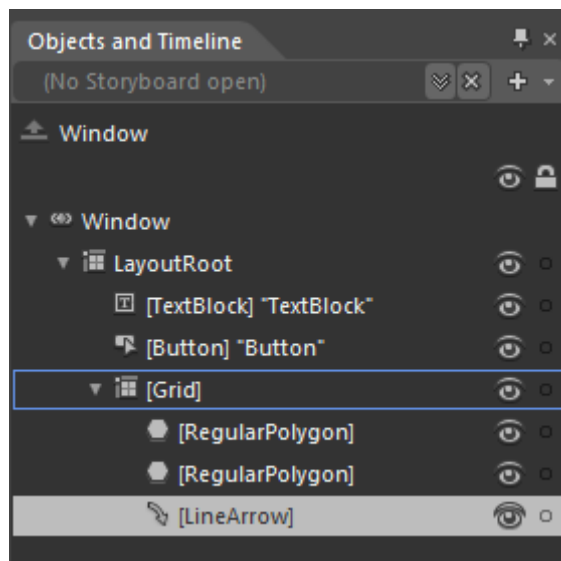


Рис. 3.3. Панель *Objects and Timeline*

Верхний узел этого иерархического представления обозначает все окно в целом (посредством элемента Window). Подчиненный ему непосредственно узел (корневой диспетчер компоновки) называется по умолчанию LayoutRoot.

Этот диспетчер компоновки может содержать любое количество элементов управления пользовательского интерфейса.

Как показано на рисунке, справа от каждого узла, отображаемого на панели Objects and Timeline, находится пиктограмма с изображением глаза, а еще дальше — кружок. Пиктограмма с изображением глаза служит для отображения и сокрытия отдельных элементов пользовательского интерфейса в визуальном конструкторе. Кружок позволяет блокировать отдельные объекты (и любые содержащиеся в них объекты), чтобы их нельзя было править в визуальном конструкторе.

Помимо обычного просмотра иерархического представления разметки, его узлы на панели Objects and Timeline позволяют быстро и просто выбирать отдельные элементы в визуальном конструкторе для последующей правки.

На панели Objects and Timeline можно также создавать объекты раскадровки, содержащие инструкции для анимации.

Используя это назначение данной панели, можно выбрать отдельный узел в иерархическом представлении разметки и видоизменить его самыми разными способами: изменить местоположение, цвет и прочие свойства. По ходу выполнения этих действий они регистрируются инструментами анимации в Expression Blend.

Над панелью Objects and Timeline находится панель Project (Проект) (рис. 3.4.), которая должна быть хорошо знакома тем, у кого имеется опыт работы в среде MS Visual Studio. Всякий раз, когда вы создаете в Expression Blend новый проект, вместе с ним автоматически создается целый ряд первоначальных файлов (разметки XAML и исходного кода), а также делаются ссылки на нужные библиотеки .NET, или так называемые сборки.

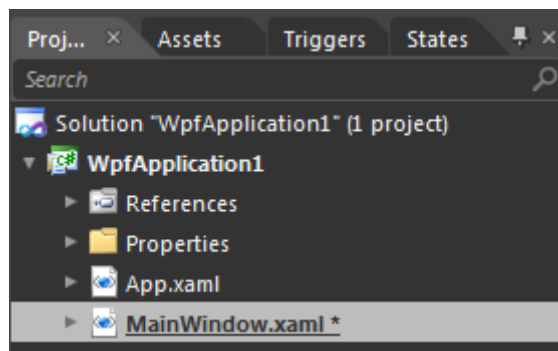


Рис. 3.5. Панель Projects

Следующий элемент интерфейса панель *Properties* (Свойства), располагаемой по умолчанию с правой стороны в рабочем окне Expression Blend (рис. 3.5.).

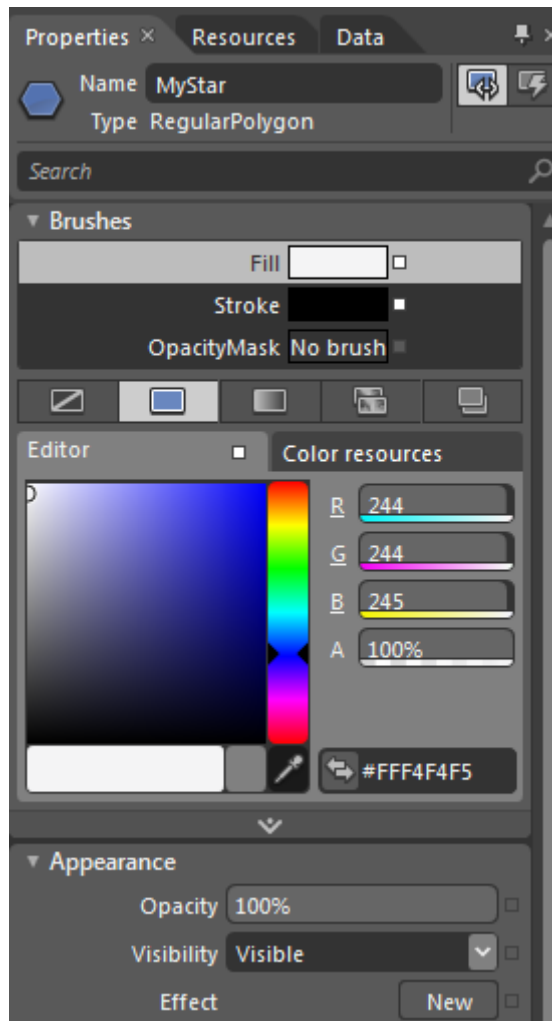


Рис. 3.5. Панель Properties

Аналогично окну Properties в среде Visual Studio, эта позволяет видоизменять выбранный элемент самыми разными способами. Однако, если видоизменить выбранный элемент на монтажном столе, например, перетащить его в другое место мышью, связанные с ним свойства на панели Properties будут обновлены.

В верхней части панели свойств можно определять имя элемента и проводить поиск элементов или свойств.

Панель Properties разделяется на различные категории свойств, каждая из которых может быть развернута или свернута независимо от остальных. Конкретные категории свойств будут динамически изменяться на панели Properties в зависимости от того, что именно выбрано в настоящий момент в визуальном конструкторе.

Так, если выбрать все окно или только отдельный элемент управления пользовательского интерфейса, на панели Properties появится целый ряд категорий свойств, в том числе:

- **Appearance** – определяются общие свойства визуализации, в том числе непрозрачность, видимость и применяемые графические эффекты (размывание, падающая тень и пр.);
- **Brushes** – Предоставляется доступ к визуальному редактору кистей;
- **Common Properties** – к этой категории относятся свойства, общие для большинства элементов пользовательского интерфейса, в том числе всплывающие подсказки, индексы перехода по табуляции, местоположение контекста данных (для операции привязки данных);
- **Layout** – обычно правятся свойства, определяющие физические размеры элементов управления, в том числе высоту, ширину, поля и пр.
- **Miscellaneous** – к этой категории, по существу, относятся все остальные свойства, доступные на панели Properties. Но самое главное, что в ней можно задать конкретный стиль оформления или шаблон для выбранного элемента;
- **Text** – настраиваются текстовые свойства выбранного элемента, в том числе параметры шрифтового оформления, разбиение на абзацы и отступы
- **Transform** – допускается выполнение графических преобразований над выбранным элементом, в том числе вращение, поворот на заданный угол, смещение и пр.

У некоторых категорий свойств имеются разворачиваемые вниз области дополнительных свойств. Так, если щелкнуть на такой области, соответствующая категория развернется для отображения ее дополнительных свойств, которые расширяют возможности данной категории, хотя используются нечасто.

На возможность дополнительной настройки некоторых свойств указывает квадратик справа. Щелкнув на этом квадратике обозначающем кнопку Advanced options (Дополнительные параметры) вы откроете еще одно окно редактора с дополнительными параметрами настройки отдельного свойства. Такая возможность оказывается полезной при выполнении операций привязки данных и обращении с ресурсами объектов.

По умолчанию слева в рабочем окне Expression Blend появляется вертикальная полоса кнопок, напоминающая панель инструментов в Visual Studio. Это и есть панель Tools (Инструменты) (рис. 3.6.), на которой доступны самые разные и чаще всего используемые элементы пользовательского интерфейса, в том числе элементы управления, диспетчеры компоновки, простые геометрические формы и пр. Эти элементы можно выбирать на панели Tools для создания собственных пользовательских интерфейсов.



Рис. 3.6. Панель Tools

Первые два инструмента служат для выделения. Первый инструмент Selection (пиктограмма в виде черной стрелки) служит для выделения объекта в целом для каких либо преобразований. Например, выделив этим инструментом прямоугольник можно далее перемещать его, изменять размеры, поворачивать или выполнять заливку.

Второй инструмент Direct Selection (пиктограмма в виде белой стрелки) служит для выделения отдельных элементов (линий, дуг) из всей фигуры, для операций над ними. Однако необходимо помнить что иногда такие операции возможны только Path (Путем). Поэтому некоторые элементы предварительно надо преобразовать в Path.

Многие инструменты работают так же как в графических редакторах, рассмотренных в курсе «Компьютерная графика». Поэтому в данном пособии подробно не рассматриваются.

Заметим, что на панели Tools отображаются далеко не все элементы, которые могут быть использованы для построения пользовательско-

го интерфейса. Когда потребуются дополнительные элементы пользовательского интерфейса, на помощь может прийти библиотека ресурсов (Assets), которую нетрудно открыть, щелкнув на крайней справа кнопке на панели Tools (на этой кнопке изображен знак »). Содержимое библиотеки ресурсов разделено на несколько категорий высокого уровня:

- **Project** – к этой категории относятся все специальные ресурсы, добавленные в текущий проект, в том числе файлы изображений, видео- и аудиозаписей специальные стили оформления и пр.
- **Controls** – к этой категории относятся все элементы управления, используемые для построения пользовательского интерфейса приложений;
- **Styles** – к этой категории относятся любые стили оформления, специально созданные для текущего проекта;
- **Behaviors** – К этой категории относятся виды поведения, которые представляют собой объекты, позволяющие фиксировать типичные события непосредственно в разметке, не прибегая к необходимости писать для этой цели специальный код C# или VB;
- **Shapes** – к этой категории относятся предварительно визуализированные геометрические формы (шестиугольники, выноски, звезды и т.д.), которые можно добавлять в текущий проект. Это дает возможность вводить стандартные формы более оперативно, чем с помощью инструментов Pen и Pencil;
- **Effects** – К этой категории относятся эффекты, с помощью которых можно изменять внешний вид элементов пользовательского интерфейса самыми разными способами.
- **Media** – эта категория подобна категории Project в том отношении, что в ней доступны специальные ресурсы текущего проекта, но к категории Media относятся только файлы изображений, видео- и аудиозаписей;
- **Categories** – к этой категории относятся все ресурсы текущего проекта, разделенные на подчиненные категории. Это дает возможность, например, быстро просматривать все элементы управления, применяемые в текущем проекте; все элементы управления, в которых используется привязка данных, и т.д.
- **Locations** – к этой категории относятся все библиотеки .NET (или так называемые сборки), содержащие различные ресур-

сы, применяемые в проектах на платформах WPF и Silverlight.

Библиотека ресурсов также доступна на одной из вкладок, располагающихся там же где и панель Projects (рис. 3.7.).

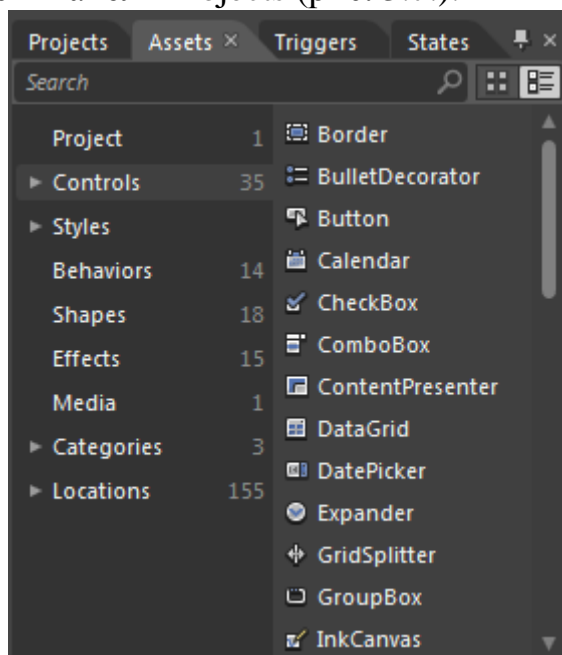


Рис. 3.7. Библиотека ресурсов

Здесь же можно получить доступ к триггерам событий.

В состав текущей версии Expression Blend включен полезный редактор исходного кода. Для того чтобы убедиться в этом, дважды щелкните на любом выбранном вами исходном файле (*.cs), доступном на панели Project.

Интегрированный в Expression Blend редактор исходного кода оказывается полезным в том случае, если требуется ввести код простой "заглушки" для обработчиков событий или написать несложный тестовый код в ходе разработки или создания прототипа приложения.

В Expression Blend существует возможность выявления синтаксических ошибок. Если вы попытаетесь выполнить приложение с синтаксическими ошибками нажав функциональную клавишу <F5> или комбинацию клавиш <Ctrl+F5>, на панели Results (Результаты) появится перечень обнаруженных ошибок.

Задания к лабораторной работе

Создайте новый проект (тематику определяем самостоятельно). При создании нового проекта необходимо указать один из шаблонов, предназначенный для построения традиционных настольных приложений на платформе WPF в виде исполняемых файлов.

Оформите отчет.

