UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Vývoj grafických aplikací s využitím WPF

Jiří Boš

Bakalářská práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jiří Boš**
Osobní číslo:	**I07580**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Vývoj grafických aplikací s využitím WPF**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Práce se bude po teoretické i praktické stránce věnovat programování aplikací s využitím Windows Presentation Foundation, především se zaměřením na vývoj grafických aplikací.
V teoretické části budou popsány nové možnosti aplikací využívajících WPF, zejména ve srovnání s aplikacemi postavených na WinForms. Především bude popsán značkovací jazyk XAML, možnosti animací, použití 3D grafiky, grafické efekty, práce s multimédii a možnost provázání dat v aplikaci pomocí DataBinding.
V praktické části budou implementovány ukázkové příklady, vhodně doplňující teoretickou část. V rámci praktické části bude vytvořen multimediální výukový materiál, seznamující uživatele se základy využívání technologie WPF.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. PETZOLD, Charles. Mistrovství ve Windows Presentation Foundation. Brno : Computer Press, 2008. 928 s. ISBN 978-80-251-2141-2.
2. ANDERSON, Chris. Essential Windows Presentation Foundation. London : Addison-Wesley, 2007. 458 s. ISBN 978-0-321-37447-9.

Vedoucí bakalářské práce: **Ing. Petr Veselý**
Katedra softwarových technologií

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne dd. mm. rrrr Jiří Boš

Poděkování

Chtěl bych poděkovat panu Ing. Petru Veselému za trpělivost, ochotu a odborné vedení, které mi věnoval během zpracování bakalářské práce. Dále bych chtěl poděkovat své rodině za podporu při studiu vysoké školy.

Anotace

Tato práce byla vytvořena za účelem poskytnutí základních učebních materiálů pro Windows Presentation Foundation. Práce je rozdělena do kapitol tak, aby čtenáři vytvořila všeobecný přehled o WPF. Velký důraz byl kladen na ukázky kódu v textu. Součástí práce je něco kolem 60 ukázkových aplikací. Ty jsou rozděleny podle kapitol. V teoretické části je pak na ně odkazováno. Některé aplikace obsahují i pokročilejší řešení některých problémů. Praktická část obsahuje webovou aplikaci s video ukázkami z některých témat této práce. U témat je uveden i stručný popis.

Klíčová slova

WPF, Windows Presentation Foundation, XAML, Data binding, 2D Grafika, 3D Grafika, animace, media, multimédia, efekty, element, atribut, události, styly, DataContext, Path, Shape, Viewport3D, Storyboard

Title

Název bakalářské práce v anglickém jazyce.

Annotation

Morbi fringilla risus ut arcu dapibus viverra ultricies dolor consequat. Fusce tempus ipsum congue sem blandit in suscipit turpis fermentum. Aliquam vel feugiat nisi. Donec mattis, purus eget sodales condimentum, sapien nunc vestibulum est, ac vehicula sem leo nec mi. Sed id luctus nulla. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Sed condimentum aliquet lorem in aliquet. Aliquam sed erat mauris. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nam eu dui in ante posuere pulvinar at vel purus. Phasellus consequat sollicitudin tempus. Ut nec elit nisi.

Keywords

Morbi, fringilla, risus, dapibus, viverra, ultricies, dolor

Obsah

1 Obsah

Seznam zkratek

WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
MSDN	Microsoft Developer Network
GUI	Graphical User Interface
UML	Unified Modeling Language
OOP	Objektově orientované programování

Seznam obrázků

Seznam tabulek

Úvod

Windows Presentation Foundation je platforma pro vytváření grafických uživatelských rozhraní (GUI). WPF bylo vytvářeno za účelem nahrazení starších technologií jako je Windows Forms. Na rozdíl od Windows Forms se ve WPF vytváří rozhraní v jazyce XAML. Tento jazyk je odvozen od jazyka XML. Jazyk XML je určen pro výměnu informací, ale může sloužit jako základ pro jiný jazyk. Důraz při vytváření rozhraní je kladen na deklarativní přístup pro vytváření rozhraní. V XAML se nepoužívají posloupnosti příkazů pro vytvoření rozhraní (imperativní přístup). Pomoci XAML se zapíše přímo jak má rozhraní vypadat. Programátor (designér GUI) není zatěžován s vytvářením objektů tříd. XML a XAML se označují jako značkovací jazyky. Značka (element) bude v XAML zastupovat objekt třídy. Při zpracování XAML dojde automaticky k vytvoření příslušných objektů. K těm je pak možné přistupovat i v kódu C# nebo Visual Basic .NET. WPF stále dovoluje použít tyto jazyky pro přidávání prvků rozhraní.

XAML dovoluje efektivně oddělit práci designérů a programátorů. Ti mohou téměř zároveň pracovat na obou částech aplikace. Tedy na rozhraní aplikace a aplikační logice. Přidáním XAML do vývoje aplikace, umožnilo vytvořit nástroje výhradně určené pro tvorbu rozhraní. Takovým nástrojem je například Microsoft Blend.

WPF obsahuje kromě vytváření oken aplikací a přidávaní ovládacích prvku i tyto části: data binding, 2D grafika, 3D grafika, Animace a Media. Tím výčet všech možností, co jsou ve WPF zdaleka nekončí. Ty co jsou vyjmenované + XAML budou tvořit obsah této práce. V poslední kapitole navíc bude ukázka přidání dokumentu do rozhraní aplikace. Což je také jednou z částí WPF.

WPF obsahuje speciální techniku pro načítání dat do rozhraní aplikace. Ta se nazývá data binding v překladu vazba dat. Slouží k vytvoření spoje mezi cílovou vlastností a zdrojem dat. Cílovou vlastností může být například nápis na tlačítku nebo text v textovém poli. Zdroj pak může představovat jinou vlastnost nebo třeba kolekci objektů. Některé ovládací prvky slouží pro zobrazení kolekcí dat. Proto je možné pomocí data bindingu do nich načíst přímo kolekci objektů. Na zobrazované objekty pak lze aplikovat datovou šablonu. Ta zobrazí data pomocí ovládacích prvků v rámci původního zobrazení.

Součástí WPF je také grafický systém dovolující přidávat grafiku do rozhraní aplikace. Ten dovoluje přidávat různé 2D a 3D objekty. Některé 2D objekty mají stejné vlastnosti jako ovládací prvky. Je tedy možné je přímo umístit do rozhraní aplikace. 2D Grafika je založena především na vektorové grafice. Co se týče 3D grafiky ta je zobrazována pomocí speciálního ovládacího prvku. WPF dovoluje vytvářet prvky definováním vrcholů trojúhelníků. Ty pak tvoří povrch 3D objektu. Na objekty je možné pak nanést materiál. Pomocí různých druhů materiálů je možné vytvořit dojem povrchu různých objektů ze skutečného světa. Objekt je možné také osvětlit pomocí několika typů světel. Zobrazování probíhá pomocí kamery. Kamera pak používá jednu z technik promítání 3D objektů na dvourozměrný povrch.

WPF obsahuje velice zdařilý systém vytváření animací. Ten je vytvořen tak, aby co nejvíce zjednodušil vytvoření animace. Pro vytvoření animace jsou k dispozici tři druhy animací. Ty se dále dělí na to, pro jaký typ vlastnosti jsou. WPF dovoluje přidat animaci do rozhraní prostřednictvím XAML. Spuštění animace se provede taktéž pomocí XAML.

WPF taktéž podporují média: obrázky, zvuk a video. Pro přehrávání zvuku a videa se používá stejný ovládací prvek. Přehrávání je možné ovládat pomocí funkcí Play, Stop a Pause. Je také možné zjistit aktuální pozici přehrávání. Zacházení s medii je ve WPF velice intuitivní.

Pro čtení textu je dobré mít základní znalosti z OOP: třída, dědění, metody… Taktéž je potřeba alespoň základní znalost programovacího jazyka C#: vlastnosti, vytvoření třídy, metody, parametry funkce, události… V některých částech práce jsou sice vysvětlovány některé tyto prvky. Nicméně není to příliš obsáhlé z důvodu zaměření samotné práce a nedostatku místa. Veškeré příklady jsou vytvořeny v prostředí Visual Studio 2012. Pro jistotu správného chodu ukázkových aplikací je potřeba mít nainstalován .NET Framework 4.5.

2 XAML

XAML je speciální typ jazyka. Tento jazyk ve WPF slouží jako prostředek pro vytváření grafického rozhraní aplikací. Syntaxe tohoto jazyka se liší od C# nebo Visual Basic .NET. XAML není zaměřen na psaní algoritmů nebo vytváření datových struktur. Jeho účelem je vytvořit objekty a logicky je uspořádat. Ve WPF nejčastěji uspořádává ovládací prvky tak, aby vytvořily rozhraní aplikace.

V XAML se nevytváří třídy, metody, datová pole, vlastnosti, události... tyto konstrukce stále patří jazykům jako je C# nebo Visual Basic .NET. XAML tyto konstrukce dokáže jenom použít. Pomocí XAML je možné vytvořit například objekt třídy. Nebo je možné nastavit hodnotu některé z objektu vlastností. Ukázka kódu XAML:

Ukázka kódu napsaného v XAML

```
<StackPanel>
     <Button Width="120" Content="Tlačítko 1" Margin="0,5"/>
     <Button Width="120" Margin="0,5">Tlačítko 2</Button>
</StackPanel>
```

StackPanel *– je jedním z layoutů uspořádávajících prvky*
Button *– tento element představuje objekt vytvořený od třídy* ***Button***
Width, Content, Margin *– jedná se o atributy nastavující hodnoty vlastností objektu*

Kód XAML se ukládá do souboru s příponou .xaml. Tento kód je většinou nezkompilován a je načítán za běhu programu.

Při vytvoření projektu WPF ve Visual Studiu je vytvořen soubor s názvem MainWindow.xaml. Tento soubor obsahuje hlavní okno aplikace + rozhraní aplikace. S tímto souborem je ještě vytvořen soubor MainWindow.xaml.cs. V tomto souboru je vytvořena třída **`MainWindow`**. Tato třída obsahuje takzvaný „code-behind“. Tedy kód tvořící interaktivní logiku aplikace. Vytváří se zde například obslužné rutiny událostí nebo libovolné další členy třídy. V této třídě se už používá jazyk C# nebo Visual Basic .NET.

Ukázka třídy MainWindow

```
public partial class MainWindow : Window
{
   public MainWindow()
   {
      InitializeComponent();

      //sem je možné psát další kód
   }

//Zde mohou být další součásti třídy (metody, datová pole, vlastnosti,
událostí...)
}
```

Nejsnazší cesta jak vytvořit rozhraní ve Windows Forms a WPF je použitím designéru. Ten je součástí Visual Studia a dovoluje vytvářet rozhraní aplikace. WPF, ale nabízí i možnost editovat rozhraní pomocí XAML. Pokud se rozhraní aplikace vytváří pomocí designéru. Kód je automaticky generován. Generovaný kód je umístěn ve Windows Forms v souboru <nazev_form>.Designer.cs (například Form1.Designer.cs). U WPF se vytváří přímo XAML kód (například v MainWindow.xaml). Ve WPF se vytvořený kód pomocí designéru může editovat. Naproti tomu u Windows Forms se to nedoporučuje.

2.1 Element objektu

V XAML existuje vícero typů elementů. Mezi tyto typy patří i element objektu. Jedná se o nejčastěji používaný typ elementu. Každý element objektu zastupuje objekt vytvořený podle třídy. Tyto elementy pak dovolují nastavovat vlastnosti objektů pomocí atributů. Nejčastěji se elementy používají pro umístění ovládacích prvků do rozhraní aplikace.

Ukázka elementu objektu s nastavením vlastnosti pomocí atributu

```
<Button Width="120" Content="Tlačítko 1" Margin="0,5"/>
```

Button *– element objektu zastupuje objekt vytvořený podle třídy* ***Button*** *(tlačítko)*
Width *– nastavuje vlastnost „šířka tlačítka“*
Content *– nastavuje vlastnost zodpovědnou za nápis na tlačítku*
Margin *– nastavuje vlastnost „odsazení tlačítka od okolních objektů“ (přidá nahoře a dole 5 pixelové místo)*
V ukázce není nastavena žádná vlastnost pro přesné umístění tlačítka v rozhraní aplikace. Protože tlačítko je umístěnou v layoutu ***StackPanel****. Tento layout řadí ovládací prvky podle toho, jak jsou postupně vytvářeny (viz. Aplikace ElementObjektu).*

Atributy (`Width, Content, Margin...`) nastavují hodnoty vlastností jako textové řetězce umístěné v uvozovkách. Samotné přiřazení hodnoty se provede pomocí znaku = (rovná se). Hodnoty je, ale nejprve potřeba převést na odpovídající typ. Ve většině případů se o to postará rovnou WPF.

Některé elementy dovolují nastavit hodnotu přímo. Hodnotu je pak možné přidat mezi počáteční a koncovou značku elementu.

Přímý obsah elementu

```
<Button Width="120" Margin="0,5">
    Tlačítko 2
</Button>
```

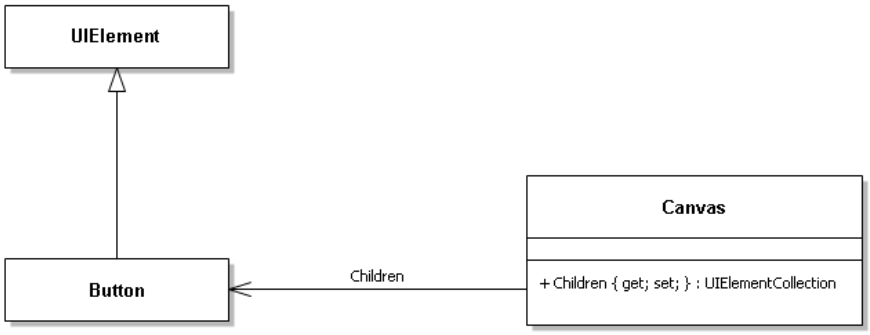
<Button Width…> *– počáteční značku elementu*
</Button> *– koncová značka elementu*
Tlačítko 2 *– přímý obsah elementu*
Přímý obsah nastavuje hodnotu pro vlastnost Content třídy Button[1].

[1] Která vlastnost je přímým obsahem elementu se dá zjistit v dokumentaci (MSDN).

Použití přímého obsahu typu kolekce u StackPanel

```
<StackPanel>
     <Button Width="120" Content="Tlačítko 1" Margin="0,5"/>
     <Button Width="120" Margin="0,5">Tlačítko 2</Button>
</StackPanel>
```

*Přímý obsah u **StackPanel** nastavuje hodnotu vlastnosti **Children**. Tato vlastnost by se jen těžko dala nastavit pomocí atributu. Proto je lepší ji nastavit pomocí přímého obsahu. Vlastnost **Children** je typu kolekce **UIElementCollection**.*



Obrázek 1 Přímý obsah elementu Canvas[2]

Na Obrázku 1 je zobrazena třída (element) **`Canvas`**. Vlastnost *`Children`* je nastavena jako přímý obsah elementu **`Canvas`**[3]. *`Children`* je typu **`UIElemetCollection`** a to umožňuje vnořovat další elementy odvozené od třídy **`UIElement`**. Na obrázku 1. je uveden příklad s elementem **`Button`**[4].

Ukázkové řešení: XAML/ElementObjektu

Ve Windows Forms sice elementy nejsou, ale je možné ovládací prvky vytvářet v kódu-za. Například je možné ovládací prvek vytvořit v konstruktoru třídy **`Form`**. A pomocí vlastnosti *`Controls`* ho přidat do rozhraní. Windows Forms ovládací prvky nedovolují používat tak bohatý obsah jako je tomu ve WPF. Například nápis na tlačítku ve WPF může být i třeba další layout s dalšími elementy.

Přidání tlačítka ve Windows Forms pomocí C#

```
Button tlacitko = new Button();
tlacitko.Location = new Point(38, 26);
tlacitko.Size = new Size(75, 23);
tlacitko.Text = "Tlacitko";

this.Controls.Add(tlacitko);
```

tlacitko *– název reference na objekt*
Location *– určí pozici tlačítka (horní levý roh)*

---

[2] Diagramy tříd byly vytvořeny za pomocí programu NClass. NClass naleznete na adrese: http://nclass.sourceforge.net/.
[3] Canvas je jedním z layoutů umožňující vizuálně uspořádat elementy (Button).
[4] Šipka od třídy Button k třídě UIElement označuje odvození (dědění) od třídy UIElement.

Size *– velikost tlačítka*
Text *– nápis na tlačítku*
Controls *– kolekce ovládacích prvků pro tento formulář (**Form1**)*

Ukázkové řešení: Windows Forms/VytvoreniOvladacihoPrvkuCSharp

2.2 Pojmenování elementu

Pojmenování elementu slouží pro odkazování se na tento element v XAML. Tak i k vytvoření reference na tento objekt v C#. V „kódu-za" je pak možné pracovat s tímto objektem. Pojmenování elementu se provede za pomoci atributu `Name` nebo `x:Name`.

Pojmenování elementu TextBox pro vypsání textu

```
<Canvas>
    <StackPanel Orientation="Horizontal"
                    Canvas.Left="10"
                    Canvas.Top="10"
                    Height="25">

        <TextBox x:Name="wpfTextBox" Width="200" />
        <Button Width="100" Click="Button_Click">WPF</Button>
     </StackPanel>
</Canvas>
```

Canvas *– tento layout umožňuje přesné umístění GUI prvku na ploše layoutu*
Canvas.Left *– umístí prvek GUI od levého okraje layoutu **Canvas***
Canvas.Top *– umístí prvek GUI od horního okraje layoutu **Canvas***
Orientation *– **StackPanel** bude GUI prvky řadit ne pod sebe, ale vedle sebe na řádek*
Height *– určí výšku **StackPanel** a v tomto případě i výšku všech GUI prvku uvnitř layout*
x:Name (Name) *– vytvoří odkaz na tento element (C#/XAML)*
Click *– atribut zastupující událost kliknutí na tlačítko + přiřazení obslužné metody pro tuto událost*
Button_Click *– název metody provedené po kliknutí na tlačítko (obslužná metoda)*
*Atributy **x:Name** a **Name** vytvoří odkaz na objekt zastoupený tímto elementem. Atributy **Left** a **Top** slouží jako souřadnice pro umístění GUI prvků.*

Použití názvu u elementu TextBox

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    wpfTextBox.Text = "Windows Presentation Foundation";
}
```

*Metoda **Button_Click** má stejný název jako hodnota u atributu **Click**. Parametry této metody předávají další informace o této události.*
wpfTextBox *– je názvem elementu, sloužící jako referenční proměnná pro objekt zastoupený elementem*
Text *– je vlastnost pro vypsání textu v TextBox*

Ukázkové řešení: XAML/PojmenovaniElementu

Pojmenování ovládacího prvku ve Windows Forms se provede pomocí vlastnosti `Name`. Ta se nastavuje ve Visual Studiu v panelu Properties[5] (Vlastnosti).

Použití názvu ovládacího prvku ve Windows Forms

```
nazevTextBox.Text = "Windows Forms";
```

nazevTextBox – název pro ovládací prvek TextBox

Ukázkové řešení: Windows Forms/ PouzitiNazvuOvladPrvku

2.3 Události

Použití událostí (event) ve WPF je velice snadné. Pokud element (třída) má nějaké události. XAML zpřístupní tyto události jako atribut. Název atributu bude stejný jako název události. Například událost *Click* u třídy **`Button`** bude přístupná pomocí atributu s názvem `Click`. Jako hodnota atributu se uvede název obslužné metody pro tuto událost.

Událost MouseLeftButtonDown u elementu Rectangle (obdélník)

```
<Rectangle
            x:Name="obdelnik"
            Canvas.Top="40"
            Canvas.Left="177"
            Fill=" #FFF4F4F5"
            Height="100"
            Width="100"
            Stroke="Black"
            MouseLeftButtonDown="obdelnik_MouseLeftButtonDown"/>
```

Rectangle *– vytvoří obdélník o výšce (**Height**) 100 a šířce (**Width**) 100, umístěný pomocí vlastností **Canvas.Top** a **Canvas.Left***
MouseLeftButtonDown *– atribut zastupující událost (kliknutí levým tlačítkem myši)*
Fill *– vyplní obdélník zadanou barvou*
Stroke *– určí barvu obrysu obdélníku*
*Je důležité, aby GUI prvek (**Rectangle**) měl nastavenou barvu výplně. Bez toho nebude fungovat určení pozice na tomto prvku.*

Zjištění pozice kliknutí myši na obdélníku

```
void obdelnik_MouseLeftButtonDown(object sender, MouseButtonEventArgs  e)
{
    Point poziceKliknuti = e.GetPosition(obdelnik);

    xTextBox.Text = poziceKliknuti.X.ToString();
    yTextBox.Text = poziceKliknuti.Y.ToString();
}
```

obdelnik_MouseLeftButtonDown *– tato metoda se zavolá, pokud dojde ke vzniku události **MouseLeftButtonDown***
sender *– objekt, který vyvolal tuto událost*

[5] Stejný panel je ve WPF použit pro nastavování atributů elementů.

e *– objekt třídy MouseButtonEventArgs nese další informace o události*
e.GetPosition(obdelnik) *– zjistí pozici myši na obdélníku, kdy došlo k této události*
Pozice myši bude relativní k objektu obdélníku. Bude počítána od levého horního okraje obdélníku

`MouseButtonEventArgs` není jediným typem argumentu vráceného událostí. Například událost *KeyDown* vrací argument typu `KeyEventArgs`. Tento objekt nese informace o klávesách, které byly stlačeny. Informace o typu argumentu události jsou uvedeny v dokumentaci na MSDN[6]. Zde je také možné zjistit jaké vlastnosti a metody tento typ poskytuje.

Ukázkové řešení: XAML/UdalostKliknutiMysiNaObdelnik

Přidání události k ovládacímu prvku ve Visual Studiu je stejný jak pro WPF tak i pro Windows Forms. V panelu Properties se nachází malá ikona blesku. Kliknutím na ni se vypíší události k danému ovládacímu prvku. Zde stačí přidat název obslužné metody události.

Přidání obslužné metody pro Windows Forms a WPF v C#

```
tlacitko.Click += tlacitko_Click;
```

Click *– událost Click u třídy Button*
`+=` *– použije se pro připojení obslužné metody*
K události může být připojeno vícero obslužných metod.

2.4 Připojené vlastnosti

GUI prvky ve WPF většinou nemají žádné vlastnosti určené pro umístění prvku na layoutu. Proto je potřeba umět tyto vlastnosti k těmto GUI prvkům připojit. Například layout `Grid` obsahuje dvojici připojených vlastností `Grid.Row` a `Grid.Column`. Tyto vlastnosti určí řádek a sloupec pro umístění GUI prvků. Tedy GUI prvek bude umístěn do buňky určené vlastností `Row` (řádek) a `Column` (sloupec).

Umístění GUI prvku Label pomocí připojených vlastností

```
<Label Grid.Row="0" Grid.Column="0" Content="Row=0 ; Column=0" FontSize="20"/>
```

Vlastnosti ***Grid.Row*** *a* ***Grid.Column*** *umístí GUI prvek* ***Label*** *do buňky se souřadnicemi [0,0].*

Připojená vlastnosti se dají také nastavit v „kódu-za“. Pro připojení vlastnosti se použije název třídy + název připojované vlastnosti.

Připojení vlastnosti v C#

```
Button tlacitko = new Button();
tlacitko.Content = "Ok"

Grid.SetRow(tlacitko, 1);
```

[6] http://msdn.microsoft.com/en-US/

```
Grid.SetColumn(tlacitko, 1);

gridUkazka.Children.Add(tlacitko);
```

*Metody **SetRow** a **SetColumn** nastaví hodnoty připojených vlastností pro objekt **tlacitko**. V tomto případě jsou hodnoty nastaveny na 1 a 1. Což značí, že tlačítko bude umístěnou ve druhém sloupci a druhém řádku. Tyto metody jsou volány **pomocí jména třídy** a názvu metody (statické metody).*

Metody sloužící pro nastavení připojených vlastností mají většinou stejnou syntaxi. Nejdříve je uvedeno slovo **Set** následované jménem vlastnosti. Například `Canvas.Top` bude v C# zapsáno `Canvas.SetTop(tlacitko, 100)`.

Ukázkové řešení: XAML/PripojeneVlastnosti

2.5 Element vlastnosti

Některé vlastnosti elementu mohou obsahovat složitější typ hodnot. Tyto hodnoty nelze jednoduše zapsat jako textový řetězec. Může se jednat například o jiný element. Proto existuje i jiný způsob jak zapsat hodnotu vlastnosti.

Ukázka použití elementu vlastnosti

```
<Button Width="242" Height="45" Canvas.Left="40" Canvas.Top="23">
   <Button.Content>
       <StackPanel Orientation="Horizontal">
          <Rectangle Stroke="Blue" Fill="Blue" Width="20" Height="20"/>
          <TextBox>Tlačítko OK</TextBox>
       </StackPanel>
   </Button.Content>
</Button>
```

<Button.Content>** – element vlastnosti určený pro nastavení hodnoty vlastnosti **Content
*Vlastnost **Content** je typu **Object** může obsahovat jakýkoliv objekt (**string, GUI prvek, DateTime**…).*

Vlastnost `Content` není přímo vlastností třídy **`Button`**. Tuto vlastnost získává od třídy **`ContentControl`**. Třída **`ContentControl`** dědí od třídy **`Control`** vlastnost **`Template`**. Tato vlastnost obsahuje šablonu vzhledu ovládacího prvku – **`ControlTemplate`**. Šablona dále používá **`ContentPresenter`** pro zobrazení obsahu GUI prvku. **`ContentPresenter`** je speciální typ objektu. Zastupuje obsah předaný pomocí vlastnosti `Content`. U tlačítka určuje místo v šabloně, kde bude zobrazen nápis.

Ukázkové řešení: XAML/ElementVlastnosti

2.6 Zdroje

WPF dovoluje vytvořit objekt jako zdroj pomocí elementu. Na tento zdrojový objekt je potom možné se odkazovat pomocí **klíče (`x:Key`).** Toto se využívá například při data bindingu, použití stylů nebo vytvoření objektu vlastní třídy.

Vytvoření zdroje pro element Canvas

```
<Canvas.Resources>
```

```
    <SolidColorBrush x:Key="vyplnObdelniku" Color="GreenYellow"/>
</Canvas.Resources>
```

Canvas.Resources *– tato vlastnost vytváří kolekci zdrojů označených klíčem*
x:Key *– vytvoří identifikační klíč ke zdroji, pomocí něho se pak odkazuje na tento objekt*
Při vytváření zdrojového objektu je nutné vytvořit klíč.

Pro použití tohoto objektu je potřeba použít speciální zápis – `{StaticResource ResourceKey=Key}`. Tento zápis pak umožní použít tento objekt například, jako hodnotu atributu.

Použití zdroje vyplnObdelniku z předchozího příkladu

```
<Rectangle Fill="{StaticResource ResourceKey=vyplnObdelniku}"
                   Height="100"
                   Canvas.Left="42"
                   Stroke="Black"
                   Canvas.Top="22"
                   Width="100"/>
```

ResourceKey *– je možné vynechat (stačí jenom zapsat hodnotu* ***x:Key*** *atributu)*

Viditelnost zdroje v ukázce je pouze v rámci potomků tohoto elementu. Například pokud by tento zdroj byl vytvořen jako `Rectangle.Resource`. Dostupnost tohoto zdroje by platila pouze pro jeho vnořené objekty (potomky).

Ukázkové řešení: XAML/VytvoreniZdroje

2.7 Styly

Styly představují další možnost nastavení vlastností na objektech. Používají se nejčastěji, kdy změny vlastností budou aplikovatelné i na dalších objektech. Tedy tyto změny budou znovu použitelné. Styly se také dají použít pro zlepšení čitelnosti kódu. Místo přímé editace vlastnosti pomocí atributu. Hodnota vlastnosti je změněna pomocí stylu. V elementu tak budou nastaveny jen některé atributy nezahrnuté v tomto stylu.

Vytvoření stylu pro určitý typ

```
<Style TargetType="ListBoxItem">
      <Setter Property="Background" Value="LightGray"/>
      <Setter Property="FontWeight" Value="Bold"/>
</Style>
```

Style *– slouží k vytvoření stylu (objektu stylu)*
TargetType *– typ (element) na jaký bude tento styl aplikován*
Setter *– nastaví hodnotu vlastnosti určené pomocí atributu* ***Property***
Property *– název vlastnosti objektu (****ListBoxItem****)*
Value *– nová hodnota pro vlastnost (****Property****)*
ListBoxItem *– tvoří položku v* ***ListBox***
ListBox *– vytvoří list položek, které je možné vybrat (třeba menu)*

*V tomto případě se aplikuje styl pouze na položky v **ListBox**. Nastaví jim barvu pozadí na **LightGray** (světle šedou). Dále nastaví váhu písma (tloušťku) na **Bold** (tučné).*

V ukázce je vytvořen styl v `StackPanel.Resources`. Tento zdroj nepotřebuje nastavit hodnotu pro `x:Key` atribut. Zde bude automaticky nastavena hodnota na `{x:Type ListBoxItem}`. Pokud bude `x:Key` nastaven na jinou hodnotu, styl se neaplikuje na všechny elementy **`ListBoxItem`**. Ale aplikuje se pouze na ty s přiřazeným stylem pomocí `{StaticResources}`.

Vytvoření stylu s x:Key atributem

```
<Style x:Key="vlastniStylTlacitka" TargetType="Button">
      <Setter Property="Background" Value="White"/>
</Style>
```

*Zde je styl pojmenován pomocí **x:Key** atributu. Nebude tedy aplikován na všechny **Button** elementy.*

Použití stylu na element Button

```
<Button Style="{StaticResource vlastniStylTlacitka}">Přidat</Button>
```

*Pomocí atributu **Style** a **StaticResource** se přidá styl **vlastniStylTlacitka**.*

Ukázkové řešení: XAML/Styly

2.8 Spínače

Spínače jsou další způsob jak přidat interaktivitu do aplikace. Na rozdíl od událostí se kód dá zapsat pomocí XAML. Pomocí spínačů je možné reagovat na události nebo změnu hodnoty vlastnosti. Reakce na sepnutí spínače může třeba změnit hodnotu vlastnosti ve stylu. V kapitole Animace budou použity ke spuštění animace.

Vytvoření spínače ve stylu

```
<Style x:Key="textBoxStyl" TargetType="TextBox">
      <Setter Property="Text" Value="White"/>

      <Style.Triggers>
          <Trigger Property="IsMouseOver" Value="true">
                <Setter Property="Background" Value="GreenYellow"/>
                <Setter Property="Text" Value="GreenYellow"/>
          </Trigger>
      </Style.Triggers>
</Style>
```

Style.Triggers *– element vlastnosti pro nastavení kolekce spínačů*
Trigger *– vytvoří konkrétní spínač s podmínkou pro sepnutí*
Property *– sledovaná vlastnost*
Value *– hodnota, kdy sepne spínač*
IsMouseOver *– nastaví se na true pokud bude kurzor myši nad tlačítkem (událost **MouseOver**)*

Trigger spínač lze použít pouze ve spojitosti se stylem. Elementy jako jsou **Button** nebo **TextBox** mají vlastnost **Triggers**. Tato vlastnost však podporuje pouze spínače spuštěné událostí (**EventTrigger**). **EventTrigger** se používají nejčastěji ke spuštění animace[7].

Ukázkové řešení: XAML/Spinace

2.9 Instance vlastní třídy

Instance vlastní třídy se vytváří pomocí elementu objektu. Nejprve je, ale nutné připojit jmenný prostor třídy. To zviditelní tuto třídu pro dokument XAML. Tento element pak může být použit, jako zdroj dat pro data binding.

Přidání jmenného prostoru třídy

```
xmlns:s="clr-namespace:InstanceVlastniTridyXAML"
```

s *– použije se pro vytvoření elementu objektu ze jmenného prostoru InstanceVlastniTridyXAML (<s:Nazev_elementu>)*
clr-namespace *– označuje, ve kterém jmenném prostoru se nachází třída pro vytvoření objektu (elementu)*
InstanceVlastniTridyXAML *– jmenný prostor třídy*
Celý tento kousek kódu je zapsán jako atribut v elementu ***Window.***

Prefix (**s**) je potřeba použít pro vytvoření elementu objektu. Bez prefixu jsou vytvářeny elementy WPF. Ty jsou umístěný v jiném jmenném prostoru.

Vytvoření objektu vlastní třídy

```
<s:Student x:Key="student"
           OsobniCislo="1"
           Jmeno="Jan"
           Prijmeni="Novák"
           Fakulta="FEI"/>
```

s:Student *– vytvoří objekt třídy Student*
OsobniCislo, Jmeno*… – nastaví hodnoty pro vlastnosti objektu třídy Student*
x:Key *– objekt je použit jako zdroj (****Grid.Resources****)*
Hodnoty budou automaticky převedeny na odpovídající typ.

Ukázkové řešení: XAML/InstanceVlastníTridy

[7] Viz. Kapitola Spuštění animace

3 Data binding

Po vytvoření rozhraní aplikace je potřeba zobrazit data aplikace. To se dá udělat pomocí kódu umístěného do těla metody. Tato metoda se pak volá z konstruktoru nebo z obslužné rutiny události. Načte potřebná data a umístí je do příslušných elementů. To je dostačující pro jednoduché aplikace, ale ne pro složité. Při vývoji aplikace je někdy nutné vyměnit rozhraní aplikace. Což znamená přepsat některé z metod načítajících data. Způsob načtení dat ze zdroje se třeba nezmění, ale způsob zobrazení dat ano. Může také dojít ke změnám v názvech elementů. V každém případě bude muset programátor tento kód opravit.

Pomocí data bindingu je možné přesunout načítání dat přímo do rozhraní aplikace. V XAML se vytvoří vazba mezi zdrojem a cílem dat. Odkud se data načtou, záleží na tom, jak je aplikace navržena. Data binding dovoluje získat data z vlastností objektů. Ty pak mohou být předány některému z atributů elementu.

Důležitá je i synchronizace mezi zobrazením dat a zdrojem těchto dat. Pokud se změní zdrojová data, je důležité, aby se změna promítla i do zobrazení. Někdy je také potřeba změnit přidružená data přímo v elementu, jež je zobrazuje. Tak, aby se automaticky upravená data v elementu promítla i do zdroje dat.

Data binding řeší většinu problémů spojených s načítáním dat do rozhraní aplikace. Vytvoří spoj mezi zdrojem a cílem dat. Vlastnosti tohoto spoje jsou uloženy v jediném objektu. Tento objekt je vytvořen jako instance třídy Binding.

3.1 Data binding – C#

Pro vytvoření data bindingu v C# je potřeba vytvořit objekt třídy **`Binding`**. Tento objekt obsahuje informace o zdroji dat a vlastnostech vytvořené vazby. Po vytvoření objektu Binding je potřeba přiřadit tento objekt k cílové vlastnosti. To se provede pomocí metody `SetBinding(DependencyProperty` **`dp`**`, BindingBase` **`binding`**`)`. Parametr **`dp`** slouží pro nastavení cílové vlastnosti. Tato vlastnost se zapisuje *<nazev_elementu>.<nazev_vlastnosti>Property*. Parametru **`binding`** se předává objekt třídy Binding.

Vytvoření objektu Binding

```
Binding binding = new Binding();
binding.Source = student;
binding.Path = new PropertyPath("Jmeno");
textBox.SetBinding(TextBox.TextProperty, binding);
```

Binding *– obsahuje informace o vytvářené vazbě*
binding *– reference na objekt třídy Binding*
Source *– reference (odkaz) na objekt (student)*
student *– objekt vytvořený od třídy Student*
Path *– cesta k datům (vlastnost objektu student)*
PropertyPath *– parametrem konstruktoru je jméno vlastnosti (objektu student)*
TextBox.TextProperty *– cílová vlastnost pro načtení dat*

Ukázkové řešení: DataBinding/DatabindingCSharp

3.2 Data binding – atribut

Někdy je potřeba získat data z atributu a zapsat je jako hodnotu jiného atributu. WPF dovoluje vytvořit data binding i v XAML. To sníží množství kódu vytvářeného v „kódu-za“. Pro připojení data bindingu stačí přidat následující kód jako hodnotu atributu: `{Binding ElementName=nazev_elementu, Path=nazev_vlastnosti}`.

```
<Slider x:Name="slider"
       Maximum="100"
       IsSnapToTickEnabled="True"
       TickFrequency="1"
       Value="0"/>

<ProgressBar Value="{Binding ElementName=slider, Path=Value}"/>
```

Slider *– posuvník měnící hodnotu vlastnosti Value*
Value(Slider) *– hodnota se mění s posunem Slideru (zdroj dat)*
ProgressBar *– zobrazuje stav probíhající operace*
ElementName *– název elementu s atributem obsahující potřebná data*
Path *– název atributu odkud se načtou data*
Value *– nastaví hodnotu ProgressBar*

Ukázkové řešení: DataBinding/DBAtribut

3.3 Data binding – vlastní třída

Data binding je možné použít i na instanci třídy v XAML. Tato třída bude vytvořena jako **Resources** s klíčem `x:Key`. Hlavní rozdíl oproti data bindingu u atributu je způsob odkazování se na element. Pro získání odkazu na element je zapotřebí použít zápis `{StaticResource klic}`.

Načtení dat z vlastní třídy

```
<Label Content="{Binding Source={StaticResource student},Path=Jmeno}"/>
```

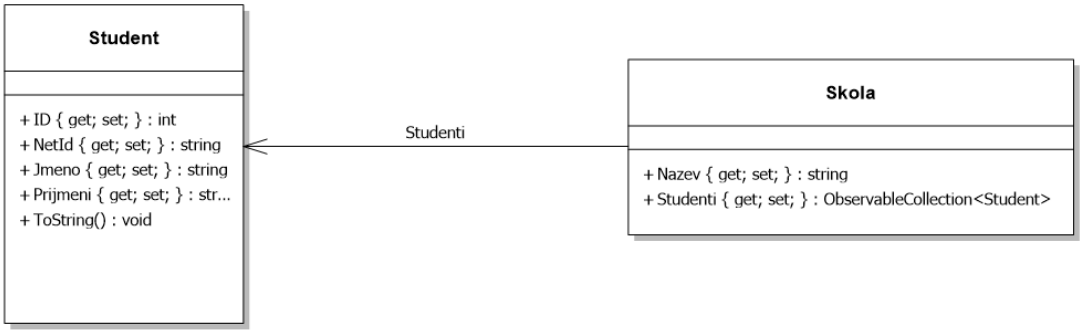
Source *– odkaz na objekt vytvořený v Resources*
student *– objekt vytvořený pomocí XAML (x:Key)*
Path *– název atributu odkud se načtou data*

Ukázkové řešení: DataBinding/DBVlastniTrida

3.4 Data binding – kolekce

Data binding dovoluje načítat kolekce objektů do elementů. WPF také obsahuje elementy určené pro zobrazování kolekce objektů (**`ListBox`**, **`ListView`** nebo **`TreeView`**). Všechny tyto elementy mají společný atribut `ItemSource`. Tento atribut slouží pro načtení objektů

z kolekce. Vytvoření Data bindingu pro kolekce je podobné jako v předchozí kapitole. Tedy za předpokladu, že třída bude vytvořena v XAML[8].



Obrázek 2 Diagram třídy Skola pro data binding

Na Obrázku 2 je ukázka diagramu třídy **`Skola`**. Tato třída má vlastnost *`Studenti`* typu **`ObservableCollection<Student>`**. Třída **`ObservableCollection`** je generickou[9] kolekcí pro různé typy objektů. Její nejdůležitější vlastností je že dokáže upozornit na změnu stavu kolekce a prvků. Nebo také, pokud se změní stav vloženého prvku[10]. To je důležité pro správné fungování data bindingu. Přidání nebo odebrání prvku z kolekce ovlivní zobrazená data.

Vytvoření objektu třídy Škola

```
<local:Skola x:Key="skola">
  <local:Skola.Studenti>
     <local:Student ID="0" Jmeno="Michal" Prijmeni="Novotný" NetId="st22222"/>
       <local:Student ID="1" Jmeno="Petr" Prijmeni="Svoboda" NetId="st22001"/>
   </local:Skola.Studenti>
</local:Skola>
```

Zobrazení studentů pomocí elementu ListBox

```
<ListBox ItemsSource="{Binding Source={StaticResource skola},Path=Studenti}"/>
```

ListBox *– zobrazí kolekci objektů (v tomto případě je vypíše pomocí ToString())*
ItemsSource *– teto atribut bude odkazovat na kolekci objektů*

Ukázkové řešení: DataBinding/DBKolekce

3.5 DataContext

`DataContext` dovoluje zpřístupnit data určitému elementu. Přesněji řečeno přidá k elementu související data. Takto zpřístupněná data je možné použít pomocí data bindingu. Připojit

[8] Další způsob může být i příklad z kapitoly Data binding – C#. Vlastnost Source bude odkazovat na objekt kolekce (Studenti). Vlastnost Path není potřeba nastavovat. Lepším řešením je použití DataContext viz. kapitola DataContext.
[9] V ostrých závorkách je uveden typ (<Student>) pro který je tato kolekce určena. To znamená, že je možné vložit pouze objekt tohoto typu nebo odvozeného typu. Pokus vložení jiného typu objektu povede k chybě při kompilaci kódu.
[10] Viz. kapitola INotifyPropertyChanged

data k elementu pomocí *DataContext* je možné vytvořit i v „kódu-za“. Vlastnost *DataContext* je možné použít u elementu odvozeného od třídy **FrameworkElement**.

Připojení dat pomocí DataContext k elementům ListBox a Label

```
Tema wpf = new Tema("Windows Presentation Foundation");

//přidání kapitol k tématu
wpf.Kapitoly.Add("XAML");
wpf.Kapitoly.Add("Data binding");

//přiřazení datového kontextu k elementům ListBox a Label
kapitolyListBox.DataContext = wpf;
temaLabel.DataContext = wpf;
```

*Třída **Tema** je podobná třídě Skola. Obsahuje kolekci **ObservableCollection<string> Kapitoly.** Jako prvky této kolekce jsou přidány názvy kapitol.*

Po připojení *DataContext* k elementu není potřeba uvádět u zápisu {Binding} název elementu nebo odkaz na objekt s daty. To platí v rámci elementu a vnořených elementů (potomků elementu).

Data binding u elementu s DataContext

```
<ListBox x:Name="kapitolyListBox" ItemsSource="{Binding Path=Kapitoly}">
   <ListBox.ItemsPanel>

        <ItemsPanelTemplate>
             <StackPanel Orientation="Horizontal"/>
        </ItemsPanelTemplate>

    </ListBox.ItemsPanel>
</ListBox>
```

***ItemsPanelTemplate** – určuje, jakým způsobem budou vizuálně uspořádány prvky v **ListBox**
V tomto případě budou prvky umístěny do **StackPanel** s horizontální orientací.*

Ukázkové řešení: DataBinding/DataContext

3.6 Směr toku dat

Směr toku dat určuje, jakým způsobem budou data přenášena. Zda se jenom načtou bez dalšího ovlivňování. Nebo budou se měnit v určitém směru. Například změna zdrojových dat se projeví i u zobrazení těchto dat. Nebo změna zobrazených dat se projeví i na zdroji těchto dat.

Nastavení směru toku dat se nastaví pomocí atributu Mode u zápisu {Binding}. V Tabulce 1 jsou uvedeny možné parametry této vlastnosti:

Tabulka 1 Možnosti nastavení vlastnosti Mode u data bindingu

Mode	Popis
TwoWay	Obousměrný tok dat. Při změně zdroje dojde ke změně zobrazený dat (cíle). Při změně cíle (zobrazených dat) dojde ke změně zdrojových dat.

	Zobrazená data, lze změnit, pouze pokud to umožňuje zobrazující element. (TextBox)
OneWay	ednosměrný tok dat. Při změně zdroje dojde ke změně zobrazených dat (cíle). Opačně to neplatí. Tedy změna zobrazených dat **neovlivní** zdrojová data.
OneWayToSource	Změna zobrazených dat se projevý na zdroji dat. Zobrazená data se nezmění po změně zdrojových dat.
OneTime	Data jsou načtena pouze jednou a to při načtení rozhraní.

Použití atributu Mode a UpdateSourceTrigger

```
<TextBox Text="{Binding ElementName=oneWayTextBox,
                        Path=Text,
                        Mode=OneWay,
                        UpdateSourceTrigger=PropertyChanged}"/>
```

oneWayTextBox *– je zdrojem dat pro tento* ***TextBox***
Mode *– způsob jakým se bude ovlivňovat cíl a zdroj dat u data bindingu*
UpdateSourceTrigger *– kdy se změna zobrazených dat má projevit na zdroji těchto dat*
PropertyChanged *– ke změně zdrojových dat dojde pokaždé, když se změní hodnota atributu* ***Text*** *tohoto elementu*

Ukázkové řešení: DataBinding/SmerTokuDat

3.7 Implementace INotifyPropertyChanged

Data binding má i své omezení. Plnohodnotný data binding lze vytvořit při dodržení několika podmínek:

1. Objekt s cílovou vlastností musí být odvozen od třídy **`DependencyObject`**
2. Cílová vlastnost musí být *`DependencyProperty`*[11]
3. Zdrojový objekt musí poskytovat informaci o změně hodnot vlastností.

Elementy co jsou součástí WPF jsou odvozeny od **`DependencyObject`**[12]**.** Jejich vlastnosti jsou vytvořeny jako *`DependencyProperty`***.** WPF elementy také poskytují upozornění na změnu hodnot vlastností. Při vytváření vlastní třídy nebo ovládacího prvku je potřeba tyto funkce vytvořit.

Pro vytvoření upozornění na změnu dat stačí implementovat rozhraní **`INotifyPropertyChange`.** Toto rozhraní obsahuje událost, která upozorní na změnu dat.

```
public event PropertyChangedEventHandler PropertyChanged[13];

protected void OnPropertyChanged(string nazevVlastnosti)
```

[11] Vlastnost DependencyProperty lze vytvořit pouze v objektu odvozeného od DependencyObject.
[12] Další informace o DependencyObject a DependencyProperty naleznete zde: http://msdn.microsoft.com/en-us/library/cc903933(v=vs.95).aspx
[13] Název události zastupuje všechny obslužné metody připojené k této události.

```
{

      if (PropertyChanged != null)
      {
         PropertyChanged(this, new PropertyChangedEventArgs(nazevVlastnosti));
      }
}
```

PropertyChangedEventHandler *– typ delegát*[14] *pro událost* ***PropertyChanged***
PropertyChanged *– událost pro upozornění na změnu dat*
OnPropertyChanged *– umístí se tam, kde může dojít ke změně dat*
PropertyChanged() *– k události je možné přistupovat jako k metodě (protože je typu delegát)*
this *– objekt, který vyvolal událost (**tento objekt**)*
PropertyChangedEventArgs *– tento argument obsahuje název vlastnosti, kde došlo ke změně*

Použití události PropertyChanged

```
private string jmeno;

public string Jmeno
{
      get { return jmeno; }
      set
      {
          jmeno = value;

          OnPropertyChanged("Jmeno");
      }
}
```

Název vlastnosti musí být přesně zapsaný. Jinak nebude fungovat upozornění na změnu v data bindingu.

Ukázkové řešení: DataBinding/ImpINotifyPropertyChanged

3.8 Data binding – Windows Forms

Windows Forms je možné vytvořit data binding pouze v kódu. Stejně jako u WPF se může zdrojem dat stát objekt (vlastnost) nebo kolekce objektů. Ovládací prvky ve Windows Forms mají pro připojení data bindingu vlastnost `DataBindings`. Jedná se o kolekci objektů třídy **`Binding`**. **Nejedná se** o stejnou třídu jako u WPF.

Vytvoření data bindingu ve Windows Forms

```
jmenoTextBox.DataBindings.Add("Text", student, "Jmeno");
prijmeniTextBox.DataBindings.Add("Text", student, "Prijmeni");
```

DataBindings – kolekce objektů Binding
Add – metoda pro přidání objektu Binding do kolekce

[14] Delegát dovoluje vytvořit odkaz na metodu. Metoda musí mít stejný typ pro návratovou hodnotu a parametry. Počet parametrů se také musí shodovat s delegátem.

V ukázce je přetížená varianta metody ***Add(cílová vlastnost, zdrojový objekt, zdrojová vlastnost)***.

Ukázkové řešení: Windows Forms/Databinding

Windows Forms také dovoluje vytvořit data binding ke kolekci. Namísto třídy **`ObservableCollection`** se použije třída **`BindingList`**. Ta obsahuje mechanizmus upozornění na změnu stavu kolekce.

Připojení kolekce dat k ovládacímu prvku ListBox ve Windows Forms

```
BindingList<string> letopocty = new BindingList<string>();
letopocty.Add("1085 - 1. český král Vratislav II.");
letopocty.Add("1212 - Zlatá bula sicilská");
letopocty.Add("1918 - vznik Československa");

listBox1.DataSource = letopocty;
```

BindingList *– kolekce podporující data binding (upozornění na změnu stavu kolekce)*
DataSource *– připojí kolekci k ovládacímu prvku*

Ukázkové řešení: Windows Forms/DatabindingKolekce

4 2D grafika

2D grafika je ve WPF založena převážně na vektorové grafice. Nicméně dovoluje vytvářet i bitmapovou grafiku. GUI elementy jsou vykreslovány pomocí vektorové grafiky. To nabízí možnosti jako je změna měřítka, zkosení nebo třeba otáčení libovolných prvků. Objekt rozhraní jsou pak vykresleny bez ztráty kvality při těchto transformacích.

WPF obsahuje objekty, které je možné rovnou použít v rozhraní aplikace. Jedná se o základní tvary (obdélník, elipsa, čára…). Tyto tvary je možné použít jako, kterýkoliv jiný GUI element. WPF ale obsahuje i objekty, které nelze přímo zobrazit v GUI. Jedná se o geometrie tvarů. Pro zobrazení těchto objektů je pak zapotřebí další element.

Objekty tak lze rozdělit do dvou základních skupin:

1. objekty použitelné stejně jako GUI elementy
2. objekty tvořené geometrií, která je popisuje

Geometrie objektu pouze popisuje jeho tvar. Neobsahuje informace o barvě čáry nebo výplně. Proto je potřeba další element, který tyto vlastnosti přidá. Mezi objekty přístupné jako geometrie patří například křivky, oblouky, obdélníky… Geometrie umožňují snadno aplikovat operátory jako sjednocení, odečtení ...

U některých příkladů jsou použity třídy z Microsoft Blend[15]. Přidávající interaktivitu do aplikace. Dovolují například hýbat s **`UIElementy`**. Nejednoduší způsob jak tyto třídy přidat do aplikace je přes NuGet[16][17].

Ukázka použití interaktivity z Blend

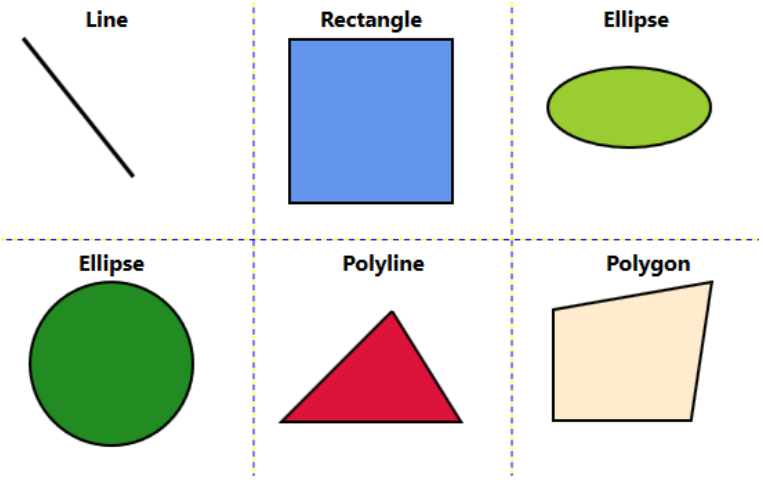
```
<i:Interaction.Behaviors>
     <r:MouseDragElementBehavior />
</i:Interaction.Behaviors>
```

Přidá interaktivní chování k libovolnému UIElementu.

4.1 Základní grafické tvary (Shape)

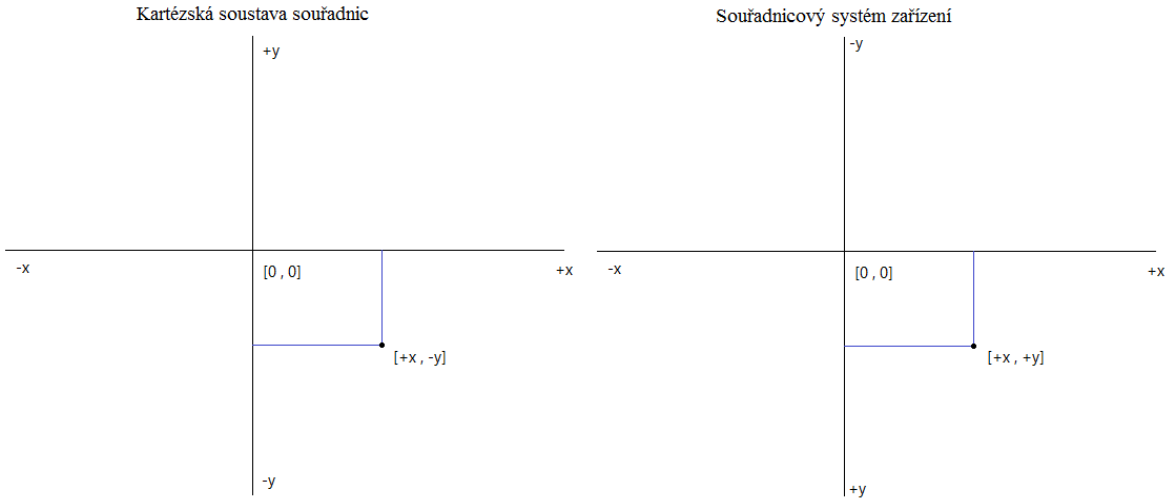
Třída **`Shape`** je odvozena od třídy **`UIElement`**. Získává tak vlastnosti typické pro ovládací prvky z rozhraní aplikace. Od této třídy jsou odvozeny už konkrétní základní typy: **`Rectangle`** (obdélník), **`Ellipse`** (elipsa), **`Line`** (úsečka), **`Polyline`** (složená čára) a **`Polygon`** (mnohoúhelník). U těchto tvarů stejně jako u ovládacích prvků je možné reagovat na události. V ukázkách kódu je použit layout **`Canvas`** pro umístění těchto tvarů. Na Obrázku 3 jsou zobrazeny všechny tyto tvary.

[15] Program na vytváření rozhraní na bázi WPF (XAML).
[16] Blend Interactivity for WPF v4.0. Na adrese: https://nuget.org/packages/Blend.Interctivity.WPF.v4.0/1.0.3.
[17] Manažer balíčků pro vývojářskou platformu – http://nuget.org/.



Obrázek 3 Základní grafické tvary

WPF má jiný souřadnicový systém než je kartézský. Pro určení pozice elementu na layoutu nebo obecně v rozhraní aplikace je použit souřadnicový systém zařízení. Ten se liší od Kartézské soustavy souřadnic převrácením osy y. Počátek soustavy je umístěn v levém horním rohu layoutu nebo okna aplikace. Na Obrázku 4 je porovnání těchto dvou soustav souřadnic.



Obrázek 4 Porovnání soustav souřadnic

Ukázkové řešení: 2D Grafika/SouradniceZarizeni

Vytvoření úsečky (Line)

```
<Line X1="20" Y1="40" X2="250" Y2="150" Stroke="Black" StrokeThickness="2"/>
```

X1,Y1 *– souřadnice prvního bodu*
X2,Y2 *– souřadnice druhého bodu*
Stroke *– barva čáry*
StrokeThickness *– šířka čáry*

Vytvoření obdélníku (Rectangle)

```
<Rectangle Canvas.Top="30"
           Canvas.Left="25"
           Width="120"
           Height="120"
           Stroke="Black"
           StrokeThickness="2"
           Fill="CornflowerBlue"/>
```

Canvas.Top *– souřadnice od levého horního rohu /osa y)*
Canvas.Left *– souřadnice od levého horního rohu (osa x)*
Width *– šířka obdélníku*
Height *– výška obdélníku*
Stroke *– barva ohraničení obdélníku*
StrokeThickness *– šířka ohraničení*
Fill *– vyplnění obdélníku zadanou barvou*

Vytvoření elipsy (Ellipse)

```
<Ellipse Canvas.Top="50"
         Canvas.Left="25"
         Width="120"
         Height="60"
         Stroke="Black"
         StrokeThickness="2"
         Fill="YellowGreen"/>
```

Atributy elementu ***Ellipse*** *mají stejnou funkci jako u elementu* ***Rectangle****.*

Vytvoření lomené čáry (Polyline)

```
<Polyline Points="100,20 20,100 150,100 100,20"
          Stroke="Black"
          StrokeThickness="2"
          Fill="Crimson"/>
```

Polyline *– vytvoří složenou čáru z úseček*
Points *– posloupnost bodů úseček*

Vytvoření mnohoúhelníku (Polygon)

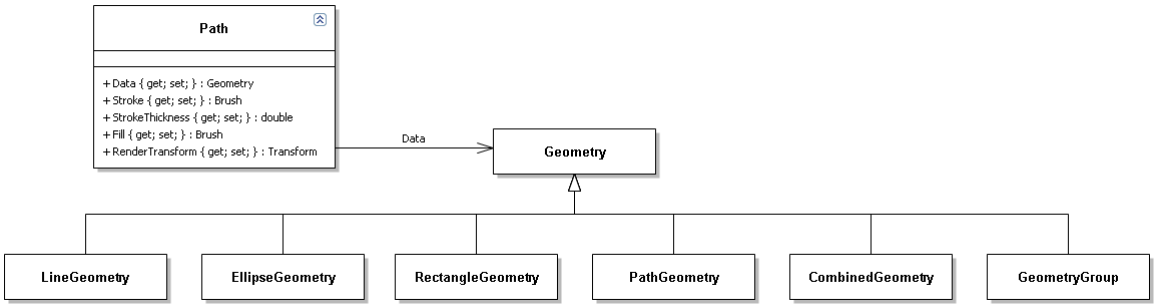
```
<Polygon Points="20,40 20,120 120,120 135,20"
         Stroke="Black"
         StrokeThickness="2"
         Fill="BlanchedAlmond"/>
```

Princip vytváření mnohoúhelníku je stejný jako u ***Polyline****. Rozdíl je v tom, že první a poslední bod se automaticky spojí.*

Ukázkové řešení: 2D Grafika/ZakladniTvaryShape

4.2 Geometrie

Geometrie ve WPF slouží stejně jako v matematice k popisu objektů. Na rozdíl od tříd odvozených od třídy **`Shape`** je nelze nakreslit bez pomocného elementu. Elementy geometrie nedisponují atributy *`Stroke`*, *`StrokeThickness`* nebo *`Fill`*. Tyto atributy se nastavují přes pomocný element.

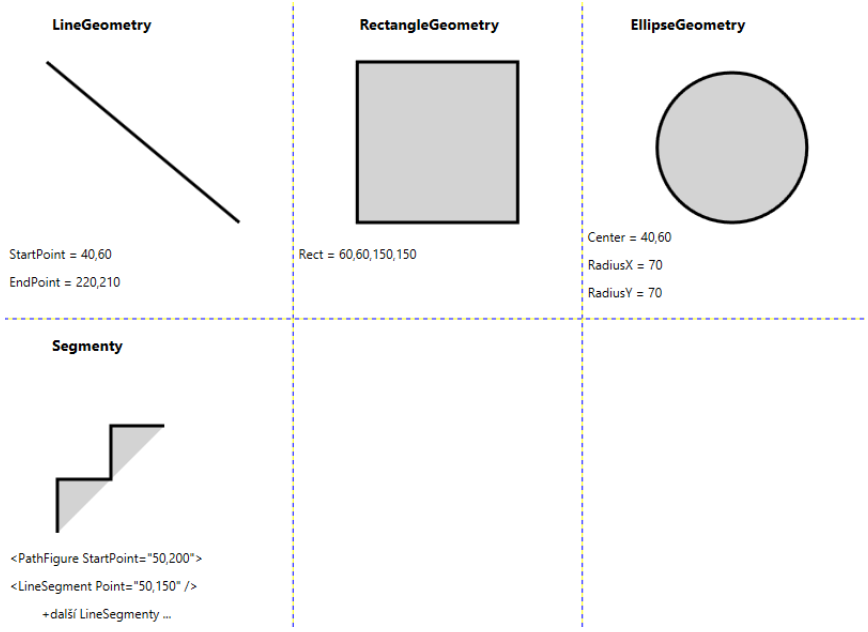


Obrázek 5 Diagram tříd pro geometrie ve WPF

Na Obrázku 5 je zobrazen element **`Path`**. Tento element slouží pro zobrazování geometrií. Nastavuje pro všechny zobrazované elementy (geometrie) atributy: *`Stroke`*, *`StrokeThickness`* nebo *`Fill`*. Atribut *`Data`* slouží pro nastavení zobrazované geometrie. Na obrázku 5 je také zobrazena třída **`Geometry`**. Od ní jsou odvozeny další třídy geometrie. *`LineGeometry`*, *`EllipseGeometry`* a *`RectangleGeometry`* vytvoří základní grafické tvary. **`PathGeometry`** slouží k vytvoření skupiny čar složených z různých segmentů (částí). Geometrie **`CombinedGeometry`** dovoluje aplikovat na geometrie operace jako je sjednocení nebo třeba odečtení. **`GeometryGroup`**[18] dovoluje sjednotit více geometrií v atributu *`Data`*.

Geometrie základních grafických tvarů mají rozdílné atributy oproti těm odvozených od třídy **`Shape`**. Na Obrázku 6 jsou zobrazeny některé z elementů geometrií a jejich atributů.

[18] GeometryGroup se chová jinak než, kdyby šlo o dva nezávislé objekty. Důležitý je atribut FillRule pro nastavení překrývání při zobrazení geometrií. Další informace můžete nalézt zde: http://msdn.microsoft.com/en-us/library/system.windows.media.geometrygroup.fillrule.aspx



Obrázek 6 Ukázky geometrií a jejich atributů

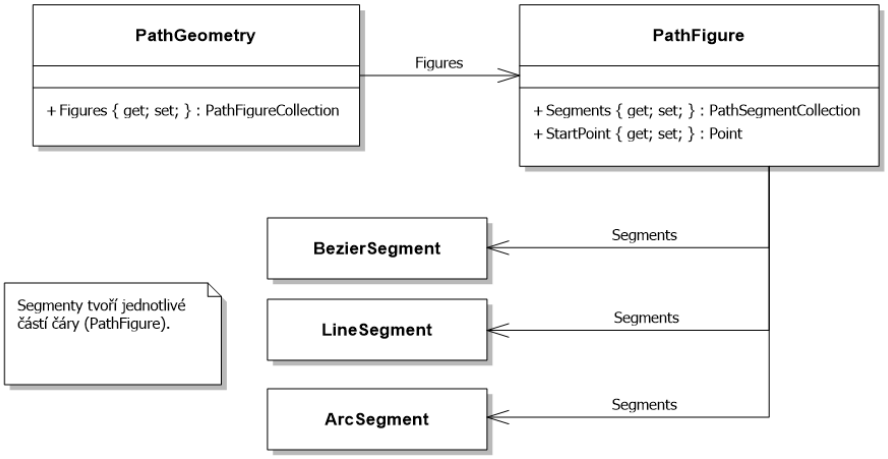
```
<Path Stroke="Black" StrokeThickness="3" Fill="LightGray">
    <Path.Data>
        <RectangleGeometry Rect="60,60,150,150"/>
    </Path.Data>
</Path>
```

Path *– zobrazí geometrie*
Path.Data *– nastavení zobrazované geometrie*
Rect *– definice obdélníku (x,y,width,height)*

Ukázkové řešení: 2D Grafika/ZakladniTvaryGeometry

4.2.1 PathGeometry

Pomocí **`PathGeometry`** je možné vytvořit skupinu čar složených ze segmentů. Každý segment může tvořit například úsečka, oblouk nebo Bézierova křivka. Čáry jsou definovány pomocí třídy **`PathFigure`**. Na Obrázku 7 je diagram třídy **`PathGeometry`**. Tato třída má atribut *`Figures`* pro vytvoření jednotlivých čar (**`PathFigure`**). Atribut *`Segmets`* je kolekcí jednotlivých částí tvořících čáru **`PathFigure`**. *`StartPoint`* udává počáteční bod celé čáry.



Obrázek 7 Diagram třídy pro PathGeometry

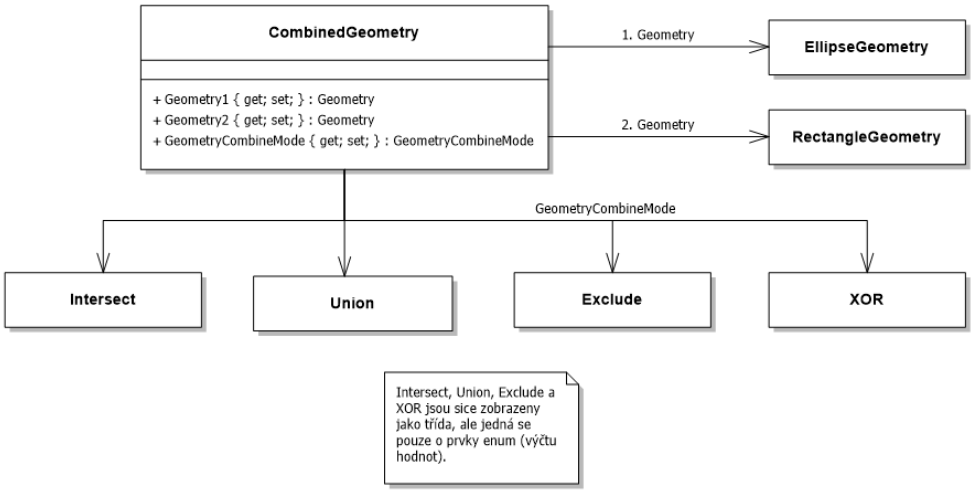
```
<PathGeometry>
    <PathFigure StartPoint="10,10">
        <LineSegment Point="200,30"/>
        <ArcSegment Point="180,100" Size="60,50"/>
    </PathFigure>

    <PathFigure StartPoint="20,100">
        <BezierSegment Point1="150,130" Point2="10,200" Point3="230,150"/>
    </PathFigure>
</PathGeometry>
```

PathGeometry *– aplikuje operaci na dva objekty*
PathFigure *– vytváří čáru složenou ze segmentů*
StartPoint *– počátek čáry vytvořené pomocí PathFigure*
LineSegment *– segment úsečky*
ArcSegment *– segment oblouku*
BezierSegment *– segment kubické Bézierovy křivky*
Koncový bod segmentu je počátečním bodem následujícího segmentu.

Ukázkové řešení: 2D Grafika/KubickaBezierovaKrivka ; 2D Grafika/Oblouk ; 2D Grafika/PathGeometry

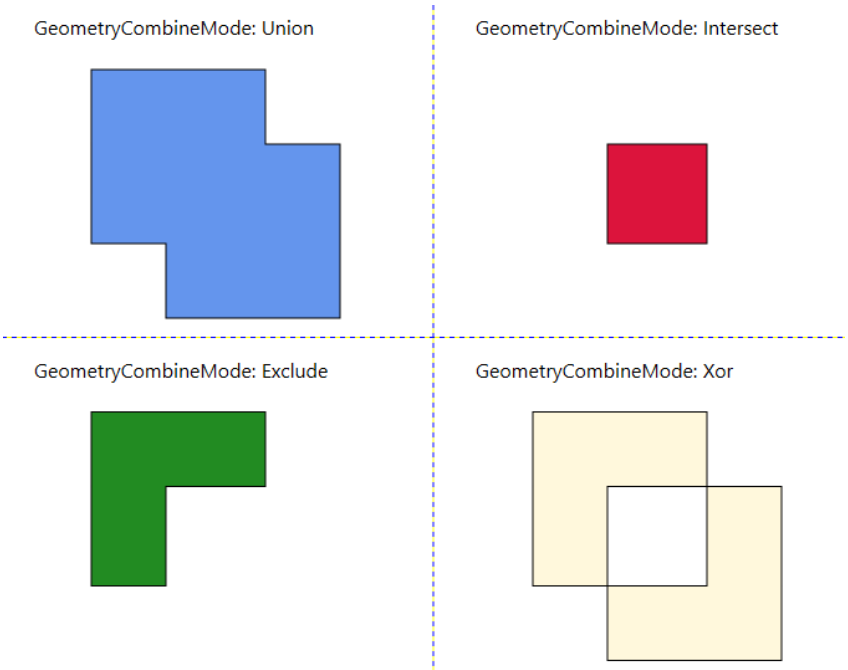
4.3 Kombinace objektů

Objekty je možné kombinovat pomocí operací: sjednocení (Union), průnik (Intersect), odečtení (Exclude) a XOR (opak průniku). Pro použití operace se používá geometrie `CombinedGeometry`.



Obrázek 8 Třída CombinedGeometry s vlastnostmi a nastaveními

Na Obrázku 8 je tato třída zobrazena. Element **`CombinedGeometry`** má dva atributy pro nastavení geometrií: *`Geometry1`* a *`Geometry2`*. Ty dva atributy jsou typu Geometry. Mohou tak obsahovat libovolný element odvozený od tohoto typu. Atributem *`GeometryCombineMode`* se nastavuje druh operace s geometriemi. Na Obrázku 8 jsou zobrazena možná nastavení pro tento atribut (Intersect, Union…). Na Obrázku 9 jsou zobrazeny všechny operace nad **`RectangleGeometry`**.



Obrázek 9 Ukázka operací s geometriemi

Použití kombinované geometrie na RectangleGeometry

```
<CombinedGeometry GeometryCombineMode="Union">
      <CombinedGeometry.Geometry1>
           <RectangleGeometry Rect="80,60,140,140"/>
      </CombinedGeometry.Geometry1>
```

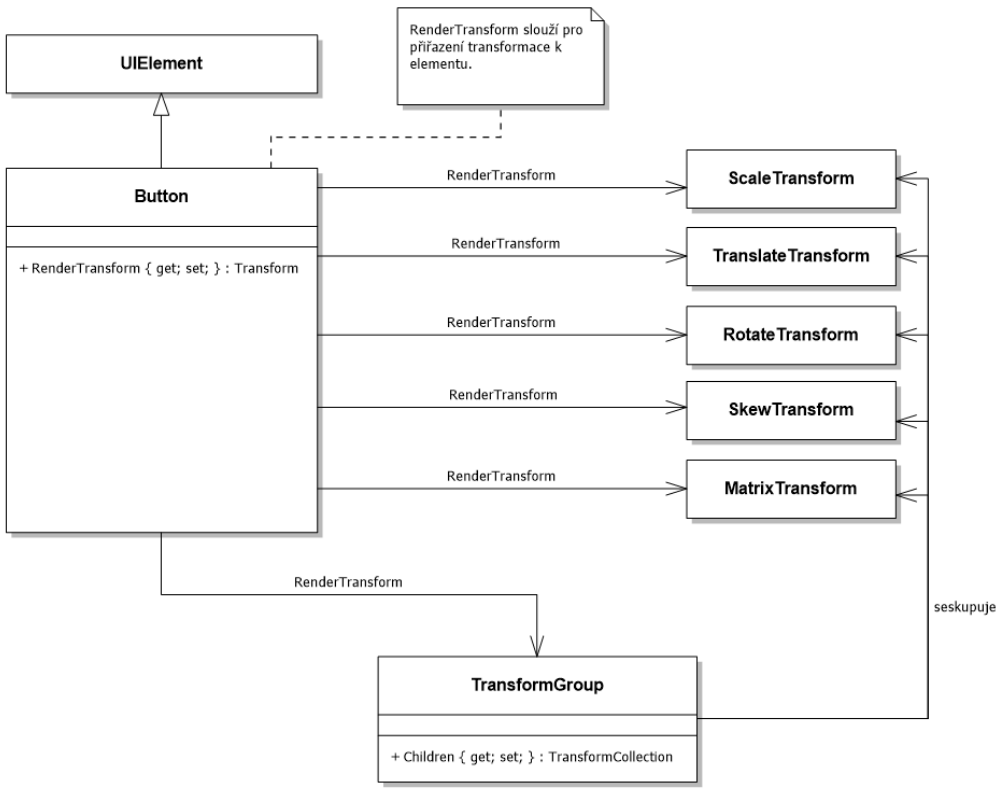
```
      <CombinedGeometry.Geometry2>
            <RectangleGeometry Rect="140,120,140,140"/>
       </CombinedGeometry.Geometry2>
</CombinedGeometry>
```

CombinedGeometry *– kombinuje geometrie*
GeometryCombineMode *– nastavuje typ operace s geometriemi (**Union, Intersect, Exclude a Xor**)*
Geometry1 *– první geometrie pro operaci*
Geometry2 *– druhá geometrie pro operaci*

Ukázkové řešení: 2D Grafika/CombinedGeometry

4.4 Transformace

Transformace je možné ve WPF aplikovat na elementy odvozené od třídy `UIElement`. Mezi takové elementy patří například: **`Button`**, **`TextBox`**, **`Rectangle`**, **`Path`**… Všechny tyto elementy mají atribut `RenderTransform` pro nastavení transformace.



Obrázek 10 Diagram tříd pro transformace

Na Obrázku 10 je zobrazen diagram tříd pro transformace. Ve WPF jsou předefinovány čtyři transformace: změna měřítka (**`ScaleTransform`**), posunutí (**`TranslateTransform`**), rotace (**`RotateTransform`**) a zkosení (**`SkewTransform`**). Na Obrázku 10 je také element **`MatrixTransform`**[19]. Tento element umožňuje vytvoření vlastní transformace. Také je možné

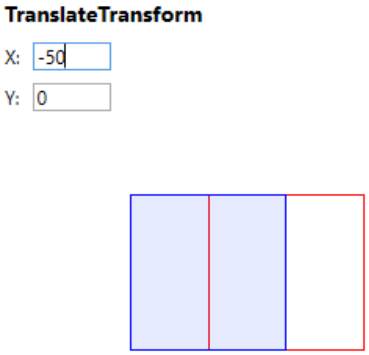
[19] Další informace o transformačních maticích a transformacích naleznete zde: http://msdn.microsoft.com/cs-cz/library/ms750596.aspx [en].

pomocí elementu `TransformGroup` použít více transformací najednou. Stačí elementy transformací umístit jako přímý obsah tohoto elementu.

TranslateTransform (posunutí elementu)

```
<Rectangle>
   <Rectangle.RenderTransform>

      <TranslateTransform X="0" Y="0" />

    </Rectangle.RenderTransform>
</Rectangle>
```

Rectangle *– posunovaný objekt*
RenderTransform *– atribut (element vlastnosti) pro nastavení transformace*
TranslateTransform *– přesune celý objekt po osách x a y z výchozího místa*
X *– posunutí o X na ose x (může být i záporné)*
Y *– posunutí o Y na ose y (může být i záporné)*
Nezáleží na tom, kde se posunovaný objekt právě nachází. Vždy bude výchozí hodnota pro X a Y rovna 0. V ukázce není objekt vůbec posunut. Nachází se ve výchozí pozici. Na Obrázku 11 je obdélník posunut o -50 bodů na ose x.



Obrázek 11 Ukázka z aplikace TranslateTransform

Ukázkové řešení: 2D Grafika/TranslateTransform20

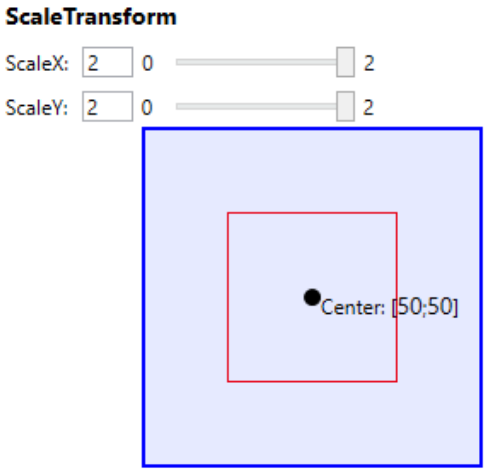
ScaleTransform (změna měřítka elementu)

```
<ScaleTransform ScaleX="1"
                ScaleY="1"
                CenterX="0"
                CenterY="0"/>
```

ScaleTransform *– změní měřítko elementu, aplikuje se i na atribut* ***StrokeThickness***
ScaleX *– změní měřítko pro osu X*
ScaleY *– změní měřítko pro osu Y*

[20] V ukázkové aplikaci je transformace TranslateTransform použita pro vytvoření interakce s obdélníkem. Obdélník je možné přesouvat kliknutím a táhnutím myši.

Center(X,Y) *– bod, ke kterému bude směřovat změna měřítka (element se zmenší nebo zvětší směrem k tomuto bodu)*
V ukázce jsou nastaveny výchozí hodnoty pro ***ScaleTransform****. Měřítko zůstává nezměněné. Na Obrázku 12 je nastaveno měřítko na dvojnásobek (****ScaleX****=2 a* ***ScaleY****=2) výchozí hodnoty. S bodem* ***Center*** *nastaveným na střed původního obdélníku. Výchozí pozice bodu* ***Center*** *se nachází v levém horním rohu elementu* ***[0,0]****.*



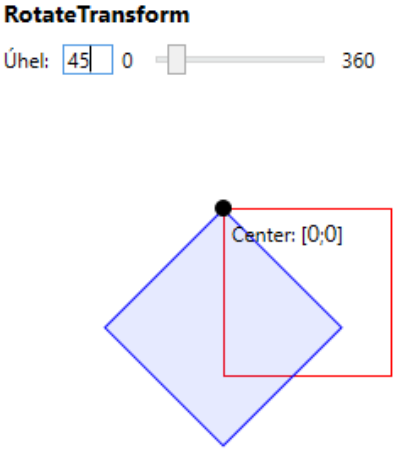
Obrázek 12 Ukázka z aplikace ScaleTransform

Ukázkové řešení: 2D Grafika/ScaleTransform

RotateTransform (rotace elementu)

```
<RotateTransform Angle="45"
                 CenterX="0"
                 CenterY="0"/>
```

RotateTransform *– rotace elementu se zadaným úhlem a středem rotace* ***Center***
Angle *– nastaví úhel rotace (ve směru hodinových ručiček)*
Na Obrázku 13 je ukázka z aplikace RotateTransform.



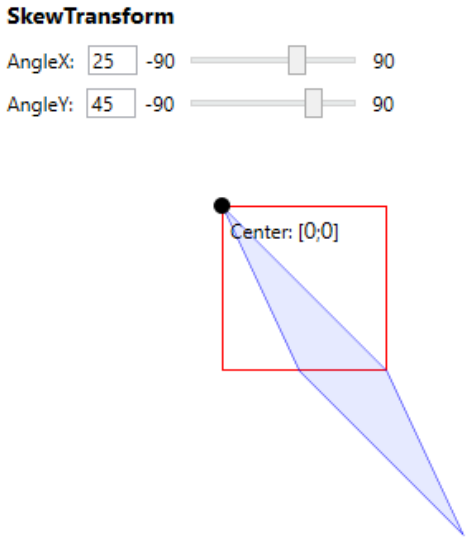
Obrázek 13 Ukázka z aplikace RotateTransform

Ukázkové řešení: 2D Grafika/RotateTransform

SkewTransform (zkosení elementu)

```
<SkewTransform AngleX="25"
               AngleY="45"
               CenterX="0"
               CenterY="0"/>
```

SkewTransform *– zkosí hrany elementu*
AngleX *– úhel zkosení pro osu x*
AngleY *– úhel zkosení pro osu y*
Na Obrázku 14 jsou nastaveny hodnoty z ukázky kódu.



Obrázek 14 Ukázka z aplikace SkewTransform

Ukázkové řešení: 2D Grafika/SkewTransform

4.5 Efekty

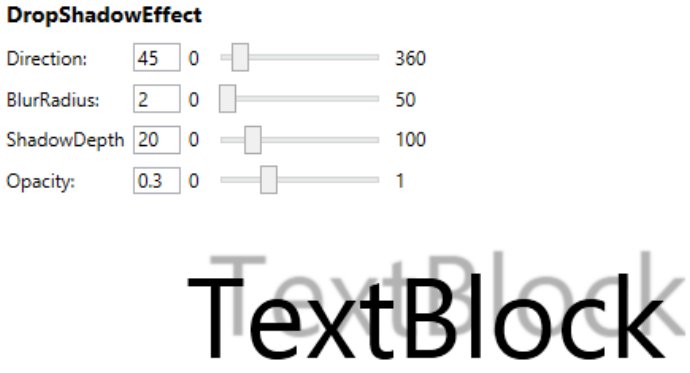
Efekty stejně jako transformace je možné přidat k elementům odvozených od `UIElement`. Tím je možné přidat efekty k ovládacím prvkům nebo grafickým prvkům. Pro přidání efektu k elementu slouží atribut *`Effect`*. WPF obsahuje dva základní efekty: `DropShadowEffect` (vytvoří stín u elementu) a `BlurEffect` (rozmaže element).

DropShadowEffect

```
<TextBlock>
    <TextBlock.Effect>

       <DropShadowEffect Color="Black"
                         BlurRadius="2"
                         Direction="45"
                         ShadowDepth="20"
                         Opacity="0.3"/>

     </TextBlock.Effect>
</TextBlock>
```

Color *– barva stínu*

BlurRadius *– rozmazání stínu*
Direction *– směr stínu (úhel)*
ShadowDepth *– vzdálenost stínu od elementu*
Opacity *– průhlednost stínu*
Na Obrázku 15 je ukázka z aplikace DropShadowEffect.



Obrázek 15 Ukázka z aplikace DropShadowEffect

Ukázkové řešení: 2D Grafika/DropShadowEffect

BlurEffect

```
<BlurEffect Radius="5" KernelType="Box"/>
```

BlurEffect *– rozmaže element*
Radius *– poloměr stínu*
KernelType *– způsob rozmazání stínu (**Box, Gaussian**)*
Na Obrázku 16 je ukázka z aplikace BlurEffect.

Obrázek 16 Ukázka z aplikace BlurEffect

Ukázkové řešení: 2D Grafika/BlurEffect

4.6 2D Grafika – Windows Forms

2D Grafika ve Windows Forms[21] je založena na GDI+[22]. Jedná se o API určené pro vytváření grafických objektů. Pro přidání grafiky do rozhraní je nejprve nutné získat kreslící plátno – **`Graphics`**. Instance této třídy se získá z události *Paint* třídy **`Form`**. **`Graphics`** pak umožňuje kreslit například objekty: úsečky, obdélníky, elipsy, křivky… WPF vykreslí objekt okamžitě po přidání příslušného elementu. Windows Forms aplikaci je potřeba nejprve spustit.

Získání grafického plátna a vytvoření úsečky

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
Graphics platno = e.Graphics;

Pen pen = new Pen(new SolidBrush(Color.Black));
platno.DrawLine(pen, new Point(50, 50), new Point(150, 150));
}
```

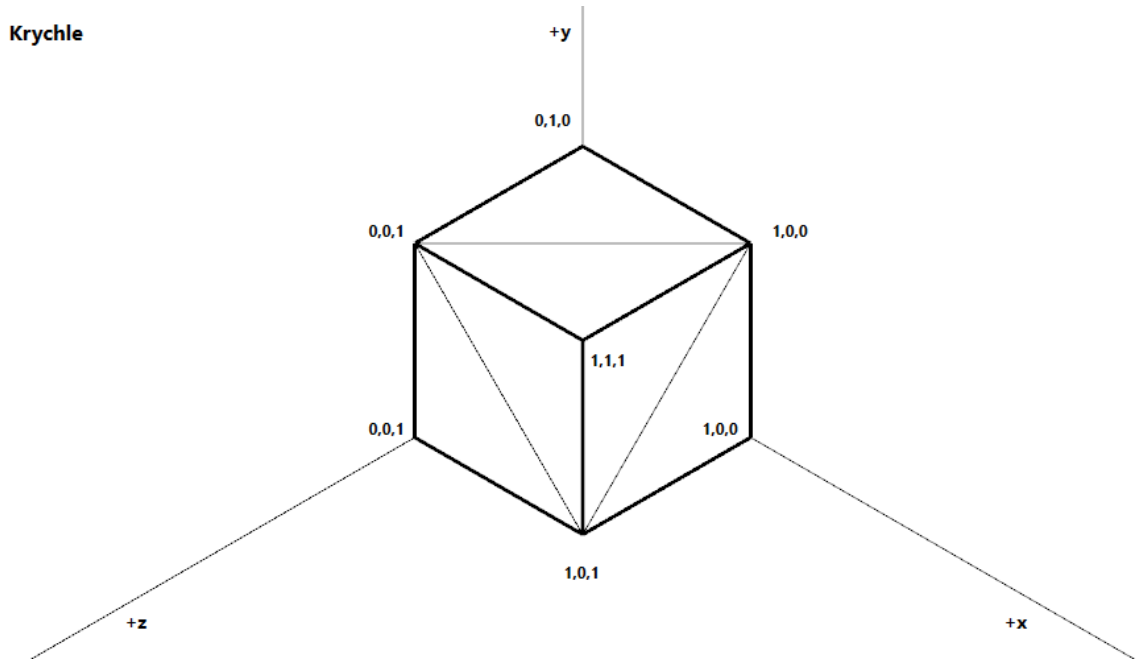
Form1_Paint *– obslužná metoda pro událost Paint*
PaintEventArgs *– odsuď, se získá objekt plátna*
Graphics *– plátno pro kreslení (použití GDI+)*
Pen *– tužka, kterou se objekt vykreslí*
SolidBrush *– nastavení štětce (podobně jako u WPF SolidColorBruhs)*
DrawLine *– nakreslí úsečku*

Ukázkové řešení: Windows Forms/VytvoreniPlatnaProKresleni

[21] Další informace o grafice naleznete zde: http://msdn.microsoft.com/en-us/library/da0f23z7.aspx [en].
[22] Další informace o této grafické knihovně naleznete zde: http://msdn.microsoft.com/en-us/library/windows/desktop/ms533798(v=vs.85).aspx [en]

5 3D Grafika

WPF obsahuje prostředky pro vytváření 3D objektů. Ty je možné nasvítí pomocí světel a umístit na ně materiály. Pro zobrazování objektů je možné použít připravené kamery. WPF obsahuje dost nízko úrovňové vytváření objektů. Neobsahuje základní objekty, jako jsou krychle, koule, válec… Tyto objekty je nutné si vytvořit nebo sehnat z nějakého jiného zdroje. Vytváření objektů je zpřístupněno na úrovni definování 3D trojúhelníků. Z těchto trojúhelníků se pak skládají celé objekty. Vytvořené objekty je možné transformovat nebo i animovat. Obrázek 17 ukazuje, jakým způsobem se vytvoří krychle pomocí trojúhelníků.



Obrázek 17 Rozložení krychle na trojúhelníky

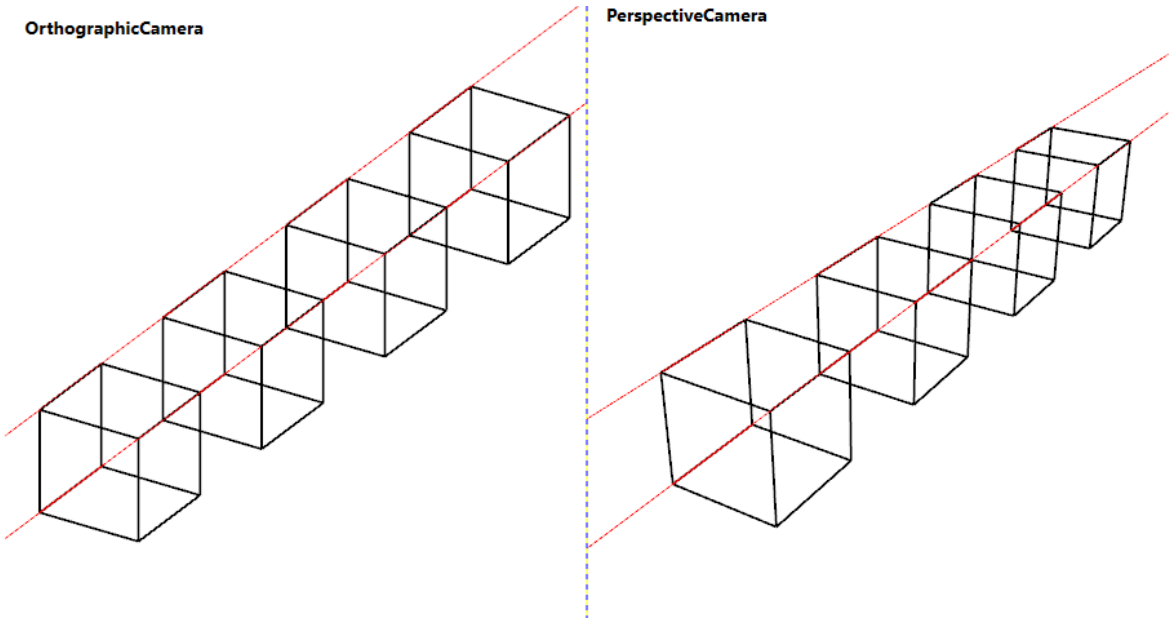
WPF zprostředkovává 3D grafiku skrze element `Viewport3D`. Ten se chová podobně jako ostatní elementy. Lze ho umístit do libovolného layoutu. Ve `Viewport3D` není souřadnicový systém stejný jako u ostatních elementů. Na Obrázku 17 jsou zobrazeny osy x, y a z. Bod tedy bude mít vždy tři souřadnice. Na Obrázku 17 nejsou zobrazeny záporné části os, ale bod může mít i záporné souřadnice.

Ukázky jako na Obrázku 17 jsou vytvořeny pomocí Helix 3D Toolkit. Tento nástroj rozšiřuje možnosti 3D grafiky ve WPF. Dovoluje například vytvářet čáry v prostoru, importovat objekty z 3D Studia Max nebo přidávat 3D orbitu[23]. Obsahuje také grafické primitivy jako jsou krychle, koule nebo třeba čajovou konvici. Helix 3D Toolkit najdete na adrese: http://helixtoolkit.codeplex.com/.

[23] Slouží k navigování ve 3D prostoru.

5.1 Kamery

Kamery slouží pro zobrazování 3D objektů. Fungují na principu zvaném – promítání. Tento princip promítne tří rozměrný objekt na dvou rozměrnou plochu (monitoru). Při takovémto zobrazování dochází ke zkreslení zobrazovaného objektu. Míru zkreslení určuje typ promítání. V této kapitole budou vysvětleny dva typy promítání a to pravoúhlé a perspektivní. Pro pravoúhlé promítání se používá kamera: `OrthographicCamera`. Perspektivního promítání je docíleno pomocí kamery: `PerspectiveCamera`. Na Obrázku 18 je zobrazen rozdíl mez těmito kamerami (promítáními).

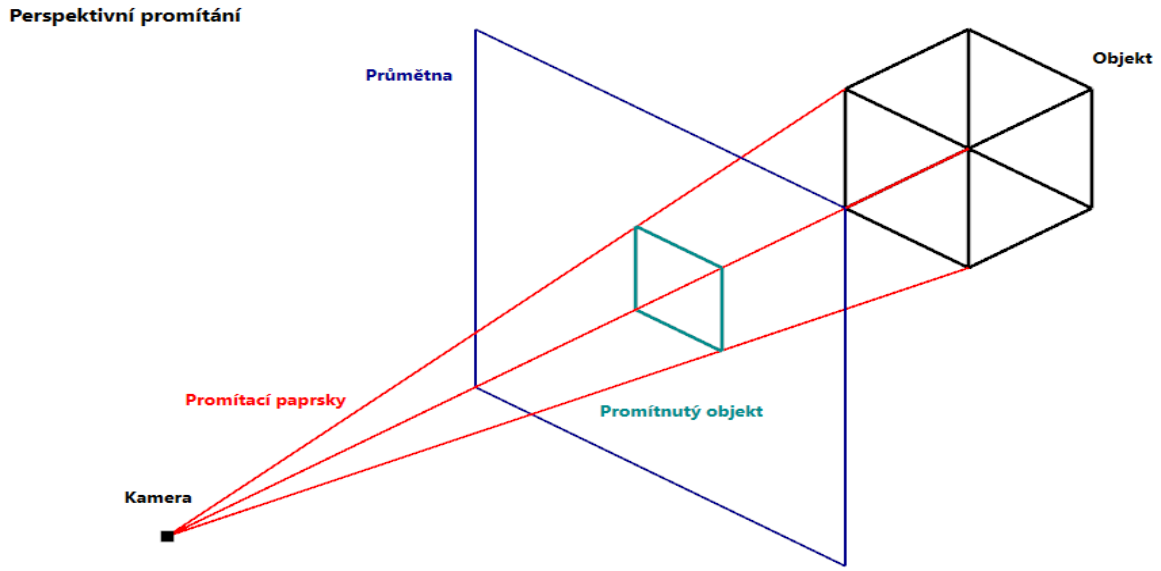


Obrázek 18 Rozdíl mezi perspektivním a pravoúhlým promítáním

Důležitými pojmy v promítání jsou: promítací paprsek a průmětna. Promítací paprsek jde od bodu na objektu směrem k průmětně. Tam, kde protne průmětnu, bude jeho obraz. Různé druhy promítání se liší ve směru promítacího paprsku k průmětně.

5.1.1 Perspektivní promítání

Perspektivní promítání se podobá lidskému zraku. Pro člověka jsou vzdálenější hrany objektu zdánlivě kratší, než ty co jsou blízko. Zdá se jako by se hrany objektu sbíhaly v nekonečnu. Toto promítání se nejčastěji používá tam, kde je potřeba zobrazit objekty tak jak je člověk vidí. Na Obrázku 19 je zobrazeno perspektivní promítání objektu.

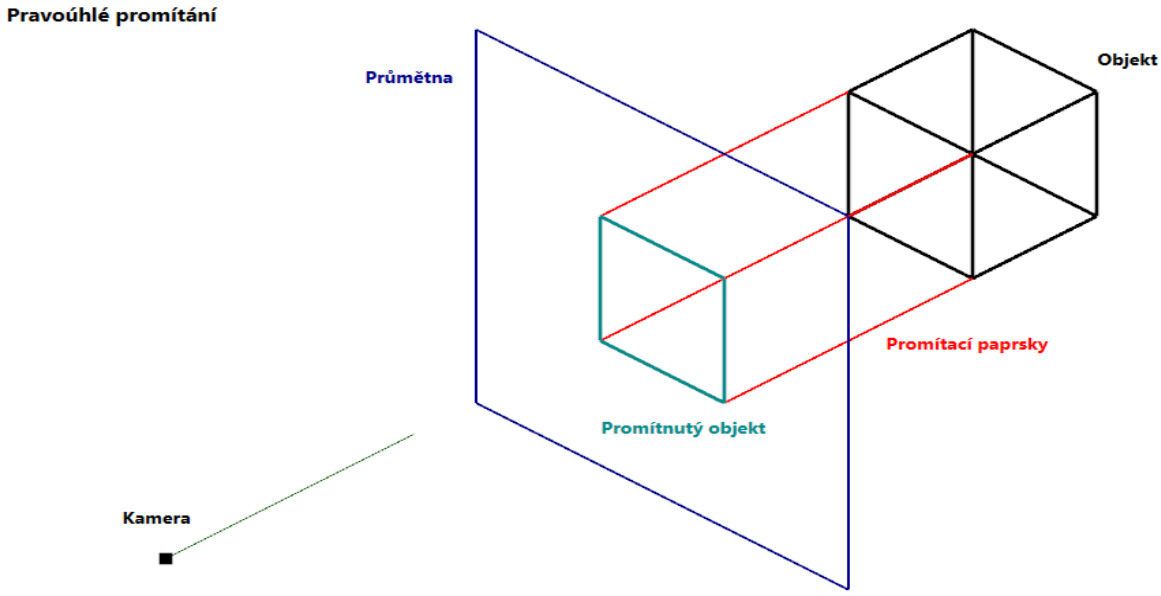


Obrázek 19 Perspektivní promítání objektu

Promítací paprsky směřují do jediného bodu. Vzdálenější hrany objektů se budou zdát kratší.

5.1.2 Pravoúhlé promítání

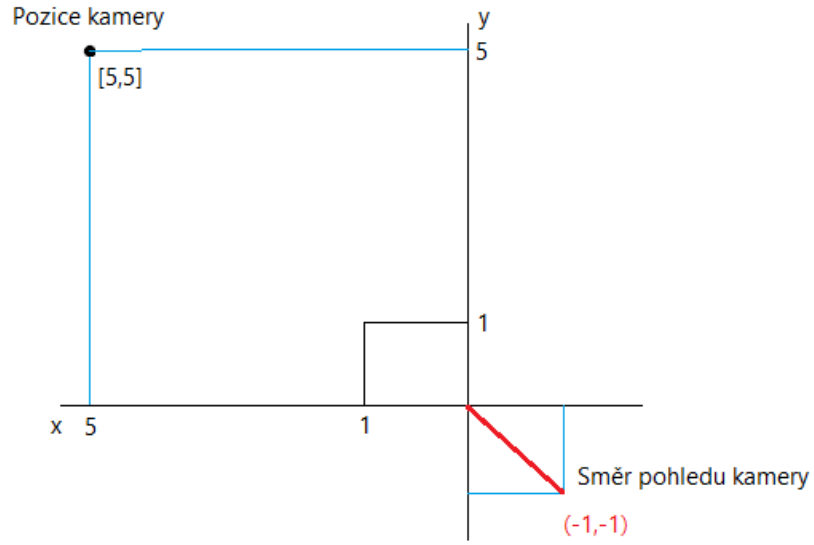
Pravoúhlé promítání nezkracuje vzdálenější hrany objektů. Je to způsobeno tím, že promítací paprsky jsou kolmé na průmětnu. Tento druh promítání se používá hlavně pro zobrazení objektů bez zkreslení. Objekty je tak snadnější modelovat pomocí nástrojů pro vytváření 3D objektů. Na Obrázku 20 je zobrazeno pravoúhlé promítání.



Obrázek 20 Pravoúhlé promítání objektů

5.1.3 Použití kamer

Obě kamery mají nastavení pozice. Ta se nastavuje jako bod v prostoru. Důležitým nastavením je směr pozorování kamery. Ta se určuje jako vektor. Na Obrázku 21 je zjednodušené nastavení kamery. Chybí zde osa z, ale jinak princip je stejný.



Obrázek 21 Nastavení pozice a směru pohledu kamery (zjednodušeně)

OrthographicCamera

```
<Viewport3D.Camera>
    <OrthographicCamera Position="5,5,5"
                        LookDirection="-1,-1,-1"
                        Width="5"/>
</Viewport3D.Camera>
```

OrthographicCamera *– pravoúhlé promítání*
Position *– pozice kamery*
LookDirection *– vektor směru pohledu kamery*
Width *– čím větší číslo tím se kamera oddálí*
*Vzdálenější pozice (**Position**) nemá vliv na oddálení kamery.*

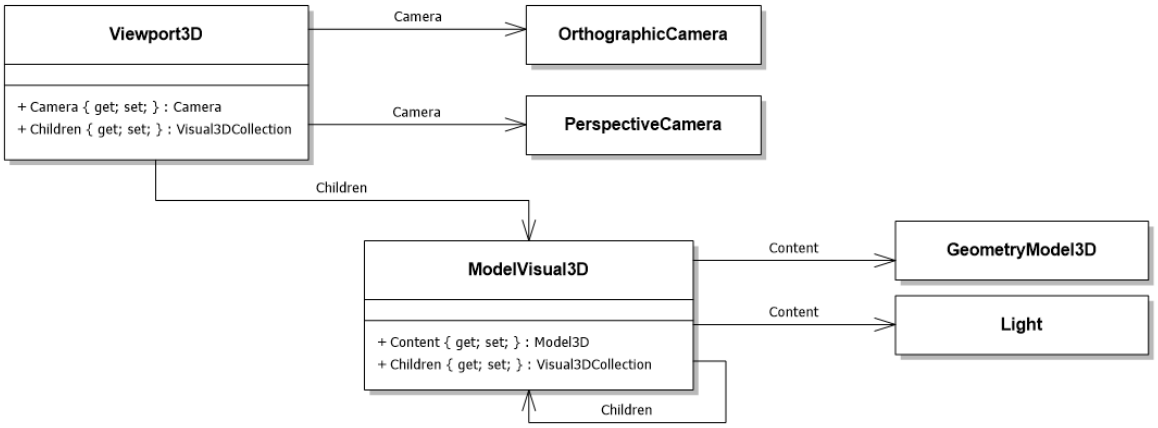
PerspectiveCamera

```
<Viewport3D.Camera>
    <PerspectiveCamera Position="5,5,5"
                       LookDirection="-1,-1,-1"
                       FieldOfView="30"/>
</Viewport3D.Camera>
```

PerspectiveCamera *– perspektivní promítání*
Position *– změní pozice kamery*
LookDirection *– směr pohledu kamery*
FieldOfView *– horizontální pohled kamery*
*Změna vzdálenosti od objektu se změní pomocí **Position** a **FieldOfView**.*

5.2 Vytvoření modelu

Objekty se vytvářejí jako 3D modely. Pro vytvoření modelu slouží element **`ModelVisual3D`**. Ten může obsahovat další modely. Nedefinuje přímo 3D objekt, ale používá další třídy pro jejich vytvoření. **`ModelVisual3D`** se přidává prostřednictvím přímého obsahu do elementu **`Viewport3D`**. Na Obrázku 22 je zobrazen diagram tříd pro vytvoření modelu.



Obrázek 22 Diagram tříd pro vytvoření modelu a jeho přidání do Viewport3D

Atribut *`Children`* u elementu **`Viewport3D`** je přímým obsahem. Není potřeba používat element vlastnosti pro jeho nastavení. **`ModelVisual3D`** má také atributu *`Children`* jako přímý obsah. Ten slouží pro vnořování dalších **`ModelVisual3D`** elementů. *`Content`* atribut nastavuje **`Model3D`**. Což může být například světlo (**Light**) nebo geometrii modelu (**`GeometryModel3D`**). **`GeometryModel3D`** pak bude sloužit pro vytvoření skutečného objektu.

5.3 Světla

Slouží k osvětlení 3D objektů z různých směrů. WPF obsahuje čtyři elementy světla: **`AmbientLight`**, **`DirectionalLight`**, **`SpotLight`** a **`PointLight`**. Světla se přidávají jako obsah (*`Content`*) elementu **`ModelVisual3D`**. Světla mají různá nastavení, ale všechny mají atribut Color (barva světla).

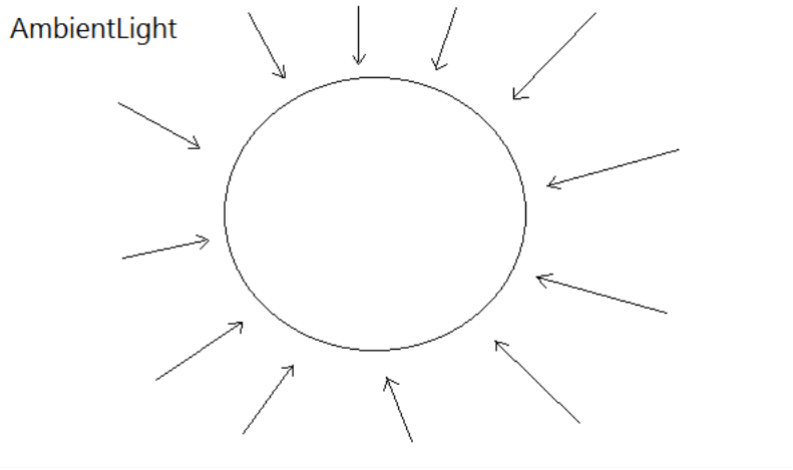
Ukázkové řešení: 3D Grafika/Svetla

AmbientLight

Vytvoří rovnoměrně rozptýlené světlo. Osvětlí všechny objekt ze všech stran. Obrázek 23 zobrazuje schéma **`AmbientLight`**.

```
<ModelVisual3D>
     <ModelVisual3D.Content>

          <AmbientLight Color="White"/>

     </ModelVisual3D.Content>
</ModelVisual3D>
```

AmbientLight *– vytvoří rozptýlené světlo*
Color *– barva světla*



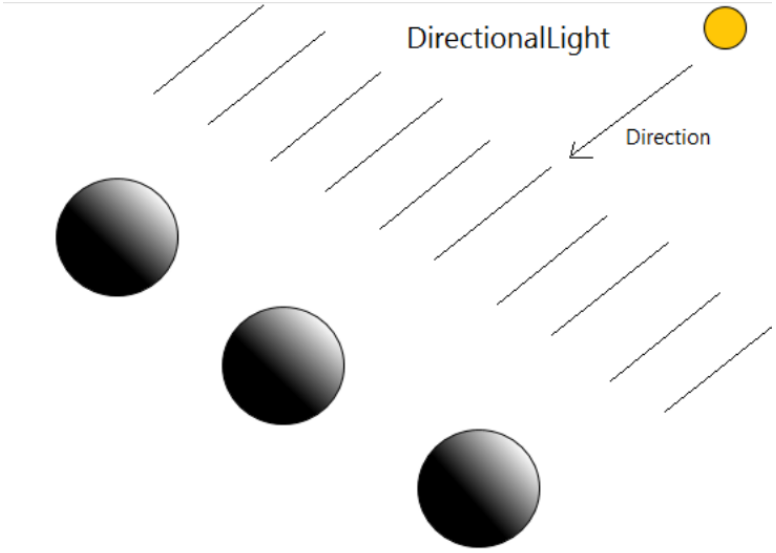
Obrázek 23 Schéma rozptýleného světla

DirectionalLight

Osvětlí objekty, které jsou ve směru světla. Příkladem může být například Slunce. Světlo je umístěno v nekonečnu. Není potřeba určovat pozici u tohoto světla. Na Obrázku 24 je schéma směrového světla.

```
<DirectionalLight Direction="0,-1,-1" Color="White"/>
```

DirectionalLight *– směrové světlo*
Direction *– vektor určující směr světla*



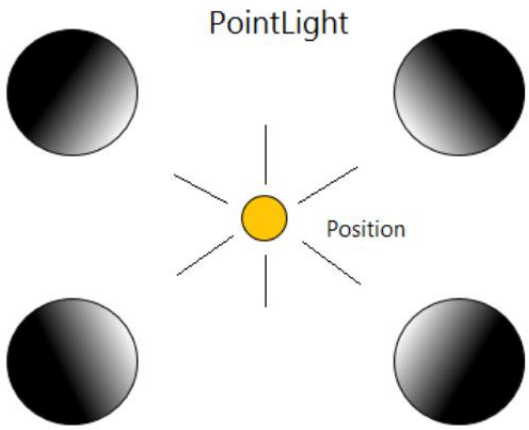
Obrázek 24 Schéma směrového světla

PointLight

Vytvoří bodové světlo (žárovka). Světlo se šíří všemi směry od zdroje. Je potřeba určit pozici světla. Na Obrázku 25 je schéma bodového světla.

```
<PointLight Position="5,5,5" Color="White"/>
```

PointLight *– bodové světlo*
Position *– pozice světla*



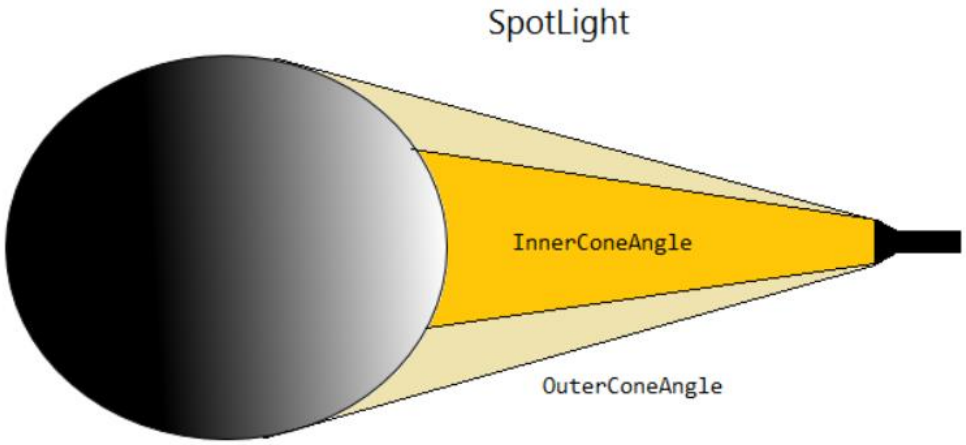
Obrázek 25 Schéma bodového světla

SpotLight

Vytvoří kužel světla směřujícího od zdroje k objektu (baterka). Skládá se z vnitřního (`InnerConeAngle`) a vnějšího kuželu (`OuterConeAngle`). Vnitřní kužel vytváří jasnější světlo. Světlo vnějšího kužele je více rozptýlené. Na Obrázku 26 je schéma světla SpotLight.

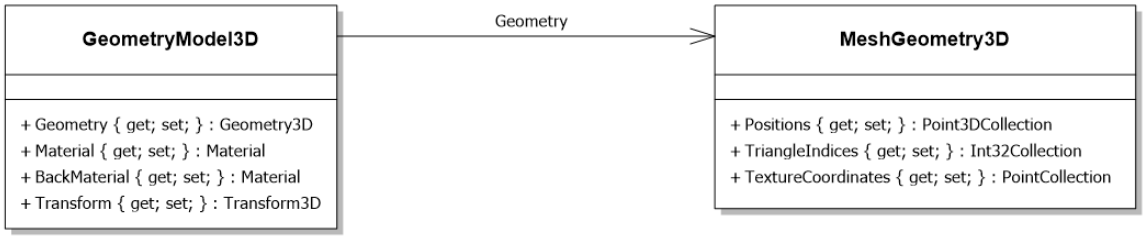
```
<SpotLight Direction="0,0,-1"
           Position="0,0,5"
           InnerConeAngle="90"
           OuterConeAngle="120"
           Color="White"/>
```

SpotLight *– vytvoří kužel světla*
Direction *– vektor určující směr světla*
InnerConeAngle *– úhel vnitřního kuželu*
OuterConeAngle *– úhel vnějšího kuželu*



Obrázek 26 Schéma světla SpotLight

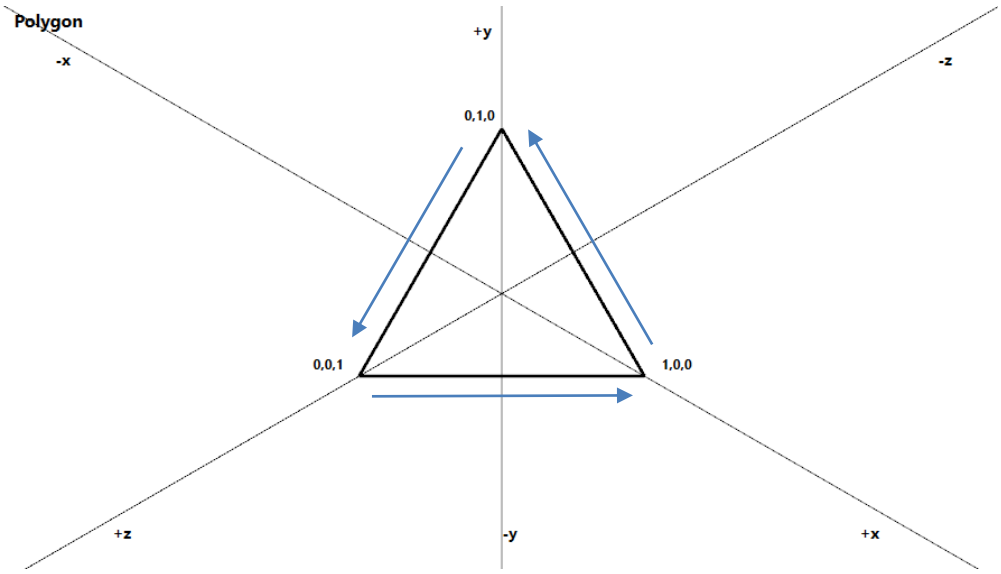
5.4 Vytvoření objektu



Obrázek 27 Diagram tříd zodpovědných za vytvoření 3D objektu

Na Obrázku 27 jsou dvě třídy, které slouží pro vytváření 3D objektů. Element **`GeometryModel3D`** se přidává jako obsah (*`Content`*) u elementu **`ModelVisual3D`.** Tento element má za úkol vytvořit geometrii objektu – *`Geometry`*. Dále obsahuje informace o materiálech, kterými je objekt pokryt – *`Material`*, *`BackMaterial`*. Umožňuje také nastavit transformaci pro objekt – *`Transform`*.

Element **`MeshGeometry3D`** vytváří geometrii objektu. Atribut *`Positions`* nastavuje kolekci vrcholů (vertexů[24]) trojúhelníků, které budou pak tvořit kostru objektu. *`TriangleIndices`* přiřazuje vertexy z *`Positions`* k trojúhelníkům. Ne ze všech vertexů se musí vytvořit trojúhelník. *`TextureCoordinates`* slouží pro upřesnění mapování obrázku (textury) na trojúhelník.



Obrázek 28 Definování vrcholů trojúhelníku

[24] Vertex je bod v Euklidovském prostoru se souřadnicemi (x, y, z).

Na Obrázku 28 jsou jednotlivé body potřebné pro vytvoření trojúhelníku. Na pořadí zápisu bodů u atributu *Positions* **nezáleží**. Zapsáním vertexů do *Position* se nevytvoří trojúhelník. Pro vytvoření trojúhelníku je potřeba zapsat *TriangleIndices*.

```
Positions="0,0,1 1,0,0 0,1,0"
```

TriangleIndices popisuje vytvoření trojúhelníků z bodů. Každý bod v *Positions* má svůj index a ten se začíná počítat od nuly. V ukázce má bod [1,0,0] index 1. Pomocí indexů jednotlivých bodů se pak definují trojúhelníky. Důležitý je směr, ve kterém budou zapisovány vrcholy trojúhelníku. Na Obrázku 28 šipky označují směr proti hodinovým ručičkám. Pokud je takto trojúhelník definován, aplikuje se z tohoto pohledu – *Material*. Ve směru hodinových ručiček se aplikuje – *BackMaterial*. Při definování trojúhelníku, nezáleží na pořadí vrcholů. Důležitý je pouze směr zápisu.

```
TriangleIndices="2 0 1"
```

Vytvoří trojúhelník z předchozího příkladu. Z tohoto pohledu bude mít – ***Material****.*

Ukázkové řešení: 3D Grafika/Trojuhelník

5.5 Materiály

Proto, aby objekt byl viditelný je zapotřebí na jeho povrch umístit materiál. Ten se nanáší na jednotlivé trojúhelníky. Materiál se aplikuje za pomoci atributu *Material* nebo *BackMaterial* u elementu **GeometryModel3D**. Všechny materiály ve WPF mají atribut *Brush* (štětec s barvou).

DiffuseMaterial

Vytvoří materiál s matným povrchem. Štětce použitelné s **DiffuseMaterial** jsou například: **SolidColorBrush** nebo **ImageBrush**. Pro použití **ImageBrush** je potřeba nastavit *TextureCoordinates*.

SolidColorBrush

```
<DiffuseMaterial Brush="Green"/>
```

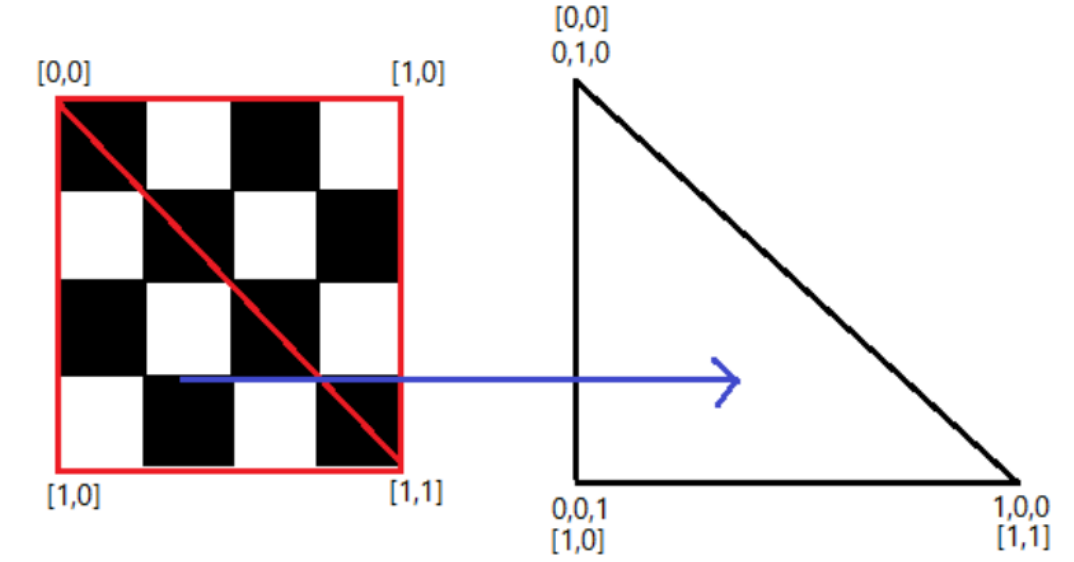
Brush *– aplikuje štětec SolidColorBrush*

ImageBrush

```
<GeometryModel3D>
      <GeometryModel3D.Material>
         <DiffuseMaterial>
            <DiffuseMaterial.Brush>
                <ImageBrush ImageSource="sach-texture.png"/>
            </DiffuseMaterial.Brush>
         </DiffuseMaterial
      </GeometryModel3D.Material>

       <GeometryModel3D.Geometry>
            <MeshGeometry3D Positions="0,0,0 0,1,0 0,0,1 0,1,1"
```

```
                              TextureCoordinates="1,1 1,0 0,1 0,0"
                              TriangleIndices="0 1 2 2 1 3"/>

        </GeometryModel3D.Geometry>
</GeometryModel3D>
```

TextureCoordinates *– slouží k mapování textury na trojúhelníky*
Levý horní roh textury (0,0) je umístěn v levém horním rohu [0,1,1] čtverce.
Levý dolní roh textury (0,1) je umístěn v levém dolním rohu [0,0,1] čtverce.
Pravý horní roh textury (1,0) je umístěn v pravém horním rohu [0,1,0] čtverce.
Pravý dolní roh textury (1,1) je umístěn v pravém dolním rohu [0,1,1] čtverce.
Na Obrázku 29 je zobrazen princip mapování textury na trojúhelníky (čtverec).



Obrázek 29 Princip mapování textur na trojúhelníky

Ukázkové řešení: 3D Grafika/DiffuseMaterial

SpecularMaterial

Vytvoří materiál s lesklým materiálem. Často se používá s materiálem **`DiffuseMaterial`**. Samotný materiál vytvoří průhledný materiál s odleskem. Atribut ***`Brush`*** nastavuje barvu odrazu. Pro přidání materiálu do skupiny materiálů slouží **`MaterialGroup`**. Pomocí atributu ***`SpecularPower`*** se nastavuje odrazivost materiálu.

```
<MaterialGroup>
        <DiffuseMaterial Brush="Green"/>
        <SpecularMaterial Brush="White" SpecularPower="40"/>
</MaterialGroup>
```

MaterialGroup *– dovoluje pro objekt použít více materiálů*
SpecularMaterial *– vytvoří materiál s lesklým povrchem*
SpecularPower *– nastavuje sílu odrazivosti*

Ukázkové řešení: 3D Grafika/SpecularMaterial

5.6 3D Grafika – Windows Forms

Windows Forms neobsahuje žádný integrovaný systém pro práci s 3D grafikou. Nicméně je možné použít GDI+ knihovnu a vytvořit sní 3D objekt. To může být velice časově náročné a nepraktické. Bude potřeba značná porce matematiky a znalosti z 3D grafiky. Například při vytváření kamery (promítání).

Jednou z možností jak přidat 3D grafiku do Windows Forms je ovládací prvek **`ElementHost`**. Ten dovoluje zobrazit elementy umístěné v **`UserControl`** (ovládací prvek WPF). **`UserControl`** funguje podobně jako element **`Window`**. Lze vněm vytvořit prvky rozhraní, včetně **`Viewport3D`** elementu.

Ukázka z UserControl

```
<UserControl x:Class="_3DInteroper.Viewport3D"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Grid>
        <ViewPort3D …/>
    </Grid>
</UserControl>
```

Další možností je použití grafických knihoven jako je Microsoft DirectX[25], Microsoft XNA[26] nebo třeba OpenGL[27]. Tyto knihovny jsou určeny především pro vytváření her.

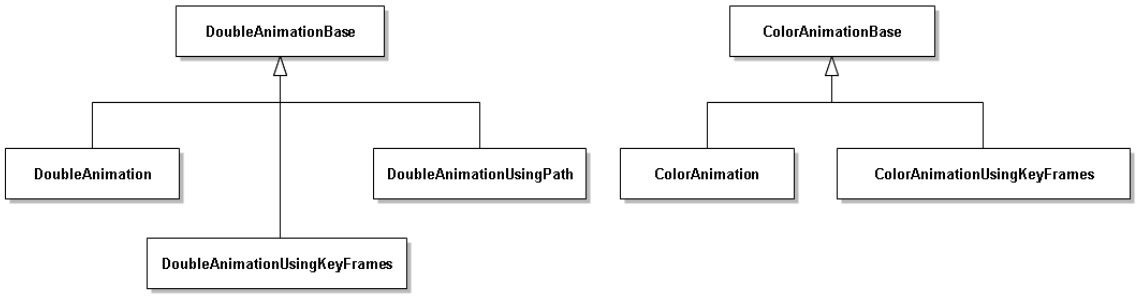
Ukázkové řešení: Windows Forms/ElementHost

[25] Informace o použití DirectX ve Windows Forms najdete například zde: http://www.godpatterns.com/2005/03/rendering-directx-as-part-of-window.html [en]
[26] Pro použití XNA můžete získat informace zde: http://www.codeproject.com/Articles/21330/Easy-Rendering-with-XNA-Inside-a-Windows-Form [en]
[27] Informace pro použití OpenGL ve Windows Forms najdete například zde: http://www.codeproject.com/Articles/16051/Creating-an-OpenGL-view-on-a-Windows-Form [en]

6 Animace

Animace je tvořená rychle se přepínajícími snímky. Každý následující snímek je nepatrně pozměněný. Snímky se přepínají tak rychle, že je to pro lidské oko nepostřehnutelné. Vytváří se tak iluze pohybu animovaného objektu. Při vytváření animace je zapotřebí zobrazit několik snímků za sekundu. Snímky mohou být například obrázky nebo změna hodnoty vlastnosti objektu. WPF dovoluje jako snímek definovat změnu hodnoty atributu. Například se může změnit hodnota atributu *Width* u elementu **Button.** Obecně se dá říct, že hodnota atributu se změní v určitý čas. Následně je změna zobrazena prostřednictvím rozhraní aplikace.

Pro vytvoření animace ve WPF slouží tří typy animací: základní animace, animace po křivce a animace s klíčovými snímky. Animovat se dají například následující typy: double, int, string, Point, Color… Pro typ string je použitelný jenom jediný typ animace a to s klíčovými snímky. Tedy, ne pro všechny typy hodnot atributů jsou použitelné všechny typy animací. Na Obrázku 30 je zobrazeny diagramy tříd pro typy animací a typy atributů (vlastností).



Obrázek 30 Diagram tříd pro typy animací

Třídy **DoubleAnimationBase** a **ColorAnimationBase** jsou určeny pro odvození animací pro typ uvedený ve jménu třídy. Element **DoubleAnimation** je základním typem animace pro typ hodnoty atributu (double). **DoubleAnimationUsingKeyFrames** je animací s klíčovými snímky pro typ **double**. **DoubleAnimationUsingPath** je animací po křivce pro typ **double.**

Syntaxe typu animace:

Základní typ

<typ_atributu>Animation

Například: DoubleAnimation, ColorAnimation

Animace s klíčovými snímky

<typ_atributu>AnimationUsingKeyFrames

Například: DoubleAnimationUsingKeyFrames, StringAnimationUsingKeyFrames

Animace po křivce

`<typ_atributu>AnimationUsingPath`

`Například: PointAnimationUsingPath`

V Tabulce 2 jsou uvedeny příklady animací s popisem jejich užití.

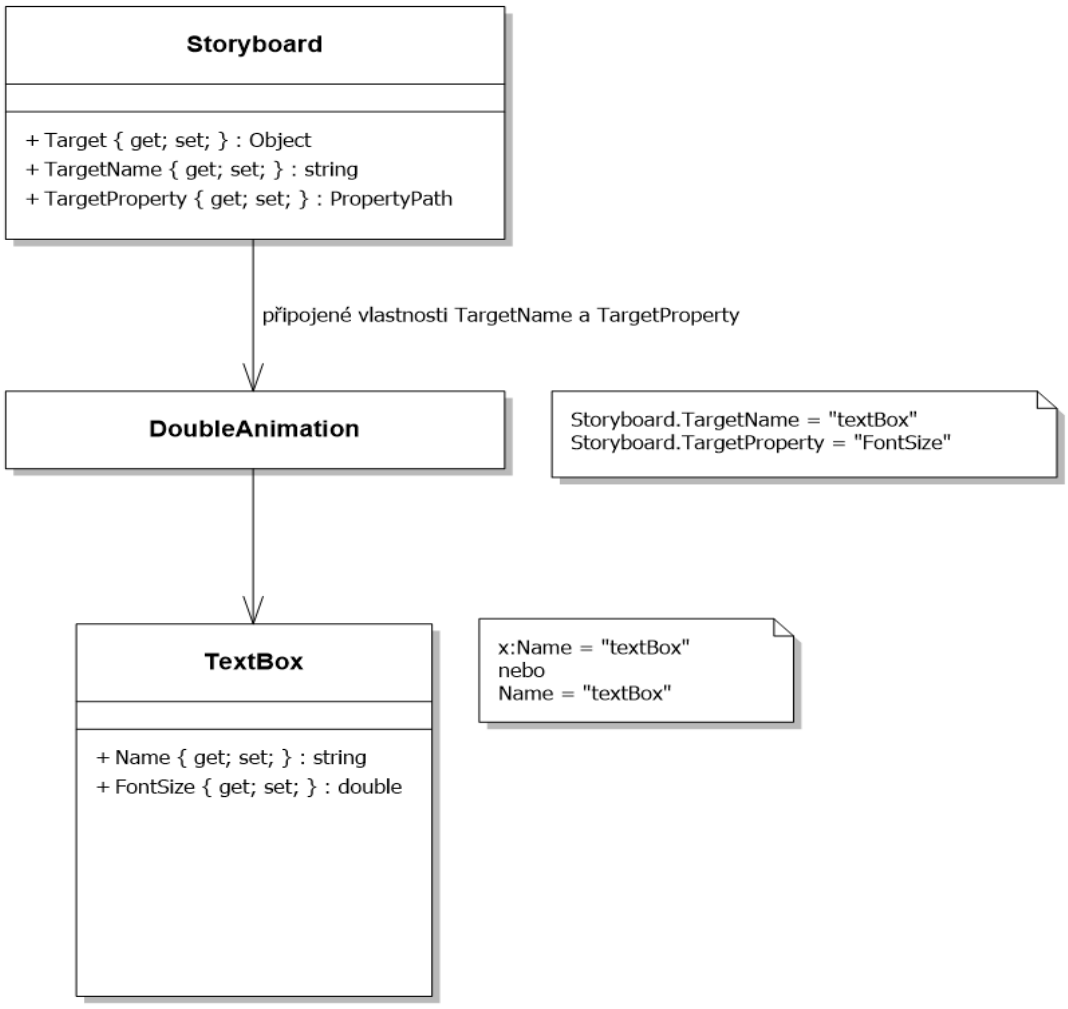
Tabulka 2 Typy animací s popisem

Typ atributu	Animation	AnimationUsingKeyFrames	AnimationUsingPath	Popis
double	✓	✓	✓	Dá se použít pro animování vlastností Width, Height, Fontsize…
string	x	✓	x	Animování textu v TextBox nebo třeba v TextBlock.
Color	✓	✓	x	Animování barvy výplně obdélníku. Animování barvy vlastnosti Background.
Point	✓	✓	✓	Střed EllipseGeometry (Center).
Rect	✓	✓	x	RectangleGeometry vlastnosti.

6.1 Storyboard

Element **`Storyboard`** funguje jako kontejner pro animace. Může obsahovat jednu nebo více animací. **`Storyboard`** poskytuje dvě důležité připojené vlastnosti: *`Storyboard.TargetName`*[28] a *`Storyboard.TargetProperty`*. Na Obrázku 31 je ukázka jak nastavit připojené vlastnosti **`Storybordu`**.

[28] Storyboard.TargetName má ještě jinou variantu a to Storyboard.Target. Tato připojená vlastnost slouží k nastavení odkazu (reference) animovaného objektu.



Obrázek 31 Použití připojených vlastností TargetName a TargetProperty

`Storyboard.TargetName` slouží pro nastavení animovaného objektu. Hodnotou pro tento atribut je název elementu nastavený pomocí *`Name`* nebo *`x:Name`*. *`Storyboard.TargetProperty`* nastaví animovaný atribut na objektu. Obě tyto vlastnosti se připojí k elementu animace.

Storyboard

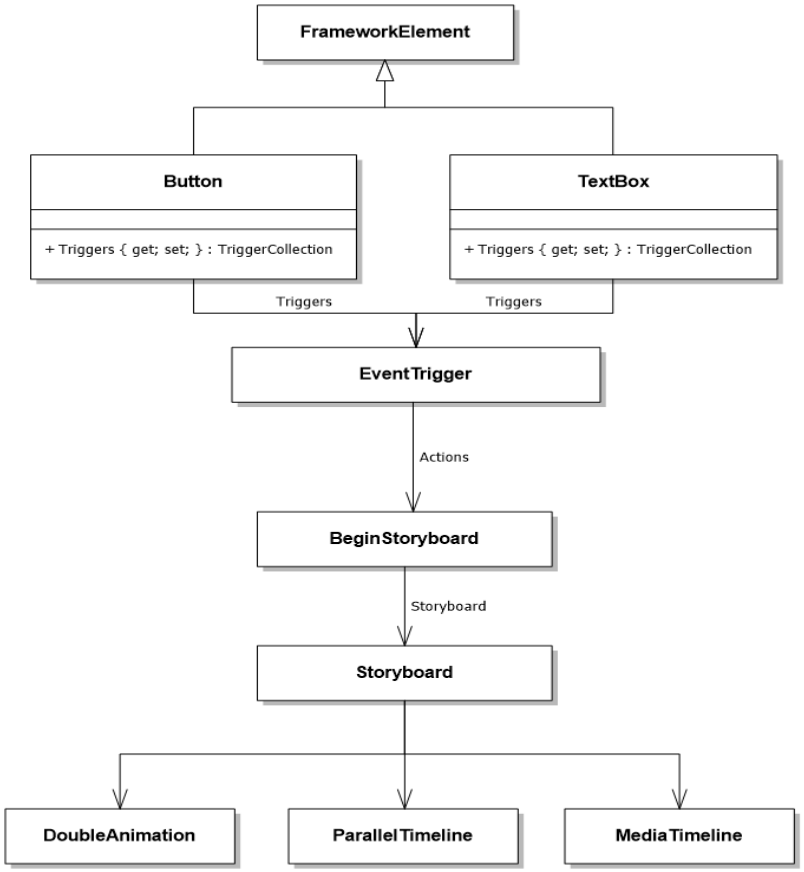
```
<Storyboard>

      <DoubleAnimation Storyboard.TargetName="animovanyObdelnik"
                       Storyboard.TargetProperty="Width"
                       BeginTime="0:0:1"
                       From="50"
                       To="250"
                       Duration="0:0:2.200"/>

</Storyboard>
```

Storyboard *– kontejner pro umístění elementů animací*
DoubleAnimation *– element animace*
Storyboard.TargetName *– připojení názvu animovaného objektu (elementu)*

Storyboard.TargetProperty *– připojení názvu animované vlastnosti (atributu)*
Ostatní atributy budou vysvětleny v kapitole Základní animace.

6.2 Spuštění animace

Pro spuštění animace v XAML je možné vytvořit spínač. Spínač je tak možné přiřadit například k tlačítku, které animaci spustí. Na Obrázku 32 je, zobrazeno jaké elementy jsou potřeba pro spuštění animace.



Obrázek 32 Uspořádání elementů pro spuštění animace

Od třídy `FrameworkElement` získávají elementy `Button` a `Textbox` atribut *`Triggers`*. Ten slouží pro přidání spínače. Element `EventTrigger` spustí při určité události všechny akce v kolekci s názvem *`Actions`*. `Storyboard` není akcí přímo spustitelnou prostřednictvím spínače. Proto potřebuje pro své spuštění element `BeginStoryboard`. Do přímého obsahu tohoto elementu se pak umístí `Storyboard`. Ten následně spustí paralelně všechny animace, co obsahuje. `BeginStoryboard` nedovoluje nastavit čas spuštění animace.

Pro spuštění animace v určitý čas se používá atribut *`BeginTime`*. Ten dovoluje nastavit zpoždění oproti startu všech animací ve `Storyboard`. Element `ParallelTimeline` slouží, k seskupení více animací. Pro ně pak lze nastavit společný čas spuštění. Animace v `ParallelTimeline` jsou spuštěny paralelně s ostatními animacemi ve `Storyboard`.

Spouštění animace

```
<BeginStoryboard>
   <Storyboard>
       <ColorAnimation Storyboard.TargetName="vyplnObdelniku"
                       Storyboard.TargetProperty="Color"
                       BeginTime="0:0:1"
                       From="LimeGreen"
                       To="Orange"
                       Duration="0:0:8"
                       RepeatBehavior="Forever"/>

       <ParallelTimeline BeginTime="0:0:2">
            <DoubleAnimation Storyboard.TargetName="fontSizeTextbox"
                             Storyboard.TargetProperty="FontSize"
                             From="12"
                             To="5"
                             Duration="0:0:2"/>

            <DoubleAnimation Storyboard.TargetName="heightTextbox"
                             Storyboard.TargetProperty="Height"
                             From="23"
                             To="60"
                             Duration="0:0:2"/>
        </ParallelTimeline>
   </Storyboard>
</BeginStoryboard>
```

BeginStoryboard *– spustí* ***Storyboard****, umístí se do elementu EventTrigger*
BeginTime *– nastaví zpoždění animace (hodiny:minuty:sekundy.milisekundy) například. 0:0:1.200*
ParallelTimeline *– seskupí animace a spustí je paralelně k ostatním ve* ***Storyboard***

Ukázkové řešení: Animace/SpusteniAnimace

6.3 Základní animace

Základní animace dovoluje animovat hodnotu atributu od hodnoty/do hodnoty/podobu. To znamená, že se nastaví počáteční a koncová hodnota atributu. A určí se čas během, kterého se změní počáteční hodnota na koncovou. Každá změna hodnoty se rovnoměrně rozdělí po celou dobu běhu animace. Správně se tento typ animace nazývá lineárně interpolační. Změna hodnoty je po celou dobu běhu animace konstantní[29]. Syntaxe základní animace je: `<typ_vlastnosti>Animation`.

Základní animace

```
<DoubleAnimation Storyboard.TargetName="animovanyObdelnik"
                 Storyboard.TargetProperty="Width"
                 BeginTime="0:0:1"
                 AutoReverse ="True"
                 From="50"
                 To="250"
```

[29] Lze změnit pomocí atributů AccelerationRatio (zrychlení animace) a DecelerationRatio (zpomalení animace).

```
                    Duration="0:0:2.200"/>
```

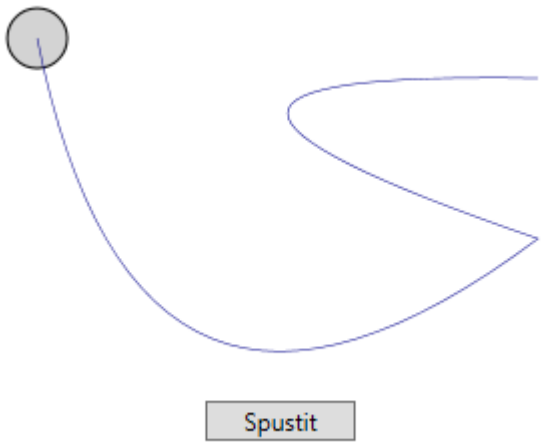
DoubleAnimation *– základní typ animace pro typ* ***double***
AutoReverse *– na konci animace se hodnoty* ***From/To*** *přehodí a animace se ještě jednou spustí*
From *– počáteční hodnota animované vlastnosti*
To *– cílová hodnota animované vlastnosti*
Duration *– délka animace (2 sekundy a 200 milisekund)*
Pokud hodnota ***From*** *se nerovná hodnotě animované vlastnosti. Pak se při spuštění animace nastaví animovaná vlastnost na hodnotu* ***From****.*

V ukázkové aplikaci jsou použity i další atributy ovlivňující běh animace. `AccelerationRatio` a `DecelerationRatio` zrychlí (Acc.) nebo zpomalí (Dec.) běh animace. Celkový čas animace zůstane stejný. V kapitole Spuštění animace byl použit atribut `RepeatBehavior = "Forever"`. To nastaví opakování animace na nekonečno.

Ukázkové řešení: Animace/DoubleAnimation ; Animace/ColorAnimation

6.4 Animace po křivce

Dovoluje animovat hodnotu vlastnosti po křivce. Například střed elipsy – `Center`. Tedy vlastnost s hodnotou typu **`Point`** (bod). Element animace po křivce má atribut `PathGeometry`. Kde se pomocí elementu **`PathGeometry`** a elementu **`PathFigure`** nastavuje cesta pro objekt. Syntaxe elementu animace po křivce je následující: `<typ_vlastnosti>AnimationUsingPath`. Na Obrázku 33 je ukázka z aplikace pro animaci po křivce.



Obrázek 33 Ukázka z aplikace PointAnimationUsingPath

Animace po křivce

```
<PointAnimationUsingPath Storyboard.TargetName="elipsa"
                         Storyboard.TargetProperty="Center"
                         Duration="0:0:6">
        <PointAnimationUsingPath.PathGeometry>

            <PathGeometry>
```

```
                <PathFigure StartPoint="50,50">
                   <QuadraticBezierSegment Point1="100,300" Point2="300,150" />
                   <QuadraticBezierSegment Point1="50,65" Point2="300,70" />
                </PathFigure>
             </PathGeometry>

          </PointAnimationUsingPath.PathGeometry>
</PointAnimationUsingPath>
```

PointAnimationUsingPath *– dovoluje animovat bod, pohybující se po křivce*
PathGeometry *– nastaví křivku po, které se bude pohybovat objekt*
Center *– střed elipsy*
Křivku je zapotřebí nakreslit samostatně.

Ukázkové řešení: Animace/PointAnimationUsingPath

6.5 Animace s klíčovými snímky

Tento druh animace v sobě spojuje více typů animací. Obsahuje lineárně interpolační animaci, diskrétní animaci a spline animaci. Každý typ animace pak představuje jeden klíčový snímek. Těch může být v animaci s klíčovými snímky vícero. Všechny klíčové snímky jsou určené pouze pro typ hodnoty vlastnosti, pro které je animace s klíčovými snímky. Nelze kombinovat například klíčové snímky pro typ **double** a typ **string**. Také ne pro všechny typy vlastností se dají použít všechny typy klíčových snímků. Například pro typ string nedávají smysl klíčové snímky: lineární a spline (animace po křivce). Pro nastavení klíčových snímků slouží atribut `KeyFrames`.

DoubleUsingKeyFrame

```
<DoubleAnimationUsingKeyFrames Storyboard.TargetName="presunCtverce1"
                               Storyboard.TargetProperty="X"
                               Duration="0:0:8"
                               FillBehavior="Stop">

          <DoubleAnimationUsingKeyFrames.KeyFrames>
                 <LinearDoubleKeyFrame Value="400" KeyTime="0:0:8"/>
          </DoubleAnimationUsingKeyFrames.KeyFrames>

</DoubleAnimationUsingKeyFrames>
```

FillBehavior *– po skončení animace se nastaví původní animovaná hodnota vlastnosti (Stop)*
KeyFrames *– vlastnost pro definování klíčových snímků*

Syntaxe animace s klíčovými snímky:

`<typ_vlastnosti>AnimationUsingKeyFrames`

`Například: DoubleAnimationUsingKeyFrames, StringAnimationUsingKeyFrames`

Syntaxe klíčových snímků:

Lineární: `Linear<typ_vlastnosti>KeyFrame`

Diskrétní: `Discrete<typ_vlastnosti>KeyFrame`

Spline: `Spline<typ_vlastnosti>KeyFrame`

Lineární klíčový snímek

Funguje podobně jako základní animace. Element lineárního snímku má hodnotu *`Value`*. Ta slouží podobně jako atribut *`To`* u základní animace. *`KeyTime`* je cílový čas během kterého se změní původní hodnota na hodnotu *`Value`*. Pokud, je snímek prvním snímkem v animaci. Původní hodnota je stejná jako hodnota vlastnosti animovaného objektu. Jinak je původní hodnota stejná jako hodnota *`Value`* u předchozího snímku.

Lineární klíčový snímek

```
<LinearDoubleKeyFrame Value="400" KeyTime="0:0:8"/>
```

LinearDoubleKeyFrame *– lineární klíčový snímek pro typ double*
Value *– cílová hodnota*
KeyTime *– cílový čas, kdy hodnota animované vlastnosti bude rovna* ***Value***

Ukázkové řešení: Animace/DoubleAnimationUsingKeyFrames

Diskrétní klíčový snímek

Diskrétní klíčový snímek změní hodnotu animované vlastnosti na hodnotu *`Value`*. Hodnota je změněna bez přechodu z původní hodnoty na hodnotu *`Value`*. Ke změně vlastnosti dojde v *`KeyTime`*.

Diskrétní klíčový snímek

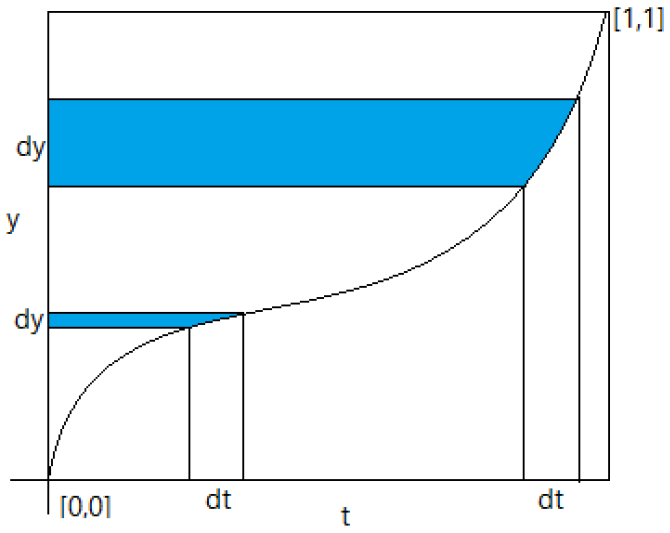
```
<StringAnimationUsingKeyFrames.KeyFrames>
         <DiscreteStringKeyFrame Value="A" KeyTime="0:0:1"/>
           <DiscreteStringKeyFrame Value="An" KeyTime="0:0:2"/>
           <DiscreteStringKeyFrame Value="Ani" KeyTime="0:0:3"/>
           <DiscreteStringKeyFrame Value="Anim" KeyTime="0:0:4"/>
           <DiscreteStringKeyFrame Value="Anima" KeyTime="0:0:5"/>
           <DiscreteStringKeyFrame Value="Animac" KeyTime="0:0:6"/>
           <DiscreteStringKeyFrame Value="Animace" KeyTime="0:0:7"/>
</StringAnimationUsingKeyFrames.KeyFrames>
```

DiscreteStringKeyFrame – diskrétní klíčový snímek pro typ String
Value – nastaví hodnotu v KeyTime
KeyTime – cílový čas

Ukázkové řešení: Animace/DiscreteStringKeyFrame

Spline snímek

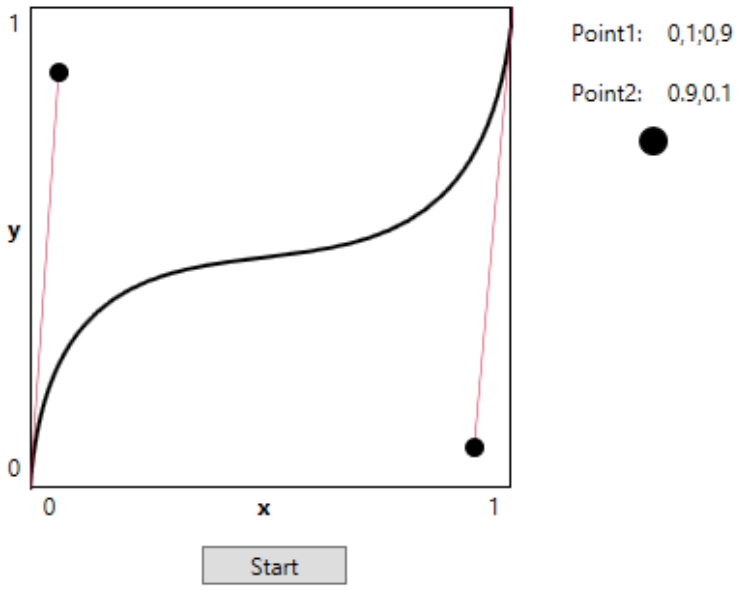
U základní animace je změna hodnoty dána lineární funkcí a časem běhu animace. U Spline snímku je změna hodnoty dána kubickou Bézierovou křivkou a časem běhu animace. Na Obrázku 34 je zobrazena tato křivka.



Obrázek 34 Spline křivka pro animování hodnoty vlastnosti

Tato křivka se nazývá Spline křivka. Počátek má v bodě [0,0] a konec křivky je v bodě [1,1]. Pro nastavení tvaru křivky slouží dva kontrolní body.

Podle tvaru křivky se v určitém časovém okamžiku (**dt**) zjišťuje hodnota změny animované vlastnosti (**dy**). Na obrázku jsou časové úseky **dt** stejně velké. Naproti tomu změna hodnoty **dy** je pro každý časový okamžik různá. Čím křivka strměji stoupá, tím budou i větší změny animované vlastnosti. Na Obrázku 35 je ukázka z aplikace **`SplineDoubleKeyFrame`**.



Obrázek 35 Ukázka z aplikace SplineDoubleKeyFrame

Spline křivka

```
<SplineDoubleKeyFrame Value="400" KeySpline="0.1,0.9 0.9,0.1" KeyTime="0:0:5"/>
```

KeySpline *– nastaví kontrolní body křivky (P1 P2)*
Kontrolní bod P1 je na Obrázku 35 vlevo.

Ukázkové řešení: Animace/SplineDoubleKeyFrame

6.6 Animace – Windows Forms

Windows Forms neobsahuje vytváření animaci jako je ve WPF. Při vytváření animace je potřeba řídit celou animaci. K obnovování snímku se používá třída **`Timer`**. U té se nastaví interval obnovování pomocí vlastnosti *`Interval`*. Dále je potřeba spočítat změnu hodnoty. Následně aktualizovat animovanou hodnotu při vzniku událost *Tick*.

Změna animované hodnoty

```
void timer_Tick(object sender, EventArgs e)
{
     if (novaPoziceTlacitka.X < cilovaHodnota)
     {
        novaPoziceTlacitka.X += dx;
        animaceButton.Location = novaPoziceTlacitka;
     }
     else
     {
        //zastaví běh animace
        timer.Stop();
     }
}
```

timer_Tick *– tato událost vznikne pokaždé, když uplyne čas daný vlastností* ***Interval***
dx *– změna hodnoty*

Ukázkové řešení: Windows Forms/Animace

7 Media

WPF dovoluje zobrazovat obrázky, přehrávat video i zvuk. Podporuje řadu formátů, které je možné zobrazit nebo přehrát. Je možné si vytvořit i svůj vlastní přehrávač nebo prohlížeč obrázků. Práce se medii je ve WPF velice intuitivní. Pro přehrávání zvuku i videa se používá jediný element.

7.1 Zobrazení obrázku

Obrázek je možné zobrazit pomocí elementu Image. Tento element má atribut Source pro určení cesty k obrázku. Podporované formáty obrázku jsou například: gif, jpg, png, bmp…

Zobrazení obrázku

```
<Image Source="zapad_slunce.jpg" Width="450" Height="350"/>
```

Source *– cesta k souboru obrázku (např. Obrazky/nazev_obrazku.jpg)*
Width *a* ***Height*** *– nastavují šířku a výšku elementu* ***Image***

Ukázkové řešení: Media/ZobrazeniObrazku

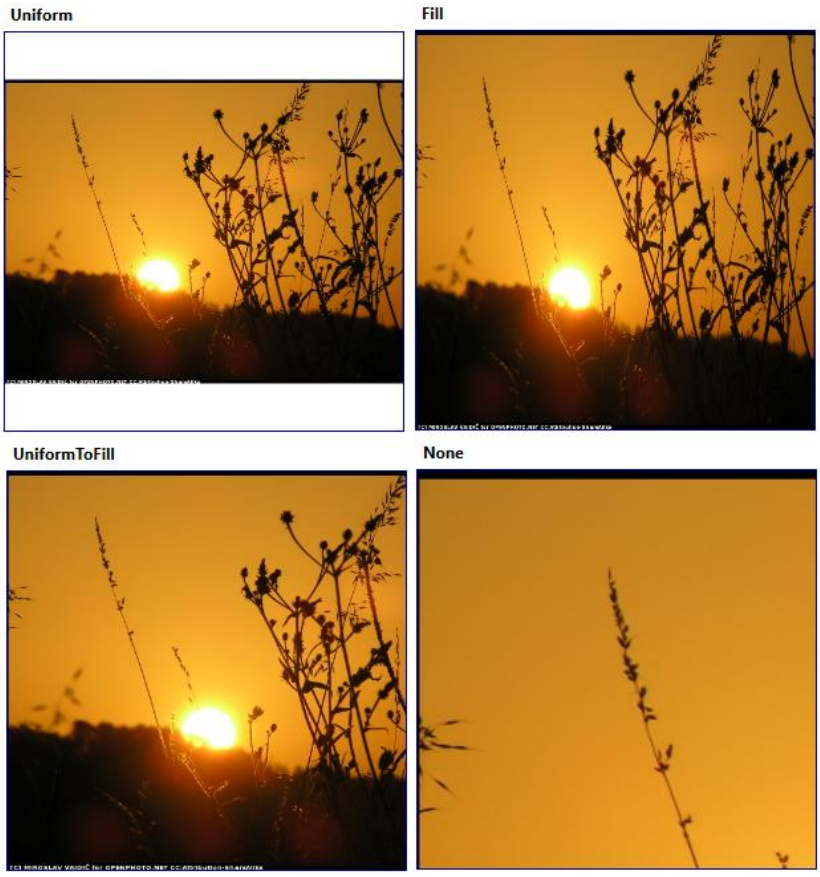
Element Image má atribut `Stretch` pro nastavení zobrazení obrázku. Pokud obrázek bude mít jinou velikost než element Image, zobrazení se upraví pomocí tohoto atributu. Tento element má čtyři nastavení a to None, Uniform, UniformToFill a Fill. V Tabulce 2 jsou uvedeny popisy pro jednotlivé volby.

Tabulka 3 Nastavení atributu Stretch pro element Image

Stretch	Popis
None	Zobrazí celý obrázek a ignoruje šířku a výšku elementu Image.
Uniform	Zobrazí celý obrázek a zachová poměr stran u obrázku. Mohou se objevit prázdná místa. **Výchozí hodnota.**
UniformToFill	Zachová poměr stran. Obrázek ořízne tak, aby nevznikla prázdná místa.
Fill	Vyplní obrázkem celý element Image. Nezachovává poměr stran u obrázku.

Na Obrázku 36[30] jsou zobrazeny všechny tyto možnosti.

[30] Zdroj obrázku:
http://openphoto.net/gallery/image.html?image_id=15458&hints=#how_to_credit_this_image-



Obrázek 36 Možnosti nastavení atributu Stretch u Image

Ukázkové řešení: Media/Stretch

U obrázku je také možné změnit formát. Je tak možné například upravit obrázek na stupně šedi. Pro změnu formátu obrázku slouží element **`FormatConvertedBitmap`**. Tento element se použije pro načtení obrázku a umístí se do atributu *`Source`* elementu **`Image`**.

Změna formátu obrázku

```
<Image>
 <Image.Source>

     <FormatConvertedBitmap Source="zapad_slunce.jpg"
                            DestinationFormat="Gray32Float"/>

 </Image.Source>

</Image>
```

FormatConvertedBitmap – načte obrázek a změní mu formát
Source – umístění souboru obrázku
DestinationFormat – cílový formát obrázku
Gray32Float – stupně šedi
Na Obrázku 37 je zobrazen původní a upravený obrázek.

Obrázek 37 Původní a upravený obrázek ve stupních šedi

Ukázkové řešení: Media/StupneSedi

Obrázek lze také rotovat o přírůstky 90 stupňů. Stejně jako u změny formátu obrázku se použije další element pro načtení. Pro rotaci se použije element **`TransformedBitmap`**. Tento element má atribut *`Transform`* pro nastavení transformace.

Rotace obrázku

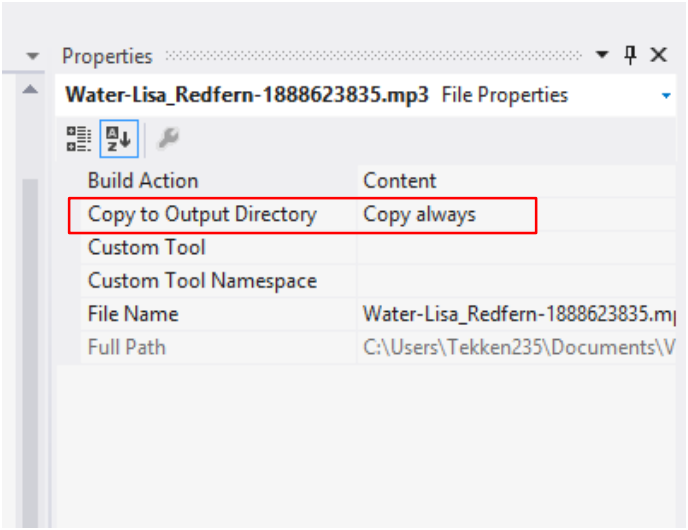
```
<TransformedBitmap Source="zapad_slunce.jpg">
        <TransformedBitmap.Transform>
             <RotateTransform Angle="90"/>
        </TransformedBitmap.Transform>
</TransformedBitmap>
```

TransformedBitmap *– transformování obrázku*
Transform *– slouží pro nastavení obrázku*
RotateTransform *– rotuje s obrázkem o přírůstky 90 stupňů (90,180,270)*
Na Obrázku 38 je ukázka rotace obrázku o 90 stupňů.

Obrázek 38 Rotace obrázku o 90 stupňů

7.2 Přehrávání videa a zvuku

Pro přehrávání videa a zvuku slouží jediný element – **`MediaElement`**. Pro přehrání je potřeba určit přehrávaný soubor. K tomu slouží atribut *`Source`*. Soubor videa nebo zvuku musí být umístěn ve složce s aplikací. Je potřeba nastavit ve Visual Studiu, aby se soubor kopíroval do výstupní složky aplikace. Na Obrázku 39 je označeno nastavení pro kopírování souboru do složky s programem. Nejprve je, ale nutné označit v **Solution Explorer** soubor. Jako hodnota se nastaví buď **Copy always** (kopíruj vždy) nebo **Copy if newer** (kopíruj, pokud soubor je upraven).



Obrázek 39 Nastavení kopírování souboru do výstupní složky

Přehrání zvuku

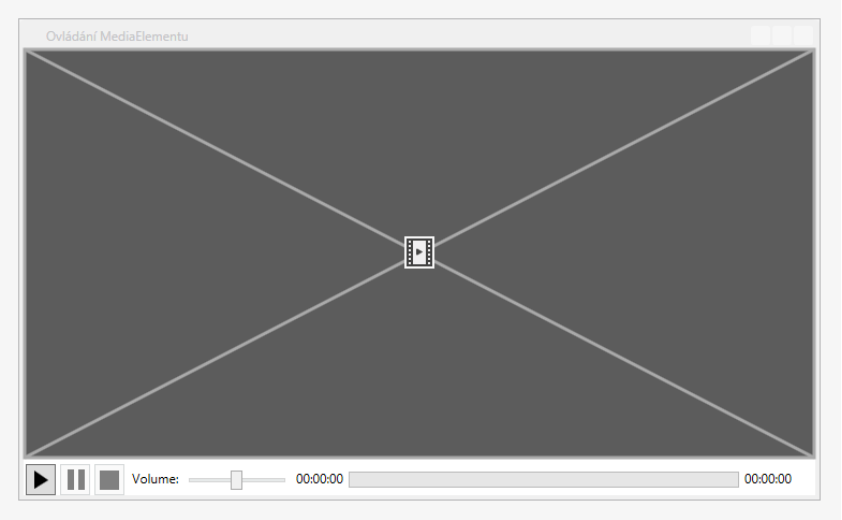
```
<!--Zdroj: http://soundbible.com/2032-Water.html-->

<MediaElement Source="Water-Lisa_Redfern-1888623835.mp3"
              Visibility="Collapsed"
              LoadedBehavior="Play"/>
```

`Source` *– cesta k souboru*
`Visibility` *– při přehrávání není potřeba, aby byl* ***`MediaElement`*** *viditelný nebo zabíral místo (Collapsed)*
`LoadedBehavior` *– co se má stát po načtení* ***MediaElementu***
Podporované formáty jsou například mp3, wav…

Ukázkové řešení: Media/PrehraniZvuku

Přehrání videa

Při nastavení *`LoadedBehavior`* na **Manual** je možné **`MediaElement`** ovládat za pomocí metod `Play()`, `Pause()` a `Stop()`. V ukázkové aplikaci jsou tyto metody použity pro vytvoření jednoduchého přehrávače. Na Obrázku 40 je ukázka z aplikace OvladaniMediaElementu.



Obrázek 40 Jednoduchý přehrávač vytvořený pomocí MediaElementu

Ukázkové řešení: Media/OvladaniMediaElementu

7.3 Media – Windows Forms

Pro zobrazení obrázku ve Windows Forms slouží ovládací prvek **`PictureBox`**. Obrázek se nastavuje pomocí vlastnosti *`Image`*. Nejednoduší způsob jak tento atribut je nastavit ho prostřednictvím panelu **Properties**.

Pro přehrání videa nemá Windows Forms ovládací prvek. Ale je možné integrovat Windows Media Player do rozhraní aplikace. Návod jak přidat Windows Media Player do rozhraní najdete například zde: http://www.c-sharpcorner.com/uploadfile/e628d9/playing-audio-and-video-files-using-C-Sharp/.

Pro přehrání zvuku[31] je možné použít třídu **`SoundPlayer`**. Ta dovoluje přehrát wav soubory. Případně je také možné použít Windows Media Player.

Přehrání zvuku pomocí třídy SoundPlayer

```
SoundPlayer player = new SoundPlayer("tada.wav");
player.Play();
```

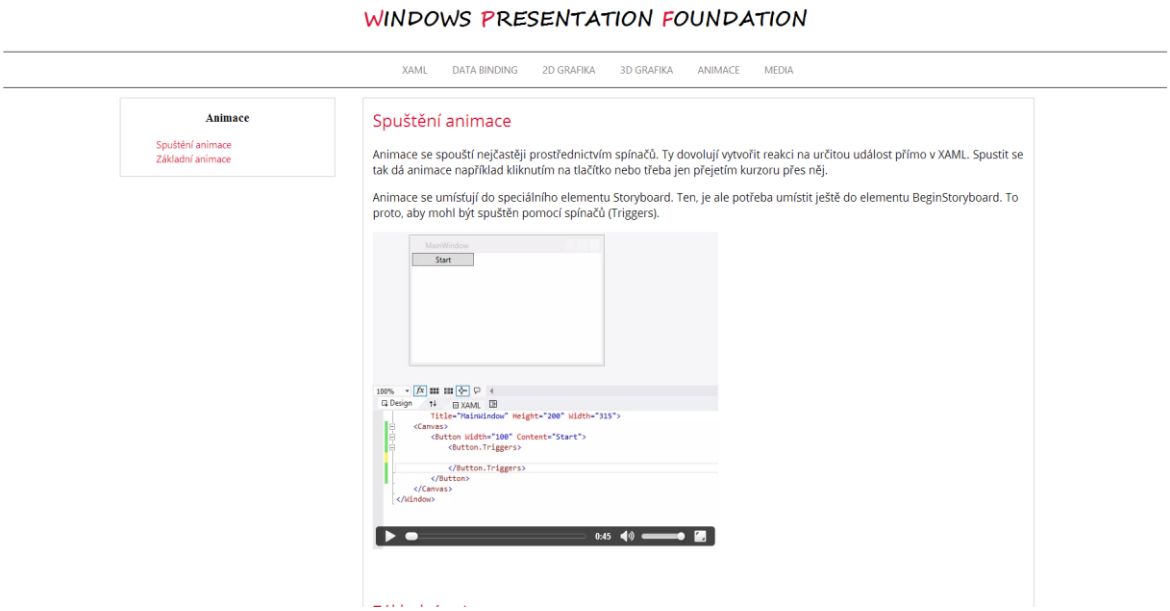
U souboru musí být nastavenou kopírování do výstupní složky.

[31] Pro lepší přehrání je možné také použít NAudio. Naleznete ho zde: http://naudio.codeplex.com/. Podporuje nejenom přehrávání, ale také nahrávání zvuku.

8 Multimediální aplikace

Aplikace byla vytvořena jako skupina navzájem propojených webových stránek. Tato aplikace obsahuje pouze statický obsah. Nedovoluje přidávat další obsah bez nutnosti editace HTML kódu. Tímto přístupem se aplikace stává snadněji přenositelnou. Není potřeba instalovat webový server.

Aplikace se spouští výběrem libovolné stránky (souboru s příponou .html). Tato stránka pak bude zobrazena v prohlížeči. Na obrázku x. je zobrazeno rozhraní stránky.



Obrázek 41 Rozhraní aplikace

Jednotlivé stránky jsou propojeny za pomocí tagu <a>. Ten dovoluje po kliknutí na tento odkaz, načíst další stránku do prohlížeče. V aplikaci je vytvořeno hlavní menu pomocí tagu <a>. Kde jsou všechny odkazy na hlavní kapitoly. Každá hlavní kapitola je vytvořena jako jeden soubor webové stránky.

Příklad odkazu na jinou stránku

```
<a href="Animace.html">Animace</a>
```

Po kliknutí na nápis Animace dojde k načtení stránky (souboru) s touto kapitolou do prohlížeče.

Stránky jsou postaveny na bázi HTML5[32]. Což je nová specifikace pro HTML jazyk. Pro tuto aplikaci je především důležitá možnost přehrávat v prohlížeči videa. Přehrávat videa v prohlížeči sice není novinkou, ale nyní je to možné provést přímo prostřednictvím HTML tagu. Není tak potřeba používat technologie jako je Flash nebo Silverlight.

Přehrání videa pomocí tagu <video>

```
<video width="580" height="512" controls>
```

[32] Další informace o HTML5 najdete například zde: http://cs.wikipedia.org/wiki/HTML5 [cz].

```
      <source src="Video/Kamera.webm" type="video/webm">
</video>
```

V HTML5 jsou zatím podporovány tři formáty videa:

- Mp4
- WebM
- Ogg

Ne všechny formáty jsou podporovány ve všech prohlížečích. V aplikaci je použit formát WebM. Ten je podporován v poslední verzi Google Chrome. Doporučuji tedy pro prohlížení použít právě tento prohlížeč.

Videa v aplikaci byla zachycena pomocí programu Microsoft Expression Encoder4 Screen Capture. Pomocí programu Expression Encoder4 byla převedena na formát Mp4. Jiný formát není podporován v této verzi aplikace. Video bylo následně převedeno z formátu Mp4 na formát WebM. Tento převod byl uskutečněn pomocí konvertoru na této stránce: http://www.online-convert.com/.

Závěr

WPF je velice rozsáhlá platforma pro vytváření rozhraní aplikací. Není možné v rámci bakalářské práce popsat všechny části. Nicméně stěžejní části WPF jsou tu uvedeny. Velký důraz byl kladen na vysvětlení práce s XAML a použití data binding. Což jsou dvě části, které dokáží velice usnadnit práci. Kapitoly nejsou psané tak, aby do hloubky probíraly některou z částí WPF. Spíše jsou psány s účelem vytvořit základy pro čtenáře. Tím, že už má čtenář základy, se může pustit do podrobnějšího studia WPF. Další informace z různých zdrojů by měli být pro něj již snadněji pochopitelné.

V kapitole XAML jsou uvedeny informace potřebné pro vytvoření aplikace. Tato kapitola je důležitá pro to, aby čtenář pochopil další části. V rámci celé práce je dávána přednost použití XAML před kódem napsaným v C#. Je to z toho důvodu, aby se čtenář naučil používat XAML. Také, aby se minimalizovalo riziko, že pude používat jenom C# pro vytváření rozhraní. Což je sice možné, ale není to rozumný nápad. Pokud vznikne problém řešitelný v C#, ale špatně v XAML. Může se podívat do dokumentace, kde jsou často uvedeny ukázky napsané v C#. Při troše zkoumání porozumí jak vytvořit stejný kód jako je v XAML i v C#.

Kapitola data binding je asi nejvíce zjednodušená. Jsou zde uvedeny základní způsoby jak vytvořit data binding. Hlavně tedy v prostředí XAML, ale je zde i ukázka pro C#. Tato část WPF je velice rozsáhlá. Proto bylo potřeba ji zmenšit a uvést základní informace jak vytvořit data binding. Tak, aby se dostalo i na ostatní kapitoly. Nicméně cílem bylo seznámit uživatele s jednotlivými částmi. Ne z nich udělat odborníky na danou problematiku. Pro další informace je možné použít literaturu nebo navštívit stránky na MSDN.

Po přečtení kapitoly 2D by mněl čtenář chápat jakým způsobem se přidává a vytváří grafika pro rozhraní aplikace. Tato kapitola se nezabývá designem rozhraní. Je na čtenáři zda přidá grafiku do rozhraní nebo ne. Z uvedených informací v této kapitole může vytvořit také jednoduchou hru.

Kapitola s 3D grafikou slouží nejenom pro vytváření 3D objektů. Ale také jako základní náhled do teorie zobrazování 3D objektů. Znalosti s této kapitoly jsou spíše informativní. Sice je možné vytvořit základní objekty jako trojúhelník nebo krychle. Ale pro vytvoření složitějších objektů bude zapotřebí použití jiných nástrojů. Nejedná se o platformu pro tvorbu 3D her.

Kapitola Animace obsahuje informace potřebné pro vytvoření a spuštění animace. V této kapitole nejsou uvedeny animace pro všechny typy vlastností. Není dostatek místa pro jejich popsání. Také by to bylo celkem zbytečné. Stačí pochopit princip, na jakém funguje systém animací ve WPF.

Poslední kapitola z teoretické části Media je celkem krátká. Tato část WPF je velice intuitivní a nepotřebuje nijak sáhodlouhé vysvětlování. Navíc pro některé zde použité

možnosti existuje lepší řešení. To, ale nebývá součástí WPF, takže je spíše jenom zmíněno. Po přečtení této kapitoly by mněl čtenář být schopen vytvořit jednoduchý přehrávač videa. Místo ušetřené na této kapitole bylo využito v důležitějších kapitolách.

Kapitola Aplikace popisuje použité vlastnosti WPF. Pro vytvoření aplikace zobrazující dokumenty. Dává možnost čtenáři se inspirovat při vytváření jeho vlastních aplikací. Obsah je vytvořen tak, aby umožnil si zrekapitulovat získané informace. Případně si vyzkoušet ukázky aplikací.

Tato práce sice není vyčerpávajícím materiálem o WPF. Nejsou zde všechny užitečné informace. Není ale na škodu se naučit je vyhledávat. Některé problémy už pravděpodobně někdo řešil. To dává možnost najít řešení vlastního problému. Co se tyče pokračování. Asi nejlepší variantou by bylo vytvoření webové stránky. Kde by se informace mohly měnit podle odezvy čtenářů.

Literatura

Anderson, Chris. 2007. *Essential Windows Presentation Foundation.* London : Addison-Wesley, 2007. str. 458. ISBN 978-0-321-37447-9.

Nathan, Adam. 2010. *WPF 4 Unleashed.* Indianapolis : Sams Publishing, 2010. str. 848. ISBN 978-0-672-33119-0.