

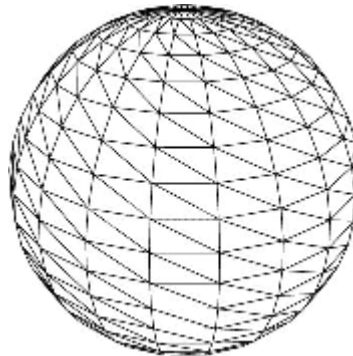
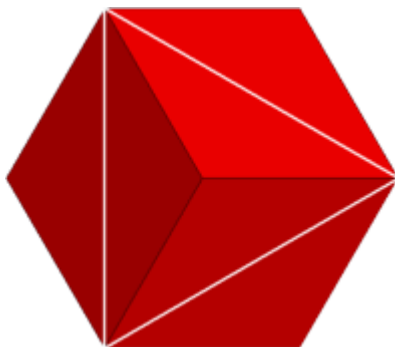
7. WPF - 3D grafika I.

Renderování trojrozměrné grafiky je další z řady funkcí Windows Presentation Foundation.

Jak se s 3D ve WPF pracuje?

Jediné co máme k dispozici a na čem musíme stavět je trojúhelník ...

Každý tvar můžeme poskládat z trojúhelníku, ať už se jedná o 2D objekt jako je čtverec (2 trojúhelníky) nebo 3D objekt jako je čtyřboký jehlan (6 trojúhelníků) nebo krychle (12 trojúhelníků). Složitější tvary jako koule se dají také poskládat z trojúhelníků, čím více jich použijeme, tím "kulatější" naše koule bude.



Proč ve WPF nejsou implicitně složitější tvary jako krychle či koule? WPF není dělané pro vývoj 3D her nebo pokročilých trojrozměrných aplikací, ve WPF by nám trojrozměrná grafika měla sloužit k vytváření 3D efektů nebo jednoduchých aplikací.

Základ je Viewport3D

Jedná se o element, který v aplikaci vytvoří trojrozměrný prostor, do kterého můžeme vkládat 3D objekty. Vytvořme si tedy takovéto trojrozměrné prostředí...

```
<Viewport3D>
  <Viewport3D.Camera>
    <PerspectiveCamera LookDirection="0,0,-1" Position="0,0,5"
      NearPlaneDistance="3" FarPlaneDistance="100" />
  </Viewport3D.Camera>
  <ModelVisual3D>
    <ModelVisual3D.Content>
      <DirectionalLight Color="White" Direction="0,0,-1" />
    </ModelVisual3D.Content>
  </ModelVisual3D>
</Viewport3D>
```

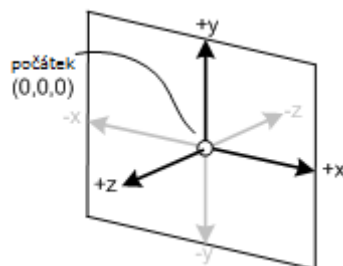
1. Vytvořili jsme trojrozměrné prostředí pomocí *Viewport3D*
2. Vložili jsme kameru, z jejíž pohledu je prostředí renderováno
 - a. *Position* - pozice kamery v prostředí
 - b. *LookDirection* - směr pohledu kamery

** Z hodnot těchto dvou atributů můžeme jednoduše spočítat, že kamera se dívá ve směru osy Z do oblasti "za monitorem"*

- c. Atributy *NearPlaneDistance*, *FarPlaneDistance* slouží k určení přední a zadní hranice vykreslování, tzn. zobrazeny budou pouze objekty vyskytující se mezi těmito dvěma hranicemi. To má zabránit vykreslení objektů které jsou příliš blízko nebo naopak příliš daleko od kamery.
3. Světlo - každé 3D prostředí potřebuje světlo. Ve WPF se světlo vkládá stejně jako model, takže jsme ho vložili mezi elementy *ModelVisual3D*.

Ve WPF máme k dispozici několi druhů světel

- a. *AmbientLight* - osvítí všechny objekty stejně, rovnoměrně na všech stranách
- b. *DirectionalLight* - můžeme přirovnat k slunečnímu světlu, svítí v jednom směru (vector). Narozdíl od *AmbinetLight* osvítí pouze tu stranu modelu, na kterou dopadá světlo.
- c. další jsou *PointLight*, *PointLightBase* a *SpotLight*

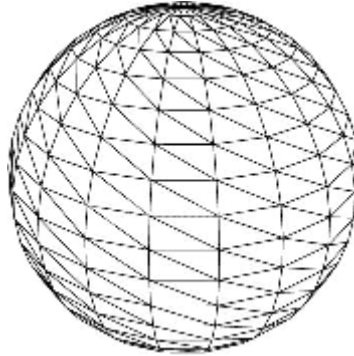


sořadný systém WPF (+z = před monitorem, -z = za monitorem)

3D modely

Každý model se skládá z povrchu (*mesh*) a nanesené textury.

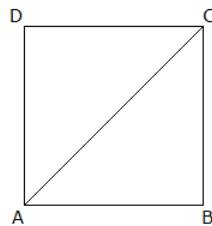
Mesh - jedná se o množinu propojených bodů (propojených do trojúhelníků), které tvoří povrch modelu.



mesh koule

Vytvoření povrchu

Vytvoříme mesh čtverce. Poskládáme čtverec ze dvou trojúhelníků, protože to je jediný tvar, který ve WPF máme.



```
MeshGeometry3D mesh = new MeshGeometry3D();

mesh.Positions.Add(new Point3D(0, 0, 0)); // bod A (0)
mesh.Positions.Add(new Point3D(1, 0, 0)); // bod B (1)
mesh.Positions.Add(new Point3D(1, 1, 0)); // bod C (2)
mesh.Positions.Add(new Point3D(0, 1, 0)); // bod D (3)

// trojuhelnik ABC
mesh.TriangleIndices.Add(0);
mesh.TriangleIndices.Add(1);
mesh.TriangleIndices.Add(2);

// trojuhelnik ACD
mesh.TriangleIndices.Add(0);
mesh.TriangleIndices.Add(2);
mesh.TriangleIndices.Add(3);

mesh.TextureCoordinates.Add(new Point(0, 1)); // bod D
mesh.TextureCoordinates.Add(new Point(1, 1)); // bod C
mesh.TextureCoordinates.Add(new Point(1, 0)); // bod B
mesh.TextureCoordinates.Add(new Point(0, 0)); // bod A
```

1. Vložili jsme body, které tvoří povrch. První vložený bod má index 0, druhý bod má index 1, atd ...
2. *TriangleIndicies* stanoví pořadí, ve kterém body uvedené v *Positions* tvoří trojúhelníky. Čtverec ABCD vytvoříme ze dvou trojúhelníků ABC a ACD. Do *TriangleIndicies* jsme zadali indexy bodů podle toho, jak tvoří tyto trojúhelníky.
3. *TextureCoordinates* je kolekce bodů určujících jak má být textura vykreslena na *mesh*. Zadané hodnoty mohou nabývat hodnot od 0 do 1.

Textura

Jakmile máme vytvořený *mesh*, stačí už pouze vykreslit texturu (*Material*).

```
DiffuseMaterial material = new DiffuseMaterial(Brushes.Red);
GeometryModel3D model = new GeometryModel3D(mesh, material);
```

** Pokud jako texturu použijeme barvu (viz. tento případ), můžeme TextureCoordinates při vytváření mesh vynechat. Pokud bychom ale použili gradient, obrázek nebo jinou texturu typu Brush, musíme TextureCoordinates uvést.*

Vytvořit jednoduchý tvar jako je trojúhelník nebo čtverec není těžké, o mnoho pracnější je ale vytvořit kouli nebo jehlan. Naštěstí na internetu je k dispozici několik implementací, které matematicky počítají tyto složité objekty a které můžeme použít v našich aplikacích.

Příklad

Vytvoříme projekt "WPF3D". V tomto projektu vytvoříme trojrozměrné prostředí, do kterého vložíme dva stejně velké čtverce (červený a zelený).

Window1.xaml

```
<Window x:Class="WPF3D.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="WPF3D" Height="300" Width="300"
  >
  <Viewport3D Name="myViewport3D">
    <Viewport3D.Camera>
      <PerspectiveCamera LookDirection="0,0,-1" Position="0,0,5"
        NearPlaneDistance="3" FarPlaneDistance="100" />
    </Viewport3D.Camera>
    <ModelVisual3D>
      <ModelVisual3D.Content>
        <DirectionalLight Color="White" Direction="0,0,-1" />
      </ModelVisual3D.Content>
    </ModelVisual3D>
  </Viewport3D>
</Window>
```

Window1.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Media3D;
using System.Windows.Controls;

namespace WPF3D
{
    public partial class Window1 : System.Windows.Window
    {
        public Window1()
        {
            InitializeComponent();

            Model3DGroup models = new Model3DGroup();

            Model3D rectangle1 = CreateRectangle(
                new Point3D(0, 0, 0),
                new Point3D(1, 0, 0),
                new Point3D(1, 1, 0),
                new Point3D(0, 1, 0),
                Brushes.Red);

            Model3D rectangle2 = CreateRectangle(
                new Point3D(0.5, 0.5, -1),
                new Point3D(1.5, 0.5, -1),
                new Point3D(1.5, 1.5, -1),
                new Point3D(0.5, 1.5, -1),
                Brushes.Green);

            models.Children.Add(rectangle1);
            models.Children.Add(rectangle2);

            ModelVisual3D visual = new ModelVisual3D();
            visual.Content = models;

            myViewport3D.Children.Add(visual);
        }

        public GeometryModel3D CreateRectangle(
            Point3D point1, Point3D point2,
            Point3D point3, Point3D point4,
            Brush brush)
        {
            MeshGeometry3D mesh = new MeshGeometry3D();
            mesh.Positions.Add(point1);
            mesh.Positions.Add(point2);
            mesh.Positions.Add(point3);
            mesh.Positions.Add(point4);

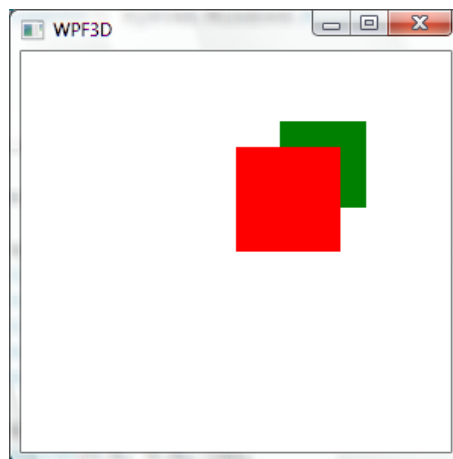
            mesh.TriangleIndices.Add(0);
            mesh.TriangleIndices.Add(1);
            mesh.TriangleIndices.Add(2);

            mesh.TriangleIndices.Add(0);
            mesh.TriangleIndices.Add(2);
            mesh.TriangleIndices.Add(3);
        }
    }
}
```

```
        return new GeometryModel3D(mesh,  
            new DiffuseMaterial(brush));  
    }  
}
```

1. Vytvořili jsme *Viewport3D* a vložili kameru a světlo.
2. Přidali namespace *Media3D*
3. *CreateRectangle()* - vytvořili jsme metodu, která vrací model čtverec podle zadaných bodů
4. Vytvořili jsme dva stejně velké čtverce (červený a zelený). Abychom mohli cokoli vložit do *Viewport3D*, musí to být typu *Visual3D*, proto jsme vložili čtverce do *ModelVisual3D* a následně do *myViewport3D*.

Aplikace po spuštění



I když se jedná o stejně velké čtverce, zelený čtverec je opticky menší protože se nachází o jednotku dál za červeným čtvercem (Z souřadnice je -1).

3D Tools for Windows Presentation Foundation

Jedná se o knihovnu napsanou WPF 3D teamem s užitečnými funkcemi jako jsou

- použití interaktivních 2D prvků na 3D povrchu
- Trackball - rotace kamery myší
- kreslení čar v prostoru
- ...

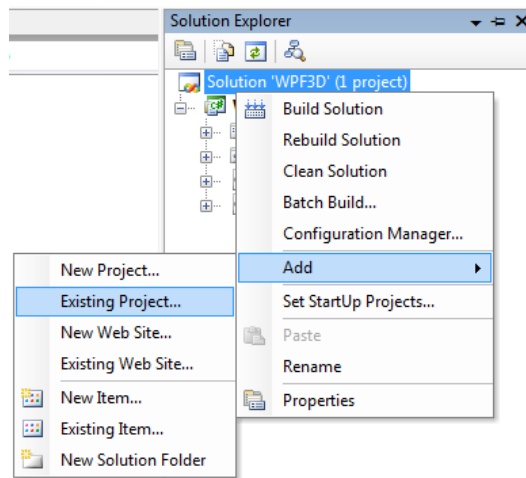
3D Tools for WPF nalezneme na adrese <http://www.codeplex.com/3DTools>, přímý odkaz na soubory je zde <http://www.codeplex.com/3DTools/Project/FileDownload.aspx?DownloadId=4662>.

Po rozbalení zde nalezneme ukázkové projekty (složka *Samples*) a zdrojové kódy (složka *3DTools*).

Použití

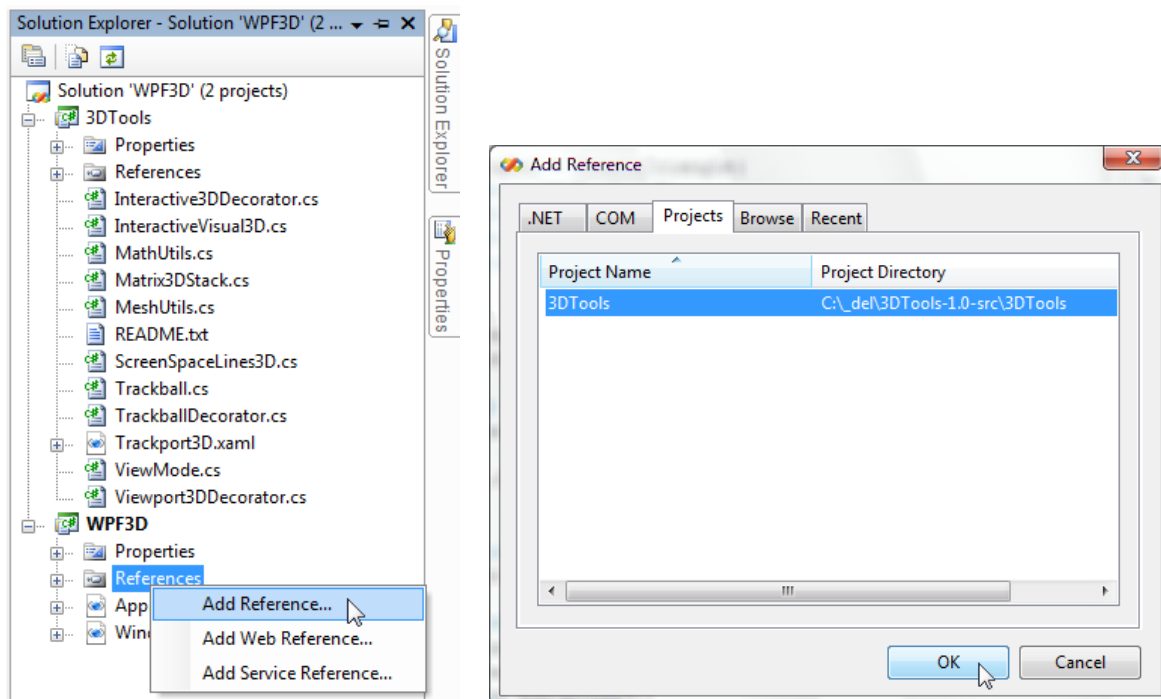
Vložíme *3D Tools for WPF* do našeho projektu:

pravým tlačítkem myši kliknout na *Solution > Add > Existing Project*



a vybereme `...\3DTools-1.0-src\3DTools\3DTools.csproj`

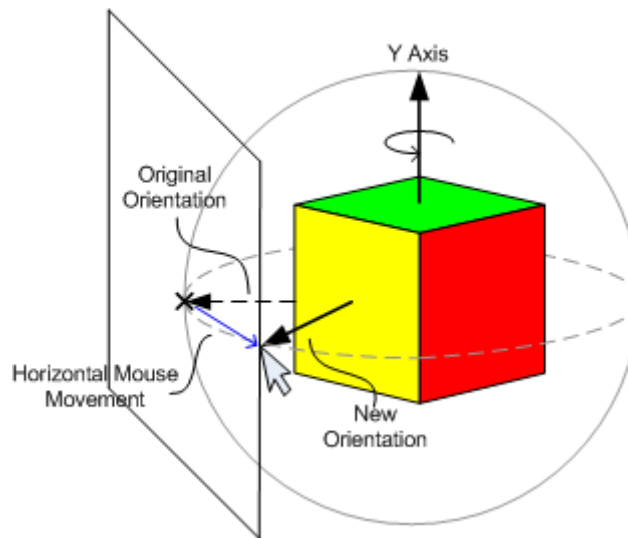
Nyní máme v našem solution dva projekty, *3DTools* a náš *WPF3D*. Abychom v našem projektu *WPF3D* mohli přistupovat k třídám z *3DTools*, vytvoříme ještě na tento projekt referenci.



Nyní pod namespace `_3DTools` máme k dispozici nástroje z *3D Tools for WPF*.

Trackball3D

Jedná se o první z užitečných funkcí *3D Tools for WPF*. Pomocí *Trackball3D* můžeme myší rotovat kameru ve *Viewport3D*, což navozuje dojem, že myší otáčíme celé prostředí. Jedná se o implementaci známou například z grafických 3D programů, kdy myší "chytíte" objekt a například táhnutím doprava ho tak otočíte kolem svislé osy.

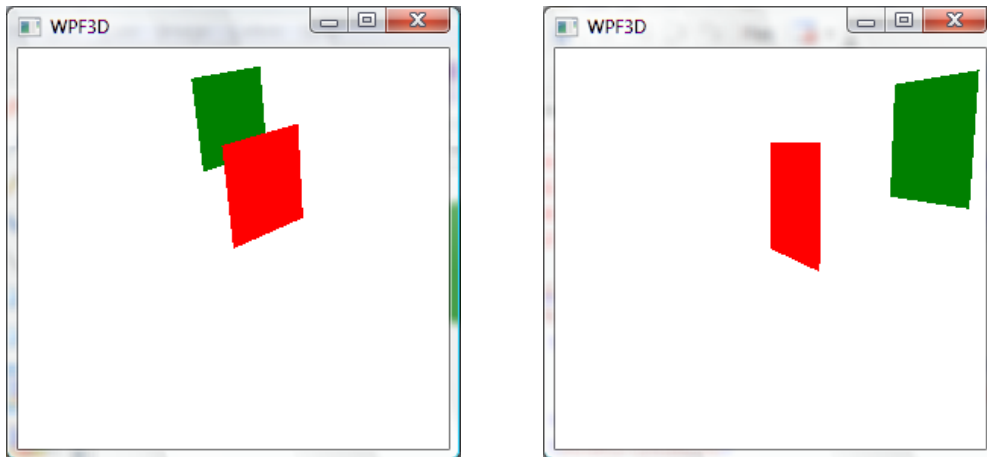


Použití *Trackball3D* v naší aplikaci.

```
<Window x:Class="WPF3D.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:Tools3D="clr-namespace:_3DTools;assembly=3DTools"
  Title="WPF3D" Height="300" Width="300"
  >
  <Tools3D:TrackballDecorator>
  <Viewport3D Name="myViewport3D">
  <Viewport3D.Camera>
  <PerspectiveCamera LookDirection="0,0,-1" Position="0,0,5"
    NearPlaneDistance="3" FarPlaneDistance="100" />
  </Viewport3D.Camera>
  <ModelVisual3D>
  <ModelVisual3D.Content>
  <DirectionalLight Color="White" Direction="0,0,-1" />
  </ModelVisual3D.Content>
  </ModelVisual3D>
  </Viewport3D>
  </Tools3D:TrackballDecorator>
</Window>
```

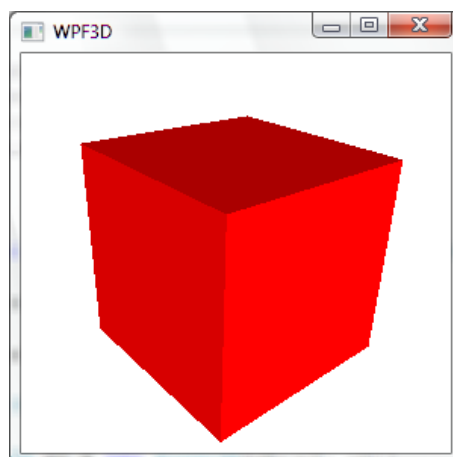
1. Namapovali jsme namespace *_3DTools* na element *Tools3D*
2. Vložili jsme *Viewport3D* mezi elementy *Tools3D:TrackballDecorator*

Pokud nyní potáhneme myší ve *Viewport3D*, otočí se scéna ve směru táhnutí.



Krychle

Krychle se skládá z šesti čtverců (4 boční, vrchní a spodní). Protože čtverec máme už naimplementovaný, vytvoření krychle bude jednoduché.



Window1.xaml

```
...
<ModelVisual3D>
  <ModelVisual3D.Content>
    <Model3DGroup>
      <AmbientLight Color="#AA0000" />
      <DirectionalLight Color="White" Direction="0.2,0.5,-1" />
    </Model3DGroup>
  </ModelVisual3D.Content>
</ModelVisual3D>
</Viewport3D>
...
```

Window1.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Media3D;
using System.Windows.Controls;

namespace WPF3D
{
    public partial class Window1 : System.Windows.Window
    {
        public Window1()
        {
            InitializeComponent();

            Model3DGroup models = new Model3DGroup();

            Point3D p0 = new Point3D(-1, -1, -1);
            Point3D p1 = new Point3D(1, -1, -1);
            Point3D p2 = new Point3D(1, -1, 1);
            Point3D p3 = new Point3D(-1, -1, 1);
            Point3D p4 = new Point3D(-1, 1, -1);
            Point3D p5 = new Point3D(1, 1, -1);
            Point3D p6 = new Point3D(1, 1, 1);
            Point3D p7 = new Point3D(-1, 1, 1);

            models.Children.Add(
                CreateRectangle(p3, p2, p6, p7, Brushes.Red)); // predni
            models.Children.Add(
                CreateRectangle(p2, p1, p5, p6, Brushes.Red)); // pravy
            models.Children.Add(
                CreateRectangle(p1, p0, p4, p5, Brushes.Red)); // zadni
            models.Children.Add(
                CreateRectangle(p0, p3, p7, p4, Brushes.Red)); // levy
            models.Children.Add(
                CreateRectangle(p7, p6, p5, p4, Brushes.Red)); // vrchni
            models.Children.Add(
                CreateRectangle(p2, p3, p0, p1, Brushes.Red)); // spodni

            ModelVisual3D visual = new ModelVisual3D();
            visual.Content = models;

            myViewport3D.Children.Add(visual);
        }

        public GeometryModel3D CreateRectangle(
            Point3D point1, Point3D point2,
            Point3D point3, Point3D point4,
            Brush brush)
        {
            ...
        }
    }
}
```

1. Nadeklarovali jsme vrcholy krychle p_0, p_1, \dots, p_7
2. Vytvořili jsme čtverce, které tvoří strany krychle
3. Nakonec jsme změnili osvětlení prostoru, aby byla krychle osvětlena ze všech stran

Závěr

Trojrozměrná grafika ve WPF není určena pro vývoj 3D her, slouží spíše pro vytvoření zajímavých trojrozměrných efektů nebo opravdu jednoduchých 3D aplikací. Pro pokročilejší práci s 3D grafikou máme k dispozici Direct3D, které můžeme ve WPF také použít.

V dalším díle věnovaném 3D grafice budeme pokračovat v implementaci kostky a ukážeme si použití dalších funkcí z *3D Tools for WPF*.

Více informací o WPF naleznete na www.unitedstatesof.net/wpf

Aleš Šturala, 17. 2. 2007

<http://hidentity.org/hid/CZ123456>

Některé obrázky v tomto tutoriálu byly použity ze stránek <http://blogs.msdn.com/danlehen>, <http://www.wikipedia.org> a <http://msdn2.microsoft.com>