ikdoeict.be

C# Programming Techniques
Advanced Applied Programming
2017 -2018

Katja Verbeeck
Joris Maervoet
Tim Vermeulen

3D in WPF

Achtergrondinfo: enkele frameworks voor grafische toepassingen

Microsoft's Graphics Display Interface (GDI 1990)
Java.awt.Graphics2D

GDI+ (Windows): 2D, niet erg performant, eenvoudig.

WPF (Windows Presentation Foundation)
2/3D, performanter en krachtiger

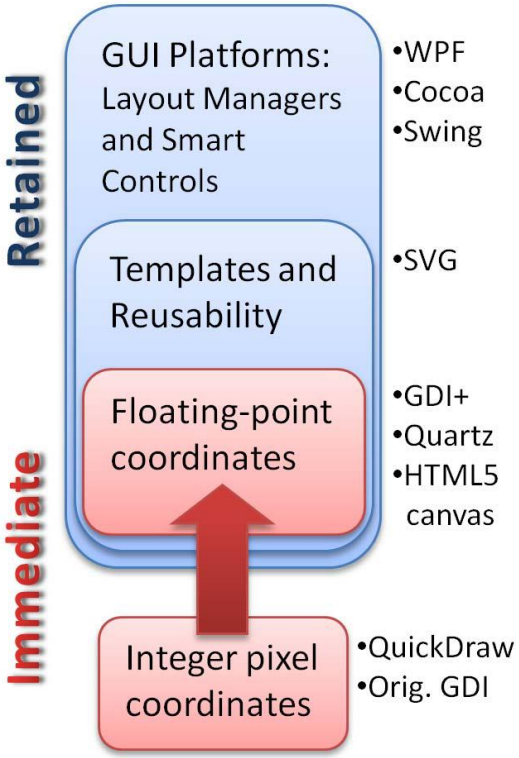
OpenGL (Silicon Graphics, Open Graphics Library …)

- Vrij complex, zeer krachtig & performant, low level
- Enkel graphics

Direct3D from DirectX (Microsoft)

High Level framework voor gaming

- Vb: XNA, Unreal Development Kit, Unity
- Gebruik maken van OpenGL, DirectX, …



Evolutie in abstractieniveau in 2D frameworks - from [Computer Graphics, Principles and Practices,3rd ed.2014 Addison-Wesley]

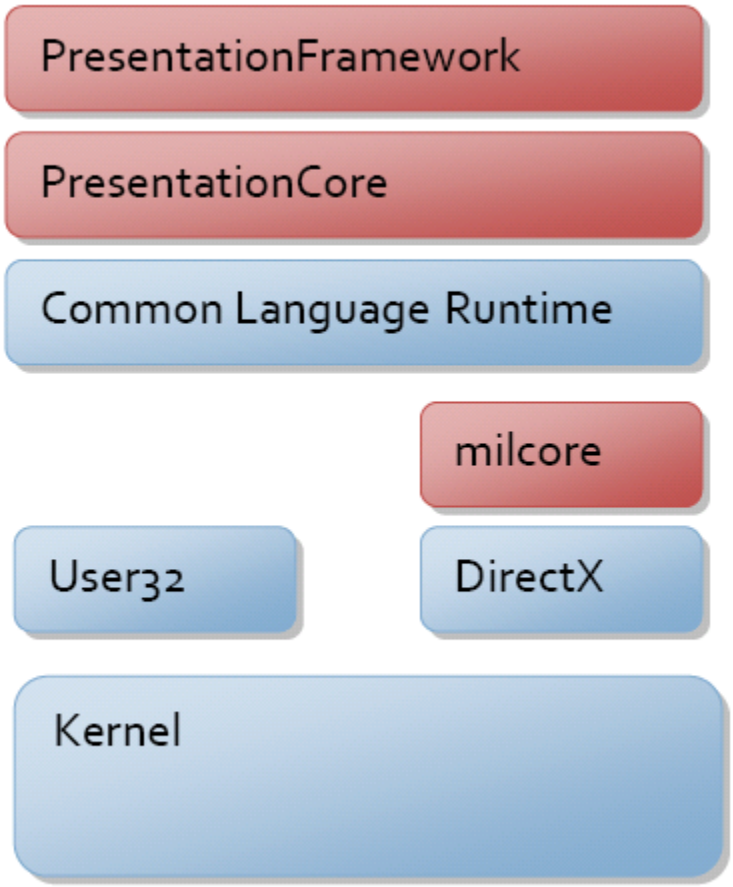
WPF vs GDI/GDI+

- WPF is not a wrapper for GDI/GDI+
- WPF is a replacement, a separate layer that works through DirectX
- WPF 's goal is to off-load as much of the work as possible on the video card so that complex graphics are render bound (limited by GPU) rather then processor bound (limited by CPU)
- Hardware Acceleration: CPU has less work and you can take advantage when newer video cards become available
- Resolution Independence: works with the system DPI setting

WPF: a higher Level API

- Web like layout model: flexible flow layout based on content
- Rich drawing model with primitive shapes
- Rich text model
- Animation as a first class programming concept
- Support for audio and video media
- Declarative User Interface: XAML – user interface is completely separated from code
- Interoperability with windows forms

Architecture of WPF

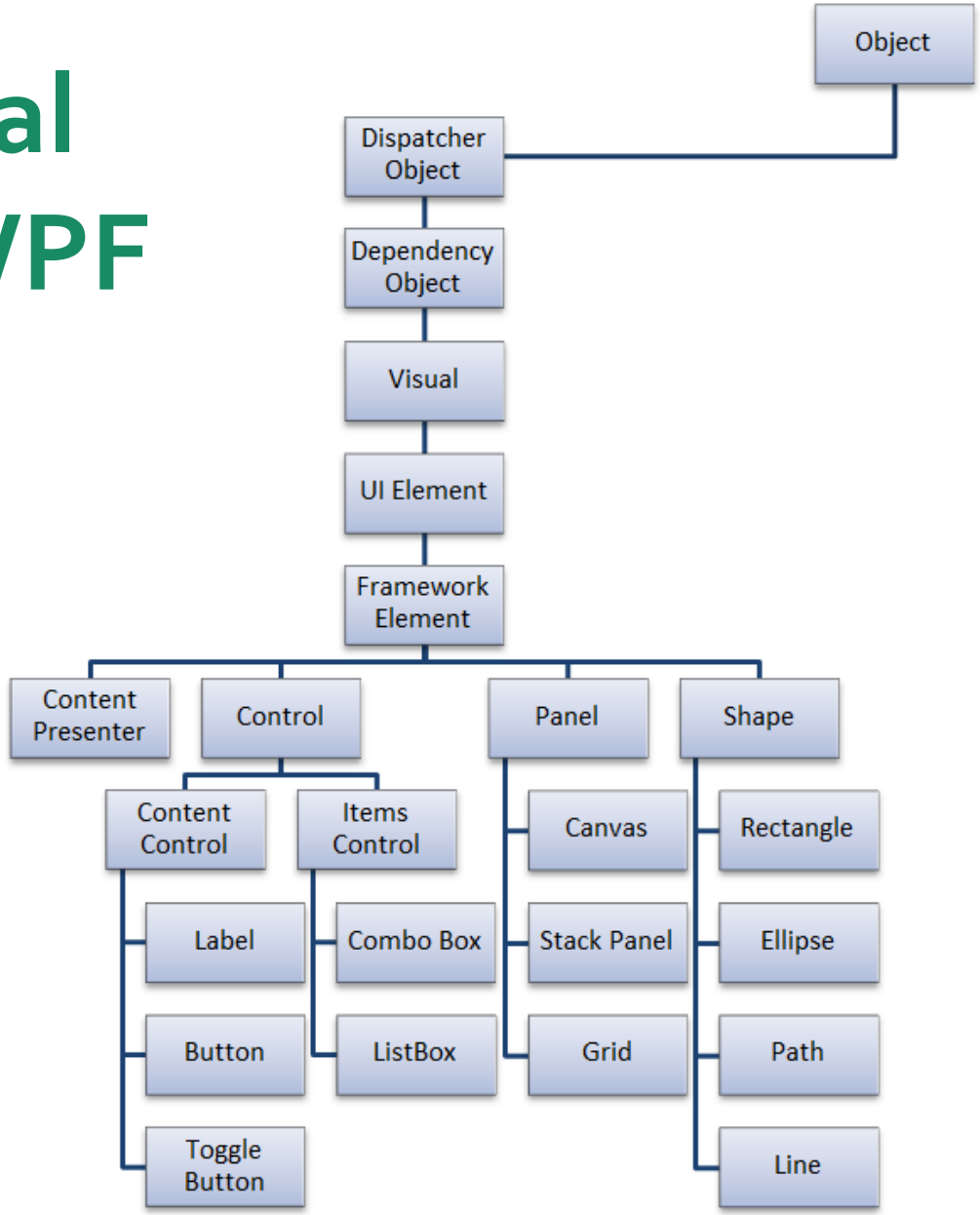


The Managed WPF API

Media Integration Layer
Unmanaged translation

Direct3D renders all the drawing in WPF

Fundamental Classes of WPF



In WPF visual items in a form are called elements (in winforms controls).
In WPF an element is a control only when they can receive focus and interact with the user

Markup and Code-Behind

Extensible Application Markup Language (XAML)

- XML-based markup language
- to implement the appearance of an application
- create windows, dialog boxes, pages, and user controls, and to fill them with controls, shapes, and graphics

C# code

- Implement the behavior
- Implement the functionality that responds to user interactions
- handling events and calling business logic and data access logic in response

XAML

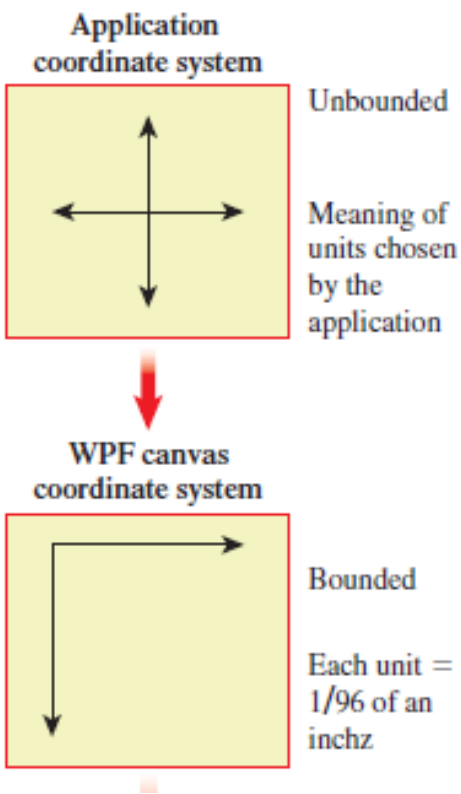
XAML in WPF http://msdn.microsoft.com/en-us/library/ms747122.aspx

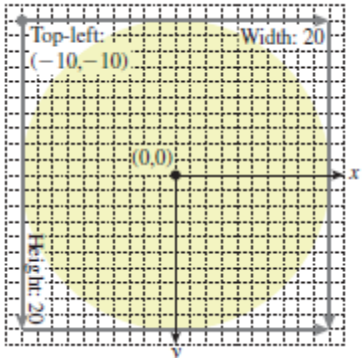
- Every element in a XAML doc maps to an instance of a .NET class
  e.g. Element <Button> instructs WPF to create a Button object
- Elements can be nested
  e.g. a Button element can reside in a Grid element.
- Properties can be set using attributes : Title, Height, Width, Name ….

```
<Window x:Class="EightBall.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Eight Ball Answer" Height="328" Width="412" >

    <Grid Name = "grid1">
        <Grid.Background>
        </Grid.Background>
    </Grid>
</Window>
```

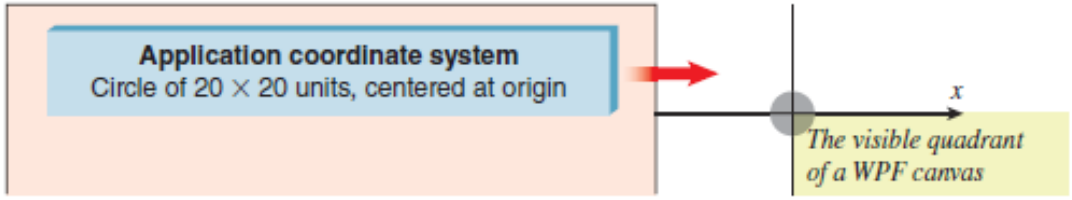
complex properties use nested tags

Remark: Application (or World) Coordinates vs. WPF Canvas Coordinates






```
<Canvas ... >
  <Ellipse
    Canvas.Left="-10.0" Canvas.Top="-10.0"
    Width="20.0" Height="20.0"
    Fill="lightgray" />
</Canvas>
```



Remark: transformations in WPF

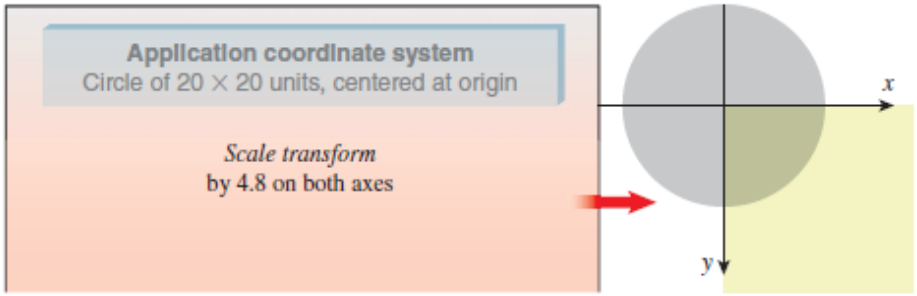


Figure 2.13: Schematic view of our application now enhanced with a scale transform.

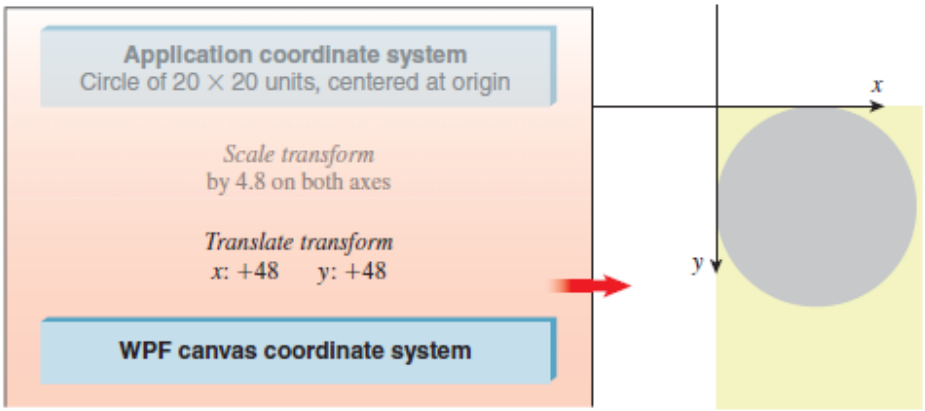
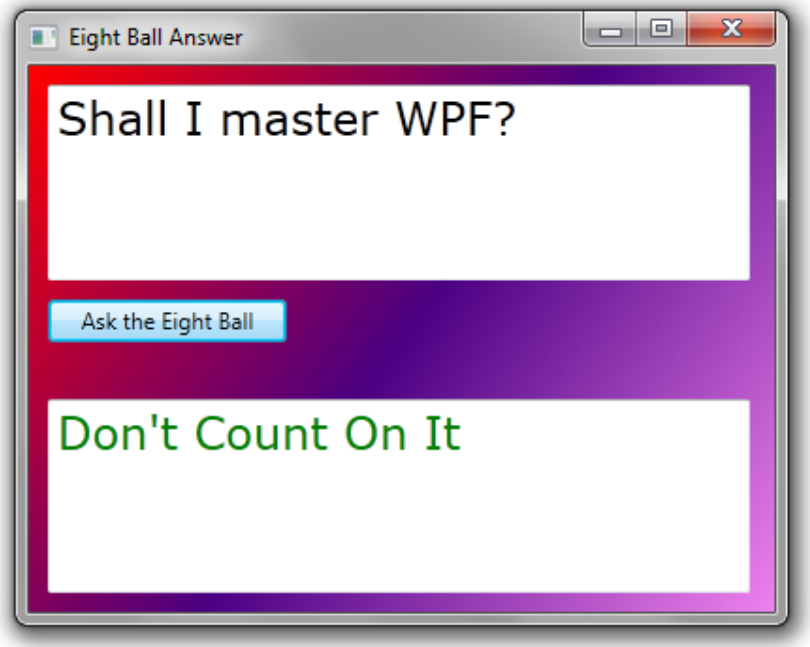


Figure 2.14: Schematic view of our application now enhanced with a two-step display-transform sequence (scale and translate).

```
<Canvas ... >
  <!- THE SCENE ->
  <Ellipse ... />

  <!- THE DISPLAY TRANSFORM ->
  <Canvas.RenderTransform>
    <TransformGroup>
     <ScaleTransform  ScaleX="4.8"  ScaleY="4.8" ... />
     <TranslateTransform X="48" Y="48" />
    </TransformGroup>
  </Canvas.RenderTransform>
</Canvas>
```

XAML the Magic 8-ball

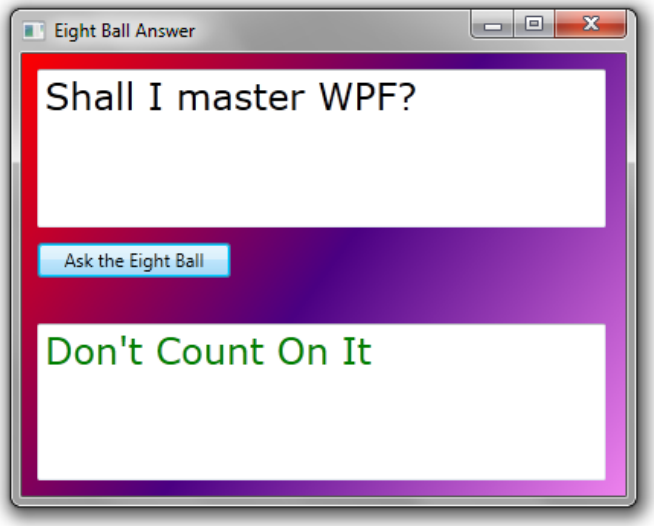


XAML the Magic 8-ball

Core WPF namespace - default

XAML namespace – x:

```
<Window x:Class="EightBall.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Eight Ball Answer" Height="328" Width="412" >
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.Background>
      <LinearGradientBrush>
        <LinearGradientBrush.GradientStops>
          <GradientStop Offset="0.00"  Color="Red" />
          <GradientStop Offset="0.50" Color="Indigo" />
          <GradientStop Offset="1.00" Color="Violet" />
        </LinearGradientBrush.GradientStops>
      </LinearGradientBrush>
    </Grid.Background>
```



XAML the Magic 8-ball vervolg

```
<TextBox VerticalAlignment="Stretch" HorizontalAlignment="Stretch"
Margin="10,10,13,10" Name="txtQuestion"
   TextWrapping="Wrap" FontFamily="Verdana" FontSize="24"
   Grid.Row="0" >
   [Place question here.]
</TextBox>
```

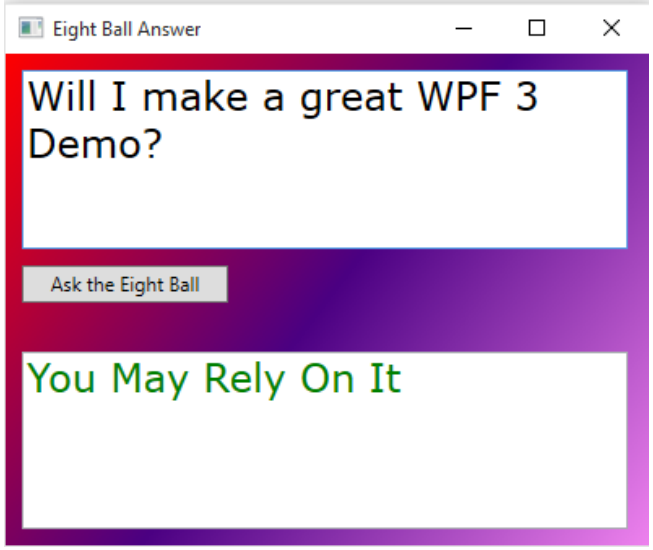
Content Property

```
<Button VerticalAlignment="Top" HorizontalAlignment="Left" Margin="10,0,0,20"
Width="127" Height="23" Name="cmdAnswer"
   Click="cmdAnswer_Click"
   Grid.Row="1">
      Ask the Eight Ball
</Button>
```

Attaching an eventhandler

```
<TextBox VerticalAlignment="Stretch" HorizontalAlignment="Stretch"
Margin="10,10,13,10" Name="txtAnswer"
   TextWrapping="Wrap" IsReadOnly="True" FontFamily="Verdana" FontSize="24"
Foreground="Green"
   Grid.Row="2">
      [Answer will appear here.]
</TextBox>
</Grid>
</Window>
```

Code behind




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

        private void cmdAnswer_Click(object sender, RoutedEventArgs e)
        {
            // Dramatic delay...
            this.Cursor = Cursors.Wait;
            System.Threading.Thread.Sleep(TimeSpan.FromSeconds(5));

            AnswerGenerator generator = new AnswerGenerator();
            txtAnswer.Text = generator.GetRandomAnswer(txtQuestion.Text);
            this.Cursor = null;
        }
    }
```

Equivalent C# Code

```
<StackPanel Margin = "10">
 <Rectangle Name = "MyRectangle"
  Width = "100"
  Height = "100"
  Fill = "Blue">
 </Rectangle>
</StackPanel>
```

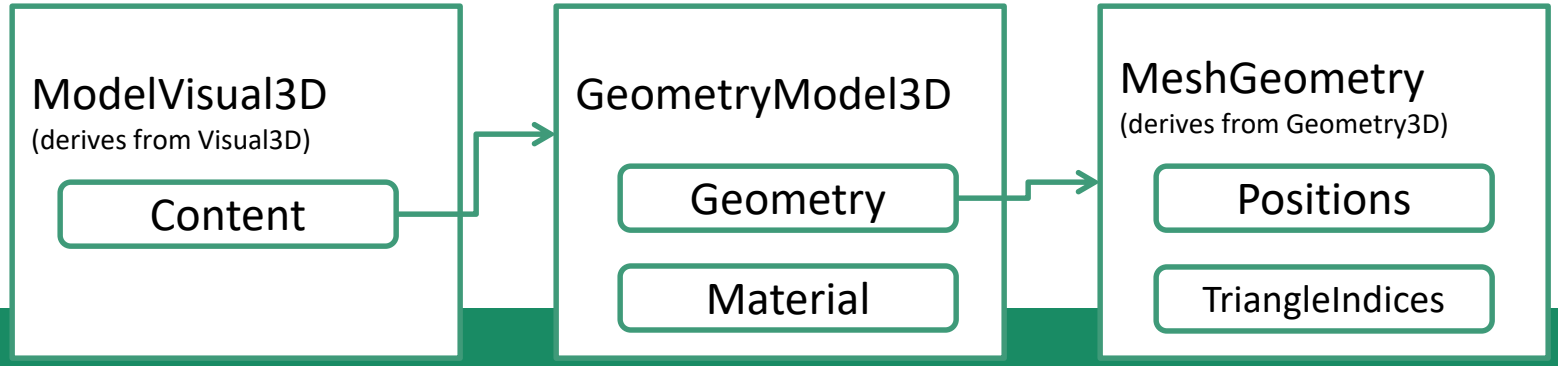
```
StackPanel myPanel = new StackPanel();
myPanel.Margin = new Thickness(10);
Rectangle myRectangle = new Rectangle();
myRectangle.Name = "myRectangle";
this.RegisterName(myRectangle.Name, myRectangle);
myRectangle.Width = 100;
myRectangle.Height = 100;
myRectangle.Fill = Brushes.Blue;
myPanel.Children.Add(myRectangle);
this.Content = myPanel;
```

XAML for 3D

A 3D drawing in WPF involves 4 ingredients:

- A viewport
- 3D object(s)
- A light source that illuminates part or all of the scene
- A camera

A viewport3D object is a container for 3D-content, i.e. Visual3Dobjects. The content of Visual3Dobjects is given by a GeometryModel3D. The Geometry of such a model is given by a Geometryobject such as a MeshGeometry



XAML For 3D triangle

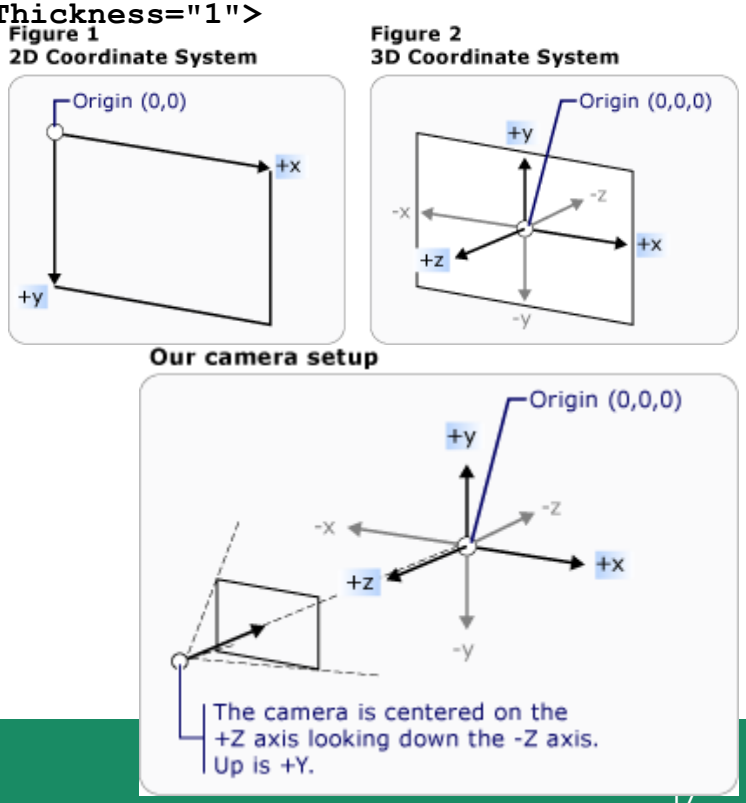
Window

```
<Window x:Class="DrawingIn3D.OneTriangleMesh"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="OneTriangleMesh" Height="300" Width="300" >

    <Grid Margin="5">
      <Border BorderBrush="Yellow" BorderThickness="1">
      <Viewport3D>
      <Viewport3D.Camera>
          <PerspectiveCamera
           Position="-2,2,2"
           LookDirection="2,-2,-2"
           UpDirection="0,1,0"
           />
        </Viewport3D.Camera>
        <ModelVisual3D>
          <ModelVisual3D.Content>
            <DirectionalLight
             Color="White"
             Direction="-1,-1,-1" />
          </ModelVisual3D.Content>
        </ModelVisual3D>
```

Viewport

Camera

Light



XAML for 3D Triangle

Points

Sequence of points

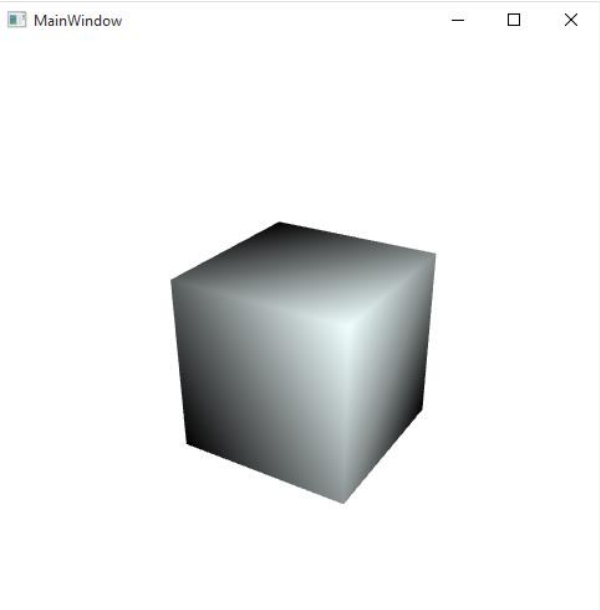
Material

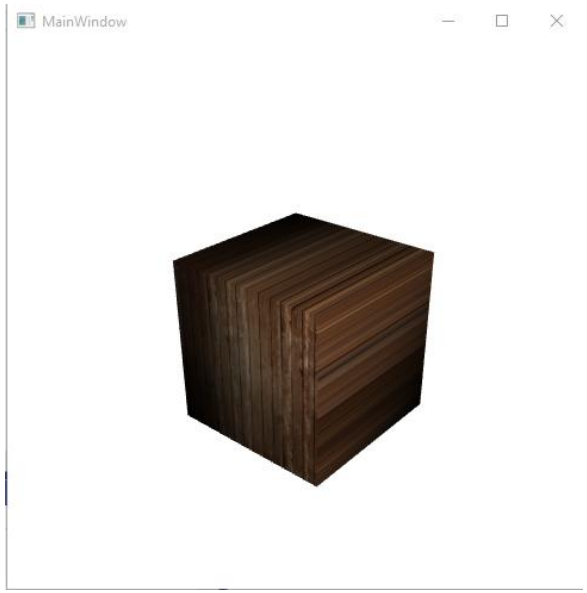
Comments

```
<ModelVisual3D>
          <ModelVisual3D.Content>
            <GeometryModel3D>
                    <GeometryModel3D.Geometry>

                        <MeshGeometry3D Positions="-1,0,0 0,1,0 1,0,0"
                                        TriangleIndices="0,2,1"
                        />

                    </GeometryModel3D.Geometry>

                    <GeometryModel3D.Material>
                       <DiffuseMaterial Brush="Yellow" />
                    </GeometryModel3D.Material>

                <!--  <GeometryModel3D.BackMaterial>
                  <DiffuseMaterial Brush="Green" />
                </GeometryModel3D.BackMaterial>-->

                 </GeometryModel3D>
               </ModelVisual3D.Content>
    </ModelVisual3D>


     </Viewport3D>
   </Border>
  </Grid>
 </Window>
```

A cube with a texture

demo





WPF Animation

In WPF, you animate objects by applying animation to their individual properties. For example, to make a framework element grow, you animate its Width and Height properties. To make an object fade from view, you animate its Opacity property.

```
<DoubleAnimation From="1.0" To="0.0" Duration="0:0:5"
AutoReverse="True" RepeatBehavior="Forever"/>
```

Or you create a Storyboard

```
<Storyboard> <DoubleAnimation Storyboard.TargetName="MyRectangle"
Storyboard.TargetProperty="Opacity" From="1.0" To="0.0"
Duration="0:0:5" AutoReverse="True" RepeatBehavior="Forever" />
</Storyboard>
```

You can use Eventtriggers, transforms, pathAnimation,databinding, …

(e.g move the camera position and its updirection)

Nuttige Links

https://msdn.microsoft.com/en-us/library/ms754130.aspx

http://wpftutorial.net/Home.html

http://www.i-programmer.info/projects/38-windows/273-easy-3d.html