3DVBVIEW - DX11Engine Version 1.1

(From build version 1054. As of April 2018)

3D graphics is, similar to virtual reality, actually a paradoxical term, as the result always is a 2-dimensional image. 3D does not only describe the spatial impression the image conveys, that can be achieved with a perspective drawing too, but rather the way how the image is created and last but not least the ability to interactively change the point of view to look at an object from all sides as if it stood in front of the observer.

A mathematical description of the 3-dimensional space, the objects to be displayed, lights, materials and a camera is taken to calculate a 2-dimensional image.

The basic process of creating an image from a description of the image content is called rendering. This process requires the execution of a sequence of work steps, this sequence is referred to as a render pipeline, the work steps as stages of the pipeline.

There are basically two types of render pipelines a 'fixed' one and a 'programmable' one. The fixed pipeline has predefined stages that, more or less, can be customized. By contrast, a programmable pipeline can be comprehensively adapted to different needs. DirectX 11 uses a programmable pipeline.

What at first glance only offers advantages, turns out to be the first and in most cases the biggest obstacle when it comes to practical implementation. The render pipeline not only can be programmed, it explicitly **must** be programmed.

The working mode of each stage and the resources and data formats used must be determinated Some stages, such as the vertex and pixel shaders even require external code to describe their function.

The variety of possible settings, data formats and applied concepts, as well as the mathematics of 3D space make getting into the matter difficult and often tedious.

This is where the concept of DX11Engine comes in. The modular design and the implementation of two levels of abstraction allow the use at different levels of difficulty. Beginners can achieve good results with just a few lines of code, but advanced user do not have to miss extensibility or flexibility.

The top level base object is a instance of the Scene class, this class provides a predefined render environment and behaves much like a fixed pipeline. The Scene class combines several core objects, which abstracts functions of the underlying render pipeline.

The use of DX11Engine according to implementation efforts and difficulty can be summarized:

- Using an existing Scene class
- Implementing a new Scene class based on existing core objects
- Changing existing or creating new core objects

Getting Started

To use the *DX11Engine* objects and methods without having to explicitly specify *DX11Engine*, the namespace is imported. Therefore, a corresponding *Imports* statement is added at the beginning of the file. Since some of the data types used are declared by *SlimDX*, an *Imports* statement should also be added for *SlimDX*.

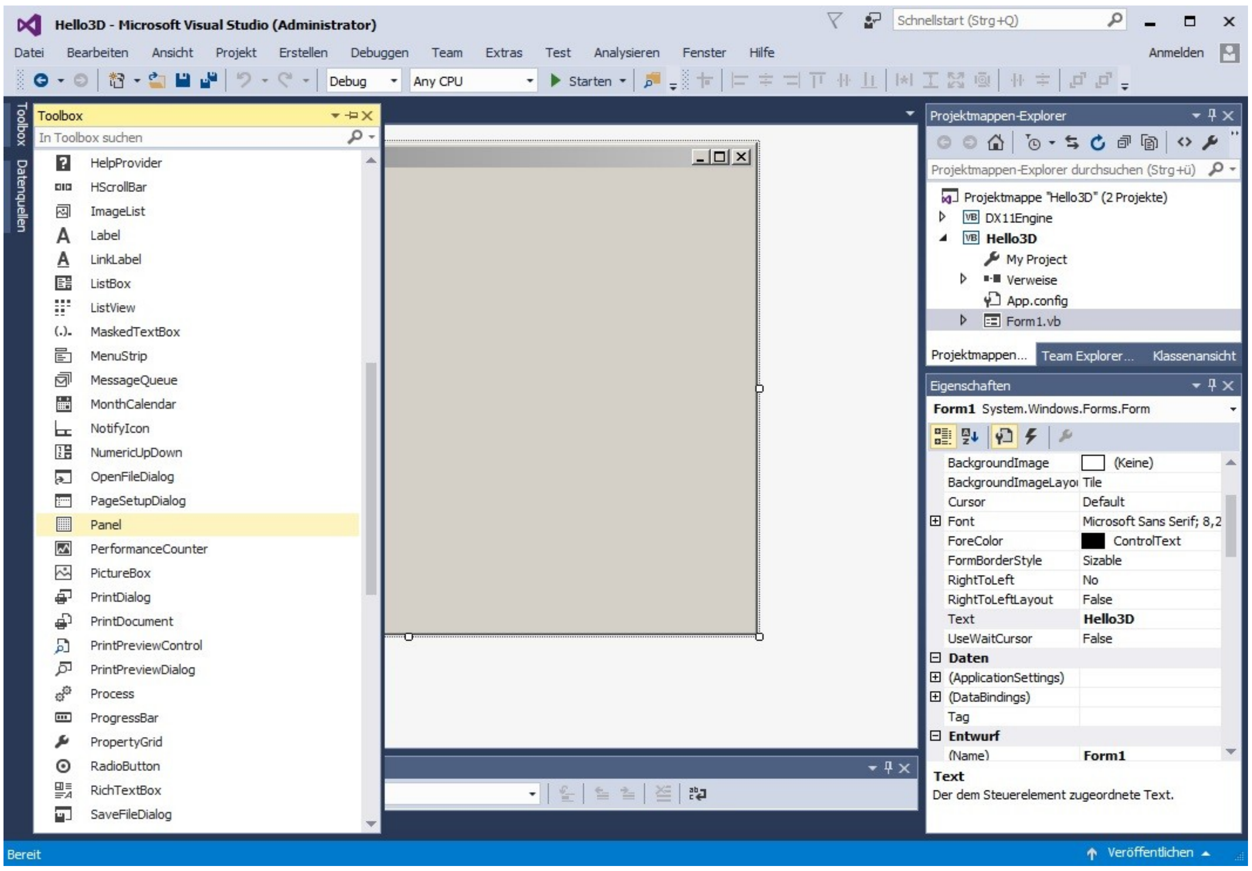
```
Imports DX11Engine
Imports SlimDX

Public Class Form1

   ...

End Class
```

To simplify integration with a user interface, *DX11Engine* uses a *panel* control to display the render result. By example the panel is named *RenderPanel,*

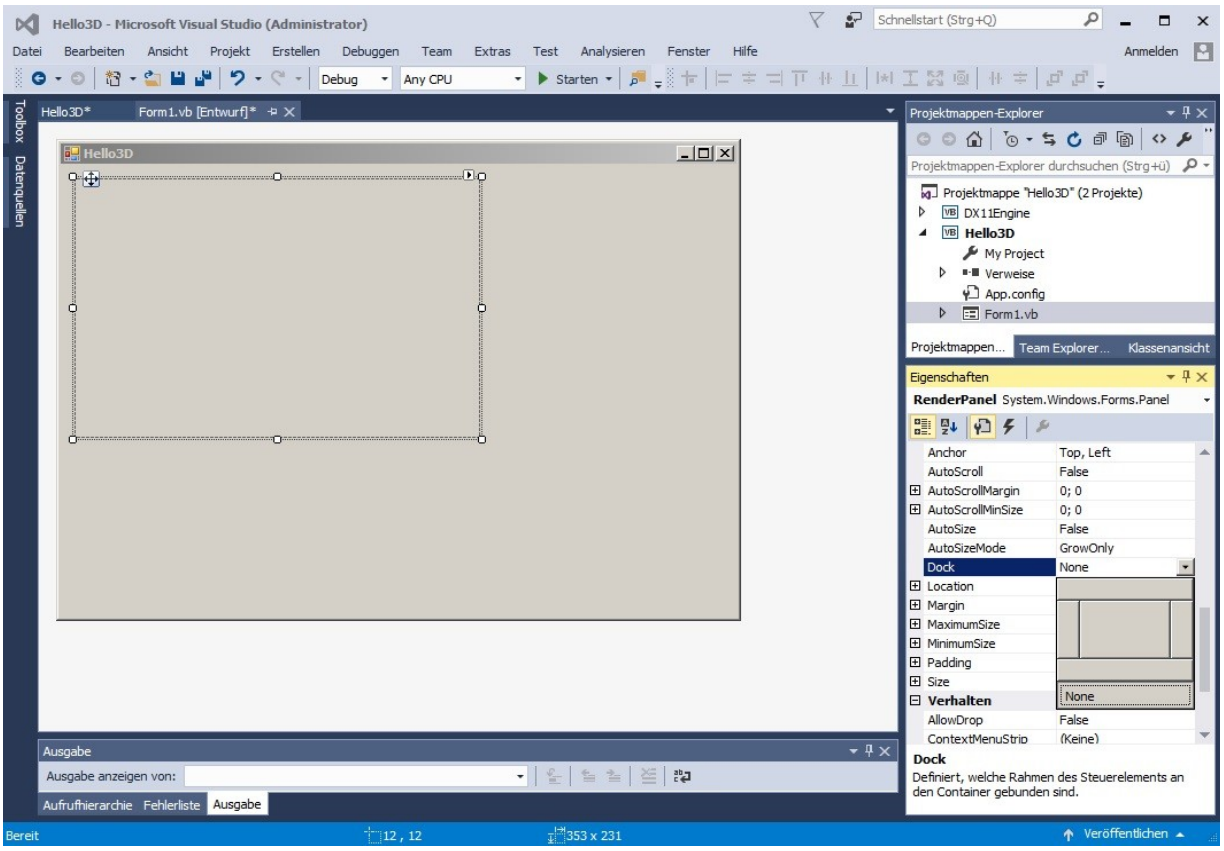


Form1 System.Windows.Forms.Form

BackgroundImage	(Keine)
BackgroundImageLayou	Tile
Cursor	Default
⊞ Font	Microsoft Sans Serif; 8,2
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	**Hello3D**
UseWaitCursor	False
⊟ **Daten**	
⊞ (ApplicationSettings)	
⊞ (DataBindings)	
Tag	
⊟ **Entwurf**	
(Name)	**Form1**

Text
Der dem Steuerelement zugeordnete Text.

and its *DockStyle* is set to *Fill*.



Eigenschaften: **RenderPanel** System.Windows.Forms.Panel

Anchor	Top, Left
AutoScroll	False
AutoScrollMargin	0; 0
AutoScrollMinSize	0; 0
AutoSize	False
AutoSizeMode	GrowOnly
Dock	None
Location	
Margin	
MaximumSize	
MinimumSize	
Padding	
Size	
Verhalten	
AllowDrop	False
ContextMenuStrip	(Keine)

Dock
Definiert, welche Rahmen des Steuerelements an den Container gebunden sind.

The base objects of *DX11Engine* are *DX11Device* and *Scene*, where the first represents the physical adapter, the graphics card and its driver, and the second represents the render environment, which includes the render pipeline and objects and methods that control the scene. For the object instances two private objects are declared in *Form1*. (Within *DX11Engine* code, a leading underscore '_' is used to mark private objects and variables)

```
Imports DX11Engine
Imports SlimDX

Public Class Form1

    Private  _DX11Device As DX11Device
    Private  _Scene As Scene

    . . .


End Class
```

For the initialization of *DX11Engine,* two new methods are created, *InitializeDevice* and *InitializeScene*. *InitializeDevice* is kept short, creating the device is an essential step, if the creation fails, there is nothing else to do here. *InitializeScene* will later be expanded by adding objects and settings that make up a real scene, for now it's comparable to empty space.

```
Private Function InitializeDevice() As Boolean

    _DX11Device = New DX11Device({FeatureLevel.Level_11_0,
                                  FeatureLevel.Level_10_1},
                                  DeviceCreationFlags.None)

    If _DX11Device.LastError <> EngineResult.OK Then

      If _DX11Device IsNot Nothing Then
        _DX11Device.Dispose()
        _DX11Device = Nothing
      End If

      Return False

    End If

    Return True

End Function
```

```
Private Function InitializeScene() As Boolean


    _Scene = New Scene(_DX11Device, RenderPanel)


    If _Scene.LastError <> EngineResult.OK Then
      Return False
    End If


    ***  Add code to initialize Scene dependencies here ***


    Return True


End Function
```

Both methods are put together in the *Form1_Load* event handler, if one of the methods fails, an error message will be shown.

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles Me.Load

    If Not InitializeDevice() Then
        MsgBox("The creation of the Direct3D Device failed.")
        Application.Exit()
        End
    End If

    If Not InitializeScene() Then
        MsgBox("The creation of the Direct3D Scene failed.")
        Application.Exit()
        End
    End If

End Sub
```

DX11Engine uses a large number of objects that refer to unmanaged resources, these resources must be released when the application is terminated. *DX11Engine* has internal methods to do this automatically, but it is necessary to ensure that the *_DX11Device* instance gets destroyed. The *Form1_FormClosing* event handler is used for this task.

```
Private Sub Form1_FormClosing(sender As Object, e As FormClosingEventArgs)
                                        Handles Me.FormClosing

    _DX11Device.Dispose()

End Sub
```

The size of the *RenderPanel* may change and such a change will affect the internals of the *_Scene* object. To handle this event one more method is created.

An *AddHandler* statement is added to the *InitalizeScene* method

```
Private Function InitializeScene() As Boolean

      _Scene = New Scene(_DX11Device, RenderPanel)

      If _Scene.LastError <> EngineResult.OK Then
         Return False
      End If

      ***  Add code to initialize Scene dependencies here ***

      AddHandler RenderPanel.Resize, AddressOf ResizeScene

      Return True

 End Function
```

and a *SceneResize* method is added to *Form1*.

```
Private Sub ResizeScene(s As Object, e As EventArgs)

      _Scene.OnResize(RenderPanel.Width, RenderPanel.Height)

 End Sub
```

The basic implementation is now complete. *Form1* now includes a Direct3D viewport that can be addressed using the properties and methods of the *_Scene* object.