3DVBVIEW - DX11Engine Version 1.1

(ab Build-Version 1054.Stand April 2018)

3D-Grafik ist ähnlich wie Virtual Reality eigentlich ein paradoxer Begriff, da das Ergebnis stets ein 2 dimensionales Bild ist. 3D beschreibt hier nicht den räumlichen Eindruck den das Bild vermittelt, das kann eine perspektivische Zeichnung auch, sondern vielmehr die Art und Weise wie das Bild entsteht und nicht zuletzt die Möglichkeit es interaktiv verändern und einen Gegenstand von allen Seiten betrachten zu können als stünde dieser real vor dem Betrachter.

Hierzu werden der 3-dimensionale Raum, die darzustellenden Objekte, Licht, Materialien und eine Kamera mathematisch beschrieben und daraus das 2-dimensionale Abbild des Raumes berechnet.

Den grundlegenden Vorgang ein Bild aus einer Beschreibung des Bildinhalts zu erzeugen bezeichnet man als Rendern. Dieser Vorgang erfordert das Ausführen einer Abfolge von Arbeitsschritten, diese Abfolge bezeichnet man als Render-Pipeline, die Arbeitsschritte als Stufen (stages).

Man unterscheidet zwischen zwei Arten der Render-Pipeline, bei der 'fixed pipeline' sind die einzelnen Arbeitsschritte vordefiniert und nur geringfügig beeinflussbar, eine 'programmable pipeline' hingegen kann umfassend an unterschiedliche Bedürfnisse angepasst werden. DirectX 11 verwendet eine programmierbare Pipeline.

Was auf den ersten Blick nur Vorteile bietet, stellt sich bei näherer Betrachtung als das erste und oft auch größte Hindernis bei der praktischen Umsetzung heraus. Die Render-Pipeline kann nicht nur weitgehend frei programmiert werden, sie **muss** explizit programmiert werden.

Für jede einzelne Stufe muss die Arbeitsweise, die von ihr verwendeten Ressourcen und deren Format festgelegt werden. Besondere Stufen wie z.B. die Vertex- und Pixelshader benötigen externen Code um ihre Funktion zu beschreiben.

Die Vielzahl an möglichen Einstellungen, verwendeten Datenformaten und angewandten Konzepten, wie auch die grundlegende Mathematik des 3D-Raumes machen den Einstieg in die Materie schwierig und oft auch langwierig.

Hier setzt das Konzept von DX11Engine an, durch den modularen Aufbau und die Implementierung von zwei Abstraktionsebenen lässt sich die Verwendung in drei Schwierigkeitsgrade unterteilen, so dass Einsteiger mit nur wenigen Befehlen zum Ziel kommen, Fortgeschrittene aber nicht auf Flexibilität und Erweiterbarkeit verzichten müssen.

Kernstück der obersten Ebene ist die Scene-Klasse, diese Klasse enthält eine vordefinierte Render-Umgebung und verhält sich ähnlich wie eine 'fixed pipeline'. Die Scene-Klasse fasst mehrere Stammobjekte zusammen, die ihrerseits Funktionen der Render-Pipeline abstrahieren.

Die Verwendung von DX11Engine nach Implementierungsaufwand und -schwierigkeit kann wie folgt zusammengefasst werden:

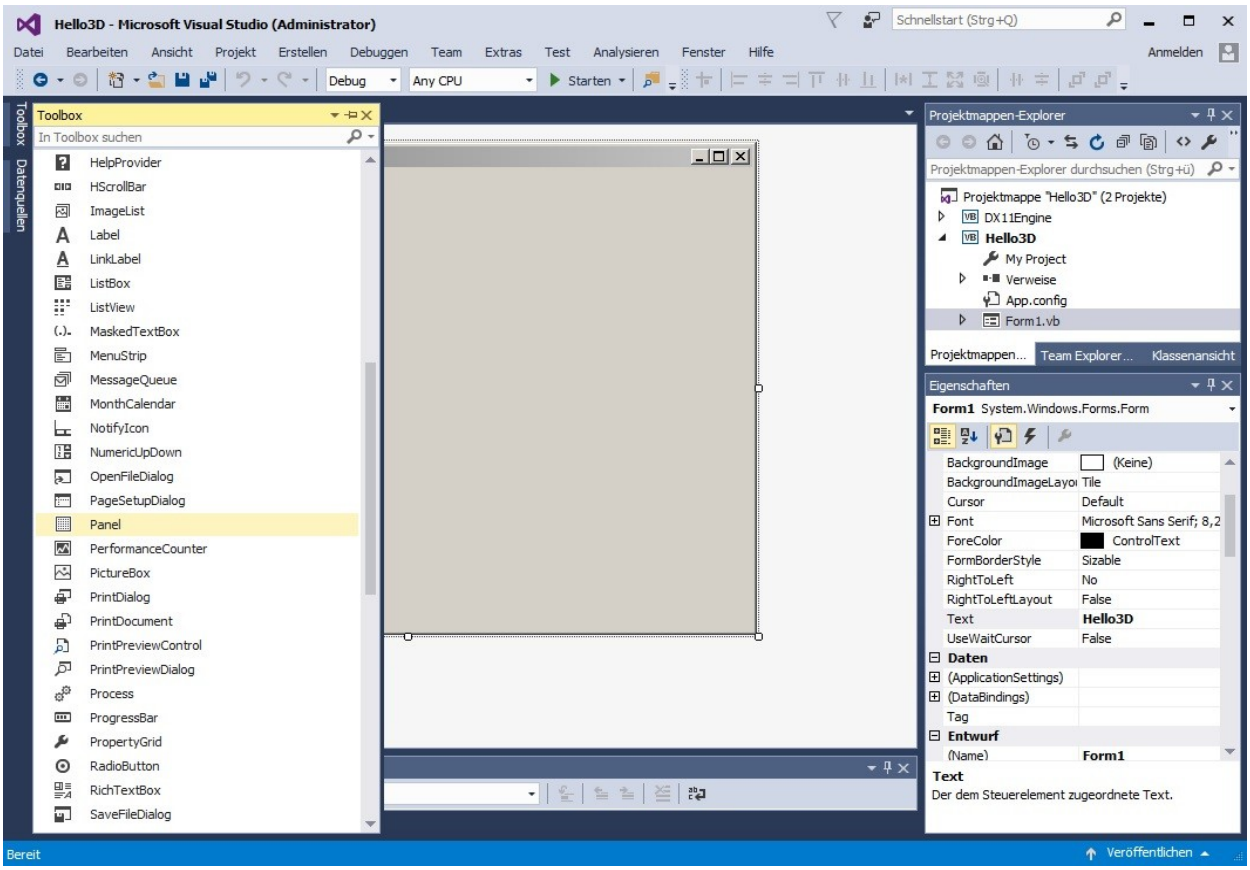
- Verwendung einer vorhandenen Scene-Klasse
- Implementierung einer eigenen Scene-Klasse auf Basis vorhandener Stammobjekte
- Ändern vorhandener oder Erstellen neuer Stammobjekte

Erste Schritte

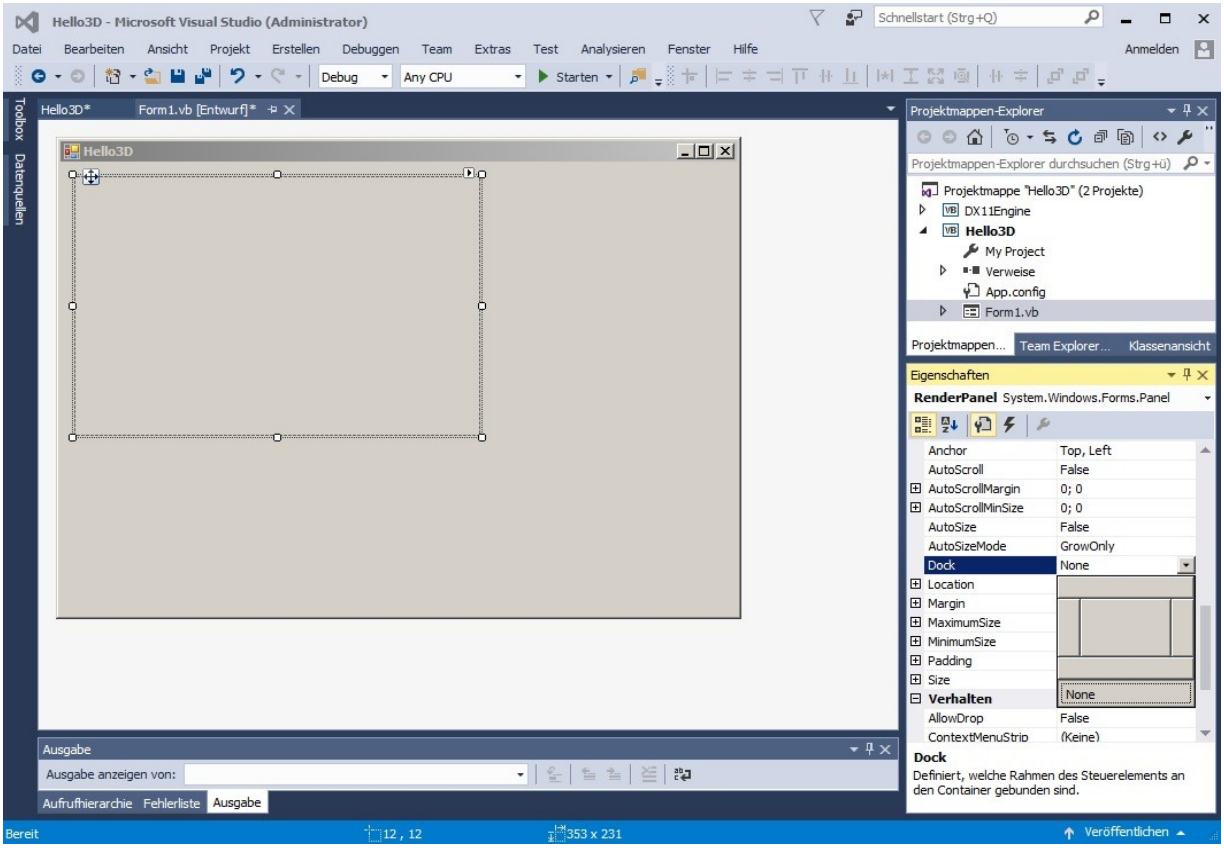
Um die Objekte und Methoden von *DX11Engine* verwenden zu können, ohne immer explizit *DX11Engine* angeben zu müssen, wird der Namespace importiert. Am Anfang der Datei wird deshalb eine entsprechende *Imports*-Anweisung hinzugefügt. Da einige der verwendeten Datentypen durch *SlimDX* deklariert werden sollte zusätzlich auch noch eine *Imports*-Anweisung für *SlimDX* hinzugefügt werden.

```
Imports DX11Engine
Imports SlimDX

Public Class Form1

    . . .

End Class
```

Um die Integration in eine Benutzeroberfläche zu vereinfachen verwendet *DX11Engine* ein *Panel*-Steuerelement um das Render-Ergebnis anzuzeigen. Das Panel wird exemplarisch mit *RenderPanel* bezeichnet



und der *DockStyle* auf *Fill* gesetzt.



Die Basisobjekte von *DX11Engine* sind *DX11Device* und *Scene*, wobei das erste den physischen Adapter, die Grafikkarte und den Grafiktreiber, repräsentiert und das zweite die Render-Umgebung, d.h. die Render-Pipeline und Objekte und Methoden die die Szene beeinflussen. Für die Objektinstanzen werden in *Form1* zwei private Objekte deklariert. (Private Objekte und Variablen werden bei *DX11Engine* durch einen Unterstrich '_' am Anfang des Bezeichners gekennzeichnet.)

```
Imports DX11Engine
Imports SlimDX

Public Class Form1

    Private _DX11Device As DX11Device
    Private _Scene As Scene

    . . .


End Class
```

Zur Initialisierung werden zwei neue Methoden erstellt, *InitializeDevice* und *InitializeScene*. *InitializeDevice* ist kurz gehalten, die Erzeugung eines Devices ist essentiell, sollte das scheitern, so wären weitere Schritte vergebens. Zu *InitializeScene* werden später Einstellungen und Objekte hinzugefügt die eine echte Szene ausmachen, zum jetzigen Zeitpunkt ist es vergleichbar mit einem leeren Raum.

```
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
```

Diese beiden Methoden werden dann im *Form1_Load* Ereignis zusammengefasst, sollte eine der Methoden nicht erfolgreich gewesen sein, so wird eine Fehlermeldung ausgegeben.

```
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 verwendet eine Vielzahl an Objekten die auf nicht verwaltete Ressourcen verweisen, diese Ressourcen müssen bei Beendigung des Programms freigegeben werden. *DX11Engine* verfügt über interne Methoden um das automatisch zu erledigen, so dass nur sichergestellt werden muss, dass das *DX11Device* selbst am Programmende zerstört wird. Hierzu wird das *Form1_FormClosing* Ereignis verwendet.

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

    _DX11Device.Dispose()

End Sub
```

Da sich die Größe des *RenderPanels* ändern kann und eine solche Änderung auch das *Scene*-Objekt beeinflusst, ist noch eine Methode erforderlich die dieses Ereignis behandelt.

Zu *InitializeScene* wird eine *AddHandler* Anweisung hinzugefügt

```
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
```

und zu *Form1* die Methode *ResizeScene.*

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

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

 End Sub
```

Die grundlegende Implementierung ist hiermit abgeschlossen. *Form1*enthält jetzt ein Direct3D Ansichtsfenster, das über die Eigenschaften und Methoden des *_Scene*-Objekts angesprochen werden kann.