MexDelphis
Enumerations-Tutorial

1. Datenstrukturen

Um die Grafikkarte und ihre Fähigkeiten zu erforschen, hat Microsoft die Enumeration erfunden. Für die meisten kleineren 3D-Projekte (Bildschirmschoner etc.) reicht es vollkommen aus, die 5 Schritte aus dem DX-SDK als Initialisierung auszuführen und sich eine aufwendige Enumeration zu sparen. Für alle anderen ist dieses Tutorial gedacht.
Im Gegensatz zu anderen Enumerationen, wird bei dieser Klasse KEINE vorauswahl getroffen. Es werden ALLE zulässigen Kombinationen enumeriert und in die Datenstrukturen gespeichert, so dass ein späterer Zugriff problemlos möglich ist ohne ein erneutes enumerieren.

Also fangen wir mal an, und überlegen, was wir von unserer Grafikkarte so alles wissen wollen und erstellen die notwendigen Datenstrukturen.

1. Wiviele Grafikkarten sind im PC ?

```
type
  PEnumInfo = ^TEnumInfo;
  TEnumInfo = Record
    AnzAdapter : Integer;
    Adapter    : Array of TAdapterInfo;
  end;
```

Ein kleines überschaubares Record.
An erster Stelle speichern wir die Anzahl der Adapter(Grafikkarten) die wir uns mit

```
IDirect3D9.GetAdapterCount
```

```
Besorgen.
```

Zum zweiten Punkt kommen wir jetzt.

2. a) Wie viele Display – Modes unterstützt unser Adapter?
   b) Welcher Display – Mode ist gerade gesetzt?
   c) Wie heißt unsere Grafikkarte ?

```
type
  PAdapterInfo = ^TAdapterInfo;
  TAdapterInfo = Record
    AnzDispModes      : Integer;
    DispModeSet       : D3DDisplayMode;
    AdIdentif         : D3DADAPTER_IDENTIFIER9;
    DispModes         : Array of D3DDisplayMode;
    DeviceInfo        : Array[0..2] of TDeviceInfo;
  end;
```

Ebenfalls recht überschaubar. AnzDispModes ist die Anzahl der unterstützten Display – Modes, DispModeSet ist der momentan gesetzte Display – Mode. AdIdentif ist eine DX-Structure wobei uns hier lediglich **Description** von Interesse ist, das ist nämlich der Name unserer Grafikkarte(n). Dann hätten wir noch DispModes, dies ist ein Dynamisches Array(die Grösse ergibt sich aus AnzDispModes) in dem wieder DX-Strucures gespeichert sind um einen Display – Mode zu beschreiben als da wären:

```
UINT Width;
UINT Height;
UINT RefreshRate;
D3DFORMAT Format;
```

Eben die Weite, die Höhe, die Wiederholungsrate sowie das Format

Die Anzahl der unterstützten Modes ergibt sich aus einem Zähler der bei jedem Schleifendurchlauf um 1 incrementiert wird, da man lediglich die Anzahl DispModes für ein bestimmtes DisplayFormat abfragen kann.

```
IDirect3D9.GetAdapterModeCount(Adapter, AdapterFormats])
```

Die aktuellen Display-Modes bekommen wir mit:

```
IDirect3D9.GetAdapterDisplayMode(Adapter,Out DispModeSet)
```

Den Adapter Identifier:

```
IDirect3D9.GetAdapterIdentifier(Adapter, 0, AdIdentif)
```

Der Zweite Parameter ist eine DWORD Flag, mit diesem wird angezeigt, was abgefragt wird. Zulässig ist 0, hier wird eben der Identifier geholt, oder D3DENUM_WHQL_LEVEL, damit wird Microsofts Windows Hardware Quality Labs (WHQL) certificates aus dem Internet downgeloadet(also Finger Weg ☺).

Kommen wir zum letzten Punkt unseres Records, das Array DeviceInfo.

Jeder Adapter(Grafikkarte) hat drei mögliche devices HAL, SW und REF
HAL ist der **hardware abstraction layer**. Alles was die Grafikkarte an Hardwareunterstützung bietet, wird hierüber genutzt, wenn für gewisse Sachen keine Hardwareunterstützung von der Karte angeboten wird, wird es über Software emuliert.

SW ist ein reiner Software Device die man sich selbst schreiben und über RegisterSoftwareDevice bei DX anmelden kann.

REF ist der reference rasterizer, er simuliert alle Direct3D Features per Software, so lassen sich bei der Entwicklung Einstellungen testen die von der Grafikkarte(weil eine Neue HighEnd zu teuer) nicht unterstützt werden.

Für jede Device einmal:

```
type
  PDeviceInfo = ^TDeviceInfo;
  TDeviceInfo = Record
    DevTyp           : D3DDEVTYPE;
    VertProc         : LONGWORD;
    Caps_Device      : D3DCaps9;
    AnzFmt_Adapter   : Integer;
    ComboInfo        : Array of TComboInfo;
  end;
```

DevTyp ist eben HAL, SW oder REF. Diesen legen wir beim Schleifendurchlauf über eine Constnte (kommt unten) fest.

VertProc ist die Vertex – Processing methode, als da sind:
HARDWARE_VERTEXPROCESSING, MIXED_VERTEXPROCESSING und SOFTWARE_VERTEXPROCESSING. Mit diesen angaben wird beim erstellen der Device festgelegt, wie DX mit den Vertexdaten verfahren soll.
Enumeriert werden diese Werte im Eigendlichen Sinne nicht, man kann lediglich prüfen ob die Gradikkarte Hardwareunterstützung bietet und somit natürlich auch HardwareVertexProcessing. Dies kann man so erledigen:

```
      If Caps_Device.DevCaps and D3DDEVCAPS_HWTRANSFORMANDLIGHT <> 0 then
```

Wenn hier TRUE herauskommt, dann haben wir HardwareSupport

Bei SOFTWARE wird das Vertexprocessing vom CPU durchgeführt. Die einstellung kann ohne Neuinitialisierung der Device NICHT geändert werden.

Bei HARDWARE wird das Vertexprocessing vom GPU durchgeführt Die einstellung kann ohne Neuinitialisierung der Device NICHT geändert werden.

Bei MIXED wird Standardmäßig die Hardware verwendet, es kann aber auf Software umgeschaltet werden

An sich verwendet man in den meisten Fällen wohl HARDWARE_VERTEXPROCESSING. Schon deshalb, weil man sich viel Platz im Speicher sparen kann und nicht genutzte Funktionalität von DX gar nicht erst lädt indem man eine D3DCREATE_PUREDEVICE beim erstellen der Device mit angibt.

```
CreateDevice(Adapter,
             DevTyp,
             HANDLE,
             D3DCREATE_PUREDEVICE  or D3DCREATE_HARDWARE_VERTEXPROCESSING,
             @ PresParams,
             ID3DDEVICE )
```

Beim erstellen einer PureDevice ist Hardware_Vertexprocessing vorgeschrieben, da KEINE Emulation von DX vorgenommen wird. Wenn die Grafikkarte also kein Hardware Vertexprocessing bietet (gibt's solche noch?) ist hier Schluss☺.
Sämtliche GET* calls sind ebenfalls weg, und stehen nicht zur Verfügung, verbrauchen aber auch keine Ressourcen ☺ .

Caps_Device: die Caps beschreiben die Fähigleiten der Grafikkarte, und sind für die Devices unterschiedlich. Zum Beispiel steht in den caps: D3DDEVCAPS_PUREDEVICE was bedeutet: Die Grafikkarte unterstützt rasterization, transform, lighting, und shading was meint, es ist eine 3D fähige Grafikkarte. Die Caps holen wir uns so:

```
IDirect3D9.GetDeviceCaps(Adapter, DevType, Out Caps_Device);
```

AnfFmt_Adapter ist die anzahl der Format Kombinationen aus Front und Backbuffer die unsere Grafikkarte unterstützt (und die sich auch vertragen). Diese werden über einen Zähler in der Schleife erfasst.

ComboInfo ist ein dynamisches Array das die Format Kombinationen enthält und sieht wie folgt aus:

```
type
  PComboInfo = ^TComboInfo;
  TComboInfo = Record
    Windowed          : BOOLEAN;  // Fenster oder Fullscreen
    fmt_Adapter       : D3DFORMAT;// Adapterformat
    AnzFmt_BackBuffer : Integer;  // Anzahl der BBuffer für fmt_Adapter
    BBInfo            : Array of TBBInfo;  // BBuffer für fmt_Adapter
  end;
```

Windowed gibt an, ob dies eine Combo ist die im Fenstermodus(TRUE) oder Fullscreen(FALSE) läuft.

Fmt_Adapter ist das Adapter Format, also der FrontBuffer.

AnzFmt_BackBuffer ist die Anzahl der passenden Backbuffer Formate.

Es hält sich hartnäckig das Gerücht, dass im Fullscreen Mode Front und Backbuffer das Gleiche Format haben müssen, dies ist FALSCH, Front und Backbuffer müssen lediglich die gleiche Bitzahl aufweisen. So ist D3DFMT_X8R8G8B8 und D3DFMT_A8R8G8B8 eine durchaus zulässige Combo.

BBInfo ist wieder ein Dynamisches Array das die einzelnen Informationen über den Backbuffer enthält, und folgende Struktur hat:

```
type
  PBBInfo = ^TBBInfo;
  TBBInfo = Record
    fmt_BackBuffer  : D3DFORMAT;  // Backbufferformat
    Anz_fmt_Stencil : Integer;    // wieviele fmt_Stencil sind da
    StencilInfo     : Array of TStencilInfo; //zu fmt_Adapter und
                                             //fmt_BackBuffer
                                             // passende Stencil Formate
  end;
```

Wir nähern uns langsam dem Ziel ☺.

Fmt_BackBuffer ist eben ein gültiges(zum FrontBuffer passendes) Format.
Dies überprüfen wir mit **CheckDeviceType**.

```
IDirect3D9.CheckDeviceType(Adapter,       // enumeriert in TEnumInfo
                           D3DDEVTYPE,    // steht in TDeviceInfo
                           AdapterFormat, // steht in TComboInfo
                           BackBufferFmt, // steht in TBBInfo
                           Windowed );    // steht in TComboInfo
```

Anz_fmt_Stencil ist die Anzahl der Depth/Stencil Formate die zu dieser Combo passen.

StencilInfo ist wieder ein dynamisches Array, das Informationen über den Depth/Stencil Buffer enthält und folgendermaßen aussieht:

```
type
  PStencilInfo = ^TStencilInfo;
  TStencilInfo = Record
    fmt_Stencil    : D3DFORMAT; // zu fmt_Adapter und fmt_BackBuffer
                                // passendes Stencil Format
    Anz_MSInfos    : Integer;
    MSInfo         : Array of TMSInfo // Multisamples die zum Stencil und
                                      //Backbuffer passen
  end;
```

fmt_Stencil ist ebenfalls ein D3DFormat für den Depth bzw. Depth/Stencil Buffer. Je nachdem, ob wir ein Format aussuchen, das Stencil zur verfügung stellt.

Das passende Depth/Stencil Format ermittelt man, indem man die DepthStencil Formate gegen den Front UND den Backbuffer prüft und zwar zum einen mit:

```
IDirect3D9.CheckDeviceFormat(Adapter,
                             D3DDEVTYPE,
                             AdapterFormat,
                             D3DUSAGE_DEPTHSTENCIL, // Stencil
                             D3DRTYPE_SURFACE,      // als Surface
                             StencilFormat);
```

Zum zweiten :

```
IDirect3D9.CheckDepthStencilMatch(Adapter,
                                  D3DDEVTYPE,
                                  AdapterFormat,
                                  BackBufferFmt,
                                  StencilFormat);
```

Wenn bei beiden Anfragen D3D_OK herauskommt, kann man das Depth/Stencil Format verwenden und der Combo hintzfügen.

Anz_MSInfos ist die Anzahl der Multisamples im Array MSInfo.

MSInfo ist ein Dynamisches Array, in dem wir alle D3DMULTISAMPLE_TYPE ´s ablegen die unsere Grafikkarete in der Verbindung Backbuffer/DepthStencil Buffer beherrscht. Wenn man die Multisamplefähigkeiten der Grafikkarte nutzen will (zB. für motion blur, depth-of-field focus effects, reflection blur …) muss man wissen, wie viele Sub-Sample-Ebenen (eben die Multisamples) zur verfügung stehen. Meist wird Multisampling aber lediglich zum Antialiasing verwendet, da die anderen Effekte über Shader deutlich schneller sind. Die Struktur sieht folgendermaßen aus:

```
type
  PMSInfo = ^TMSInfo;
  TMSInfo = Record
    MSTyp            : D3DMULTISAMPLE_TYPE; // multisample type
    MSQuality        : LongWord;  // Quality Level
  end;
```

Was hier noch hinzukommt ist der Quality – Level. Wenn man in seiner Engine lediglich Antialiasing haben will und auf Multisample verzichtet, kann man durch erfragen des *QualityLevels* unter verwendung von D3DMULTISAMPLE_NONMASKABLE die einstellungen bezüglich Antialiasing beim initialisieren der Device einstellen

```
IDirect3D9.CheckDeviceMultiSampleType
                (Adapter,                    // Integer
                 DeviceType,                 // D3DDEVTYPE :HAL/SW/REF
                 SurfaceFormat,              // D3DFORMAT  :Backbuffer
                 Windowed,                   //
                 MultiSampleType,            // D3DMULTISAMPLE_TYPE
                 pQualityLevels)             // OUT LongWord
```

wenn wir hier bei MultiSampleType D3DMULTISAMPLE_NONMASKABLE abgeben, erhalten wir über pQualityLevels das QualityLevel die unsere Grafikkarte unterstützt, die können wir dann in die PresentParameter beim Initialisieren der Device eintragen aber eben NUR mit D3DMULTISAMPLE_NONMASKABLE .. bei allen anderen Multisamples würde die Device nicht initialisiert.

```
PresParams.MultiSampleType    := D3DMULTISAMPLE_NONMASKABLE;
PresParams.MultiSampleQuality := pQualityLevels - 1
```

Um diesen ganzen Wulst an Strukturen zu erfassen, hier noch einmal eine Strukturierte Darstellung wie diese aufgebaut sind.

```
TEnumInfo []
| - AnzAdapter
|
- + TAdapterInfo []
  | - DispModeSet
  | - AdIdentif
  | - DispModes []
  |
  - + TDeviceInfo [0..2]
    | - DevTyp
    | - VertexProcessing
    | - Caps_Device
    | - AnzFmt_Adapter
    |
    - + TComboInfo []
      | - Windowed
      | - fmt_Adapter
      | - AnzFmt_BackBuffer
      |
      - + TBBInfo []
        | - fmt_BackBuffer
        | - Anz_fmt_Stencil
        |
        - + TStencilInfo []
          | - fmt_Stencil
          | - Anz_MSInfos
          |
          - + TMSInfo []
              - MSTyp
              - MSQuality
```

Die Strukturen sind wie in einem Verzeichnisbaum abgeordnet und orientieren sich am Aufbau am Caps-Viewer aus dem MS-SDK.

2. Die Constanten

Um später in unserer Enumeration bequem die einzelnen Firmate in einer Schleife abarbeiten zu können, werden diese in Constanten abgelegt.

Fangen wir mit dem DeviceTyp an:

```
const d3dDevTypes  : Array[0..2] of D3DDEVTYPE  = (D3DDEVTYPE_HAL,
                                                   D3DDEVTYPE_SW,
                                                   D3DDEVTYPE_REF);
```

Mit diesem Typ enumerieren wir später die deviceTypes … eben HAL / DW und REF.

Weiter mit den Adapter – Formaten:

```
const AdapterFormats : Array[0..2] of D3DFORMAT = (D3DFMT_X8R8G8B8,
                                                   D3DFMT_X1R5G5B5,
                                                   D3DFMT_R5G6B5);
```

In diesem ConstantenArray liegen alle zulässigen Adapter – Formate für den FrontBuffer. Einen habe ich hier unterschlagen, - A2R10G10B10 - dieser ist, für den FrontBuffer, NUR für FULLSCREEN zulässig.

Die Backbuffer – Formate:

```
const BackBufferFmt : Array[0..5] of D3DFORMAT = (D3DFMT_A2R10G10B10,
                                                  D3DFMT_A8R8G8B8,
                                                  D3DFMT_X8R8G8B8,
                                                  D3DFMT_A1R5G5B5,
                                                  D3DFMT_X1R5G5B5,
                                                  D3DFMT_R5G6B5);
```

Und weiter geht's zu den Depth/Stencil Formaten :

```
const DepthStencilFmt : Array[0..5] of D3DFORMAT = (D3DFMT_D16,
                        //16-bit z-buffer bit depth.
                                                    D3DFMT_D32,
                        //32-bit z-buffer bit depth.
                                                    D3DFMT_D24X8,
                        //32-bit z-buffer bit depth using 24 bits
                        //for the depth channel.
                                                    D3DFMT_D24S8,
                        //32-bit z-buffer bit depth using
                        //24 bits for the depth channel and
                        //8 bits for the stencil channel.
                                                    D3DFMT_D24X4S4,
                        //32-bit z-buffer bit depth using
                        //24 bits for the depth channel and
                        //4 bits for the stencil channel.
                                                    D3DFMT_D15S1);
                        //16-bit z-buffer bit depth where
                        //15 bits are reserved for the depth channel and
                        //1 bit is reserved for the stencil channel.
```

Wie man hier schön sieht, Gibt es eben Formate, die lediglich den Depth Buffer zur verfügung stellen aber keinen Stencil. Dies muss man bei einer spätern Auswahl natürlich

beachten. Wenn bei der Initialisierung der Engine zB. ein DepthFormat 24 Bit und Stencil 8 Bit verlangt wird, so muss man nur alle Combos durchgehen und schauen ob eine Gültige Combo mit diesem Depth/Stencil Format vorhanden ist.

Kommen wir zu den Multisample Typen:

```
const MultiSampleTypes : Array[0..16] of D3DMULTISAMPLE_TYPE
                                                =(D3DMULTISAMPLE_16_SAMPLES,
                                                D3DMULTISAMPLE_15_SAMPLES,
                                                D3DMULTISAMPLE_14_SAMPLES,
                                                D3DMULTISAMPLE_13_SAMPLES,
                                                D3DMULTISAMPLE_12_SAMPLES,
                                                D3DMULTISAMPLE_11_SAMPLES,
                                                D3DMULTISAMPLE_10_SAMPLES,
                                                D3DMULTISAMPLE_9_SAMPLES,
                                                D3DMULTISAMPLE_8_SAMPLES,
                                                D3DMULTISAMPLE_7_SAMPLES,
                                                D3DMULTISAMPLE_6_SAMPLES,
                                                D3DMULTISAMPLE_5_SAMPLES,
                                                D3DMULTISAMPLE_4_SAMPLES,
                                                D3DMULTISAMPLE_3_SAMPLES,
                                                D3DMULTISAMPLE_2_SAMPLES,
                                                D3DMULTISAMPLE_NONMASKABLE,
                                                D3DMULTISAMPLE_NONE );
```

Die MultiSampleTypes – eine etwas längere Liste, und ob 16 Samples in näherer Zukunft realistisch sind, möchte ich anzweifeln aber wir wollen ja für alles gewapnet sein … also rinne damit ☺

Eine Anmerkung zu den MultiSamples:
Wer sich mit der Enumeration schon einmal herumgeschlagen hat, wird feststellen dass ich hier die Samples auch in umgekehrter Reihenfolge stehen habe … um später in der Schleife diese abzubrechen wenn ein MultisampleTyp unterstützt wird und diesen als Maximal – Marke zu speichern. Dies setzt voraus, dass bei einer Unterstützung von 9 Samples auch alle darunterliegenden automatisch unterstützt werden. **DEM IST NICHT SO!** Meine Grafikkarte z.B. unterstützt im REF Modus bei einigen Combos 9 Samples … jedoch KEINE Samples von 5 – 7 erst wieder 2 – 4 werden supported.

Damit hätten wir den ersten Teil schon überstanden. Und die Grundlagen einer Enumeration geschaffen. Im zweiten Teil kümmern wir uns um die eigentliche Klasse und füllen unsere Strukturen mit Daten ☺