Fortis-Akademie
Chemnitz

Facharbeit

in der Fachrichtung Technik

im Fach Informatik

Grafikkartenenumeration und deren Umsetzung mit Direct3D

von
Florian Oeser
FOS T1/04

Betreuer:

Ort, Datum:

Inhaltsverzeichnis

1 Einleitung....3

2 Grundlagen....3

2.1 Erstellen des *IDirect3D9*-Interfaces....3

2.2 Erstellen des *IDirect3DDevice9*-Interfaces und die Präsentationsparameter....4

3 Die Enumeration....5

3.1 Die Adapterinformationen....5

3.2 Die Videomodis....6

3.3 Die „Caps“-Struktur....8

3.4 Die Gerätetypen....8

3.5 Die *BehaviorFlags*....9

3.6 Das Multisampling....9

3.7 Der BackBuffer....10

3.8 Der Z-Stencil-Buffer....11

4 Die Umsetzung....13

4.1 Code-Sample 1: Enumeration der Adapterinformationen....13

4.2 Code-Sample 2: Enumeration der Videomodis....14

4.3 Code-Sample 3: Enumeration der Gerätetypen....14

4.4 Code-Sample 4: Enumeration der Backbuffer....15

4.5 Code-Sample 5: Enumeration des Z-Stencil-Buffers....15

4.6 Code-Sample 6: Enumeration der Multisamples....16

5 Quellen....18

1. Einleitung

Eine Grafikkarte ist heute mehr denn je eines der wichtigsten Bestandteile des PC. Ohne diese wäre der Computer bzw. das Betriebssystem mit seiner graphischen Oberfläche nicht in der Lage, unter anderem per Dialogbox, mit uns zu kommunizieren. Die tägliche Arbeit mit dem PC wäre eine andere. Hohe Ansprüche an die Grafikkarte stellen aber vor allem die heutigen CAD-Programme und aktuelle Spiele. Dabei spielt die Darstellung von dreidimensionalen Grafiken eine sehr große Rolle. Gerade hier hat sich in den letzten Jahren sehr viel getan. Denn nicht nur die CPU-Taktraten erhöhen sich kontinuierlich, sondern auch die GPU-Taktraten und die VRAM-Größe auf den Grafikboards. Doch das alleine ist nicht unbedingt entscheidend, denn es gibt unzählige Eigenschaften und Fähigkeiten, die eine Grafikkarte besitzen kann. Angefangen von den verschiedenen Auflösungen, bis hin zur hardwaregestützten Lichtberechnung, über Pixel- und Vertexshader und Oberflächenformaten. Da diese von Grafikkarte zu Grafikkarte sehr unterschiedlich sein können bzw. sind und sich nicht die Hardware unseren Programmen anpasst, müssen wir uns der Hardware anpassen. Wir müssen also alle wichtigen Leistungen und Fähigkeiten auflisten. Der Fachbegriff dafür, der aus der Mathematik und Informatik kommt, heißt Enumeration. Eine ordentliche Enumeration ist das A und O einer Grafikanwendung. Denn nichts ist schlimmer als ein Programmabsturz, nur weil man einfach bestimmte Dinge voraussetzt, die aber die Grafikkarte gar nicht unterstützt.

2. Grundlagen

Zunächst ein paar Worte zu *Direct3D,* mit dessen Hilfe die Enumeration geschrieben wird. Die *Direct3D* Bibliothek, speziell für die Darstellung von graphischen Primitiven ausgelegt, ist Bestandteil der *DirectX* API, eine Multimediaschnittstelle von *Mircosoft,* die mehrerer solcher Teil-API's besitzt. Die *DirectX Audio* API zum Beispiel ist für den Sound zuständig und die *DirectInput* Komponete für die Maus- und Tastaturbehandlung. *DirectX* baut auf dem *COM* auf. Dieses von *Microsoft* entworfene Modell vereinfacht die Organisation von großen Funktions- und Klassenmengen, die *DirectX* zweifelsohne nun einmal besitzt. Darauf ist auch die Einteilung in diese Teil-API's zurückzuführen.
Wir werden jedoch nur von der *Direct3D*-Schnittstell e Gebrauch machen. Deshalb beleuchten wir sie etwas näher. Wie oben erwähnt ist sie ausschließlich für die Darstellung dreidimensionaler Grafik zuständig und bildet quasi den Vermittler zwischen Programm und dem Grafikkartentreiber. Durch den einheitlichen Befehlssatz den *Direct3D* bietet, wäre theoretisch eine Benutzung auf jeder Grafikkarte möglich, die einen Treiber liefert. Auch das *Direct3D* Interface an sich beinhaltet noch eine Hilfsbibliothek, die *Direct3D extensions* (D3DX), die unter anderem sehr schnelle Mathefunktionen zur Verfügung stellt. Die beiden wichtigsten Schnittstellen von *Direct3D* sind *IDirect3D9* und *IDirect3DDevice9*. Beide sind wie jede *COM*-Schnittstelle von *IUnknow* abgeleitet. *IDirect3D9* stellt quasi die Vermittlungsschnittstelle von *Direct3D* dar. Sie wird durch *Direct3DCreate9()* Methode erzeugt und bietet hauptsächlich Methoden für die Enumeration an. Die Methode *CreateDevice()* vom *IDirect3D9*-Interface erzeugt uns dann eine Instanz der *IDirect3DDevice9* Schnittstelle. Diese wiederum repräsentiert einen einzelnen *Direct3D*-Adapter (unter *Direct3D* ein Gerät, was für die Grafikausgabe zuständig ist). Sie ist also die Anlaufstelle für nahezu alles, da wir an sie die Befehle zum Zeichnen von 3D-Grafiken weiterleiten.
Direct3D besitzt weiterhin sein eigenes Ressourcensystem. Eine Ressource ist ein Speicherbereich in dem zum Beispiel Geometriedaten, Oberflächen, Meshs und so weiter gespeichert werden. Der Vorteil ist, dass diese Ressourcen direkt im VRAM der Grafikkarte liegen und so der Zugriff schneller erfolgen kann.

2.1 Erstellen des *IDirect3D9* Interfaces

Wie oben bereits erwähnt, benötigen wir nur eine Funktion, um die *IDirect3D9*-Schnittstelle anzulegen.

```
IDirect3D9 *WINAPI Direct3DCreate9(
  UINT SDKVersion
);
```

Die Methode erwartet lediglich einen Parameter, welcher die Schnittstellenversionsnummer beinhaltet. Anzugeben ist hier das *D3D_SDK_VERSION* Makro welches in der d3d9.h definiert ist und die aktuelle Version des installierten DirectX-SDK's bereitstellt.

```
LPDIRECT3D9 lpD3D = Direct3DCreate9(D3D_SDK_VERSION);
if(!lpD3D)
        MessageBox(NULL, „Couldn't initialize D3D“, „ERROR“, MB_OK);
lpD3D->Release;
```

Wichtig ist, die Initialisierung immer auf Erfolg zu überprüfen. Falls diese fehlschlägt ist meistens eine zu niedrige DirectX Version die Ursache. Ebenfalls wichtig ist, dass jede Schnittstelle nach dem benutzten dereferenziert werden muss. Dies erfolgt mit dem ->Release Aufruf. Nach dem die Schnittstelle jetzt angelegt ist, können wir anfangen, mit dessen bereitgestellten Methoden, die Enumeration durchzuführen. Jedoch widmen wir uns erst noch der Erstellung vom *IDirect3DDevice9*, damit wir einen Überblick bekommen, was überhaupt zu enumerieren ist.

2.2 Erstellen des *IDirect3DDevice9*-Interfaces und die Behandlung der Presentparameters

Die *IDirect3D*-Schnittstelle stellt uns weiterhin die *CreateDevice()* Methode zur Verfügung. Sie ist eine sehr wichtige, da sie uns ein *IDirect3DDevice9*-Interface erzeugt.

```
HRESULT CreateDevice(
  UINT Adapter,
  D3DDEVTYPE DeviceType,
  HWND hFocusWindow,
  DWORD BehaviorFlags,
  D3DPRESENT_PARAMETERS *pPresentationParameters,
  IDirect3DDevice9 **ppReturnedDeviceInterface
);
```

Der erste Parameter erwartet eine so genannte Adapter-ID(siehe 3.1). Für den zweiten Parameter gilt es einen geeigneten Gerätetyp zu finden(siehe 3.4). Der dritte Parameter erwartet das Handle vom Fenster in dem gerendert wird soll. Der vierte Parameter erwartet die „Verhaltensflags“, die das Verhalten der Schnittstelle beeinflussen (siehe 3.5). Die *D3DPRESENT_PARAMETERS*-Struktur, welche der fünfte Parameter erwartet, wird von der Methode mit weiteren wichtigen Informationen gefüllt. So zum Beispiel über das Multisampling oder Z-Bufferformat. Der letzte Parameter erwartet die Adresse eines Zeigers auf die *IDirect3DDevice9*-Schnittstelle.

Element	***Beschreibung***
UINT BackBufferWidth	Breite des Videomodus und der Bildbuffer

Element	*Beschreibung*
UINT BackBufferHeight	Höhe des Videomodus und der Bildbuffer
D3DFORMAT BackBufferFormat	Format der Bildbuffer
UINT BackBufferCount	Anzahl der Bildbuffer
D3DMULTISAMPLE_TYPE MultiSampleType	Multisampling-Typ
DWORD MultiSampleQuality	Multisampling-Qualität
D3DSWAPEFFECT SwapEffect	Einstellung zum Bildbuffertausch
HWND hDeviceWindow	Handle vom Fenster in dem gerendert wird.
BOOL Windowed	Fenster- oder Vollbildmodus
BOOL EnableAutoDepthStencil	Legt fest ob Direct3D automatisch einen Stencil- bzw. Depth(Z)buffer erzeugen soll oder nicht
D3DFORMAT AutoDepthStencilFormat	Format der Buffer
DWORD Flags	Flags die den Umgang mit den Buffern beschreiben.
UINT FullScreen_RefreshRate	Die Bildwiederholungsfrequenz vom Videomodus.
UINT PresentationInterval	Dient wieder zum Beschreiben der Bildbuffer

Tabelle 1: Die Präsentationsparameter

Zusammenfassend muss also folgendes enumeriert werden: Es wird eine Adapter-ID benötigt, eine Auflösung in der gerendert werden soll, ein Gerätetyp, die entsprechend zur Verfügung stehenden *BehaviorFlags*, ein Backbufferformat, ein Multisampletyp mit Qualitätsstufe, ein Z-Stencil-Format und die Bildwiederholungsrate. Es gibt aber auch Parameter und Elemente der Präsentationsparameter, die nicht enumeriert werden müssen. Das wäre das Fensterhandle, welches in der *CreateDevice()* Methode anzugeben ist und das *hDeviceWindow* Element der Präsentationsparameter, welches ebenfalls ein Fensterhandle erwartet. Die drei weiteren Elemente der Struktur, die wir nicht betrachten, sind *BackBufferCount*, *SwapEffect* und *PresentationsInter*val. Mit dem *BackBufferCount* Element lässt sich angeben wie viele BackBuffer man erzeugen will. So lassen sich Bilder als Art Reserve vorrendern. Das *SwapEffect* Element beschreibt, wie das Tauschen der Backbuffer geschehen soll. *Direct3D* stellt uns dafür *D3DSWAPEFFECT enum*-Aufzählung zur Verfügung. Zum einen gibt es das *D3DSWAPEFFECT_DISCARD* Element mit dem wir angeben, das es uns egal ist, was nach dem Kopieren des Backbuffers in den VRAM geschieht und erleichtern so die Arbeit für die Grafikkarte. Kommen mehrere Backbuffer zum Einsatz ist *D3DSWAPEFFECT_FLIP* die beste Wahl. Wird nur ein Bildbuffer benutzt, der ohne Veränderungen einfach kopiert wird, geben wir *D3DSWAPEFFECT_COPY* an. Das *PresentationInterval* Element gibt an, wie oft der Frontbuffer an sich sichtbar gemacht wird. Um die maximal Framerate zu erzielen geben wir hier *D3DPRESENT_INTERVAL_IMMEDIATE*, da jetzt nicht mehr auf den vertikalen Strahlenrücklauf vom Monitor gewartet wird.

3. Die Enumeration

3.1 Adapterinformationen

Zu den Adapterinformationen zählen die Anzahl der Adapter und die Beschreibung der Adapter. Die Anzahl der potentiell verfügbaren Adapter erhalten wir mit der Methode *GetAdapterCount()*

unserer *IDirect3D9*-Schnittstelle.

```
UINT GetAdapterCount(VOID);
```

Diese Methode erwartet keine Parameter und gibt uns die Anzahl als unsigned Integer zurück. Wobei der Rückgabewert eine ID darstellt. So ist die 0 der erste Adapter, 1 der zweite und so weiter. Statt 0 für den ersten Adapter zu verwenden kann man alternativ auch das *D3DADAPTER_DEFAULT* Makro benutzen. Nun wird jeder einzelne Adapter auf seine Beschreibung hin abgefragt. Dafür bedienen wir uns der *D3DADAPTER_IDENTIFIER9* Struktur. Die Methode *GetAdapterIdentifier()* ist für das Füllen zuständig.

```
HRESULT GetAdapterIdentifier(
   UINT Adapter,
   DWORD Flags,
   D3DADAPTER_IDENTIFIER9 *pIdentifier
);
```

Bei dem ersten Parameter ist die Apapter-ID anzugeben, für den eine genaue Beschreibung gewünscht wird. Für den zweiten Parameter gibt man entweder *D3DENUM_WHQL_LEVEL* an, um ein recht zeitaufwendiges Zertifizierungsverfahren zu starten, das einhundertprozentige Kompatibilität zwischen Adapter und Treiber sicherstellt oder NULL für den normalen Modus. Der letzte Parameter erwartet einen Zeiger auf die zu füllende *D3DADAPTER_IDENTIFIER9*-Struktur, die wir jetzt ein wenig genauer Betrachten.

Element	***Beschreibung***
char Driver[]	Dateiname des verwendeten Treibers
char Description[]	Name des Adapters(Grafikkarte)
LARGE_INTERGER DriverVersion	Treiberversion

Tabelle 2: Die *D3DADAPTER_IDENTIFIER9*-Struktur

Siehe hierzu auch *4.1 Code-Sample 1: Enumeration der Adapterinformationen.*

3.2 Die Videomodis

Der Videomodus bestimmt die Breite, Höhe, das Pixelformat und die Bildwiederholungsfrequenz. Es ist klar, dass nicht jede Grafikkarte und jeder Monitor hier die gleichen Eigenschaften liefern kann. Deshalb legen wir keinen Videomodus fest, sondern lassen den Benutzer entscheiden. Wie bei der Adapterinformation müssen wir erst einmal die Anzahl der verfügbaren Videomodis erfragen. Dafür stellt uns *Direct3D* die *GetAdapterModeCount* Methode zur Verfügung.

```
UINT GetAdapterModeCount(
   UINT Adapter,
   D3DFORMAT Format
);
```

Dem ersten Parameter der Methode übergeben wir die oben enumerierte Adapter-ID, damit *Direct3D* weiß, welcher Adapter abgefragt werden soll. Der zweite Parameter erwartet ein Pixelformat. *D3DFORMAT* ist eine *enum*-Aufzählung mit vielen Einträgen. Diese geben an, wie viele Bits oder Bytes ein Pixel einer Oberfläche an Speicher benötigt. Es ist wichtig zu wissen, dass nicht jedes Format für alles verwendet werden kann. So gibt es zum Beispiel Formate für Texturen, die aber nicht als Backbufferformate geeignet sind.

Format	*Beschreibung*
D3DFMT_A8R8G8B8	8 Bits für den Alphawert und 8 Bits für jede Farbkomponente
D3DFMT_X8R8G8B8	8 Bits unbenutzt und 8 Bits wieder für jede Farbkomponente
D3DFMT_R5G6B5	Ein 16-Bit-RGB-Format: 5 Bits für Rot und Blau. 6 Bits für Grün.
D3DFMT_R8G8B8	Ein 24-Bit-RGB-Format: 8 Bits für jede Farbkomponente
D3DFMT_R3G3B2	Ein 8-Bit Texturenformat: 3 Bits für Rot und Grün. 2 Bits für Blau.
D3DFMT_A16B16G16R16	Ein 64-Bit Oberflächenformat. 16 Bits für jede Farbekomponente.

Tabelle 3: Wichtige Formate der *D3DFORMAT enum*-Aufzählung

Die Methode liefert uns nun die Anzahl der verfügbaren Videomodis zurück, wieder in Form einer ID, die auf dem entsprechenden Adapter mit unterschiedlichen Formaten vorhanden sind. Wieder sehr ähnlich zu den Adapterinformationen, gehen wir jetzt jeden Videomodus durch und fragen wieder nähere Informationen ab. Die von *IDirect3D9* bereitgestellte Methode dafür ist *EnumAdapterModes()*.

```
HRESULT EnumAdapterModes(
  UINT Adapter,
  D3DFORMAT Format,
  UINT Mode,
  D3DDISPLAYMODE* pMode
);
```

Der erste Parameter erwartet wieder die Adapter-ID, dessen Videomodus abgefragt wird. Bei dem zweiten geben wir wieder das *D3DFORMAT* an, welches wir auch bei der vorhergehenden Methode angeben haben. Die Videomodi-ID, die wir mit *GetAdapterModeCount* erfragt haben, wird beim dritten Parameter angeben. Dem vierten und letzten Parameter ist ein Zeiger auf die *D3DDISPLAYMODE*-Struktur zu übergeben. Diese wird dann von der Methode mit den genaueren Informationen über den Modus befüllt.

Elemente	*Beschreibung*
UINT Width	Die horizontale Anzahl der Pixel(Breite)
UINT Height	Die vertikale Anzahl der Pixel(Höhe)
UINT RefreshRate	Die Bildwiederholungsfrequenz in Hz
D3DFORMAT Format	Das Format des Videomodis

Tabelle 4: Die D3DDISPLAYMODE-Struktur

Wenn im Windowed-Modus gearbeitet wird, braucht der Videomodus nicht geändert werden. Das heißt, wir nehmen das Format, welches auch gerade der Desktop benutzt. Um den aktuellen Videomodus herauszubekommen, benutzen wir die *GetAdapterDisplayMode()* von der *IDirect3D9*-Schnittstelle.

```
HRESULT GetAdapterDisplayMode(
```

```
  UINT Adapter,
  D3DDISPLAYMODE *pMode
);
```

Die Adapter-ID ist wieder beim ersten Parameter anzugeben. Der zweite Parameter erwartet einen Zeiger auf die oben besprochene *D3DDISPLAYMODE*-Struktur.

Nach diesen Schritten wissen wir welche Adapter uns zur Verfügung stehen, kennen die unterstützten Auflösungen und Bildwiederholungsfrequenzen und die verfügbaren Videomodeformate.

Siehe hierzu auch *4.2 Code-Sample 2: Enumeration der Videomodis.*

3.3 Die D3DCAPS9-Struktur

Einfach gesagt enthält diese Struktur, nach dem sie befüllt worden ist, alle hardwareabhängigen Fähigkeiten, die der Adapter hergibt. So lässt sich dann zum Beispiel abfragen wie hoch bzw. breit eine Textur nur sein darf oder wie viele Lichter es maximal geben darf. Zur Enumeration benötigen wir die Caps-Struktur lediglich um einen Gerätetyp zu finden und um die Unterstützung der Hardwarebeschleunigten Transformation und Beleuchtung abzufragen. Bevor dies geschehen kann, muss erst noch die Struktur mit Hilfe der *GetDeviceCaps* Methode vom *IDirect3D9*-Interface befüllt werden.

```
HRESULT GetDeviceCaps(

  UINT Adapter,
  D3DDEVTYPE DeviceType,
  D3DCAPS9 *pCaps
);
```

Logischerweise wird beim ersten Parameter wieder die Adapter-ID angegeben, von welchem wir die Informationen sammeln. Der zweite Parameter erwarten eine Gerätetypangabe(siehe 3.4). Ein Zeiger auf unsere zu füllende *D3DCAPS9*-Struktur muss beim dritten Parameter angegeben werden.

3.4 Die Gerätetypen

Der Gerätetyp gibt an, welcher Rasterizer zum Einsatz kommen wird. Der Rasterizer ist quasi der *Direct3D*-Kern. Dieser sorgt dafür, dass wir letztendlich Grafik auf unserem Bildschirm sehen können. Der Rasterizer war früher über die Software implementiert und ist heute hardwarebeschleunigt und so sehr viel schneller. Für die Gerätetypen stellt uns *Direct3D* die *D3DDEVTYPE enum*-Aufzählung zur Verfügung. Die drei wichtigsten sind *D3DDEVTYPE_HAL*, *D3DDEVTYPE_REF* und *D3DDEVTYPE_SW*. Bei *D3DDEVTYPE_HAL* wird der Rasterizer durch die Hardware vertreten. Im Gegensatz zu *D3DDEVTYPE_REF* muss man aber einige hardwarebedingte Begrenzungen in Kauf nehmen. So zum Beispiel die begrenzte Anzahl an Lichtern. Mit dem *D3DDEVTYPE_REF* Gerätetyp kann man zwar alles implementieren, egal ob das die Grafikkarte unterstützt oder nicht. Dies ist aber wegen der Softwareimplementierung enormst langsam und somit nur für den Entwickler geeignet. Der letzte Gerätetyp *D3DDEVTYPE_SW* ist ein benutzerdefinierter Software-Rasterizer. Mit dem DirectX-DDK kann man diesen selber schreiben und implementieren. Da das Vorhandensein des hardwarebeschleunigtem *D3DDEVTYPE_HAL* Gerätetyps nicht vorausgesetzt werden kann, müssen wir dies wieder mit einer Methode vom *IDirect3D9*-Interface überprüfen. Es kommt die vorher besprochene *GetDeviceCaps()* Methode zum Einsatz. Jetzt testen wie alle Gerätetypen durch,

indem wir sie jeweils einzeln dem zweiten Parameter der Methode übergeben. Schlägt diese dann fehl, wissen wir, dass dieser Gerätetyp nicht unterstützt wird.

Siehe hierzu auch *4.3 Code-Sample 3: Enumeration der Gerätetypen*

3.5 Die *BehaviorFlags*

Die so genannten *BehaviorFlags* oder „Verhaltensflags“ werden vom vierten Parameter der wichtigen *CreateDevice()* Methode erwartet. Die „Verhaltensflags“ sind eine Ansammlung von mehreren Flags, welche Einfluss auf das Verhalten der Schnittstelle haben. Es ist also möglich mehrerer solcher Flags getrennt mit dem bitweisen Oder-Operator(„|“) anzugeben. Insgesamt gibt es zehn Flags, von denen wir aber nur die Wichtigsten betrachten.

Flag	***Beschreibung***
D3DCREATE_HARDWARE_VERTEXPROCESSING	Die Transformations- und die Beleuchtungsberechnungen werden explizit von der Hardware durchgeführt.
D3DCREATE_SOFTWARE_VERTEXPROCESSING	Die Transformations- und die Beleuchtungsberechnungen werden explizit von der Software durchgeführt.
D3DCREATE_MIXED_VERTEXPROCESSING	Die Transformations- und die Beleuchtungsberechnungen werden von der Hardware und der Software durchgeführt.
D3DCREATE_PUREDEVICE	Direct3D emuliert fehlende Hardwarevorrausetzungen nicht mehr. Es wird quasi die „nackte“ Grafikkarte benutzt.
D3DCREATE_DISABLE_DRIVER_MANAGEMENT	Das Ressourcenmanagement übernimmt Direct3D an Stelle des Treibers.

Tabelle 5: Die wichtigsten BehaviorFlags

Da nicht jedes Device die Unterstützung für Transform and Lightning mit sich bringt, wird erst wieder geschaut, welche Flags überhaupt unterstützt werden. Wie bei den Gerätetypen greifen wir wieder auf die *D3DCAPS9*-Struktur zurück.

```
if(!(d3dCaps.DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT)){

        MessageBox(NULL, "HU", "HU", MB_OK);
        EnableWindow(GetDlgItem(m_hDlg, DLGHANDLES.RadioVertexProc),
                                        FALSE);
        return FALSE;

}
```

3.6 Multisampling

Das Multisampling (auch Anti-Aliasing oder Kantenglättung) sorgt für glatte und „weiche“ Bilder. Normalerweise teilt der Rasterizer die relativ genauen Vektorkoordinaten in das relativ ungenaue Pixelraster ein. So kann es passieren, dass ein Pixel an der Stelle 500,501 gerendert

wird, obwohl es an der Stelle 500,500 sein müsste. Obwohl das nur ein Pixel Unterschied ist, kann das bei niedrigen Auflösungen sehr schlecht aussehen. Das Multisampling sorgt jetzt dafür, dass Punkte, die nicht einem Pixel zugeordnet werden können, auf die anderen umgebenen Pixel verteilt werden. So scheint alles sehr glatt und detailreich auszusehen. Es müssen zum Füllen der Präsentationsparameter die Glättungsstärke und die Multisamplingqualität enumeriert werden. Für die Glättungsstärke stellt uns *Direct3D* die *D3DMULTISAMPLE_TYPE enum*-Aufzählung zur Verfügung. Wenn zum Beispiel kein Multisampling erwünscht ist, geben wir *MULTISAMPLE_NONE* an. Erwähnenswert sind auch die Konstanten *D3DMULTISAMPLE_2_SAMPLES* bis *D3DMULTISAMPLE_16_SAMPLES*. Die Zahl gibt die Anzahl der Samples (Durchgänge) an. Grundsätzlich gilt, je höher desto glätter und je kleiner desto schneller. Um zu überprüfen, ob überhaupt Multisampling unterstützt wird und wenn ja, mit wie vielen Samples und Qualitätsstufen, bedienen wir uns der *CheckDeviceMultiSampleType()* Methode vom *IDirect3DDevice9*-Interface.

```
HRESULT CheckDeviceMultiSampleType(

    UINT Adapter,
    D3DDEVTYPE DeviceType,
    D3DFORMAT SurfaceFormat,
    BOOL Windowed,
    D3DMULTISAMPLE_TYPE MultiSampleType,
    DWORD* pQualityLevels
);
```

Die Adapter-ID ist wieder beim ersten Parameter anzugeben. Der zweite erwartet den gewählten Gerätetyp. Beim dritten Parameter wird das Oberflächenformat vom Backbuffer bzw. vom Z-Stencilbuffer erwartet. Der vierte Parameter benötigt die Angabe, ob im Windowed- oder im Vollbildmodus gerendert wird. Der zu testende Multisampletyp ist beim vierten Parameter anzugeben. Zuletzt muss ein Zeiger auf eine *DWORD*-Variable angeben werden, die dann mit dem entsprechendem Qualitätslevel gefüllt wird. Ob nun überhaupt das eingesetzte Multisampling verfügbar ist, entscheiden wir wieder mit Hilfe des Rückgabewerts. Schlägt die Methode fehl, ist der gewünschte Multisampletyp nicht verfügbar. Weiterhin ist es wichtig, dass der Multisampletyp mit dem gewählten Backbufferformat, als auch mit dem Z-Stencil-Format überprüft wird.

Siehe hierzu auch *4.6 Code-Sample 6: Enumeration der Multisamples.*

3.7 Der Backbuffer

Der Backbuffer(oder auch Bildbuffer) ist eine gewöhnliche *Direct3D*-Oberfläche und liegt als Speicherbereich im VRAM der Grafikkarte. Alles was später auf dem Bildschirm sichtbar ist, wird also in den Backbuffer geschrieben. Um überhaupt zu wissen welche Backbufferformate auf dem Device verfügbar sind, müssen wir dies mit der *CheckDeviceFormat()* Methode von der *IDirect3D9*-Schnittstelle überprüfen. Diese wird übrigens für jegliche Ressourcenformatüberprüfung verwendet.

```
HRESULT CheckDeviceFormat(

    UINT Adapter,
    D3DDEVTYPE DeviceType,
    D3DFORMAT AdapterFormat,
    DWORD Usage,
    D3DRESOURCETYPE RType,
    D3DFORMAT CheckFormat
```

```
);
```

Der erste Parameter erwartet wieder die Adapter-ID, wessen Formate abgefragt werden sollen. Beim zweiten Parameter ist ein entsprechender Gerätetyp anzugeben. Der dritte Parameter erwartet das ausgewählte Videomodusformat oder falls Windowed aktiv ist, dass Format vom aktuellen Videomodus. Der Verwendungszweck für das Format wird mit dem vierten Parameter festgelegt. Da wir ein Backbufferformat suchen, geben wir hier *D3DUSAGE_RENDERTARGET* an. Der nächste Parameter entscheidet über den Ressourcentyp. Da der Backbuffer eine normale Oberfläche ist, geben wir hier *D3DRTYPE_SURFACE* an. Der fünfte und letzte Parameter erwartet nun das zu testende Format. Es wird überprüft ob es auf dem Adapter, mit dem Gerätetyp und in Form einer Ressource als Backbuffer einsetzbar ist. Schlägt der Aufruf der Methode fehl, wissen wir, dass das getestete Format so in der Form nicht verfügbar ist. Schlägt jedoch das *SUCCEEDED* Makro an, ist klar, dass das Format von der Hardware unterstützt wird. Jetzt wird aber noch eine weitere Methode benötigt um die Kompatibilität zwischen Gerätetyp und Format zu gewährleisten. Dazu wird die *CheckDeviceType()* Methode vom *IDirect3D9*-Interface benutzt.

```
HRESULT CheckDeviceType(

    UINT Adapter,
    D3DDEVTYPE DeviceType,
    D3DFORMAT DisplayFormat,
    D3DFORMAT BackBufferFormat,
    BOOL Windowed
);
```

Der erste Parameter erwartet wie gewohnt die Adapter-ID. Der Gerätetyp, welcher mit dem Format kompatibel sein soll, wird dem zweiten Parameter übergeben. Das Format des Videomodis(oder im Windowedmodus das des aktuellen Desktopformates) wird dem dritten Parameter übergeben. Der vierte Parameter erwartet das zu verwendende Backbufferformat. Als letztes wird angegeben, ob das Programm im Vollbild- oder Windowedmodi laufen wird. Wieder über den Rückgabewert entscheiden wir, ob die Methode Erfolg hatte oder nicht. Ist alles in Ordnung können wir die enumerierten Backbufferformate nutzen.

Siehe hierzu auch *4.4 Code-Sample 4: Enumeration der Backbuffer*

3.8 Der Z-Stencil-Buffer

Neben dem Backbuffer gibt es noch andere wichtige Oberflächen, wie etwa den Z-Buffer und den Stencil-Buffer. Der Z-Buffer ist quasi ein Speicherbereich in dem die Tiefenwerte der zu zeichnenden Pixel gespeichert werden. Daher auch der Name, denn die Z-Achse ist die Achse, die in die Tiefe geht. Werden nun Pixel gezeichnet, wird erst im Z-Buffer nachgeschaut, ob ihn schon ein Pixel dieser Tiefe gespeichert hat. Ist die Tiefe des neuen Pixels kleiner oder gleich, wird sein Tiefenwert in den Buffer geschrieben, da es näher am Beobachter ist. Liegt der neue Pixel aber tiefer als das vorhandene, kann es nicht sichtbar sein und wird daher verworfen und nicht gezeichnet. Wie gesagt, ist der Z-Buffer eine Oberfläche, obwohl er keine Farbinformationen beinhaltet. Somit gibt es auch wieder die verschiedenen Formate die ein Z-Buffer annehmen kann.

Format	***Beschreibung***
D3DFMT_D16_LOCKABLE	Ein 16 Bit-Z-Buffer, der gesperrt werden kann
D3DFMT_D32	Ein 32 Bit-Z-Buffer

Format	*Beschreibung*
D3DFMT_D24X8	Ein 32-Bit-Z-Buffer von dem aber nur 24-Bits benutzt werden
D3DFMT_D32F_LOCKABLE	Ein 32-Bit-Z-Buffer jedoch als Fließkommazahl, der ebenfalls gesperrt werden kann

Tabelle 6: Wichtige Z-Bufferformate aus der *D3DFORMAT enum*-Aufzählung

Grundsätzlich lässt sich sagen, je mehr Bits der Z-Buffer besitzt, desto genauer ist er. Diese Genauigkeit ist aber nicht überall gleich, denn auf kurzen Entfernungen ist sie sehr gut und bei weitentfernten Objekten schlecht. Als Alternative kann man den Z-Buffer auch zu einem so genannten W-Buffer umfunktionieren. Dieser Buffer speichert nicht die Z-Koordinaten der Pixel sondern deren W-Koordinaten. Diese entstehen bei der Projektion ins Pixelraster. Diese Werte sind genauer in der Tiefe verteilt. Leider kann man eine Hardwareunterstützung nicht voraussetzen und sollte dann auf die normalen Z-Buffer zurückgreifen.

Der Stencil-Buffer oder auch Schablonenpuffer, ist ein Speicherbereich, der für jeden Pixel einen zusätzlichen Wert speichert. Streng genommen ist er ein Teil des Z-Buffers. Verwendet wird der Stencil-Buffer für viele Effekte. So zum Beispiel für die Berechnung von Schatten (Shadow Mapping/Shadow Volume Rendering) oder auch zum maskieren von Pixeln, die dann zum Beispiel nicht gerendert werden. Auch der Stencil-Buffer bringt wieder einige neue Oberflächenformate mit sich.

Format	*Beschreibung*
D3DFMT_D24S8	Ein 32-Bit-Z-Buffer mit 24 Bits für die Tiefe und 8 Bits für den Stecil-Buffer
D3DFMT_D15S1	Ein 16-Bit-Z-Buffer mit 15 Bits für die Tiefe und einem Bit für den Stencil-Buffer
D3DFMT_D24FS8	Ein 24-Bit-Fließkomma-Z-Buffer mit 8 Bits für den Stencil-Buffer
D3DFMT_D24X4S4	Ein 32-Bit-Z-Buffer mit 24 Bits für die Tiefe, 4 Bits Stencil-Buffer und 4 ungenutzten Bits

Tabelle 7: Wichtige Stencil-Bufferformate aus der *D3DFORMAT enum*-Aufzählung

Um erst einmal wieder alle Z-Stencil-Bufferformate herauszubekommen die das Device unterstützt, rufen wir wie beim Backbuffer die *CheckDeviceFormat()* Methode auf. Nur mit dem Unterschied, dass wir beim vierten Parameter *D3DUSAGE_DEPTHSTENCIL* angeben, damit die Methode weiß, welchen Verwendungszweck wir wünschen. Natürlich müssen auch dem fünften Parameter entsprechend andere Formate übergeben werden. Um noch die Kompatibilität der Z-Stencil-Formate mit dem Videomodusformat bzw. Bildbufferformat zu gewährleisten, benötigen wir eine weitere Methode. *CheckDepthStencilMatch()* vom *IDirect3D9*-Interface.

```
HRESULT CheckDepthStencilMatch(
  UINT Adapter,
  D3DDEVTYPE DeviceType,
  D3DFORMAT AdapterFormat,
  D3DFORMAT RenderTargetFormat,
  D3DFORMAT DepthStencilFormat
);
```

Der erste Parameter verlangt wie immer die Adapter-ID. Der entsprechende Gerätetyp ist dem zweiten Parameter zu übergeben. Der dritte und vierte Parameter erwarten jeweils das Videomodusformat bzw. das Backbufferformat. Beim fünften Parameter wird schließlich das zu überprüfende Z-Stencil-Format angegeben. Ist die Methode erfolgreich, kann man sicher sein und das Format verwenden.

Siehe hierzu auch *4.5 Code-Sample 5: Enumeration des Z-Stencil-Buffers*

4. Die Umsetzung

Nun haben wir alle möglichen Eigenschaften und Fähigkeiten enumeriert um die *CreateDevice()* Methode aufzurufen und um die Präsentationsparameter zu füllen. Man sollte aber nicht vergessen, dass einige Optionen von wieder anderen Optionen abhängig sind. So kann es sein, dass bestimmte Backbufferformate zwar vom Device unterstützt werden aber nicht mit dem aktuell gewählten Videomodusformat. Aus diesem Grund muss eine hierarchische Abhängigkeitsliste erstellt werden:

- Adapterinformationen abfragen
- Den Gerätetyp abfragen
- Vollbild- oder Windowedmodus
- Videomodusinformationen holen
- Die Backbufferformate enumerieren
- Die Z-Stencil-Formate abfragen
- Den Multisampletyp- und Qualität bestimmen

Das heißt, wenn der Benutzer einen anderen Videomodus wählt, müssen die darunter folgenden Eigenschaften erneut abgefragt werden.

4.1 Code-Sample 1: Abfragen der Adapterinformationen

```
VOID CDXEnum::EnumAdapter(VOID){

        int iNumAdapter;

        iNumAdapter = m_lpD3D->GetAdapterCount();

      D3DADAPTER_IDENTIFIER9* m_pAdapterIdent = new D3DADAPTER_IDENTIFIER9[iNumAdapter];

        for(int iAdapter = 0; iAdapter < iNumAdapter; iAdapter++)
        {

                m_lpD3D->GetAdapterIdentifier(iAdapter, 0, &m_pAdapterIdent[iAdapter]);

                char Info[1024];
                sprintf(Info, "Driver: %s  Driverversion: %d Description: %s",
                        m_pAdapterIdent[iAdapter].Driver,
```

```
                    m_pAdapterIdent[iAdapter].DriverVersion,
                    m_pAdapterIdent[iAdapter].Description);

            //Ausgabe zum Beispiel in MessageBox
        }

}
```

4.2 Code-Sample 2: Abfragen der Videomodis

```
VOID CDXEnum::VideoModes(VOID){


INT iNumVideoModes;
char cVideoModes[1000] = "";
char cFormat [512] = "";
D3DDISPLAYMODE* pVideoModes = NULL;

D3DFORMAT d3dFormats[] = {D3DFMT_R5G6B5, D3DFMT_X8R8G8B8,
                          D3DFMT_A2R10G10B10, D3DFMT_X1R5G5B5};

for(int iVideoFormats = 0; iVideoFormats < 4; iVideoFormats++)
{


        iNumVideoModes = m_lpD3D->GetAdapterModeCount(m_iChoosenAdapter,
                                                      d3dFormats[iVideoFormats]);

        pVideoModes = new D3DDISPLAYMODE[iNumVideoModes]

        for(int iVideoModes = 0; iVideoModes < iNumVideoModes; iVideoModes++)
        {

            m_lpD3D->EnumAdapterModes(m_iChoosenAdapter, d3dFormats[iVideoFormats],
                                      iVideoModes,
                                      &pVideoModes[iVideoModes]);

            ReturnFormatString(pVideoModes[iVideoModes]->Format, chFormat);

            sprintf(cVideoModes, "NumModus %d: %d x %d x %d Hz with %s",
                iVideoModes,
                pVideoModes[iVideoModes]->Width,
                pVideoModes[iVideoModes]->Height,
                pVideoModes[iVideoModes]->RefreshRate,
                cFormat);

            //Ausgabe zum Beispiel in MessageBox

        }

        delete[] pVideoModes;

}
}
```

4.3 Code-Sample 3: Enumeration der Gerätetypen

```
VOID CDXEnum::DeviceType(){


        if(!SUCCEEDED(m_lpD3D->GetDeviceCaps(m_iChoosenAdapter,
                                             D3DDEVTYPE_HAL,
                                             &d3dCaps)))
```

```
            //HAL ist verfügbar

        if(!SUCCEEDED(m_lpD3D->GetDeviceCaps(m_iChoosenAdapter,
                                             D3DDEVTYPE_REF,
                                             &d3dCaps)))
            //REF ist verfügbar

        if(!SUCCEEDED(m_lpD3D->GetDeviceCaps(m_iChoosenAdapter,
                                             D3DDEVTYPE_SW,
                                             &d3dCaps)))
            //SW ist verfügbar

}
```

4.4 Code-Sample 4: Enumeration der Backbuffer

```
VOID CDXEnum::GetBackBufferFormats(VOID){

INT iVideoModes;
D3DDISPLAYMODE* pVideoModes = NULL;

D3DFORMAT d3dFormats[] = {D3DFMT_X8R8G8B8, D3DFMT_A8R8G8B8,
                          D3DFMT_R5G6B5, D3DFMT_A2R10G10B10,
                          D3DFMT_A1R5G5B5, D3DFMT_X1R5G5B5};
char cFormats[256];
char cBufferFormats[256];
D3DFORMAT* pFormat = NULL;


for(int iNumFormats = 0; iNumFormats < 6; iNumFormats++)
{


        if(SUCCEEDED(m_lpD3D->CheckDeviceFormat(m_iChoosenAdapter,
                                                m_d3dDevType,
                                    m_bWindowed ? m_pCurrentMode.Format : pVideoModes->Format,
                                                D3DUSAGE_RENDERTARGET,
                                                D3DRTYPE_SURFACE,
                                                d3dFormats[iNumFormats])))
        {


            if(SUCCEEDED(m_lpD3D->CheckDeviceType(m_iChoosenAdapter,
                                                  m_d3dDevType,
                                    m_bWindowed ? m_pCurrentMode.Format : pVideoModes->Format,
                                                  d3dFormats[iNumFormats],
                                                  FALSE)))
                {

                    ReturnFormatString(d3dFormats[iNumFormats], cFormats);
                    sprintf(cBufferFormats, "%s",cFormats);

                    //Ausgabe zum Beispiel in MessageBox

                }
        }
}
}
```

4.5 Code-Sample 5: Enumeration des Z-Stencil-Buffers

```
VOID CDXEnum::Get_Z_StencilFormats(VOID){
```

```
INT iVideoModes;
D3DDISPLAYMODE* pVideoModes = NULL;
char cZStencilFormats[256];
char cAvailableFormats[256];
D3DFORMAT* pZStencilFormat;

D3DFORMAT d3dZStencilFormats[] = {D3DFMT_D16_LOCKABLE, D3DFMT_D32,
                                  D3DFMT_D15S1, D3DFMT_D24S8,
                                  D3DFMT_D24X8, D3DFMT_D24X4S4,
                                  D3DFMT_D32F_LOCKABLE,
                                  D3DFMT_D24FS8, D3DFMT_D16};

for(int iNumFormats = 0; iNumFormats < 9; iNumFormats++)
{

        if(SUCCEEDED(m_lpD3D->CheckDeviceFormat(m_iChoosenAdapter,
                                                m_d3dDevType,
                                  m_bWindowed ? pVideoModes->Format : m_pCurrentMode.Format,
                                                D3DUSAGE_DEPTHSTENCIL,
                                                D3DRTYPE_SURFACE,
                                                d3dZStencilFormats[iNumFormats])))
        {

                if(SUCCEEDED(m_lpD3D->CheckDepthStencilMatch(m_iChoosenAdapter,
                                                             m_d3dDevType,
                                  m_bWindowed ? pVideoModes->Format : m_pCurrentMode.Format,
                                                             *pSelectedBackBuffer,
                                                             d3dZStencilFormats[iNumFormats])))
                {

                        ReturnFormatString(d3dZStencilFormats[iNumFormats], cZStencilFormats);
                        sprintf(cAvailableFormats, "%s", cZStencilFormats);

                        //Ausgabe zum Beispiel in MessageBox


                }
        }
}
}
```

4.6 Code-Sample 6: Enumeration der Multisamples

```
VOID CDXEnum::CheckMultiSampling(VOID){


D3DMULTISAMPLE_TYPE d3dMultiSampleType[] = {
        D3DMULTISAMPLE_NONE, D3DMULTISAMPLE_NONMASKABLE,
        D3DMULTISAMPLE_2_SAMPLES, D3DMULTISAMPLE_3_SAMPLES,
        D3DMULTISAMPLE_4_SAMPLES, D3DMULTISAMPLE_5_SAMPLES,
        D3DMULTISAMPLE_6_SAMPLES, D3DMULTISAMPLE_7_SAMPLES,
        D3DMULTISAMPLE_8_SAMPLES, D3DMULTISAMPLE_8_SAMPLES,
        D3DMULTISAMPLE_9_SAMPLES, D3DMULTISAMPLE_9_SAMPLES,
        D3DMULTISAMPLE_10_SAMPLES, D3DMULTISAMPLE_11_SAMPLES,
        D3DMULTISAMPLE_12_SAMPLES,
D3DMULTISAMPLE_FORCE_DWORD};

DWORD dwpQualityLevels;
char    cMultiSampling[256];

for(int iNumSamples = 0; iNumSamples < 16; iNumSamples++)
{
```

```
        if(SUCCEEDED(m_lpD3D->CheckDeviceMultiSampleType(m_iChoosenAdapter,
                                                         m_d3dDevType,
                                                        *pSelectedBackBuffer,
                                                         m_bWindowed,
                                                         d3dMultiSampleType[iNumSamples],
                                                         &dwpQualityLevels)))
        {

                if(dwpQualityLevels > 0)
                {

                        if(SUCCEEDED(m_lpD3D->CheckDeviceMultiSampleType(m_iChoosenAdapter,
                                                                         m_d3dDevType,
                                                                        *pSelectedZStencilBuffer,
                                                                         m_bWindowed,
                                                                         d3dMultiSampleType[iNumSamples],
                                                                         &dwpQualityLevels)))
                        {

                                sprintf(cMultiSampling, "MultiSamplingTypes: %d QualityLevel: %d",
                                        d3dMultiSampleType[iNumSamples], dwpQualityLevels);

                                //Ausgabe zum Beispiel in MessageBox


                        }

                }

        }

}
```

Hier noch die in den Samples vorkommende *ReturnFormatString() Methode*:

```
VOID CDXEnum::ReturnFormatString(D3DFORMAT d3dFormat, char* pcFormat){

switch(d3dFormat)
{

case D3DFMT_X8R8G8B8:           strcpy(pcFormat, "X8R8G8B8");           break;
case D3DFMT_R5G6B5:             strcpy(pcFormat, "R5G6B5");             break;
case D3DFMT_A8R8G8B8:           strcpy(pcFormat, "A8R8G8B8");           break;
case D3DFMT_A2R10G10B10:        strcpy(pcFormat, "A2R10G10B10");        break;
case D3DFMT_X1R5G5B5:           strcpy(pcFormat, "X1R5G5B5");           break;
case D3DFMT_A1R5G5B5:           strcpy(pcFormat, "A1R5G5B5");           break;
case D3DFMT_D16_LOCKABLE:       strcpy(pcFormat, "D16_LOCKABLE");       break;
case D3DFMT_D32:                strcpy(pcFormat, "D32");                break;
case D3DFMT_D15S1:              strcpy(pcFormat, "D15S1");              break;
case D3DFMT_D24S8:              strcpy(pcFormat, "D24S8");              break;
case D3DFMT_D24X8:              strcpy(pcFormat, "D24X8");              break;
case D3DFMT_D24X4S4:            strcpy(pcFormat, "D24X4S4");            break;
case D3DFMT_D32F_LOCKABLE:      strcpy(pcFormat, "D32F_LOCKABLE");      break;
case D3DFMT_D24FS8:             strcpy(pcFormat, "D24FS8");             break;
case D3DFMT_D16:                strcpy(pcFormat, "D16");                break;

}


}
```

5. Quellen

Microsoft:

LPDIRECT3D9
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/_idirect3d9.asp

IDirect3D9::GetDeviceCaps
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/GetDeviceCaps.asp

D3DDEVTYPE Enumerated Type
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/enums/d3ddevtype.asp

D3DCAPS9 Structure
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/structures/d3dcaps9.asp

IDirect3D9::GetAdapterModeCount
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/GetAdapterModeCount.asp

D3DFORMAT
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/enums/d3dformat.asp

IDirect3D9::GetAdapterIdentifier
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/GetAdapterIdentifier.asp

D3DADAPTER_IDENTIFIER9
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/structures/d3dadapter_identifier9.asp

IDirect3D9::GetAdapterDisplayMode
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/GetAdapterDisplayMode.asp

D3DDISPLAYMODE Structure
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/structures/d3ddisplaymode.asp

IDirect3D9::GetAdapterCount
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/GetAdapterCount.asp

IDirect3D9::EnumAdapterModes
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/EnumAdapterModes.asp

IDirect3D9::CreateDevice Method
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/CreateDevice.asp

D3DCREATE
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/constants/D3DCREATE.asp

IDirect3D9::CheckDeviceType Method
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/CheckDeviceType.asp

IDirect3D9::CheckDeviceMultiSampleType Method
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/CheckDeviceMultiSampleType.asp

D3DMULTISAMPLE_TYPE Enumerated Type
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/enums/d3dmultisample_type.asp

IDirect3D9::CheckDeviceFormat Method
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/CheckDeviceFormat.asp

D3DUSAGE
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/constants/D3DUSAGE.asp

D3DRESOURCETYPE Enumerated Type
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/enums/d3dresourcetype.asp

IDirect3D9::CheckDepthStencilMatch Method
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3d9/CheckDepthStencilMatch.asp

IDirect3DDevice9 Interface
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/interfaces/idirect3ddevice9/_idirect3ddevice9.asp

D3DPRESENT_PARAMETERS Structure
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/structures/d3dpresent_parameters.asp

D3DSWAPEFFECT Enumerated Type
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/enums/d3dswapeffect.asp

D3DPRESENTFLAG
http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_dec_2004/directx/graphics/reference/d3d/constants/D3DPRESENTFLAG.asp

David Scherfgen:
3D-Spieleprogrammierung mit DirectX9 und C++
2., erweiterte und aktualisierte Auflage