OpenGL und DirectX

OpenGL und DirectX

OpenGL (= Open Graphics Library) ist ein Softwareinterface zur Graphikhardware bestehend aus ca. 250 Kommandos in 2 Bibliotheken `oglcore` und `oglutilities`.
Entwickelt ab 1990 von SGI (Silicon Graphics Inc.). Ziel: Hardware- und Betriebssystem-unabhängige Graphik (gerichtet an Hersteller von Graphikkarten und an Programmierer).
Links:
www.opengl.org
www.sgi.com/products/software/opengl
www.miszalok.de/C_3D_OpenGL\Index_of_Course.htm

DirectX ist der Sammelname von 10 Windows-Bibliotheken (siehe Tabelle weiter unten) für extrem Hardware-nahe Programmierung in Hardware-unabhängiger Form.
Dauerentwicklung von Microsoft seit 1994 (damals "Games SDK") in aufsteigenden Versionsnummern (derzeit 10.1). Ziel und Hauptanwendung: Windows als Plattform für Multimedia.
Fast alle Graphik-, Sound-, Radio-, Video-, TV-Karten haben DirectX-Treibersoftware.
Bis incl. DirectX 9.0 unterscheiden sich die DirectX-Bibliotheken prinzipiell von allen anderen Windows-APIs (Application Programming Interfaces). Sie garantieren nicht, dass die Aufrufe ausgeführt werden.
Jeder Programmierer muss selbst herausfinden, ob die via DirectX zu programmierende Hardware auf seinen Zielrechnern vorhanden ist und wenn ja, ob und was man mit der Zielhardware programmieren kann.
In der Praxis vertrauen viele Programmierer hoffnungsvoll auf den DirectX-Hardwaretreiber.
Die Hoffnung ist, dass der Treiber der Zielhardware so gut ist, dass er unverdauliche Calls via DirectX nicht einfach verweigert, sondern diese mit Hilfe einer geeigneten Notlösung abarbeitet (siehe HEL weiter unten).
Auf dieser Hoffnung basiert auch Course 3DMDX von Prof. Miszalok.

Es gibt mehrere Zugänge zur DirectX-Programmierung:
1.) C++ und das DirectX Software Development Kit = DXSDK mit C++ (alte PC-Spiele).
2.) Managed DirectX mit C# ist fast genauso schnell aber einfacher als 1.) und ist enthalten im DXSDK.
3.) XNA mit C# ist Nachfolger von 2.) bei neuen PC und XBox-Spielen.
4.) Auf DirectX10 basiert die Programmierung der Windows Vista Benutzeroberfläche (enthalten im DXSDK).
5.) Windows Presentation Foundation = WPF mit XAML und C# →
DirectX10 liegt verhüllt in einer enormen Klassenbibliothek.

Managed DirectX wurde schnell sehr populär, weil es einen einfachen und eleganten Zugang zu DirectX mit voller Geschwindigkeit bietet und war von 2002 bis 2007 die modernste Plattform für PC-Spieleentwicklung unter WindowsXP. 2007 hat Microsoft die Weiterentwicklung von Managed DirectX eingestellt zugunsten zweier neuer DirectX-APIs:
1) XNA, welches einen gegenüber Managed DirectX erstaunlich vereinfachten Zugang zu DirectX für die PC- und XBox-Spieleentwickler bietet.
2) WPF, mit dem Ziel der vollständigen und nahtlosen Integration von DirectX in alle Programmoberflächen und Internetseiten.

Links:
www.microsoft.com/windows/directx/default.aspx
www.dotnet-magazin.de/itr/online_artikel/psecom,id,636,nodeid,31.html
www.it-academy.cc/content/article_browse.php?ID=732
Sie finden 7 Managed-DirectX-Anwendungen unter:
www.miszalok.de/C_3D_MDX/Index_of_Course.htm.

OpenGL und der harte Kern von Direct3D sind funktionell gleichwertig und ziemlich ähnlich.
Man beachte jedoch folgende Unterschiede:

	OpenGL	DirectX
objektorientiert	nein	ja
Verpackung in Klassenbibliothek	QT	Managed DX, XNA, WPF
unterstützt Audio/Video/Game Input Devices	nein	ja
Betriebssysteme	viele	nur Windows und seine Varianten
brauchbare Treiber vorhanden für	hochwertige Graphikkarten	fast alle Graphikkarten
Qualität der Treiber	oft schlecht	oft besser als OpenGL-Treiber
verwendet von	Uni, Forschung, CAD	Spiele-Industrie
neue Version	alle 5 Jahre (sonst nur "Extensions")	alle 15 Monate
Eigentum von	Silicon Graphics Inc.	Microsoft

OpenGL Bibliotheken und DirectX Namespaces

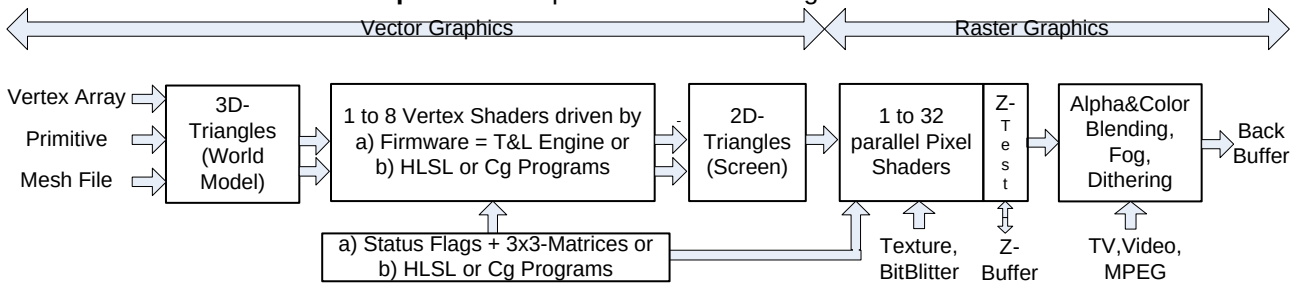
OpenGL besteht aus zwei Bibliothekn (DLLs), die sich ausschließlich der Graphik gewidmet sind.
Microsoft hat unter dem Namen DirectX alles versammelt, was am Betriebssystem vorbei direkt auf die Hardware zugreift. In Managed DirectX sind diese Bibliotheken in Form von Namespaces verhüllt.
Nur die ersten vier dieser Namesspaces beschäftigen sich mit Graphik.

OpenGL lib	covers **DirectX** functionality from
oglcore	Microsoft.DirectX, Microsoft.DirectX.DirectDraw, Microsoft.DirectX.Direct3D
oglutilities	Microsoft.DirectX.Direct3DX
.NET Namespace	**API** = Application Programming Interface
Microsoft.DirectX	common basic functions
Microsoft.DirectX.DirectDraw	subset of Direct3D-lib: basic 2D functions, bitmaps, window management
Microsoft.DirectX.Direct3D	API for 3D graphics: wireframes, textures, light, Vertex and Pixel Shaders
Microsoft.DirectX.Direct3DX	3D utilities library, Mesh class and scene graph
Microsoft.DirectX.DirectPlay	network support for multiplayer games, host administration for DirectPlay sessions
Microsoft.DirectX.DirectSound	contains DirectMusic, API for real time multichannel mixer, 3D sound
Microsoft.DirectX.DirectInput	API for keyboard, mouse, joystick, trackball, touchpad, gamepad, wheel, force feedback
Microsoft.DirectX.AudioVideoPlayback	API for simple sound and video
Microsoft.DirectX.Diagnostics	system diagnostics API
Microsoft.DirectX.Security	system security API

OpenGL & Direct3D Pipeline

Der Graphikchip einer modernen Grafikkarte enthält mehrere in Form einer Pipeline (=Eimerkette) kaskadierend arbeitende Graphik-Prozessoren. Die erste Hälfte dieser Prozessoren beschäftigt sich mit Vektorgraphik, die zweite mit Rastergraphik.
Die Befehlsketten von OpenGL und Direct3D spiegeln das Prinzip der Prozessorkette der Graphikchips wieder. Folglich verteilt sich der Befehlsumfang von OpenGL und Direct3D auch etwa je zur Hälfte auf 3D-Vektorgraphik und auf Rastergraphik. Der fundamentale Unterschied zwischen Vektor- und Rastergraphik wird dabei verhüllt und versteckt, um den Programmierer, soweit irgend möglich, von den Problemen der Umwandlung von Vektor- in Rastergraphik zu verschonen.
Ebenso wird er abgeschirmt von den Problemen der Arbeitsteilung zwischen CPU und Graphikkarte, wie überhaupt von den gravierenden Unterschieden zwischen allen möglichen Varianten von Graphikchips.
Bei vielen Anfängern entsteht so zunächst die Illusion, Kenntnisse von Hardware und Rastergraphik seien nicht nötig.

Schema der 3D-Pipeline von OpenGL & Direct3D folgt der Architektur der GPU:

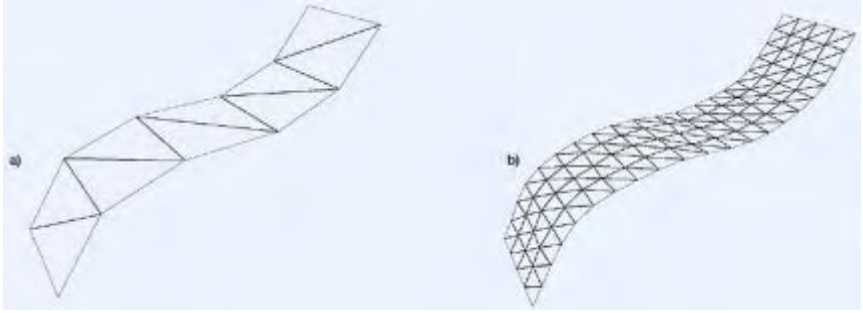


In DirectX kann man den gesamten Vektorgraphikteil der GPU abschalten durch den Flag `CreateFlags.SoftwareVertexProcessing` im Konstruktor von `Decice`.
Dann oder wenn die Graphikkarte oder das Motherboard keine vollständige oder überhaupt keine GPU besitzt, wird die Pipeline durch OpenGL/DirectX in der CPU simuliert.
Dann geben die Begriffe Vertex Shader, T&L Engine, HSSL/Cg Programme keinen Sinn und man spricht besser von CPU-basierter Graphik, siehe unten Kapitel **HEL und HAL**.

Vertex Shader = Kaskade von hintereinander geschalteten Microprozessoren innerhalb der GPU.
Moderne GPUs enthalten bis zu 8 solcher Kaskaden parallel. Der Begriff ist zweideutig: Ein Programm, das in **HLSL** oder **Cg** für einen Vertex Shader geschrieben ist, nennt man auch Vertex Shader.
Aufgabe des Vertex Shaders: Umwandlung von 3D-Dreiecken (Weltmodell) zu 2D-Dreiecken (Bildschirmansicht).
Prozesse im Vertex Shader: Tesselation, Koordinatentransformationen, 3D-Scroll+Zoom+Rotation, Clipping, Back Face Culling.
T&L Engine = Transform & Light Engine = Fixed Vector Pipeline = Bezeichnung für einen oder bis zu 8 parallele Vertex Shader, die durch vorgefertigte Firmware programmiert sind, die wenig Freiheiten lässt.
Man muss diese Firmware von außen steuern durch
a) Zustandsflags, Beispiel: `device.Lights[0].Enabled = true;`) und
b) 3x3-Matrizen, Beispiel:

```
device.Transform.View = Matrix.LookAtLH( new Vector3( 0f, 0f,-4f ),
                                         new Vector3( 0f, 0f, 0f ),
                                         new Vector3( 0f, 1f, 0f ) );.
```

Tesselation: Erzeugt aus Polygonen Dreiecksnetze und verfeinert ein grobmaschiges Dreiecksnetz.
Modern: Verfeinerung abhängig vom Betrachterabstand = adaptive Verfeinerung = Depth-Adaptive Tesselation = Level Of Detail based Tesselation = LOD based Tesselation = feine Dreiecke in niedrigem Abstand zum Eye Point und grobe Dreiecke bei weitem Abstand. image source: www.hartware.net

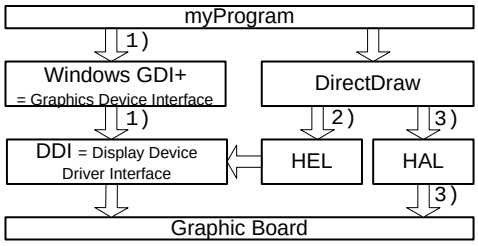


Clipping = Abschneiden von Linien und konvexen Polygonen am Bildrand mit dem Cohen-Sutherland-Algorithmus
Back Face Culling: ca. 50% der Dreiecke zeigen dem Betrachter ihre Rückseite. Wenn man sie entfernt, sind die folgenden Rasteroperationen doppelt so schnell.
Pixel-Shader = Rasterizer = dem Vertex-Shader nachgeschalteter Spezialprozessor des Graphikchips, spezialisiert auf Rastergraphik = Texturen und Rendering einzelner Pixel, mit **HLSL** oder **Cg** programmierbar, Graphikchip enthält bis 32 parallele Pixel-Shader.
Information und Tools zur Shader-Programmierung siehe:
http://developer.nvidia.com/object/fx_composer_home.html
Texture = deformieren = verzerren eines rechteckigen Rasterbildes so, dass es auf ein Polygon passt.
BitBlitter = Abkürzung für Bit Block Transfer = Annahme von gerasterten Buchstaben, Linien, Rechtecken, Ellipsen etc.
Z-Test = Depth Test = Entsorgung verdeckter Pixel
Alpha & Color Blending = maskiertes Überlagern bei Transparenz = Durchsichtigkeit
Fog = mit der Entfernung zunehmender Nebel
Dithering = Glätten stufiger Farbverläufe von 4-, 8- und 16-Bit Bildern durch gleitendes Mischen

HEL und HAL

Bei ihrer Installation verankern sich die Treiber von Graphikkarte, Soundkarte, Joystick etc. im Betriebssystem in Form je eines Device Driver Interface DDI. Mit Hilfe des entsprechenden DDIs initialisiert jede DirectX-Teilbibliothek beim Start einen Hardware Emulation Layer HEL und einen Aufruf-identischen Hardware Abstraction Layer HAL. HEL enthält die low-level Aufrufe für Basisfunktionen und CPU-Code,
HAL die Aufrufe für die externen, autonomen Microprogramme von Graphikkarte, Soundkarte etc.
HAL hat Aufruf-Priorität vor HEL, aber alle Bibliotheks-Aufrufe funktionieren, auch dann,
wenn HAL wenig oder gar nichts kann. Die HEL-Graphiken, HEL-Animationen, HEL-Audios, HEL-Videos etc. sind in der Regel langsam, aber sie funktionieren immer.
Die CPU-Hersteller Intel und AMD kämpfen gegen die Graphik- und Multimediakarten. Sie verbessern laufend die Graphik- und Soundfähigkeit und die Busarchitektur der CPUs, um HEL gegen HAL zu stärken.
Und durchaus mit Erfolg: Bei einfachen Spielen und Multimedia bemerkt man kaum einen Unterschied und für Büroanwendungenein genügt ein einfacher on-board-Videocontroller ohne Video Memory
(alle Graphikbuffer im Main Memory).

Beispiel: Zeichnen mit GDI+ oder mit DirectDraw HEL/HAL
Es gibt drei Wege, um etwas zu zeichnen:
`1)` normaler Windows-Befehl ohne DirectX führt über GDI+ und DDI.
Beispiel: `graphics.DrawLine( mypen, 0, 0, 100, 100 );`
`2)` über DirectDraw, HEL und DDI
`3)` über DirectDraw und HAL



Falls der Aufrufweg `3)` existiert, ist `2)` gesperrt.
`3)` ist schneller als `2)` und `2)` ist schneller als `1)`. GDI+ und DirectDraw-Zeichenbefehle sind beliebig mischbar.
GDI+ Info: **http://msdn.microsoft.com/library/GDIPlus.asp**

Vorsicht: Unausgereifte und schlechte Graphikkartentreiber installieren oft ein unvollständiges DDI und entwerten damit eine gute Graphikkartenhardware. Die Klasse **Device** kennt die Graphikkarte nämlich nur über deren DDI und benutzt nur die im DDI verzeichneten Features und Microprogramme.
Abhilfe: Im Internet beim Hersteller Ihrer Graphikkarte nachsehen, ob es dort einen neueren Treiber gibt und diesen installieren.

Direct3D Device

ist die wichtigste Direct3D-Klasse, sie steuert die Graphikkarte. Ihre sichtbarste Methode ist `Device.Present`, die den BackBuffer der Graphikkarte auf FrontBuffer umschaltet und damit die Szene auf den Monitor bringt. Weiter enthält sie die Eigenschaften/Methoden für Vektorgraphik (z.B. Viewport, Vertex Format, Transform) als auch für die Rastergraphik (z.B. Material, Texture, Adressen und Längen der Output-Buffer).

Beim Start muss jedes Direct3D-Programm diese Klasse instanziieren und erhält dadurch Ressourcen und Zugriffsrechte auf die Graphikkarte. Leider sind beide nicht von Dauer, sie können jederzeit verloren gehen und müssen dann zurückgefordert werden. Man merkt den Verlust der `Device` meist dadurch, dass `Device.Present` eine `DeviceLostException` wirft und nicht mehr funktioniert. Die Probleme, die zum Verlust des `Device` führen:
1) Der User verändert mit der Maus die Fenstergröße des Programms: `Form.Resize`.
2) Ein Bildschirmschoner übernimmt exklusiv die Graphikkarte.
3) Das Windows-Betriebssystem übernimmt exklusiv die Graphikkarte.
4) CPU oder Graphikkarte wechseln in Standby-Betrieb.
5) Der Deckel des Notebooks wird zu- und aufgeklappt.
In den Übungen von **Course3DCis** ist Problem 1) dadurch beseitigt, dass die Initialisierung von `Device` in den `OnResize()`-EventHandler verlegt ist.
Vorsicht: Damit alles einfach bleibt, sind die Probleme 2) bis 5) ungelöst. Professionelle Wege finden Sie unter **http://pluralsight.com/wiki/default.aspx/Craig.DirectX/DeviceRecoveryTutorial.html** und unter **www.jkarlsson.com/Articles/devicelost.asp**

Important **Properties**	of Direct3D class "**Device**"
DeviceCaps	Gets a struct representing the capabilities of the hardware; this is the property to query when determining whether the hardware supports the particular features that an application may require
Viewport	Gets/sets the rectangular region on the device on which to render
Material	Gets/sets the material to use in rendering
Lights	Gets the collection of lights that can be activated for rendering
RenderState	Gets the collection of render states that are used to control the different stages of the Direct3D pipeline
VertexDeclaration	Gets/sets a description of the vertex format being used with a vertex shader
VertexFormat	Gets/sets a description of the vertex format being used with the Fixed Vector Pipeline
VertexShader, PixelShader	Gets/sets the vertex/pixel shader to use for rendering

Important **Methods**	of Direct3D class "**Device**"
BeginScene	Prepares the device to render a frame of primitives; BeginScene must be called before any primitive is rendered for the frame
EndScene	Signals to the device that all the primitives have been rendered for a frame; EndScene must be called after all the primitives are rendered for the frame
DrawPrimitives	Renders a primitive
Clear	Clears the viewport in preparation for another frame of rendering
Present	rendered into and prepares the next buffer for rendering; Present is called after EndScene and before the next BeginScene (for the next frame)
GetTransform, SetTransform	Gets/sets the world, view, projection or other transform; transforms are applied to vertex positions and normals, and/or to texture coordinates
GetTexture, SetTexture	Gets/sets the texture associated with a given texture stage

Managed DirectX Programmierung von Device
(Unter XNA ist die Device-Programmierung viel einfacher und unter WPF ist sie ganz versteckt.)

Beispiel vor dem Initialisieren von "Device": Abfrage der verfügbaren Pixelformate und Bildwiederholraten

```
1: StringBuilder s = new StringBuilder();
2: AdapterInformation ai = Manager.Adapters(0);
3: foreach DisplayMode dm in ai.SupportedDisplayModes
4:   s.Append( dm.Format + " " + dm.RefreshRate + "\r\n" );
```

Zeile 2: Graphikkarte Nr. 0 nehmen. (Es können 2 auf dem Bus stecken !)
Zeile 3: Enumeration aller verfügbaren `SupportedDisplayModes` durchlaufen.
Zeile 4: Zeile anhängen: `Format`-String + Blank + `RefreshRate`-String + Wagenrücklauf + neue Zeile.

Beispiel Initialisieren von "Device":
Eine solche Initialisierung ist notwendig bei jeder Größenänderung des Fensters (`OnResize`-Ereignis), weil dabei die alte `Device` verlorengeht.

```
1: presentParams = new PresentParameters();
2: presentParams.Windowed = true;
3: presentParams.SwapEffect = SwapEffect.Discard;
4: Device device = new Device( 0, DeviceType.Hardware, this,
CreateFlags.SoftwareVertexProcessing, presentParams );
```

Zeile 1: Speicherplatz reservieren für die Struktur `PresentParams`
Zeile 2: Festsetzen, dass Direct3D in einem Fenster ablaufen soll. Alternative: ganzes Display
Zeile 3: `SwapEffect` ausschalten.
Zeile 4: neues `Device`: Speicherplatz reservieren und mit den vorgefundenen
und einigen selbstbestimmbaren Eigenschaften füllen.

Beispiel Benutzen von "Device":

```
1: device.Clear( ClearFlags.Target, Color.Blue, 1.0f, 0 );
2: device.BeginScene();
3:   (Mesh.Teapot( device )).DrawSubset( 0 );
4: device.EndScene();
5: device.Present();
```

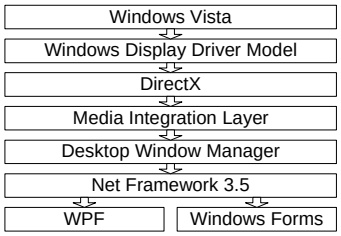
Zeile 1: Zeichenfläche im BackBuffer löschen.
Zeile 2: Klammer auf
Zeile 3: Teekanne in den BackBuffer zeichnen.
Zeile 4: Klammer zu
Zeile 5: BackBuffer und FrontBuffer flippen.

XNA

XNA ist ein auf Spieleprogrammierung spezialisiertes Derivat von DirectX. Es vereint moderne Komponenten:
1. Visual Studio 2005 als Entwicklungsumgebung
2. C# als Programmiersprache
3. Auf DirectX 10 basierende Bibliotheken speziell für Games
4. Spiele sind sowohl auf Windows-PCs als auch auf Xbox 360-Konsolen lauffähig
5. Auch für Anfänger geeignet, siehe **../../C_3D_XNA/Index_of_Course.htm**

Weiterführende Links:
1. **http://msdn.microsoft.com/directx/XNA**
2. **http://msdn.microsoft.com/directx/xna/videos**
3. **http://en.wikipedia.org/wiki/Microsoft_XNA**
4. **http://xbox360homebrew.com/content/XNATutorialMasterList.aspx**
5. **http://entwickler.de/zonen/portale**

DirectX und Windows Vista



DirectX10 wird zusammen mit und exklusiv für Windows Vista ausgeliefert.
DirectX10 stellt insofern einen Quantensprung dar, als es nicht mehr ein Zusatz zum Betriebssystem, sondern integraler Bestandteil von Vista ist. Vista läuft überhaupt nicht mehr ohne DirectX10.
Microsoft schreibt den Graphikchip-Herstellern detailliert vor,
was Chip und Treiber als Vermittler zwischen Vista und DirectX leisten muss.
Siehe:
Windows Display Driver Model WDDM
Vista Display Driver Model
Vista Driver Development

Vorteile:
1) Man kann sich darauf verlassen, dass Vista die Graphikhardware voll nutzt.
   Es gibt weder DDI noch HEL, sondern nur noch HAL.
2) Man kann sich darauf verlassen, dass jeder Vista-fähige Graphiktreiber
   den minimalen Befehlssatz "Direct3D 10 compatible" anbietet.
3) Das User Interface von Vista bietet schnelle Hochqualitätsgraphik, Transparenz, Animation, 3D und Video.
4) Internet-Seiten können über WPF schnelle DirectX-Graphik nutzen.

Nachteile:
1) Alte Graphikkarten, Drucker, Scanner (ohne DirectX10-Treiber) laufen nicht unter Vista.
2) Alte DirectX-Spiele laufen nicht unter Vista.

Weiterführender Link: **Direct3D and Vista**

Windows Presentation Foundation WPF

DirectX war früher ein schmutziger Trick zur Umgehung des langsamen Betriebssystems.
Mit Vista ist es aber zur zentralen Schnittstelle jedweder graphischen Ausgabe aufgestiegen.
Die Grundlagen der Vista-Graphik sind:
a) der Desktop Window Manager = DWM
b) das Windows Vista Display Driver Model = WVDDM oder kürzer WDDM,
   das alle Vista-Graphikkarten unterstützen müssen.
Ablauf:
1.) Die Fenster und Graphiken (auch Schrift) von Vista sind Vektorgraphik.
2.) Die Vektor-Graphik-Daten und Befehle werden vom DVM zwischengespeichert, verwaltet und positioniert.
3.) Der DVM übergibt Daten und Befehle dem WVDDM-Graphiktreiber.
4.) Der Treiber wandelt alles in DirectX-Daten und Befehle und lädt diese in seine Graphikkarte.
5.) Die Graphikpipeline der Graphikkarte rendert autonom (d.h. ohne Beteiligung der CPU)
    mit maximaler Geschwindigkeit in den Back-Buffer.
6.) Ist der Back-Buffer komplett, schaltet die Graphikkarte diesen dann (als Front-Buffer) zum Bildschirm durch.

WPF ist die Programmierschnittstelle (Application Programming Interface API) von Windows Vista.
Mit WPF kann man sowohl eigenständige EXE-Programme als auch Frontends von verteilten Anwendungen und Browser-Anwendungen schreiben. WPF enthält genau genommen zwei APIs, die sich gegenseitig ergänzen und die man frei mischen kann: Man kann in C# programmieren oder in XAML oder beidem. WPF wird die bisherige Windows-Forms- und Active-Server-Page-Programmierung aufsaugen und ersetzen.
Vorteile:
1) WPF generiert konsequent Vektorgraphik (bis auf Texturen und Videos) wie Flash.
2) WPF rendert mit DirectX. Folge: Animation, Transparenz, Anti-Aliasing sind viel schneller als bei Flash.
3) WPF ist ein reichhaltiger und vorbildlich strukturierter Objekt-Baukasten für Windows- und Web-GUIs.
4) WPF everywhere bietet eine einheitliche Oberfläche für Windows, Web-Seiten und mobile Geräte.
5) Mit Expression Studio können dumme Nicht-Informatiker Oberflächen und Web-Seiten zusammenklicken.
Nachteile:
1) WPF geht nur unter Windows (Ausnahme: Silverlight siehe unten).
2) WPF ist eine reiche und tief gestaffelte Klassenbibliothek und schwer zu lernen.
3) WPF verleitet zu nutzlosen graphischen Spielereien jeder Art.
Beispiele: Course 2D WPF

Silverlight ist eine kleine Untermenge von WPF in Form eines Plugins, das für fast alle Browser verfügbar ist. Eine mit Silverlight 2.0 erstellte Seite besteht aus XAML und C# Code. Man kann damit WPF-Programme schreiben, die in den gängigen Browsern ohne Installations- und Sicherheitsprobleme auf allen Plattformen lad- und ausführbar sind. Siehe: Course 2D_SL und Silverlight 2.0 Poster.