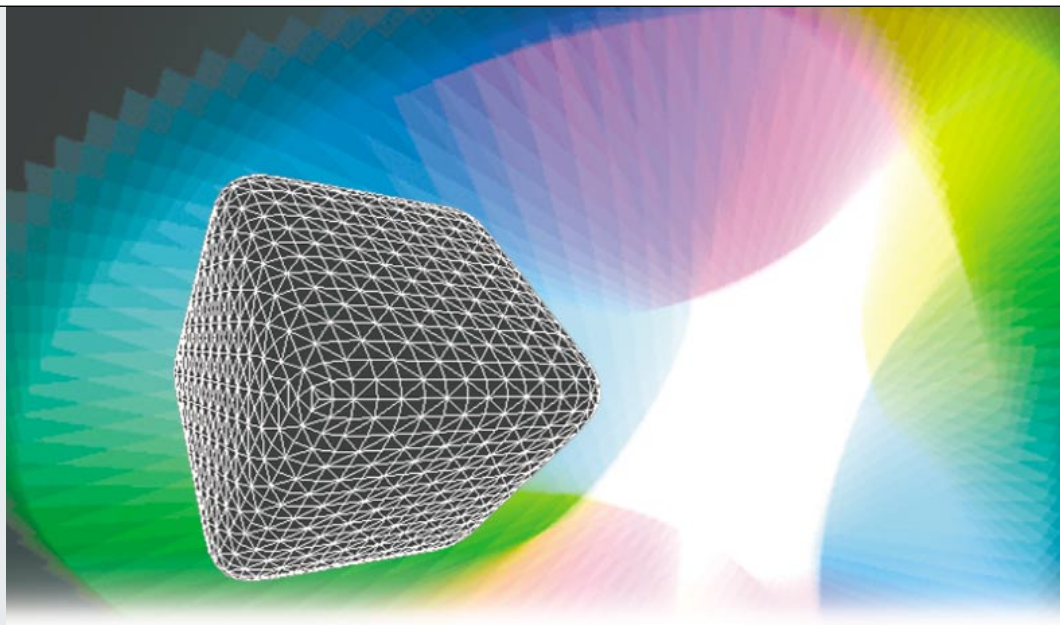
Workshop

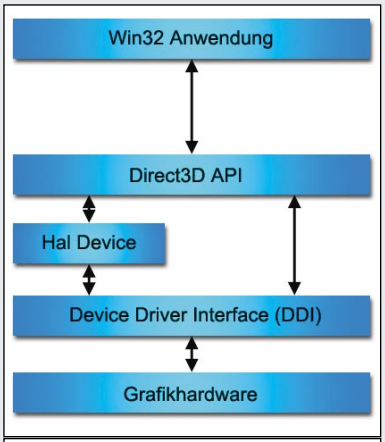


Direct-3D-9

Kleine Schritte zur großen Grafik

Die Direct-3D-Komponente des vor kurzem vorgestellten DirectX 9 bietet eine einheitliche Schnittstelle, um Grafikbeschleuniger zu programmieren.

Carsten Dachsbacher



Schichtenmodell: Die Bestandteile beim Rendering von 3D-Objekten.

Das vor kurzem erschienene DirectX 9 enthält – neben den aktualisierten Komponenten DirectSound, DirectMusic etc. – auch ein überarbeitetes Direct3D-Interface, das die neueste Grafikhardware nutzen kann. Grund genug, Direct3D9 einen Platz in der Reihe der PC-Underground-Artikel zu reservieren. Wir zeigen Ihnen in dieser Ausgabe, wie Sie Direct3D9 sowohl für eine Fenster- als auch Vollbild-Anwendung korrekt initialisieren und verwenden. Damit legen Sie die Grundlage für weitere Programme und Grafikeffekte.

Für eine Direct3D-Anwendung benötigen Sie zunächst ein normales Windows-Fenster. Dieses können Sie mit den MFC (Microsoft Foundation Classes) anlegen, wenn Sie nicht einfach die Win32-API verwenden wollen. Wir haben den zweiten Weg gewählt, da dieser einfacher zu überschauen ist und weniger Overhead verursacht. Somit entsteht ein einfaches Framework für Direct3D-Anwendungen.

Fenster auf

Wir zeigen Ihnen hier die vollständige *WinMain*-Funktion des Programms, in der Sie zunächst eine eigene Fensterklasse anlegen. Dazu füllen Sie die Felder der *WNDCLASSEX*-Struktur aus. Darin sind alle Informationen über den Stil, Cursor, Icon usw. der Fensterklasse enthalten. Unter anderem müssen Sie auch einen Zeiger auf die Window-Prozedure angeben. Diese Funktion bearbeitet alle Nachrichten wie Mausklicks und Tastatureingaben, die an ein Fenster verschickt werden. Das Beispielprogramm fragt diesen Klick auf den *Schliessen*-Button des Fensters oder ein Drücken der *[Esc]*-Taste ab und verschickt gegebenenfalls eine *WM_QUIT*-Nachricht. Die Fensterklasse registrieren Sie dann mit *RegisterClassEx*:

```
int WINAPI WinMain(
    HINSTANCE hInst,
    HINSTANCE hPrevInst,
    LPSTR     commandLine,
    int       commandShow )
{
  WNDCLASSEX wndClass;
  MSG        msg;

  // wndClass Struktur ausfüllen
wndClass.lpszClassName = "PCUvsD3D9";
 wndClass.lpfnWndProc   = WindowProc;
  ...

  // Fensterklasse registrieren
if( RegisterClassEx( &wndClass )==0 )
      return E_FAIL;
```

Wenn die Fensterklasse registriert ist, erzeugen Sie Ihr Direct3D-Fenster und bringen es auf den Bildschirm:

```
  gHWND = CreateWindowEx(
     NULL, "PCUvsD3D9", "Direct3D9",
    WS_OVERLAPPEDWINDOW | WS_VISIBLE,
```

```
   0, 0, 640, 480, NULL, NULL, hInst,
     NULL );

  if( gHWND == NULL ) return E_FAIL;

  ShowWindow( gHWND, commandShow );
  UpdateWindow( gHWND );
```

Mit den folgenden Direct3D-Programmteilen verbinden Sie drei Funktionen, die initialisieren, rendern und die Ressourcen freigeben: *initialize3D(), render3D() und shutdown3D()*. Diese finden Sie im letzten Teil der *WinMain*-Funktion, die fortwährend die Rendering-Funktion aufruft, bis Sie das Programm beenden:

```
 // Initialisierung
  initialize3D();

  ZeroMemory( &msg, sizeof( msg ) );

  // render3D(), bis zum Programmende
  while( msg.message != WM_QUIT )
  {
   if( PeekMessage( &msg, NULL, 0, 0,
                    PM_REMOVE ) )
    {
      TranslateMessage( &msg );
      DispatchMessage( &msg );
    } else
      render3D();
  }

  // und aufräumen
  shutdown3d();

  UnregisterClass("MY_WINDOWS_CLASS",
                  wndClass.hInstance);

  return msg.wParam;
}
```

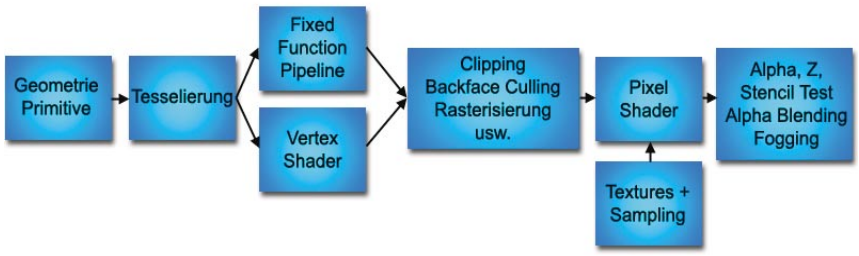
Die obige *WinMain*-Funktion erzeugt ein Fenster. Für eine Vollbild-Anwendung ist an dieser Stelle nur ein anderer *CreateWindowEx*-Aufruf notwendig:

```
  gHWND = CreateWindowEx(
     NULL, "PCUvsD3D9", "Direct3D9",
     WS_POPUP|WS_SYSMENU|WS_VISIBLE,
  0, 0, 640, 480, NULL, NULL, hInst,
     NULL );
```

Direct3D im Fenster

Bei der Initialisierung von Direct3D mit der jeweiligen *init3d()*-Funktion sind die Unterschiede von Fenster- und Vollbild-Betrieb schon größer. Deshalb initialisieren Sie zuerst Direct3D für den Fenstermodus und anschließend für Vollbildanwendungen.

Als erstes erzeugen Sie sich mit *Direct3DCreate9(...)* eine Instanz eines *IDirect3D*-Objekts. Der Parameter lautet dabei immer *D3D_SDK_VERSION*. Dies dient dazu, für das Rendering Direct3D-Objekte zu erzeugen, deren Fähigkeiten auszulesen, Grafikmodi aufzulisten und die Parameter einzustellen. Achten



Rendering Pipeline: Nach diesem Schema arbeitet jeder Grafikbeschleuniger. Das aktuelle DirectX9 bleibt allerdings nur der neuesten Grafikhardware vorbehalten.

Sie darauf, Fehler abzufragen, um einen Programmabsturz zu vermeiden. In unserem Beispielcode übernimmt dies die fiktive Funktion *error()*:

```
LPDIRECT3D9       pD3D      = NULL;
LPDIRECT3DDEVICE9 pD3DDevice = NULL;

pD3D = Direct3DCreate9
          ( D3D_SDK_VERSION );

if ( pD3D == NULL ) error();
```

Da Sie im Fensterbetrieb keinen neuen Grafikmodus festlegen, lesen Sie die Parameter des aktuellen aus. Diese sind neben Breite, Höhe und Bildwiederholfrequenz ein Format-Parameter, alles verpackt in eine *D3DDISPLAYMODE*-Struktur. Der Format-Parameter enthält z.B. die Farbtiefe.

```
D3DDISPLAYMODE dm;

if( FAILED(
      pD3D->GetAdapterDisplayMode(
            D3DADAPTER_DEFAULT, &dm )
  )       )
 error();
```

Jetzt prüfen Sie, ob das Direct3D-Gerät (Standard Device, identifiziert durch *D3DADAPTER_DEFAULT*) die Programmanforderungen erfüllen kann wie z.B. eine bestimmte Z-Buffer-Genauigkeit. Solche Format bezogenen Details fragen Sie mit *CheckDeviceFormat* ab:

```
HRESULT hr;

hr = pD3D->CheckDeviceFormat(
  D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
    dm.Format, D3DUSAGE_DEPTHSTENCIL,
    D3DRTYPE_SURFACE, D3DFMT_D16 );

if (hr==D3DERR_NOTAVAILABLE) error();
```

Der zweite Parameter (*D3DDEVTYPE_HAL*) steht für ein Hardware beschleunigtes Direct-3D-Device. Sie könnten ihn z.B. durch *D3DDEVTYPE_REF* ersetzen, um den Software Referenz Rasterizer zu verwenden.

Die Fähigkeiten einer Grafikkarte, die so genannten *Caps* (Capabilities) fassen Sie in einer *D3DCAPS9*-Struktur zusammen. Darin sind alle Features enthalten, deren umfangreiche Liste im DirectX9-SDK dokumentiert ist.

```
D3DCAPS9 caps;

if( FAILED(
      pD3D->GetDeviceCaps(
                D3DADAPTER_DEFAULT,
                D3DDEVTYPE_HAL,
                &caps
  )       )
    error();
```

Die weiterhin benötigten Caps beschreiben, ob die Grafikkarte Vertex Processing (also Transformation, Beleuchtung usw.) in Hard- oder Software ausführt. Diese Information verwenden Sie, um die *Behaviour Flags* zu setzen:

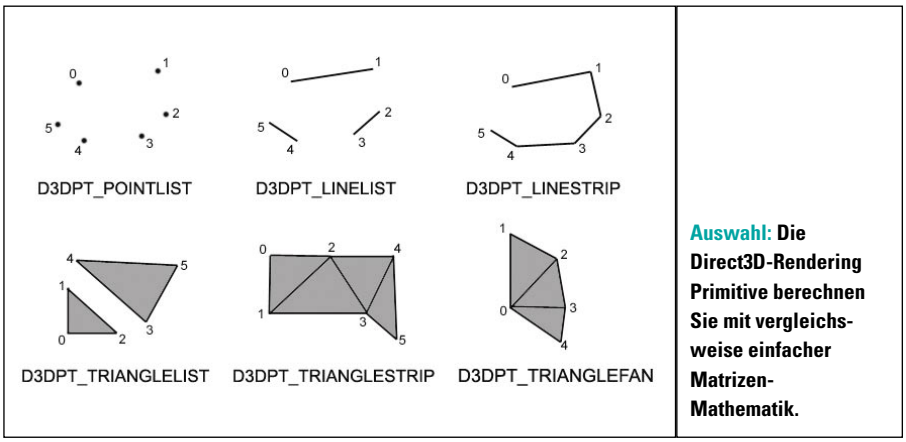
```
DWORD flags;

if( caps.VertexProcessingCaps != 0 )
  flags |=
 D3DCREATE_HARDWARE_VERTEXPROCESSING;
else
  flags |=
 D3DCREATE_SOFTWARE_VERTEXPROCESSING;
```

Als letzte Aufgabe der Initialisierung erzeugen Sie das Direct3D-Device. Dazu benötigen Sie noch die so genannten *Presentation Parameters*. Diese beschreiben z.B. die Anzahl der Backbuffers und deren Format oder das Z-Buffer-Format. Im Falle des Fensterbetriebs, müssen Sie nicht alle Parameter setzen:

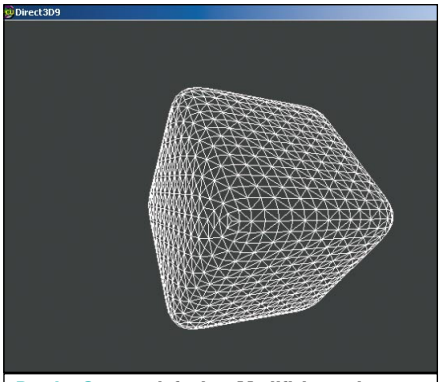
```
D3DPRESENT_PARAMETERS pp;
```

Direct3D-Vollbild

Die Initialisierung eines Vollbild-Direct3D-Modus unterscheidet sich prinzipiell in einem Punkt: Sie sind nicht darauf angewiesen, den gerade aktuellen Grafikmodus des Desktops zu verwenden, sondern Sie können sich einen Modus aussuchen. Dazu fordern Sie eine Liste aller unterstützten Grafikmodi an, die eine *D3DDISPLAYMODE*-Struktur beschreibt. Die Anzahl der Modi (hier mit 32-Bit-Farbtiefe, bestimmt durch *D3DFMT_X8R8G8B8*):



Auswahl: Die Direct3D-Rendering Primitive berechnen Sie mit vergleichsweise einfacher Matrizen-Mathematik.



Render States: einfaches Modifizieren des Rendering

```
int nMaxModes =
  pD3D->GetAdapterModeCount(
    D3DADAPTER_DEFAULT,
    D3DFMT_X8R8G8B8 );
```

Jetzt überprüfen Sie alle *nMaxModes*, bis Sie einen gewünschten gefunden haben. Hierzu prüfen Sie für jeden Modus die Breite, Höhe, Bildwiederholfrequenz und die Format-Flags:

```
D3DDISPLAYMODE dm;
bool foundMode = false;

....

if ( foundMode == false )
  // kein passender Modus gefunden
  exit();
```

Der letzte Aspekt, den Sie beim Vollbildbetrieb noch beachten müssen, ist die Abfrage, ob für den gewählten Grafikmodus Hardware-Beschleunigung zur Verfügung steht. Die ersten beiden Parameter bezeichnen dabei wie gehabt das Direct3D-Device, gefolgt von den Formaten für Frame- und Back-Buffer und einem *FALSE* für Nicht-Fenster-Betrieb.

```
if ( FAILED(
       pD3D->CheckDeviceType(
           D3DADAPTER_DEFAULT,
           D3DDEVTYPE_HAL,
           D3DFMT_X8R8G8B8...
```

Der Rest der Initialisierung, d.h. die Überprüfung der Caps, machen Sie so wie zuvor beschrieben.

Direct3D-Shutdown

Der Vollständigkeit halber zeigen wir Ihnen an dieser Stelle, wie Sie Direct3D wieder korrekt verlassen. Dies beschränkt sich lediglich auf zwei Aufrufe, die das Direct3D-Device und Objekt freigeben:

```
void shutdown3D()
```

Nach der Initialisierung können Sie sich nun endlich der Rendering Schleife Ihres Programms widmen, die Sie in der *render3D()*-Funktion implementieren. Diese ist fest nach Schema aufgebaut: Als erstes löschen Sie den Frame-, Depth- und/oder Stencil-Buffer, je nach dem, was Sie für das Device angefordert haben. Wenn Sie jeweils den ganzen Buffer löschen wollen, sind die ersten beiden Parameter *0* bzw. *NULL*. Welcher Buffer betroffen ist, legen Sie im dritten Parameter durch eine *Oder*-Verknüpfung der *D3DCLEAR*-Flags fest. Die Farb-, Tiefen- und Stencil-Werte, welche die Buffers beschreiben, bilden die letzten drei Parameter:

```
pD3DDevice->Clear( 0, NULL,
  D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,
  D3DCOLOR_COLORVALUE( 0, 1, 0, 1 ),
  1.0f, 0 );
```

Alle folgenden Rendering-Vorgänge befinden sich zwischen den *BeginScene*- und *EndScene*-Aufrufen:

```
pD3DDevice->BeginScene();
// Rendering !
pD3DDevice->EndScene();
```

Zuletzt bringen Sie den Inhalt des Backbuffers, also das Resultat des Renderings auf den Bildschirm. Da Sie auch hier jeweils den ganzen Buffer sehen wollen, sind alle Parameter *NULL*:

```
pD3DDevice->Present
 ( NULL, NULL, NULL, NULL );
```

Transformationen

Für das Rendering benötigen Sie Transformationen, die Sie über die *SetTransform*-Methode Ihres Direct3D-Device setzen. Es gibt eine Transformation (definiert durch eine 4x4 Matrix) für die 3D-2D-Projektion (*D3DTS_PROJECTION*), eine Kamera-Abbildung (*D3DTS_VIEW*) und die so genannte World-Transformation (*D3DTS_WORLD*), die die Transformation eines Objektes in den World Space angibt. Zwar gibt es mehrere dieser World Matrizen, um Vertex Blending bei Animationen zu verwenden, doch bleibt das für unseren Einsatz zunächst nebensächlich. Die Transformationen bilden nur einen kleinen Teil der Rendering Pipeline und sind für die *Fixed Function Pipeline* relevant. Dieser Teil übernimmt die normale Transformations- und Beleuchtungsberechnung.
Um solche Transformationen elegant zu handhaben, verwenden Sie am besten die Direct3D-Erweitungen (*D3DX*). *D3DX* sammelt umfangreiche Direct3D-Hilfsroutinen für vielfältige Zwecke wie für Mathematik und Texturen. Darin ist u.a. der *D3DXMATRIX*-Typ definiert, der eine 4x4-Matrix darstellt. Außerdem verfügen Sie damit über zahlreiche Methoden, um Matrizen zu erzeugen und zu berechnen. Um eine Matrix für eine perspektivische Abbildung zu erhalten, platzieren Sie die folgenden Code-Fragment in der Render-Schleife:

```
D3DXMATRIX mProjection;
```

Für die World Matrizen können Sie z.B. die Funktionen *D3DXMatrixTranslation* / *D3DXMatrixRotationAxis* verwenden, um Abbildungen zu verschieben oder zu drehen. Die Kamera-Matrix erzeugen Sie intuitiv mit *D3DXMatrixLookAtLH*.

Rendering Primitive

Jetzt haben Sie eine vollständige Umgebung geschaffen, um geometrische Primitive zu rendern. Damit sind Punkte, Linien, Dreiecke usw. gemeint. Die von Direct3D unterstützten Primitive sehen Sie im Bild.
Am besten rendern Sie mit den so genannten Vertex Buffers. Unter einem Vertex Buffer können Sie sich einen Speicherbereich vorstellen, der nur Vertex Daten wie z.B. die Eckpunkte eines Dreiecksnetzes und damit assoziierte Daten enthält. Das Format der Vertices kann dabei sehr unterschiedlich sein: untransformiert,

transformiert, beleuchtet oder nicht, mit oder ohne Textur-Koordinaten usw. Das Format beschreiben Sie über das *Flexible Vertex Format* (FVF). Die möglichen Vertex-Attribute sehen Sie in der Tabelle.
Um einen Vertex Buffer anzulegen, legen Sie zunächst das Format fest und erzeugen entsprechende Daten (hier zum Beispiel mit einem konstanten Array), indem Sie eine Vertex-Struktur und die dazugehörigen *FVF*-Flags (eine Kombination der *D3DFVF*-Konstanten) definieren:

```
#define FVF_VERTEX3D
     ( D3DFVF_XYZ | D3DFVF_DIFFUSE )
```

Während der Initialisierung von Direct3D erzeugen Sie den Vertex Buffer:

```
LPDIRECT3DVERTEXBUFFER9 pDreieckVB;
```

Der erste Parameter gibt die Größe des Vertex Buffers in Bytes an. Mit dem zweiten Parameter können Sie so genannte *D3DUSAGE*-Parameter spezifizieren, z.B. um den Vertex Buffer *write-only* zu deklarieren. Sie sollten die Fähigkeiten eines Vertex Buffers immer so weit wie möglich einschränken, um eine größtmögliche Performance zu erzielen! Der nächste Parameter gibt Auskunft über das *FVF*, gefolgt von einer *D3DPOOL*-Konstante, die bestimmt, in welchem Speicherbereich (z.B. Haupt-oder Grafikkarten-Speicher) der Vertex Buffer abgelegt wird. Wenn Sie dafür die Funktion *D3DPOOL_MANAGED* wählen, können Sie nichts falsch machen: Direct3D kümmert sich um die Daten, platziert Sie am sinnvollsten und behält ein Backup im Systemspeicher. Der vorletzte Parameter ist ein Zeiger auf das Vertex Buffer Interface, das mit dem Vertex Buffer assoziiert ist. Der letzte Parameter ist immer *NULL*.
Nun können Sie Ihre Daten in den Vertex Buffer kopieren. Dazu müssen Sie diesen verschließen (lock). Sie erhalten einen Zeiger auf einen Speicherbereich, in den Sie die Daten schreiben:

```
VERTEX3D *pData = NULL;
```

Nachdem Sie die obigen Schritte während der Initialisierung vorgenommen haben, können Sie in der Render-Schleife das Dreieck auf den Bildschirm bringen. Dazu müssen Sie Direct3D zwei Dinge – jeweils vor dem Rendering-Kommando – mitteilen: Wo sind die Daten, also welcher Vertex Buffer wird gerade verwendet, und welches Format haben die Daten?
Das Rendering Kommando lautet dann für ein Primitiv ab der Position *0* im Vertex Buffer:

```
pD3DDevice->DrawPrimitive(
  D3DPT_TRIANGLELIST, 0, 1 );
```

In der *shutdown3D()*-Funktion geben Sie die Ressourcen des Vertex Buffers bei Programmende wieder frei. Dies muss erfolgen, bevor Sie das Direct3D-Device freigeben:

```
pDreieckVB->release();
```

Render States

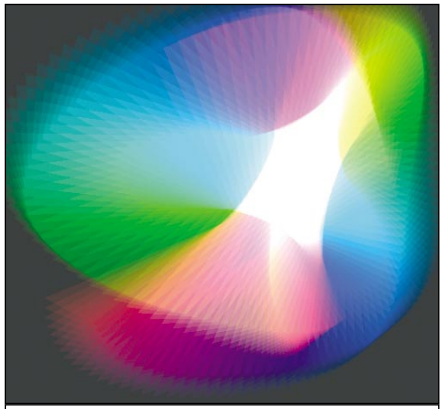
Wie Sie vielleicht von OpenGL wissen, gibt es eine riesige Anzahl von so genannten Render States: Zustände bzw. Variablen, deren Wert das Rendering beeinflusst. Zum Beispiel *Culling Modi*, Z-Buffer oder Alpha Tests, Beleuchtungsparameter usw. Alle diese Einstellungen sind in Direct3D in der *SetRenderState*-Methode des Direct3D-Device Objektes zusammengefasst. Diese Methode akzeptiert zwei Parameter: Der erste gibt an, welchen *State* (*D3DRS*-Konstante) Sie modifizieren wollen, gefolgt von einem Wert. Dieser kann dabei entweder ein numerischer Wert oder eine vordefinierte Konstante sein. Sämtliche Render States listet wiederum das DirectX-SDK auf.
Unser Beispielprogramm verwendet die Render States z.B., um zwischen dem Rendering von ausgefüllten Dreiecken und Dreieckskanten zu wählen:

```
// ausgefüllt
pD3DDevice->SetRenderState(
 D3DRS_FILLMODE, D3DFILL_SOLID );
pD3DDevice->SetRenderState(.....
```

Mit den Render States für Alpha Blending erzeugt unser Beispielprogramm weitere interessante Effekte.
Ein Direct3D-Device kann sich entweder im *Betriebszustand* oder im *Lost State* befinden. Letzteres tritt z.B. ein, wenn einer Vollbild-Anwendung der Fokus (z.B. durch Drücken von *Alt-Tab*) entzogen wird oder auch durch Power Management Funktionen. Im Lost State haben Rendering Kommandos keinen Effekt, obwohl Sie *D3D_OK* als Rückgabewert liefern. Der Lost State ist nur am *D3DERR_DEVICELOST*-Rückgabewert der *Present(...)*-Methode zu erkennen.

Lost Devices

Dieses Ereignis müssen Sie in Ihrem Programm abfragen und warten, bis das Device wiederhergestellt werden kann. Anschließend sind alle Ressources im Video-Speicher freizugeben und neu zu erzeugen. Der benötigte Programmcode für die Wiederherstellung ist dabei ähnlich oder sogar identisch, um Vertex Buffers und anderer Ressourcen zu initialisieren. Dieser Vorgang ist aber nicht notwendig, wenn Sie die Ressourcen mit *D3DPOOL_MANAGED* angelegt haben. Deshalb können wir die detaillierte Behandlung der Lost Devices zunächst außen vor lassen.
Mit dem Beispielprogramm haben Sie so die Grundlagen geschaffen, um saubere Direct3D-Programme zu entwickeln, die die Leistung Ihrer Grafikkarte ausschöpfen können. Es dient als Basis für weitere PC-Underground-Programme, welche Sie schrittweise ausbauen. : et



Alpha Blending: Farbeffekte durch Akkumulation der Farbwerte

Flexible Vertex-Formats

Vertex-Attribut	transformierte/untransformiert Vertices	Daten
Vertex Position	✔/✔	X, Y, Z (Float)
RHW	✔/-	RHW (Float)
Blending Gewichte	✔/✔	1, 2 oder 3 Floats/DWORD
Vertex Normale	-/✔	Nx, Ny, Nz (Float)
Vertex Punktgröße	✔/✔	1 Float
Farbe Diffus	✔/✔	RGBA (DWORD)
Farbe Spekular	✔/✔	RGBA (DWORD)
8x Textur Koordinaten	✔/✔	1 bis 4 Floats