DirectX und OpenGL

Proseminar Multimedia-Hardwareerweiterungen

Michel Weimerskirch

Zusammenfassung Diese Arbeit, die im Rahmen des Proseminars *Multimedia Hardwareerweiterungen* ausgearbeitet wurde, soll einen allgemeinen Überblick über DirectX und OpenGL verschaffen.

Nach einer kurzen Einführung im ersten Kapitel liefern die Kapitel 2.1 und 2.2 eine Übersicht über die Hauptfunktionalität von DirectX und seiner Teilkomponenten beziehungsweise über die Hauptfunktionalität von OpenGL. Anschliessend wird in den Kapiteln 3.1 und 3.2 der sprachliche Aufbau von OpenGL sowie von Direct3D, der Grafik-Komponente von DirectX, erläutert. Diese beiden Kapitel schliessen jeweils mit einem veranschaulichendem Beispiel ab.

Abschliessend verdeutlichen die folgenden zwei Kapitel wichtige Aspekte der besprochenen APIs. Es handelt sich dabei um die Utility-Toolkits sowie die unterstützten Plattformen.

Inhaltsverzeichnis

1 Einführung

Millions of years ago, a group of our ancestors celebrated the invention of the wheel. The new invention was no doubt put into use right away. Today low cost real-time hardware accelerated 3D graphics is available for the average PC, and it's time to put it in use. ([8], S. XX)

Als vor etwas 20 Jahren die ersten Graphikkarten mit 3D-Unterstützung den Heimbereich erreichten, herrschte noch ein Mangel an einheitlichen APIs, um die noch recht neue Technik vollständig zu nutzen. Die heute wohl am meisten benutzen Multimedia-APIs DirectX und OpenGL haben seit den ersten Versionen Mitte der 90er Jahre beide einen hohen Reifegrad sowie eine weite Verbreitung erreicht. Handelt es sich bei DirectX um ein kommerzielles Produkt, das nur für Microsoft-Plattformen verfügbar ist, so existieren für das etwas ältere OpenGL Implementierungen für die verschiedensten Plattformen von Microsoft Windows über Apple Macintosh bis hin zu vielen Unix-Plattformen und Linux.

Die beiden APIs unterscheiden sich jedoch keineswegs nur durch die unterstützen Plattformen, sondern, wie im folgenden gezeigt, vor allem durch den sprachlichen Aufbau. Nach einer Übersicht über die DirectX-Komponenten und über das OpenGL-Framework in Abschnitt 2.1 beziehungsweise 2.2, wird in Abschnitt 3 ein Vergleich des sprachlichen Aufbaus von OpenGL und Direct3D, der Grafik-Komponente von DirectX, durchgeführt.

2 Übersicht über DirectX und OpenGL

2.1 Übersicht über DirectX und seine Komponenten

DirectX wurde 1995 in einer ersten Version veröffentlicht. Das damals noch unter dem Namen *Windows Games SDK* laufende Produkt diente dazu, Windows den entscheidenden Schub in den Markt der Spieleentwicklung zu geben. Durch das "protected memory model" von Windows war ein direkter Zugriff auf die Hardware nämlich nicht mehr ohne weiteres möglich, was einen erheblichen Performanceverlust für Spiele bedeutet hätte. Um Windows dennoch auch für die Spieleentwicklung attraktiv zu machen, wurde die *Windows Games SDK*, die wenig später unter dem Namen *DirectX* vermarktet wurde, entwickelt. (vgl. [10])

Die aktuelle Version von DirectX, Version 9.0c, besteht aus den folgenden Teilkomponenten (vgl. [4]):

- **DirectX Graphics** besteht aus der Teilkomponente **Direct3D** sowie der **D3DX Utility Library**, durch die viel graphische Programmierarbeit gespart werden kann.
- **DirectInput** erlaubt die Hardware-nahe Kommunikation mit einer Vielzahl von Eingabegeräten wie Joysticks, Lenkräder oder ähnliches. Sie bietet ausserdem eine volle Unterstützung der force-feedback Technologie von Microsoft.

- **DirectSound** wird bei der Entwicklung von hochperformanten Audio-Anwendungen genutzt.
- **DirectMusic** wird zur Wiedergabe von synthetischer Musik benutzt (beispielsweise im MIDI-Format).

Neben diesen aktuellen Komponenten existiert noch eine Reihe älterer Komponenten, deren Benutzung nicht mehr empfohlen wird:

- Die **DirectDraw** Komponente die bisher für die zweidimensionale Darstellung verwendet wurde, wird seit der Version 9c von DirectX nicht mehr empfohlen. Die zweidimensionale Darstellung wird nun ebenfalls von *Direct3D* sowie den Hilfsfunktionen aus der *D3DX Utility Library* übernommen.
- Die **DirectPlay** Komponente wurde bisher zur Netzwerk-Kommunikation bei Online-Spielen benutzt. Microsoft empfielt die Verwendung dieser Komponente jedoch nicht mehr, und rät stattdessen zur Benutzung von *Windows Sockets* sowie den *Windows Firewall API*s.
- Die **DirectShow** Komponente erlaubt die Verarbeitung von Audio- und Video-Streams. Von der Verwendung der Komponente in der Spieleentwicklung wird seit der aktuellen Version von DirecX abgeraten.

2.2 Übersicht über OpenGL

OpenGL steht für **"Open Graphics Library"** und bezeichnet eine Software-Schnittstelle zur Grafik-Hardware. Diese Schnittstelle besteht aus mehreren hundert Prozeduren und Funktionen, die es dem Programmierer erlauben, zwei- und dreidimensionale Objekte zu spezifizieren, um hochwertige Bilder dieser Objekte zu erstellen. (vgl. [7], S. 1)

Eine erste Version wurde am 1. Juli 1992 vom *OpenGL Architectural Review Board (ARB)*, das von SGI[1] gegründet wurde, veröffentlicht. Sie entstand ursprünglich aus einer von SGI entwickelten 3D-API namens IrisGL, hatte aber ihr gegenüber den wesentlichen Vorteil, Funktionen auch dann bereitzustellen wenn sie nicht von der jeweiligen Hardware unterstützt wurden, so dass sie per Software emuliert werden konnten. (vgl. [7] S. 306, [11])

OpenGL ist eine reine Grafik-Schnittstelle. Sie bietet keine Unterstützung für Eingabegeräte (z.B. Maus, Joystick) oder andere Ausgabegeräte (z.B. Soundkarte). Entwickler müssen also auf andere Mechanismen zurückgreifen um Benutzereingaben zu verarbeiten und um Sound auszugeben. (vgl. [7], S. 4)

Für den systemnahen Zugriff auf solche Geräte wird in vielen Projekten die plattformübergreifende *Simple DirectMedia Layer (SDL)*[2] Bibliothek verwendet. Alternativ greifen viele Projekte auf die von *Creative Labs* entwickelte *Open Audio Library (OpenAL)*[3] zurück, die hochwertigen drei-dimensionalen Sound ermöglicht.

[1] SGI steht für Silicon Graphics, Inc.
[2] `http://www.libsdl.org/`
[3] `http://www.openal.org/`

Extensions

Die OpenGL-Spezifikation erlaubt es, Extensions mit zusätzlichen Funktionen zu definieren. Durch den Extension-Mechanismus wird es Hardware-Herstellern beispielsweise ermöglicht, neue Technologien in OpenGL einzubinden und direkt voll zu unterstützen, ohne auf eine neue Version der API warten zu müssen.

Die Namensgebung der Extensions folgt ebenso wie bei OpenGL einem strengen Schema[4]. Alle Hersteller haben eine eigenes Kürzel, das es erlaubt, Befehls-, Konstanten- und Typennamen einer Extensions dem jeweiligen Hersteller zuzuordnent. Bei *ATI Technologies* lautet dieses beispielsweise "ATI".

Wird eine Extension vom *ARB* für gut befunden, so werden sie in der *OpenGL Extension Registry*[5] veröffentlicht. Das kennzeichnende Kürzel lautet dann "ARB". Je nachdem wie wichtig die Extension eingestuft wird, kann ihre Funktionalität in einer neuen Version von OpenGL in den offiziellen Kern der Spezifikation übernommen werden. (vgl. [11], [7] S. 345-346)

3 Sprachlicher Aufbau der API

3.1 OpenGL

OpenGL ist eine prozedurale Sprache bei der alle Befehle in einer festen Reihenfolge abgearbeitet werden.

In OpenGL folgen Befehls-, Konstanten- und Typennamen einem strikten Schema. Sämtliche Befehle haben das Präfix *gl*. Konstanten und Typen haben das Präfix *GL*. Durch dieses einheitliche Namensschema werden Konflikte mit anderen Bibliotheken vermieden, und eine größtmögliche Transparenz für den Entwickler wird gewahrt.

Schematische Darstellung

Abbildung 1 zeigt die Schritte, welche die einzelnen Befehle durchlaufen: Im ersten Schritt (Evaluator) werden Polynomfunktionen der Eingabedaten ausgewertet. Dann werden verschiedene Berechnungen auf den geometrischen Objekten durchgeführt, um sie dann im nächsten Schritt in zwei-dimensionale Fragmente zu *rastern*. Danach werden noch Operationen auf diesen Fragmenten ausgeführt, bevor der Framebuffer geändert wird. Wenn nötig können die ganzen Vertex[6]-Operationen auch übersprungen werden indem Pixelwerte direkt in den Framebuffer geschrieben werden. (vgl. [7], S. 10)

[4] Siehe hierzu Abschnitt 3.1.
[5] http://oss.sgi.com/projects/ogl-sample/registry/
[6] Der Begriff *Vertex* bezeichnet einen Eckpunkt einer geometrischen Form.

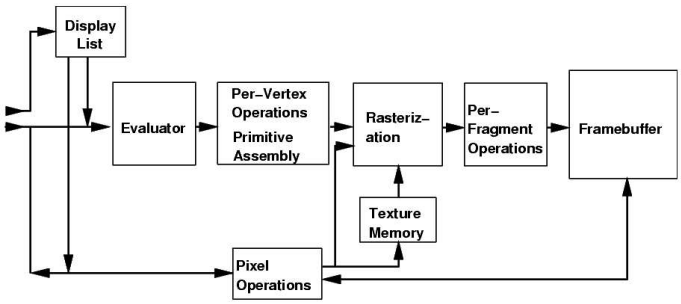

Abbildung 1. Blockdiagramm OpenGL (Quelle: [7] S. 10)

Anwendungsbeispiel: Dreieck

Der folgende Code-Ausschnitt, mit dem das in Abbildung 3 gezeigte Dreieck dargestellt wird, soll die Struktur eines einfachen OpenGL-Programmes deutlich machen. Im nächsten Unterkapitel werden dann anhand eines ähnlichen Beispiels, das mit Direct3D realisiert wurde, die großen Unterschiede in der Struktur der beiden APIs verdeutlicht.

Mit *glColor3f* wird die aktuelle Farbe anhand von 3 Fließkommazahlen auf Schwarz festgelegt. Dann wird der Hintergrund mit der eben festgelegten Farbe gelöscht. Zwischen *glBegin* und *glEnd* werden Eckpunkte (vertices) für ein einzelnes Dreieck definiert. Dabei wird vor jedem der drei Eckpunkte die Farbe jeweils auf Rot, Grün und Blau gesetzt. Mit *glFlush* wird dann das Dreieck angezeigt.

```
glColor3f(0.0, 0.0, 0.0); // Farbe = Schwarz

glClear(GL_COLOR_BUFFER_BIT); // Hintergrund loeschen

glBegin(GL_TRIANGLE_STRIP);
        glColor3f(1.0, 0.0, 0.0); // Farbe = Rot
        glVertex2f(  0.0,  0.75); // 1. Ecke
        glColor3f(0.0, 1.0, 0.0); // Farbe = Gruen
        glVertex2f(-0.75, -0.75); // 2. Ecke
        glColor3f(0.0, 0.0, 1.0); // Farbe = Blau
        glVertex2f( 0.75, -0.75); // 3. Ecke
glEnd();

glFlush(); // Dreieck darstellen
```

3.2 Direct3D

Anders als bei OpenGL sind Direct3D-Programme nicht prozedural aufgebaut und Befehle werden nicht direkt ausgeführt. Stattdessen werden die Koordinaten

der Eckpunkte in einem Puffer gespeichert und die meisten Berechnungen werden erst ausgeführt, wenn sie explizit gefordert werden. (vgl. [3])

Funktionen, Macros, Strukturen, Konstanten und Enumerations haben das Präfix *D3D*. Interfaces haben das Präfix *IDirect3D*. [7] (vgl. [4])

Schematische Darstellung

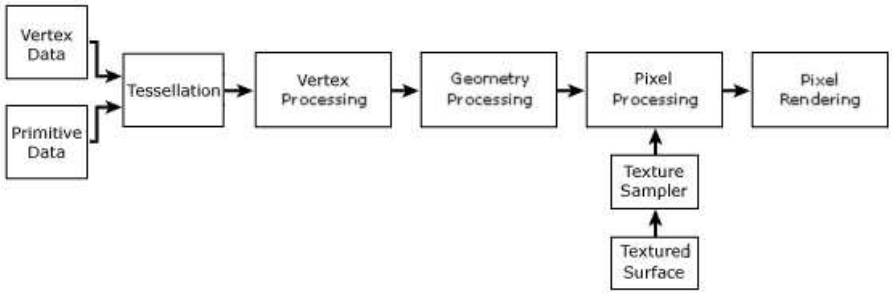

Abbildung 2. Blockdiagramm Direct3D (Quelle: [3])

Abbildung 2 zeigt die *Direct3D Graphics Pipeline*. In der *Vertex Data*-Einheit werden die Eckpunkte gespeichert, die wiederum aus der *Primitive Data*-Einheit heraus referenziert werden. Der *Tesselator* wandelt dann Objekte höherer Ordnung in einfachere Eckpunkte um, die dann wiederum in der *Vertex Data*-Einheit abgelegt werden. Beim *Vertex Processing* werden nun spezielle Direct3D-Umwandlungen angewendet, bevor weitere geometrische Berechnungen (wie beispielsweise Clipping) sowie das Rastern beim *Geometry Processing* durchgeführt werden.

Mittels der *IDirect3DTexture9*-Schnittstelle werden Textur-Koordinaten aus der *Textured-Surface*-Einheit ausgelesen. Je nach gewünschter Detailtiefe filtert der *Texture Sampler* die Texturen bevor beim *Pixel Processing* durch die Positionen und Farben der Eckpunkte sowie durch die Texturen die Farbwerte der einzelnen Pixel berechnet werden. Abschließend werden beim *Pixel Rendering* die Farbwerte der Pixel noch einmal verändert durch Effekte wie Nebel oder Alpha Blending[8].

Anwendungsbeispiel: Dreieck

Um die Koordinaten des Dreiecks zu speichern wird eine *Vertex*-Struktur definiert, welche die Position und den Farbwert der drei Eckpunkte (*vertices*)

[7] Analog dazu haben die anderen DirectX-Komponenten ähnliche Präfixes, wie beispielsweise *DI* bzw. *IDirectInput* für DirectInput.

[8] Beim Alpha Blending werden zwei Farbwerte kombiniert um Transparenz-Effekte zu bewirken.

des Dreiecks speichert. Zusätzlich wird ein entsprechendes *flexible vertex format (FVF)* zur Beschreibung der *Vertex*-Struktur definiert:

```
struct CUSTOMVERTEX {
  FLOAT x, y, z, rhw; // Position der Vertex
  DWORD color;        // Farbwert
};

#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZRHW|D3DFVF_DIFFUSE)
```

Nachdem die Struktur und das Format definiert sind, wird ein Array mit den drei Eckpunkten des Dreiecks und deren Farbwert erstellt:

```
CUSTOMVERTEX Vertices[] = {
  //    x,     y,    z, rhw,                        color,
  {150.0f, 50.0f,0.0f,1.0f,D3DCOLOR_XRGB(255,  0,  0),},
  {250.0f,250.0f,0.0f,1.0f,D3DCOLOR_XRGB(0,    0,255),},
  { 50.0f,250.0f,0.0f,1.0f,D3DCOLOR_XRGB(0,  255,  0),},
};
```

Als nächstes wird ein *Vertex*-Buffer in der Größe des *Vertex*-Arrays erstellt. Der Buffer wird dann gesperrt, die drei Eckpunkte werden per *memcpy* hineinkopiert und danach wird der Buffer wieder entsperrt.

```
g_pd3dDevice->CreateVertexBuffer( 3*sizeof(CUSTOMVERTEX),
                          0, D3DFVF_CUSTOMVERTEX,
                          D3DPOOL_DEFAULT, &g_pVB, NULL );

VOID* pVertices;
g_pVB->Lock( 0, sizeof(Vertices), (void**)&pVertices, 0 );
memcpy( pVertices, Vertices, sizeof(Vertices) );
g_pVB->Unlock();
```

Um mit dem Rendern anzufangen wird die Hintergrundfarbe nun auf Schwarz (Farbwert 0,0,0) gesetzt und danach wird *BeginScene* aufgerufen. Dann wird als Quelle des Streams "0" der eben erstelle Buffer angegeben. Als *flexible vertex format (FVF)* wird die oben erstellte *D3DFVF_CUSTOMVERTEX* angegeben.

Durch den Aufruf von *DrawPrimitive* wird angegeben um welche primitive Form es sich handelt. Abschliessend wird das Rendern mit *EndScene* beendet und mit *Present* wird das Dreieck aus Abb. 3 im Fenster angezeigt. (vgl. [5])

```
g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET,
                          D3DCOLOR_XRGB(0,0,0), 1.0f, 0 );

g_pd3dDevice->BeginScene();
g_pd3dDevice->SetStreamSource( 0, g_pVB, 0,
                          sizeof(CUSTOMVERTEX) );
g_pd3dDevice->SetFVF( D3DFVF_CUSTOMVERTEX );
g_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLELIST, 0, 1 );
g_pd3dDevice->EndScene();
g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
```

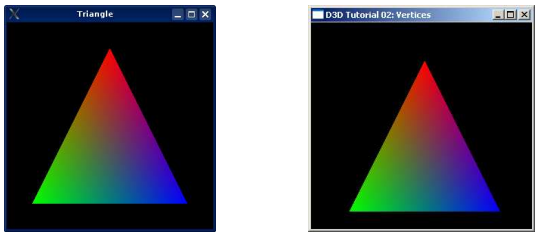

Abbildung 3. Die vom OpenGL- und vom Direct3D-Beispiel erzeugten Dreiecke.

4 Utility Toolkits

4.1 GL Utility Toolkit (GLUT)

Da OpenGL eine reine Grafik-Bibliothek ist, wird keine direkte Unterstüzung für Fenster-basierte Programmierung geboten. Für diesen Zweck gibt es das **GL Utility Toolkit (GLUT)**. Dieses Toolkit bietet eine plattformunabhängige Möglichkeit, Fenster zu erstellen und zu verwalten, sowie Eingabesignale von Maus und Tastatur zu verarbeiten. Folgender Code-Abschnitt erklärt die Verwendung von GLUT anhand eines einfachen Beispiels. (vgl. [1], S. 40-41)

```
int main(int argc, char** argv) {
    // GLUT initialisieren
    glutInit(&argc, argv);

    // Darstellungsmodus durch Bitmaske spezifizieren
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);

    // Fensterposition und -dimensionen festlegen
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(300, 300);

    // Fenster erstellen
    glutCreateWindow("Fenstertitel");

    // CallBack-Funktionen anmelden (optional)
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);

    // In eine perpetuelle Schleife eintreten
    glutMainLoop();
}
```

Die Namen der CallBack-Funktionen die in diesem Beispiel angemeldet werden (*display*, *reshape*, *keyboard* und *mouse*), erwarten gewisse Parameter, die Details über das aufgetretene Ereignis liefern, wie beispielsweise den Code der

gedrückten Taste bei der Keyboard-Funktion. Darauf werde ich hier aber nicht genauer eingehen.

4.2 DirectX Utility Toolkit (DXUT)

Ähnlich wie das GLUT vereinfacht das DXUT die fensterbasierte Programmierung für Direct3D unter Windows. Neben der Möglichkeit, Fenster zu erstellen, vereinfacht das DXUT außerdem noch die Erstellung von D3D-Devices. Direkte Aufrufe von Funktionen der Windows-API werden dadurch größtenteils überflüssig. Das folgende Code-Beispiel zeigt die Verwendung des Toolkits (Quelle: [2]).

```
INT WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, INT) {
    // CallBack-Funktionen anmelden (optional)
    DXUTSetCallbackDeviceCreated( OnCreateDevice );
    DXUTSetCallbackDeviceReset( OnResetDevice );
    DXUTSetCallbackDeviceLost( OnLostDevice );
    DXUTSetCallbackDeviceDestroyed( OnDestroyDevice );

    DXUTSetCallbackFrameRender( OnFrameRender );
    DXUTSetCallbackFrameMove( OnFrameMove );

    // DXUT initialisieren, Fenster öffnen, device erstellen.
    DXUTInit( TRUE, TRUE, TRUE );
    DXUTCreateWindow( L"BasicHLSL" );
    DXUTCreateDevice( D3DADAPTER_DEFAULT, TRUE, 640, 480 );

    // In eine perpetuelle Schleife eintreten
    DXUTMainLoop();

    return DXUTGetExitCode();
}
```

Nachdem die Callback-Funktionen angemeldet sind, wird ein leeres Fenster und dann ein D3D-Device erstellt. Danach tritt das Programm in eine Endlos-Schleife.

5 Unterstützte Plattformen

OpenGL-Implementierungen existieren für eine Vielzahl von Plattformen. Darunter Microsoft Windows, Linux, MacOS-X sowie diverse andere Unix-Derivate, aber auch Spielekonsolen wie die Playstation 3[9] von Sony. DirectX und seine Teilkomponenten hingegen sind auf Microsoft-Plattformen beschränkt; mit anderen Worten auf das Betriebsystem Windows und die Spielekonsole X-Box.

[9] Die Playstation 3 benutzt eine leicht modifizierte API namens *OpenGL ES* (ES steht hier für *Embedded Systems*). Siehe hierzu auch `http://www.khronos.org/opengles/`.

5.1 Direct3D mit Wine unter Linux und Unix

Um diese Einschränkung wenigstens teilweise zu umgehen, arbeiten einige Entwickler beim Wine[10]-Projekt daran, die DirectX-API nachzuprogrammieren. Größtes Problem bei diesen Anstrengungen bleibt allerdings die unvollständige oder gar fehlende Dokumentation. Im Gegensatz zu OpenGL handelt es sich bei DirectX nicht um einen offenen Standard dessen Spezifikationen frei verfügbar sind. Doch obwohl dieses Problem durch geschicktes Reverse Engineering gelöst werden kann, beschränkt die Natur des Wine-Projektes die Lauffähigkeit auf die x86-Plattform. Systeme mit anderen Prozessoren (wie PowerPC) bleiben also auch hier außen vor. (vgl. [12], [9])

5.2 DirectX und OpenGL in Windows Vista

Für die kommende Version von Windows hat Microsoft angekünigt, die Grafik-Architektur von Windows komplett zu überarbeiten. Als Basis für das neue Grafik-System *Windows Presentation Foundation* dient zukünftig Direct3D. Die OpenGL-Software-Implementierung kommuniziert dann künftig nicht mehr direkt mit der Hardware, sondern durch Direct3D, also durch eine weitere Software-Schicht. Wird ein Treiber mit einer eigenen OpenGL-Implementierung verwendet, werden die graphischen Effekte (wie beispielsweise Transparenz) der *AeroGlass*-Oberfläche für das jeweilige Fenster ausgeschaltet. (vgl. [6])

6 Schlussfolgerung

Direct3D und OpenGL weisen trotz einem vergleichbaren Funktionsumfang starke Unterschiede vor allem in der sprachlichen Gestaltung auf. Da beide APIs aber schon seit langem nebeneinander koexistiern und beide sehr stark verbreitet sind (Direct3D mehr im Spiele-Bereich und OpenGL mehr im wissenschaftlichen Bereich), ist es allerdings unwahrscheinlich dass sich eine der beiden APIs in der nächsten Zeit gegenüber der anderen durchsetzen wird.

Mit der rezenten Entwicklung von Shader-Sprachen[11], die in den neuen Versionen von Direct3D und OpenGL erstmals breitflächig zum Einsatz kommen, wird die Erstellung dreidimensionaler Objekte und Umgebungen noch einmal wesentlich effektiver.

Es bleibt also weiterhin interessant zu verfolgen, wie die 3D-Hardware und damit auch die zugehörigen Software-APIs sich in den kommenden Jahren weiterentwickeln. Der Trend verläuft momentan auf jeden Fall in Richtung hohe Abstraktion von der Hardware.

[10] *Wine* ist ein sogenanntes rekursives Akronym und steht für "Wine Is Not an Emulator". Es handelt sich um eine Software, mit deren Hilfe Windows-Programme auf Linux und Unix-Systemen benutzt werden können.

[11] Shader-Sprachen sind höhere Programmiersprachen, die von einer Rendering-API definiert sind. Die in solch einer Sprache geschriebenen Programme werden von der GPU (Graphical Processing Unit) abgearbeitet.

Literatur

[1] F.S. Hill, Jr.: *Computer Graphics Using OpenGL*. Second Edition. Prentice Hall, 2001

[2] Microsoft: *DXUT Overview*. http://msdn.microsoft.com/library/en-us/directx9_c/DXUT_Overview.asp

[3] Microsoft: *Microsoft Developer Network Library - Direct3D Architecture*. http://msdn.microsoft.com/library/en-us/directx9_c/Direct3D_Architecture.asp

[4] Microsoft: *Microsoft Developer Network Library - DirectX*. http://msdn.microsoft.com/library/en-us/dnanchor/html/anch_directx.asp

[5] Microsoft: *Microsoft Developer Network Library - DirectX - Tutorial 2: Rendering Vertices*. http://msdn.microsoft.com/library/en-us/directx9_c/Tutorial_2___Rendering_Vertices.asp

[6] Microsoft: *Windows Presentation Foundation and GDI, GDI+, DirectX and Direct3D*. http://msdn.microsoft.com/windowsvista/support/faq/presentation/

[7] Segal, Marc ; Akeley, Kurt ; Frazier, Chris ; Leech, Jon ; Brown, Pat: *OpenGL 2.0 Specification*. http://opengl.org/documentation/spec.html. Version: 22. Oktober 2004

[8] Trujillo, Stan: *Cutting-Edge Direct 3D Programming*. Coriolis Group Book, 1996

[9] Wikipedia: *Direct3D vs. OpenGL*. http://en.wikipedia.org/w/index.php?title=Direct3D_vs._OpenGL&oldid=36202759

[10] Wikipedia: *DirectX*. http://en.wikipedia.org/w/index.php?title=DirectX&oldid=36171785

[11] Wikipedia: *OpenGL*. http://en.wikipedia.org/w/index.php?title=OpenGL&oldid=36190683

[12] Wine-Projekt: *Wine Status - DirectX DLLs*. http://www.winehq.org/site/status_directx