

# Grundlagen der Spieleprogrammierung

Teil I: 3D-Graphik

---

## Kapitel 4: Die Realität DirectX und OpenGL (Übersicht)

Peter Sturm  
Universität Trier

## Outline

---

1. Übersicht und Motivation
2. Mathematische Grundlagen
3. Das Ideal: Photorealistisch (Raytracing, Radiosity)
4. Die Realität: DirectX und OpenGL (Übersicht)
5. Schritt 1: Drahtgitter
6. Schritt 2: Texturen
7. Schritt 4: Licht, Filter, etc.
8. Schritt 5: Fortgeschrittene Techniken (Vertex-, Pixel-Shader, ...)
9. 3D-Hardware
10. 3D-Engines im Überblick, Cg von nvidia
11. Spielekonsolen
12. Zusammenfassung und Ausblick

## Einordnung

- Raytracing als Darstellungsmethode für 3D-Echtzeit unrealistisch
  - Enorm hoher Aufwand
  
- Ausgangspunkt 3D-Szene
  - Welche Objekte sind auf dem Bildschirm sichtbar?
  - Ggf. Clipping
  - Aufwand hängt von Szenenkomplexität und Detaillierung ab

## 3D-Pipeline

- Photorealismus häufig unnötig
- Vereinfachung
  - Keine exakte Lichtberechnung
  - Keine Reflektionen/Refractionen
- Tricks
- Reine Softwarelösung
  - CPU berechnet alle Schritte
  - Leistungsfähige CPU notwendig
- Hardware-Beschleunigung
  - Wesentliche Schritte übernimmt Spezial-Hardware

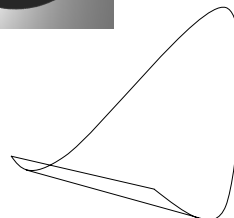
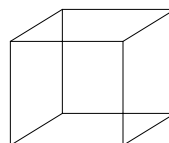
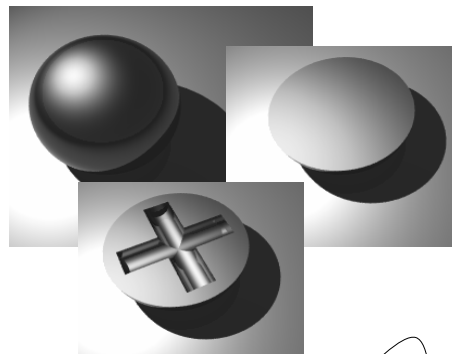


---

## Aufgaben der CPU

## 1. 3D-Modellierung

- Beschreibung der 3D-Szene
- Volumenorientiert (CSG)
  - Elementarvolumina: Quader, Zylinder, Kugel, ...
  - Mengenoperationen
- Flächenorientiert
  - „Patchwork“
  - Reguläre Flächen
    - Polygone
    - Dreiecke
  - Freiformflächen
    - Bezierkurven
    - Splines

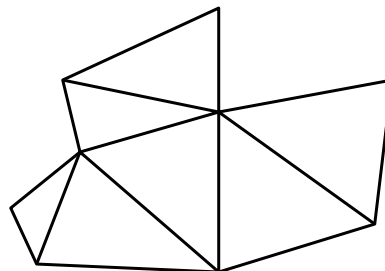
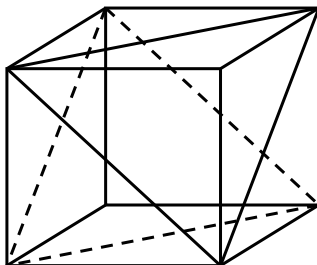


## Retained- und Immediate-Mode

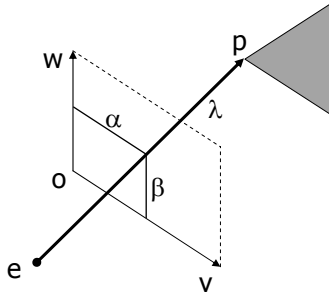
- 3D-Laufzeitumgebungen unterscheiden zwei Modi
- Retained Mode
  - 3D-Modell wird in der Umgebung gespeichert und verwaltet
  - Zugriff ähnlich einer Hauptspeicher-DB
  - Implizite Umsetzung
- Immediate Mode
  - 3D-Modell Bestandteil der Anwendung
  - Aufruf aller notwendigen Zeichenoperationen
  - Explizite Umsetzung
- OpenGL, DirectX realisieren nur Immediate Mode
  - Retained scheint aber anstrebenswertes Ideal

## 2. Tessellation / Triangulation

- 3D-Hardware unterstützt nur bestimmte Elementarflächen
  - Z.B. nur Dreiecke
- Umwandlung des 3D-Modells in Elementarflächen
  - Bei regulären Flächen einfach
  - Triangulation der Freiformflächen aufwendiger
- Hohe Auflösung = Viele Dreiecke = Hoher Rechenaufwand



### 3. 2D-Projektion



$$\begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix} + \alpha \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} + \beta \cdot \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + \lambda \left( \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} - \begin{pmatrix} e_x \\ e_y \\ e_z \end{pmatrix} \right)$$

- Sichtbarkeit einzelner Dreiecke?
- Ort in der 2D-Projektion?
- Lösung eines linearen Gleichungssystems für jeden Punkt der Szene
  - $\alpha$  und  $\beta$ : Ort und Sichtbarkeit
  - $\lambda$  Tiefeninformation
- Effizienz des Lösungsverfahrens
  - Faktor 10-20 in der Laufzeit
- Probleme
  - Teilweise sichtbare Dreiecke
  - e innerhalb eines Objekts

### Schnittstelle CPU und GPU

- Leistungengpaß
  - Benötigter Speicherdurchsatz
    - n Anzahl der Dreiecke
    - Punktgröße: 3 Floats/Double, Farbinformation, ...
    - Fps: Frames per Second
  - Beispiel: n=20000, Punkt=3\*8 Byte, 2 Byte Zusatz, 50 fps
    - Ca. 85 Mbyte/s Durchsatz
- Konsequenz
  - Unsichtbare Dreiecke vorher identifizieren
  - Redundanzen minimieren

---

## Aufgaben der 3D-Hardware

---

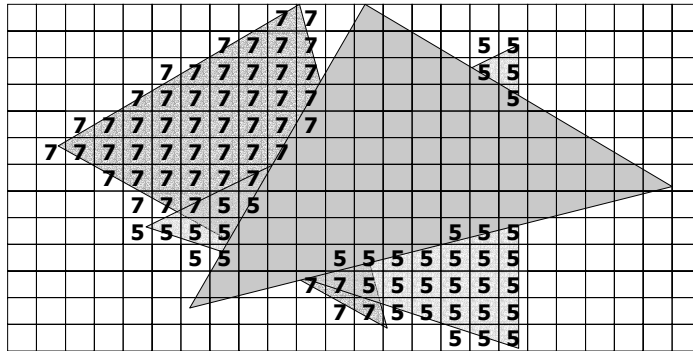
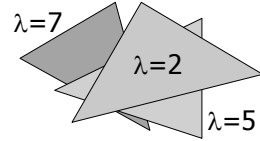
## 4. Finish

- Z-Buffer
  - Tiefeninformation
- Dreiecksflächen füllen
  - Farbe
  - Texturen
- Schatten
- Transparenzeffekte
- Nebel
- Anti-Alias
  - Kantenglättung
- ...



## 4. Tiefeninformation: Z-Buffer

- Einfaches Beispiel: Dreiecke parallel zur Projektionsfläche
- Auflösung des Z-Buffers (16 oder 32 Bit)

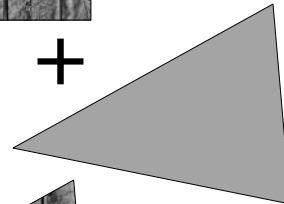


## Texturen

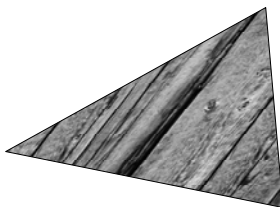
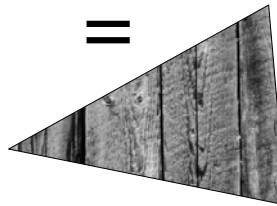
- „Bekleben“ der Dreiecksfläche
  - Wiederholung derselben Textur bei großer Fläche (Tiling)
  - Große Texturen wünschenswert
    - Viel Speicher
    - Komprimierung
- Photos häufig Ausgangspunkt



+



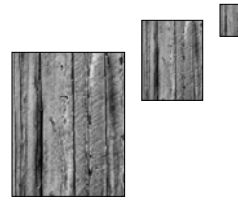
=



- Perspektivische Korrektur abhängig von Tiefeninformation

## Texturen 2

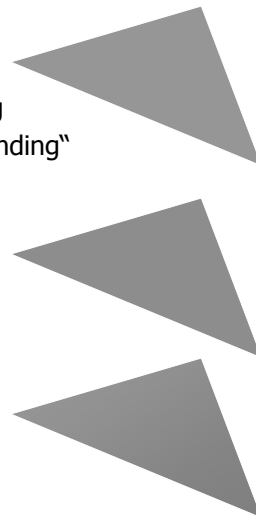
- Mip-Mapping
  - Unterschiedlich aufgelöste Texturen
  - Wahl entfernungsabhängig



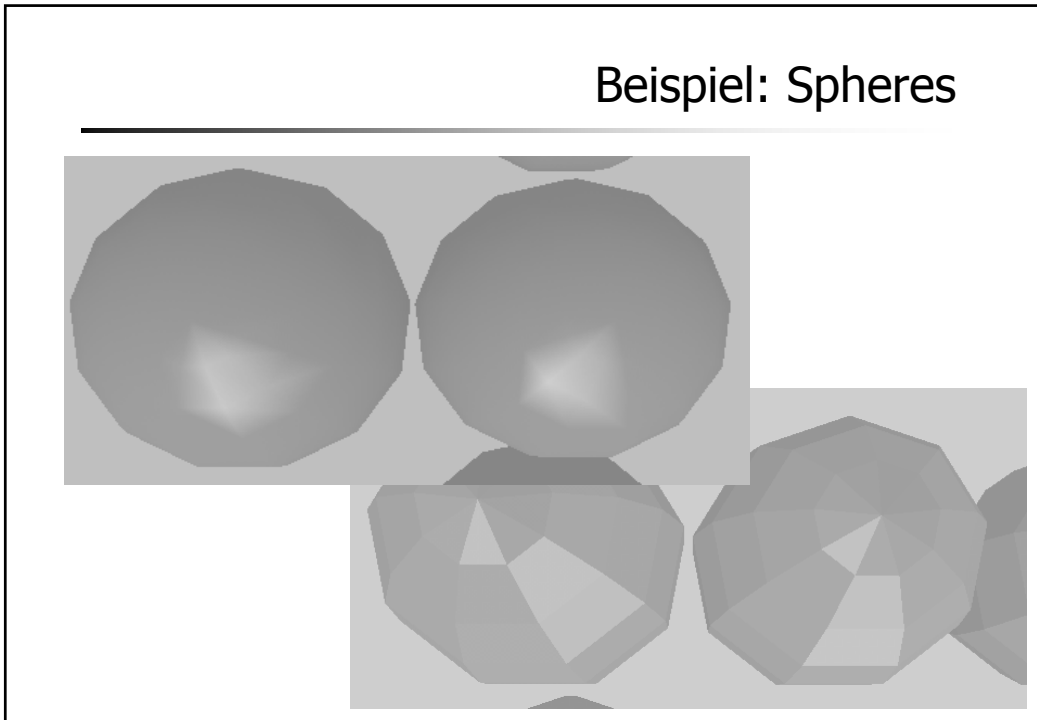
- Bump Mapping
  - Textur ist Höhenfeld
  - Kombination mit Shading
  - „Rauhe Oberfläche“
- Environmental Mapping
  - Texture wird dynamisch aus der Umgebung berechnet
  - Spiegelungen

## Schatten

- Licht ...
  - Lichteinfluß auf Fläche berechnet CPU
  - Zusätzliche Texturen bestimmen Lichtwirkung
  - Transparente Leuchteffekte durch „Alpha-Blending“
- ... Und Schatten
  - Einzelne Dreiecke sind abgedunkelt
- Einfaches Shading
  - Einheitliche Abdunklung
- Gouraud Shading
  - Iterative Zunahme der Abdunklung
  - Ausgangspunkt







### Transparenz

$P_{HG}$   $P_{VG}$

Alpha

- Überlagerung zweier Bilder
  - Hintergrundbild
  - Eingblendetes Vordergrundbild
- Alpha-Blending
  - Alphawert pro Pixel

$$P_{Gesamt} = P_{HG} + \alpha \cdot P_{VG}$$

## Ausgabe

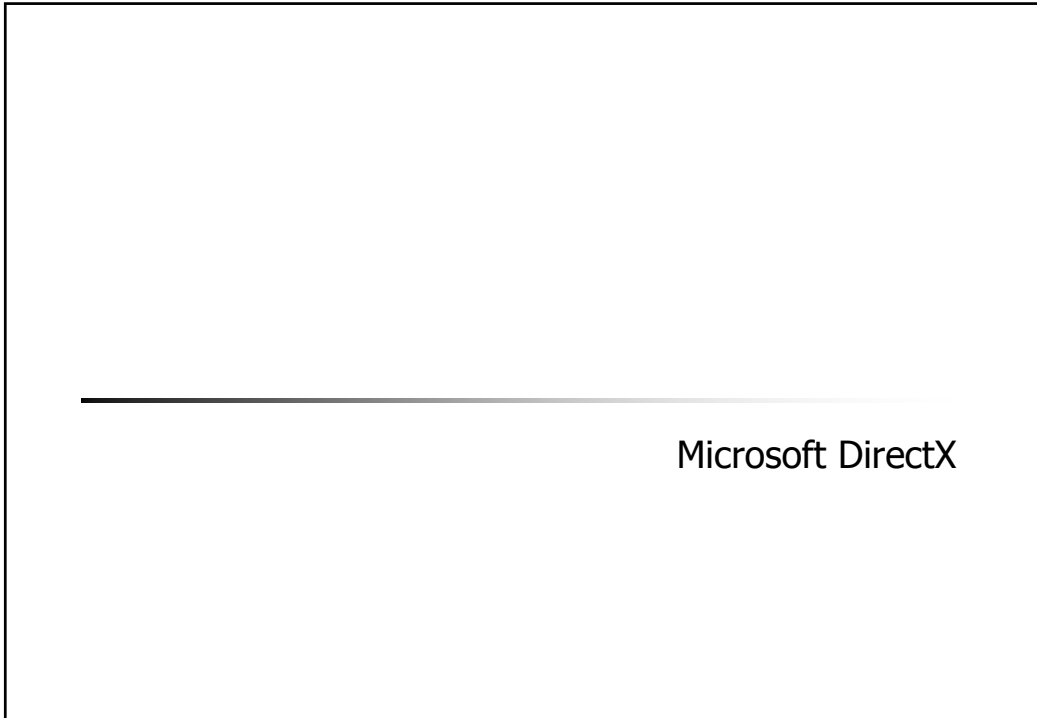
---

- Sauberer Wechsel zwischen Einzelbildern
  - Double-Buffer, Triple-Buffer, ...
  - Steuerung über VSYNC

## Trends

---

- Ziel Auslastung CPU und GPU jeweils 100%
- Trend
  - Aktuell mehr Steigerungspotential in der GPU
    - Parallelisierbarkeit der Abläufe
  - Verlagerung der Schnittstelle nach oben
    - z.B. Lichtquellen in HW (zuerst 8, ...)
    - Ansätze für Retained Mode
- Ausführliche Diskussion im Kapitel Hardware



## DirectX

---

- Laufzeitunterstützung für multimediale Anwendungen
  - Direkter Zugang zu HW-Funktionalität
  - Trotzdem hohe Kompatibilität
- Eigenentwicklung von Microsoft
  - In Zusammenarbeit mit diversen HW-Herstellern
- Aushandeln der Funktionalität
- Seit DirectX 7.0
  - TnLHAL (Transform and Lightning)

```

            graph TD
                DirectX[DirectX] --- HAL[Hardware Abstraction Layer  
HAL]
                HAL --- Hardware[Hardware]
                DirectX --- HEL[Hardware Emulation Layer  
HEL]
            
```

Hardware Abstraction Layer  
HAL

Hardware

Hardware Emulation Layer  
HEL

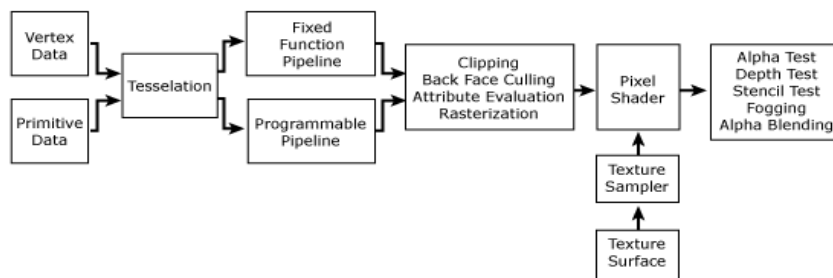
## Funktionsumfang

- DirectX Graphics
  - DirectDraw: 2D-Graphiken
  - Direct3D (D3D): 3D-Darstellung
- DirectInput: Eingabegeräte
- DirectPlay: Networking (Multiplayer)
- DirectSetup: Installation von DirectX-Anwendungen
- DirectMusic: Alles was mit Ton zu tun hat
- DirectSound: (Low-Latency Sound)
- DirectShow: Streaming Media

## Die Render-Pipeline in D3D

- Klassische Grundstruktur
- Neu
  - Vertex Shader: Kleine Maschinenprogramme in GPU für Knoten
  - Pixel Shader: Kleine Maschinenprogramme in GPU für Pixel

### Graphics Pipeline



DirectX Version	Major Features
1.0	DirectX 1.0 included DirectDraw, DirectInput, DirectPlay, and DirectSound.
2.0	DirectX 2.0 introduced Direct3D.
3.0	DirectX 3.0 introduced special DirectInput files, a special joystick control panel applet, and an updated virtual math coprocessor device file to support the Intel® MMX technology. A DirectSound3D API was included to supplement DirectSound.
3.0a and 3.0b	These two versions fixed some minor problems, but added no new features.
5.0	DirectX 5.0 introduced several new features and improvements including vertex buffers, support for force-feedback controllers, 3D audio hardware acceleration, multimonitor support (in Windows 98), a new game controllers control panel, better MMX support, and an improved user interface. Also introduced in DirectX 5.0, DirectX Setup provides a database of known configurations to simplify user setup on leading hardware platforms. The media layer, with its collection of high-level services, was introduced with DirectX 5.0.
5.2	DirectX 5.2 included several minor changes and an updated version of DirectPlay.
6.0	DirectX 6.0 introduced several new 3D features, including support for single-pass multitexturing, bump mapping, texture compression, and stencil buffers. DirectX 6.0 also added a new DirectX control panel, improved MMX performance, and support for AMD's 3DNow technology.
6.1	Released in February 1999, DirectX 6.1 added the DirectMusic API, which had been planned for release in DirectX 6.0.

## Historie

- Erste offizielle Version DirectX 5.0
- TnL mit Version 7.0
  - GPU übernimmt zunehmend höhere Aufgaben
- Aktuell Version 9.0a
  - Managed-Zugang
  - Neue Möglichkeiten bei den Shadern
- ...

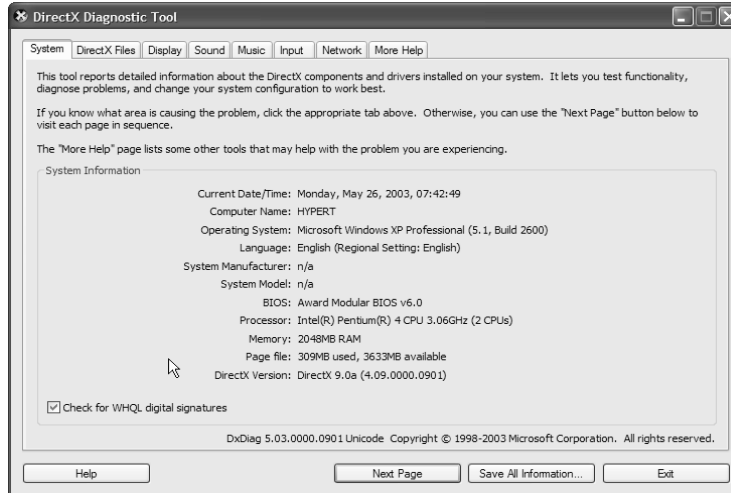
## DirectX SDK

- Wo, Wie groß, Installation
  - <http://msdn.microsoft.com>
  - 222 MB aktuelles 9.0a SDK (20.5.03)
  - 18 MB Help für Visual Studio .NET
- Dokumentation
  - C/C++ ist Teil des SDK
  - C# (allg. managed) nur Online
- Viele Beispiele
  - Sample Browser



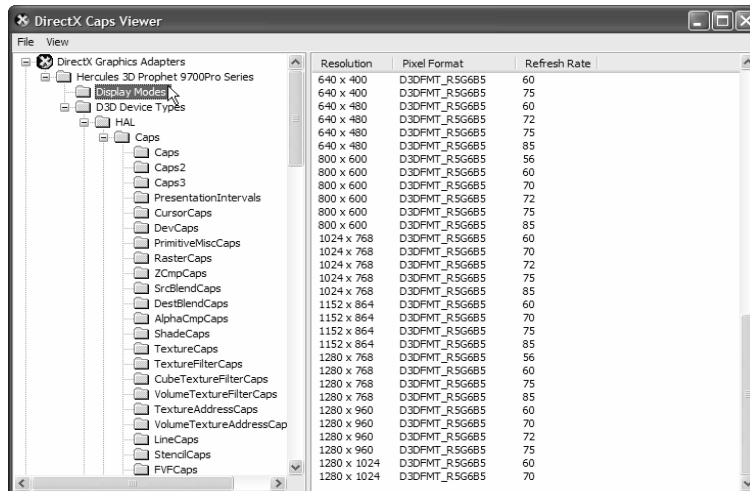
## Tools: dxdiag

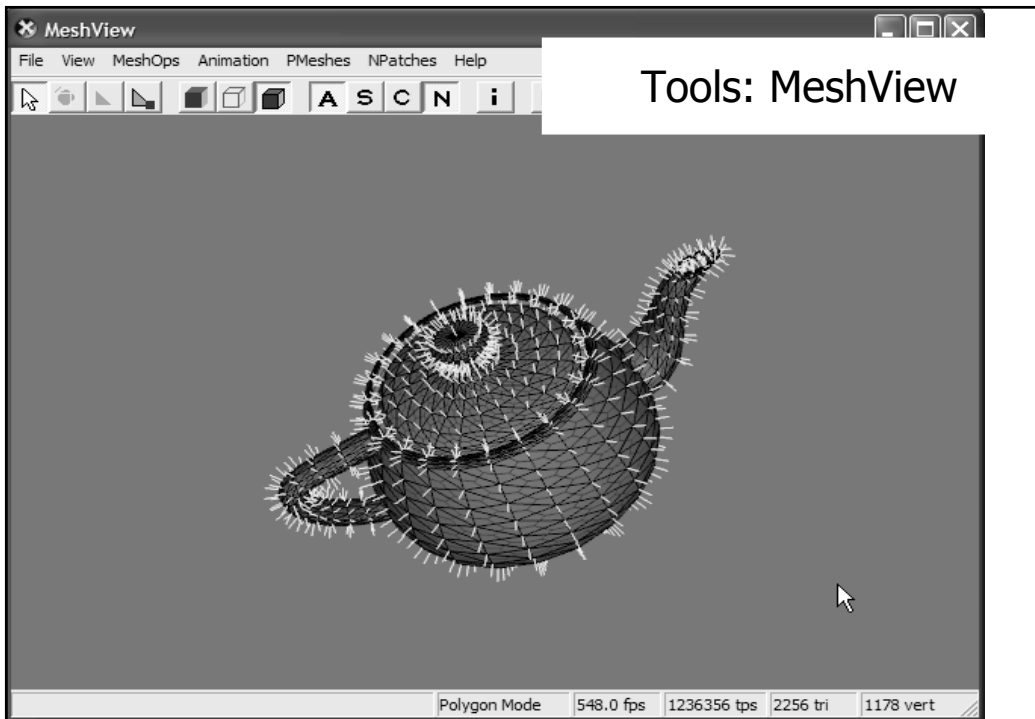
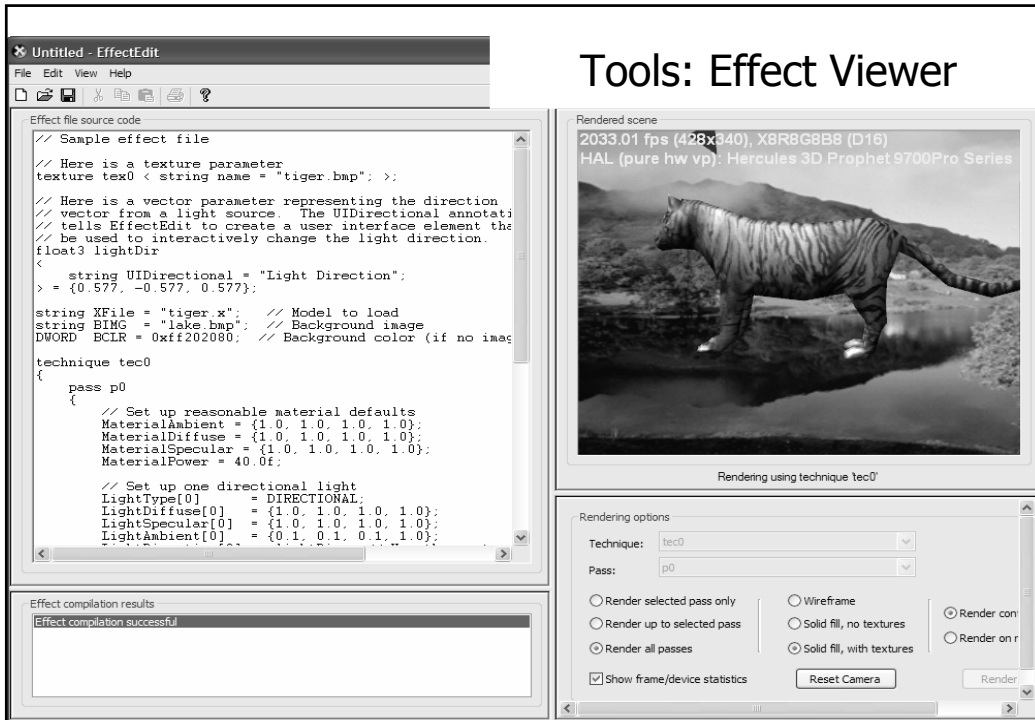
- DirectX-Installation: dxdiag



## Tools: capsviewer

- Darstellung der Capabilities





## Weitere Tools

---

- D3DSpy
  - Beobachten der Direct3D-Funktionsaufruf (lokal und remote)
- DirectPlay Network Simulator
- DirectX Error Lookup
  - Hexcode ⇒ Textuelle Beschreibung
- Texture Tool
- DMO Tester (DirectX Media Objects)
  - Streaming etc.
- Force Feedback Editor

## OpenGL

---

- Verbreiteter Graphikstandard im Bereich CAD/CAM/CAE
  - Nur Immediate Mode
  - Alle 3D-Kartenhersteller bieten OpenGL-Bibliotheken an
  - Visual Studio liefert OpenGL-Includes mit
  - Portierbar zwischen Windows und Linux
- Einzige ernsthafte Basis für 3D im Linux-Umfeld
- Einige Spieleentwickler setzen auf OpenGL
- Konkurrent nur zu Direct3D
  - Eindruck: Bewegt sich langsamer als DirectX