OpenGL & DirectX

Grafik-APIs

1 l Was sind OpenGL und DirectX

Prinzipiell denkt man wenn man diese beiden Begriffe hört zunächst erstmal an Grafik. Grob gesehen ist das auch richtig. Es handelt sich um Grafik-APIs.
DirectX ist allerdings noch mehr als das. DirectX ist ein Paket, das unter anderem die Grafik-APIsDirekt3D und Direkt2D beinhaltet.
Grafik-APIs können in Low-Level-Grafik-APIs und High-Level-APIs unterteilt werden.
Low-Level-Grafik-APIs sind prinzipiell nichts anderes als Treiber-Spezifikationen, also vorgeschriebene Schnittstellen, die der Treiber unterstützen muss.
High-Level-Grafik-APIs hingegen sind Libraries, die die Schnittstelle zum Treiber abstrahieren und somit einfacher machen. Spiele-Engines gehen noch einen Schritt weiter und abstrahieren zudem noch den Zugang zu anderer Hardware, wie etwa Sound, Vibration, Maus, Tastatur und anderen Controllern. Sie können entweder ihre eigene High-Level-Grafik-API entwickeln oder auf einer anderen aufbauen.
Die Anwendungen haben die Wahl, ob sie direkt die Low-Level-APIs oder die High-Level-APIs ansprechen, wobei Abstraktionsschichten das Fehlerrisiko erheblich reduzieren.

2 l Grafikpipeline

Stark vereinfacht gesprochen machen Grafikkarten “nur” drei Dinge:
1 l Gefüllte Dreiecke malen
2 l Zwischen zwei Punkten interpolieren
3 l Führen unmengen and Code gleichzeitig durch
Genauer betrachtet verarbeitet eine Grafikkarte Input-Daten in der sogenannten Grafikpipeline.
Daten kommen binär an und werden im Vertex-Shader so verarbeitet, dass die Grafikkarte diese als Vertecies (Also Punkte im dreidimensionalen Raum) verarbeiten kann.
Diese werden dann je nach mitgegebener Topologie (hier: Dreieck) gerastert.
Im Fragment-Shader wird den Vertices dann eine Farbe zugeordnet.
Dies wird dann zu einem Ausgabebild zusammengestellt und an das Fenster weitergegeben.
Bei OpenGL war die Grafikpipeline bis zu v1.5, bei DirectX bis v7.0 eine fixed-function Pipeline. Erst mit v2.0, bzw. v8.0 war sie programmierbar (programmable pipeline).
Bei den fixed-function Pipelines waren die shader vorprogrammiert, d.h. die API hat vorgegeben, wie die Daten verarbeitet werden.
Erst mit der programmierbaren Pipeline kann und muss man diese Shader selbst schreiben.
OpenGL: GLSL (OpenGL Shader Language)
Direct3D: HLSL (High Level Shading Language)

3 l Shadertypen

Neben den Vertex- und Fragment-Shadern (Der bei Direct3D Pixel-Shader genannt wird) gibt es heute auch andere Shadertypen.
Der Tesselationshader unterteilt bestehende Topologie-Ebenen in mehrere kleinere und benutzt Tesselation Informationen um die Positionsinformationen der neuen Verticies zu berechnen.
An der Tafel das Prinzip zeigen danach Beispiel auf der Folie
Der Geometry-Shader kann aus bereits vorhandenen Primitiven neue primitive Geometiren erzeugen und diese erneut in die Grafikpipeline einfügen. Durch die Einführung des Tesselation-Shaders wurde der Geometry-Shader weitestgehend obsolet. Für die Berechnung von *Point Sprites* ist dieser Shader allerdings noch wichtig.
Der Compute-Shader funktioniert auf eine andere Weise als die anderen Shader. Er wird hauptsächlich zur Berechnung von Dingen benutzt, die nichts mit dem malen von Dreiecken oder Pixeln zu tun haben.

4 l Shader-Techniken

Objektbezogen:
Diffuse - Berechne Farbe basierend auf Licht
Bumped diffuse: Täuscht Lichteffekte auf einem Objekt vor ohne Geometrie hinzuzufügen
Specular: Reflektion eines Objektes (Glanz)
Parrallax specular: Erzeugt Verschiebungen; Ein Objekt schaut dadurch von verschiedenen Blickwinkeln und Abständen anders aus
Transparent: Durchsichtigkeitsberechnungen basierend auf Farb- und Alpha-Level
Szenenbezogen:
Depth of Field: Nur ein Teilbereich des Bildschirms ist scharf, erzeugt Fokussierung auf ausgewählten, scharfen Bereich
Bloom/glow: Erzeugt einen Glüheffekt um Kanten eines hellen Objekts / Lichtquelle
Motion blur: Durch verschmelzen mehrerer Frames wird Bewegung suggeriert.

OpenGL

1 l OpenGL Überblick

- Low-Level-Grafik-API und somit eine Grafiktreiber-Schnittstellenspezifikation
- Erste Version erschien 1992
- Zunächst vom Industrie Konsortium ARB (OpenGL Architecture Review Board) entwickelt
- Seit 2006 vom 2000 gegründetem Industrie Konsortium “Khronos Group” weiterentwickelt. In der “Khronos Group” sind über 100 Mitglieder, darunter unter anderem AMD, Intel, NVIDIA, Google, Oracle, und mehr.
- Polygonales Darstellungsmodell
- Bis v1.5 fixed function seit v2.0 programmierbare Pipeline
- Client-Server-Modell (dazu nachher mehr)
- Keine Windowing, Physik oder Game Library! Zu verschiedenen Schnittstellen später noch mehr.
- Eigens entwickelte Shadersprache GLSL

2 l OpenGL Versionsgeschichte

Keine Angst ich geh nicht die Komplette Versionsgeschichte durch, will aber ein zwei interessante Stellen aufzeigen.

- V1.0 Start 1992
- V1.5 -> V2.0 == fixed function pipeline -> programmable pipeline
- V2.1 -> v3.0: Nach übergabe an Khronos Group: Longs Peak (Angekündigte Featureliste, wie Wahlversprechen in der Politik)
  Für September 2007 angekündigt, als riesige Revision mit vielen neuen Features angekündigt
  Oktober 2007 Bekanntgabe der Verzögerung
  Bis Veröffentlichung Funkstille
  Am Ende nur weniger der angekündigten Features umgesetzt
- V3.1 “Longs Peak Reloaded”
- V4.0 Tesselationshader
- V4.5 Actuelle Version (Emulation für leichtere Portierung von DirectX Programmen)

3 l Client-Server-Modell

Das Modell geht zurück auf die Terminal-Server Computer Infrastruktur, bei der die Terminals damals nur sehr leistungsschwach waren und aufwendige Grafikprozesse auf den stärkeren GPUs der Server ausgeführt wurden. Damit wäre es heute prinzipiell auch noch möglich die Grafikkarte eines anderen Rechners für berechnungen zu nutzen (z.B. mit Compute Shader). Im Normalfall ist das in der heutigen Zeit aber nicht mehr von Bedeutung, lässt sich im Code aber dennoch wiedererkennen, wenn man weiß nach was man schauen muss (Beispiel folgt nachher). Prinzipiell werden alle Berechnungen, die auf der Grafikhardware laufen auf dem Server ausgeführt. Alle anderen OpenGL Prozesse laufen auf dem Client.

4 l Schnittstellen-Libraries

Context und Fenster Toolkits:

- GLFW (OpenGLFrameWork; OpenGLFreeWindow, GraphicsLibraryFramework)
  Neben Fenstern und OpenGL Contexts auch Inputverarbeitung von Joystick, Keyboard, Mouse, Zeit, Clipboard,…
- SDL (Simple DirectMedia Layer) Video, Audio, Input Devices, Threads, shared object loading, networking und timer; Kann sowohl mit Direct3D, als auch OpenGL contexten umgehen.
- SFML (Simple and Fast Multimedia Library) Window Creation, OpenGL Contexts, Simple Hardware-Beschleunigte 2D Graphiken, Audio, Basic Tcp & UDP communication
- Qt (Q weil der Buchstabe gut aussieht, t für toolkit) Cross-Platform Application Framework
  Erweiterungs-Libraries:
- GLEW (OpenGL Extension Wrangler Library) Hilft bei Anfragen und Laden von OpenGL extensions
- GLM (OpenGL Mathematics) Header für Mathematik, die auf den GLSL Spezifikationen basiert
  Mesa 3D (Open Source Implementation der OpenGL Spezifikation)

5 l Vorschau (Vulkan)

- Entwickelt von der “Khronos Group”
- Am 3. März 2015 auf der GDC (Game Developers Conference) vorgestellt
- Auf Komponenten von Mantle aufgebaut
- Binary Format für Shaderprogramme (SPIR-V) -> Nicht mehr an bestimmte Shadersprache gebunden. Driver muss nur noch SPIR-V verarbeiten keine ganze Source Sprache
- Komplett Cross-Plattform (Auch Mobile - OpenGL hatte eine spezielle Ausführung für Mobile -> OpenGL ES)
- Besserer Multithreading Support
- Bottleneckerkennung und Workload-Verteilung

DirectX

1 l DirectX Überblick

- Sammlung COM-basierter APIs
- Erscheinungsjahr 1995
- Entwickelt durch Microsoft
- Nur Windows Betriebssysteme
- Aktuelle Komponenten:
  Direct3D (3D Graphiken), DXGI (Erzeugung von Direct3D Swap chains und Aufzählung von Geräte-Adaptern)
  Direct2D (2D Graphiken), DirectWrite (Schriftart- und Textrendering innerhalb Direct2D Applikationen)
  XAudio2 (Lower-Level Audioverarbeitungs API) , XACT3 (Higher-Level Audioverarbeitungs API. basiert auf XAudio2)
  DirectCompute (Mehrzweck Multithreadingberechnungen)
  DirectInput (Inputerkennung von Maus, Tastatur, Joystick), XInput (Inputerkennung von Xbox 360 Controllern - bzw. allen Controllern, die mit der Xbox 360 kompatibel sind)
  DirectSetup (DirectX Verionsmanager um die neuste Version von DirectX zu installieren)
  XNA Math (Mathematik Library)
  Windows Games Explorer (Präsentieren des Spieles in der Windows Benutzeroberfläche)

2 l Component Objekt Model (COM)

- Bausteinsystem (Austauschbar, nur einzelne Bausteine wählen)
- Interprozesskommunikation
- Dynamische Objekterzeugung
- Sprachunabhängig (In jeder Sprache implementierbar)
- Client/Server Prinzip
- Zugriff über Interfaces

3 l Direct3D

1992 von der Firma “RenderMorphics” unter dem Namen “Reality Lab” veröffentlicht.
1995 von Microsoft aufgekauft
1996 Erste Version von Direct3D (DirectX 2.0)
Shadersprache HLSL

4 l Versionsgeschichte

- V1.0 “retained mode” und “immidiate mode”
- V8.0 Shaderprogramme (HLSL)
- V11.0 Tesselation, Computeshader
- V12.0 Auf der GDC vorgestellt
  Hardwarenähere Programmierung (Somit den CPU Overhead verringern)
  Erste Spiele mit DX12 Unterst+tzung werden für Ende 2015 erwartet

Einsatz von OpenGL und Direct3D

Einsatz	Grafik-API	Typ
JMonkey	OpenGL	Grafik-Engine
OGRE	Direct3D, OpenGL	Grafik-Engine
Irrlicht	Direct3D, OpenGL	Grafik-Engine
Unity	Direct3D, OpenGL	Game-Engine
Unreal	Direct3D, OpenGL (, AMD Mantle)	Game-Engine
Blender	Direct3D, OpenGL	3D-Modellierungs- und Animationsprogramm
CryEngine	Direct3D, AMD Mantle	Game-Engine
Quake-Engine	Direct3D, OpenGL	Game-Engine
Source Engine	Direct3D, OpenGL	Game-Engine
Horde3D	Direct3D, OpenGL	Grafik-Engine

Beispiel

Link-Liste

- OpenGL Beispiel THI
- OpenGL Tutorial 1
- OpenGL Tutorial 2
- OpenGL on Reddit
- OpenGL Forum