



KMS UXA DRM OMG WTF BBQ?

Durchblick im Linux-Grafikdschungel

Martin Fiedler
Dream Chip Technologies GmbH

Agenda

- Konsole und Framebuffer
- X Window System
- OpenGL, Mesa und Gallium3D
- DRI Direct Rendering Infrastructure
- KMS Kernel Mode Setting
- Compositing
- Treiber-Übersicht
- Andere Grafiksysteme Android, Wayland und Mir
- Videobeschleunigung
- Hybridgrafik

Konsole und Framebuffer

Damals ...

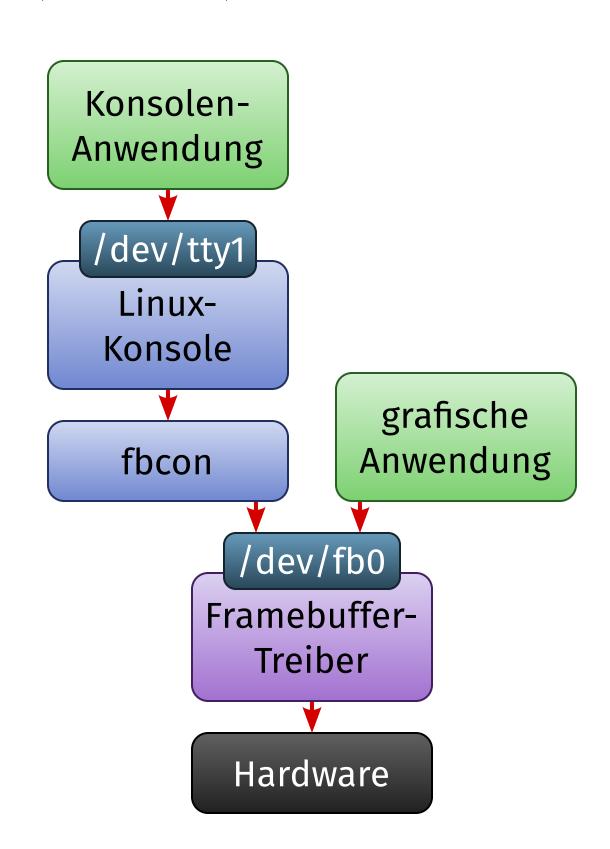
Die Situation zum Beginn der Linux-Zeit:

- Linux-Konsolentreiber steuerte VGA-Hardware direkt an
 - ... natürlich im Textmodus ©
- erste Programme, die Grafik anzeigen wollten, brachten ihre eigenen Treiber mit
- erste Libraries zur Grafikdarstellung, z.B. SVGALib
- Grafik-Applikation hinterlässt die Hardware im Originalzustand
 - beim Start: sichern des Zustands der Grafikhardware
 - beim Beenden: Wiederherstellen des alten Zustands
 - gilt auch heute noch für den X-Server

Framebuffer Devices

Erstes Grafik-Framework im Kernel: Framebuffer Devices (»fbdev«)

- nötig geworden wegen Portierbarkeit:
 viele Plattformen haben gar keinen Textmodus
- hardwarespezifische Kerneltreiber mit einheitlichem API
 - ▶ z.B. intelfb, atifb
 - vesafb: hardwareunabhängig, nutzt das VESA-BIOS der Grafikkarte
 - efifb: dito, für UEFI
- vom Userspace aus nutzbar: /dev/fbX
- sehr einfaches API
- **fbcon**: Emulation der Textkonsole mit Bitmap-Fonts (und Pinguinen ⓒ)
 - ▶ im Kernel, nicht Userspace

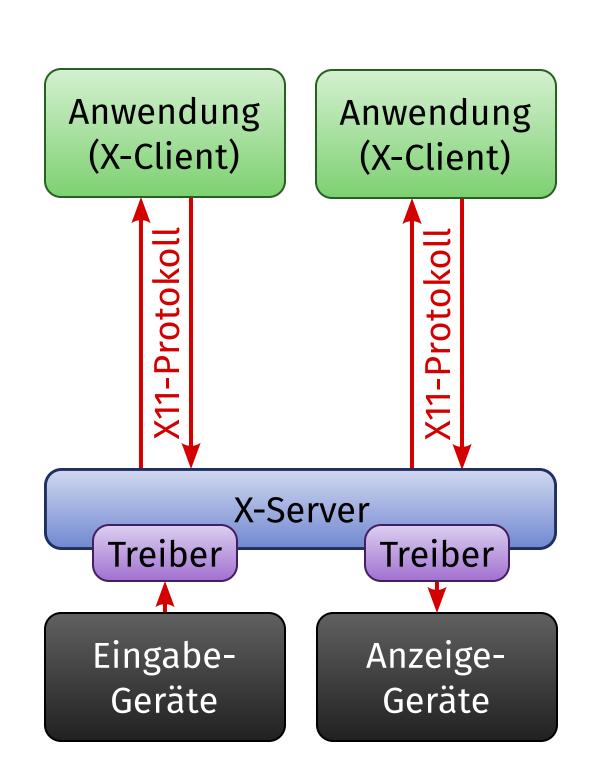


X Window System

X Window System

Das verbreitetste grafische System unter Linux: Das X Window System (»X11«, »X«)

- auf allen Unix-Derivaten verbreitet
- Client/Server-Architektur
 - Client = Anwendung
 - Server verwaltet Ein- und Ausgabe
- netzwerktransparent: Client und Server müssen nicht auf dem selben Rechner laufen
 - ► Kommunikation via TCP/IP
 - oder lokal über Unix Domain Sockets
- X11 ist der Name des Protokolls
- X-Server verwaltet eine Fensterhierarchie
 - root window = Desktophintergrund
 - top-level windows = Anwendungsfenster
 - subwindows = Steuerelemente (Knöpfe etc.)



X-Clients und -Server

- X-Clients implementieren X11-Protokoll nicht direkt, sondern nutzen Libraries:
 - traditionell Xlib
 - ► neuer, schlanker: **XCB** (»X11 C Bindings«)
 - auch Toolkits (Motif, Gtk, Qt, ...) setzen auf Xlib oder XCB auf
- Windowmanager: spezieller X-Client, der die Positionen der Top-Level-Fenster steuert und Fensterrahmen (»Dekorationen«) zeichnet
- X-Server erledigt Eingabe (Tastatur, Maus, ...) und Ausgabe (nur Grafik)
 - generischer Teil: DIX (»Device Independent X«)
 - hardwareabhängiger Teil: **DDX** (»Device Dependent X«)
 - enthält Treiber für Eingabegeräte und Grafikhardware
- populärste X-Server-Implementierung: XFree86, heute X.Org
 - ▶ DDX-Teil modular aufgebaut: Treiber sind gekapselte Module
 - ► DDX-Interface ändert sich oft zwischen Revisionen

X-Extensions

Das X-Protokoll ist mit **Extensions** erweiterbar, um zusätzliche Funktionalität bereitzustellen. Beispiele:

- XSHM (»X Shared Memory«) schnellere lokale Anzeige von Bitmap-Grafiken
- Xv (»X Video«) hardwarebeschleunigte Darstellung von Videos
- GLX OpenGL unter X
- Xinerama Multi-Monitor-Support
- XRandR (»X Resize and Rotate«) Konfiguration des Grafikmodus ohne X-Server-Neustart
- XRender moderne 2D-Grafik mit Antialiasing und Alpha-Blending
 - heute Grundlage für (fast) jegliche 2D-Grafikdarstellung

2D-Grafikbeschleunigung in X

Mehrere Ansätze für hardwarebeschleunigte 2D-Grafik in XFree86 bzw. X.Org:

- **XAA** (»XFree86 Acceleration Architecture«, 1996)
 - einfache Beschleunigung von Linien und Fülloperationen
- EXA (2005) abgeleitet aus KAA (»Kdrive Acceleration Architecture«, 2004)
 - gezielte Beschleunigung von XRender
- UXA (»Unified Memory Acceleration Architecture«, 2008)
 - von Intel als EXA-Nachfolger entwickelt
 - hat sich (außer auf Intel-Hardware) nicht durchgesetzt
- **SNA** (»Sandy Bridge New Acceleration«, 2011)
 - sehr Intel-spezifisch, aber sehr schnell
- **Glamor** (2011)
 - ▶ implementiert sämtliche Beschleunigungsfunktionen mit OpenGL
 - dadurch hardwareunabhängig

OpenGL

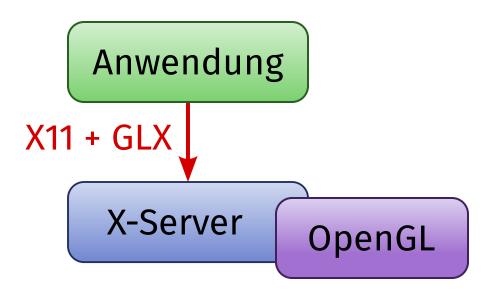
OpenGL

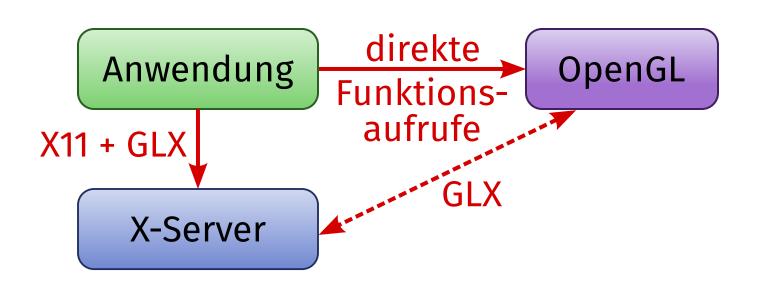
- OpenGL (»Open Graphics Language«) ist das Standard-API für 3D-Grafik.
- Industriestandard, herausgegeben vom Konsortium »Khronos Group«
- Funktion: hardware-beschleunigtes Zeichnen von texturierten Dreiecken
- OpenGL ES = »OpenGL for Embedded Systems«
 - (weitestgehend) Teilmenge von OpenGL, ~90% kompatibel
- OpenGL (ES) 2.0 und neuer beherrschen programmierbare Shader
 - C-ähnliche Sprache GLSL (»OpenGL Shading Language«)
- Extension-Mechanismus (ähnlich wie bei X11)
- zusätzliches systemspezifisches API zur Verwendung nötig:
 - GLX im X Window System
 - ► WGL (Windows), AGL (Mac OS X)
 - ▶ **EGL** für OpenGL ES (Embedded Linux, Android, iOS, ...)
 - systemübergreifend verfügbar, wird GLX etc. langfristig ablösen

Indirect vs. Direct Rendering

Wie sieht die Verwendung von OpenGL unter Linux mit X.Org in der Praxis aus?

- GLX = Teil des X-Protokolls
- Indirect Rendering
 - OpenGL-Kommandos werden durch das GLX-Protokoll übertragen
 - früher war so keine Hardware-Beschleunigung möglich
- Direct Rendering
 - nur lokal möglich
 - Client linkt gegen libGL.so und benutzt diese direkt
 - ► libGL.so enthält eine (evtl. hardwarespezifische) OpenGL-Implementation





Mesa

Unter Linux kommen zwei Arten von OpenGL-Implementationen zum Einsatz:

- die proprietären Treiber von nVidia und AMD
- oder Mesa

Mesa ist eine Open-Source-Implementierung von OpenGL

- ... einschließlich GLX, EGL und OpenGL ES
- zunächst nur Software-Rendering
- heute Grundlage für alle Open-Source-3D-Treiber

Gallium3D

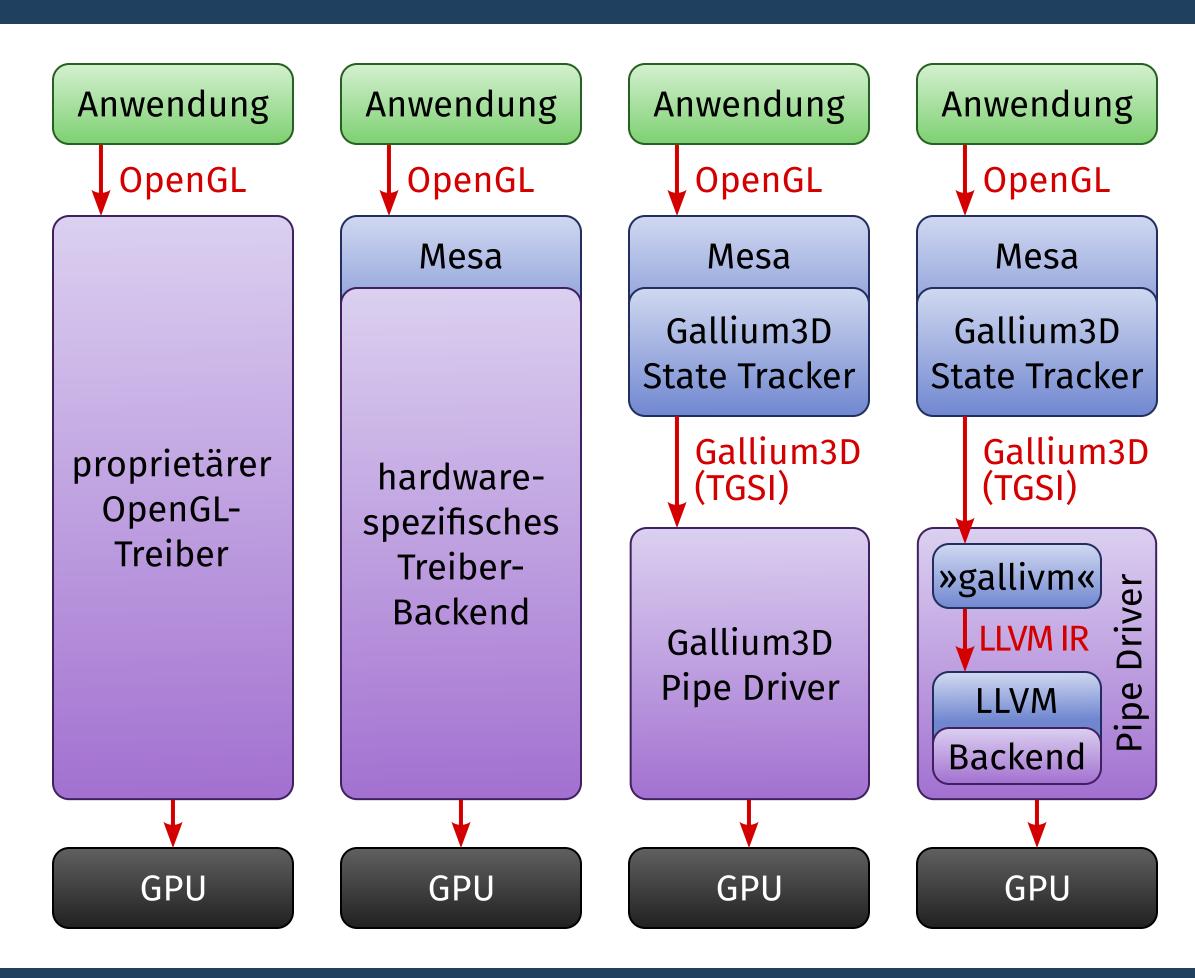
Gallium3D ist ein Framework zur betriebssystemunabhängigen Implementierung von Treibern für GPUs.

- setzt teilweise auf Mesa auf
- nicht nur 3D, auch Compute und Videodecodierung
- konzeptionelle Dreiteilung:
 - ▶ State Tracker: Implementation eines Client-APIs
 - z.B. OpenGL (über Mesa), OpenCL für Compute, VDPAU und OpenMAX für Video
 - ▶ WinSys Driver: Implementation der GLX- oder EGL-Schicht
 - ▶ Pipe Driver: Hardware-Backend für konkrete GPUs
 - z.B. llvmpipe (vergleichsweise schneller Software-Renderer)
 - nv30, nv50, nvc0, nve0 (nVidia-GPUs); r300, r600, radeonsi (AMD-GPUs)
- benutzt intern die Shadersprache TGSI (»Tungsten Graphics Shader Infrastructure«)
 - zusätzlicher Umweg über LLVM in einigen Backends

OpenGL-Treiberstacks

Insgesamt vier mögliche Treiberstacks für OpenGL:

- Proprietärer Treiber
 - ersetzt libGL.so
- »Mesa Classic«
 - generische libGL.so
 - hardwarespezifisches Backend in Mesa
- Mesa + Gallium3D
 - Mesa als State Tracker
 - Gallium3D-Backend (TGSI)
- Mesa + Gallium3D + LLVM
 - Mesa als State Tracker
 - Gallium3D-Backend (LLVM)



OpenCL

- Aktuelle GPUs eignen sich nicht nur für Grafik
 - enthalten Dutzende bis Tausende schneller Floating-Point-Recheneinheiten
 - ► **GPGPU** (»General Purpose GPU«) oder **Compute**-Anwendungen
- Standard-API für Compute: OpenCL (»Open Compute Language«)
 - ebenfalls von der Khronos Group herausgegeben
 - Linux-Support ähnlich wie bei OpenGL:
 - Closed-Source-Treiber bringen eigene Implementation mit
 - Gallium3D: State Tracker Clover
 - Beignet für Intel-GPUs
- weiteres verbreitetes Compute-API: CUDA
 - nVidia-proprietär, nur in Closed-Source-Treibern verfügbar

Direct Rendering Infrastructure

DRI & DRM

- OpenGL-Treiber läuft im Userspace als Teil der Anwendungsprozesse
- Zugriff zur Grafikhardware erfordert jedoch einen Kerneltreiber
 - außerdem nötig für Koordination mehrerer paralleler Prozesse
- Proprietäre Grafiktreiber benutzen proprietäre Kerneltreiber-APIs
- Für Open-Source-Treiber existiert ein entsprechendes Framework: die **Direct Rendering Infrastructure (DRI)**
- Mehrere Schichten:
 - hardwareunabhängige Userspace-Library (libdrm.so)
 - ▶ hardware- und treiberabhängige Userspace-Library (z.B. libdrm_intel.so)
 - eigentliches Kernelmodul: Direct Rendering Manager (DRM)
- DRM stellt Devices /dev/dri/cardX zur Verfügung
 - ▶ aber: Interface zwischen libdrm_XXX.so und DRM ist teilweise treiberabhängig

DRI-Versionen

Es existieren drei wesentliche Generationen der DRI:

- DRI 1 (1998)
 - erste, eingeschränkte Implementierung
 - sehr ineffizient bei mehreren gleichzeitig laufenden 3D-Anwendungen
- DRI 2 (2007)
 - löst die gravierendsten Probleme von DRI 1
 - die derzeit aktuelle, verbreitetste Version
- DRI 3 (2014?)
 - viele Detailverbesserungen gegenüber DRI 2
 - derzeit in der Entwicklung

Die folgenden Folien beziehen sich auf DRI 2, soweit nicht anders angegeben.

DRM Master und Render Nodes

DRM-Clients sind nicht gleichberechtigt – es gibt einen »**DRM Master**«

- typischerweise der X-Server
- läuft als root
- verwaltet allein die GPU
 - es ist immer nur ein DRM Master pro GPU aktiv
- kann andere Prozesse autorisieren, die GPU zu benutzen
- Problem: GPU-Nutzung ohne X-Server dadurch nicht möglich
 - ärgerlich für Compute-Anwendungen
- Lösung: **Render Nodes** in DRI 3
 - /dev/dri/renderDXX
 - eingeschränkte Funktionalität keine Grafikausgabe
 - keine Autorisierung durch den DRM Master nötig

Speicherverwaltung und Buffer-Sharing

Eine wesentliche Aufgabe der DRI ist die Verwaltung des Grafikspeichers.

- der Intel-Treiber verwendet dafür GEM (»Graphics Execution Manager«)
- die meisten anderen Treiber verwenden das API von GEM, aber eine andere Implementierung dahinter: TTM (»Translation Table Manager«)
- wichtiges Feature: Weitergabe und Teilen (»Sharing«) von Puffern im Grafikspeicher über Prozessgrenzen hinweg
 - essenziell für Compositing (»3D-Desktops« wie Compiz)
- unter GEM: flink-Mechanismus
 - globale numerische ID für geteilte Buffer
 - ► Sicherheitsproblem: IDs sind leicht erratbar
- neueres, sichereres Sharing-API ab Linux 3.3: DMA-Buf
 - Buffer erhalten Filedeskriptoren
 - Filedeskriptoren können über Unix Domain Sockets sicher übertragen werden

Kernel Mode Setting

Probleme mit User Mode Setting

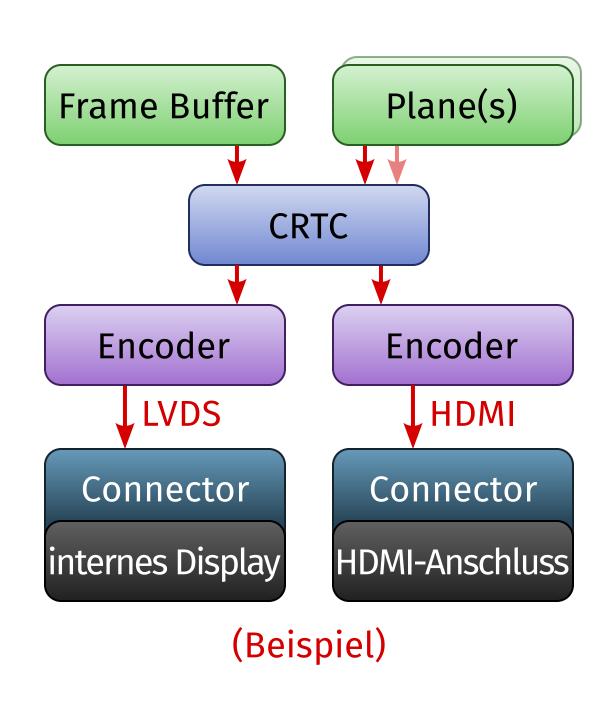
Klassische Grafikmoduseinstellung (»User Mode-Setting«) ist problematisch:

- Grafikhardware wird mehrfach initialisiert
 - erst vom BIOS für die Bootmeldungen ...
 - ... dann vom Framebuffer-Treiber für die Bootkonsole ...
 - ... und schließlich vom X-Server
- Geflacker beim Booten
- Geflacker beim Wechsel zwischen virtuellen Terminals und X-Server-Instanzen
- Teile des Treibercodes doppelt
 - Framebuffertreiber und DDX
- Probleme mit Suspend und Resume
- VESA-Framebuffertreiber kann Displayauflösung nicht ermitteln
 - startet in irgendeiner Standard-Auflösung
 - Resultat: hässliche Bootmeldungen ©

Kernel Mode Setting

Lösung: Kernel Mode Setting (KMS)

- ein Treiber im Kernel,
 benutzt von Framebuffer und X-Server
- Subsystem der DRI
 - keine neuen Device Nodes
- flexibles Displaykonzept, angelehnt an die Möglichkeiten moderner Displayhardware:
 - Frame Buffer
 - ► Plane = Overlay
 - CRTC = Displaycontroller
 - ► Encoder, z.B. HDMI-Transmitter
 - Connector = physischer Anschluss
- Frame Buffer und Planes sind DRI-Buffer



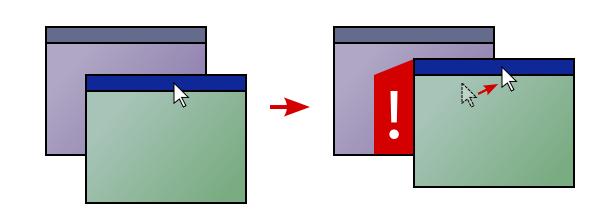
KMS: Ausblick

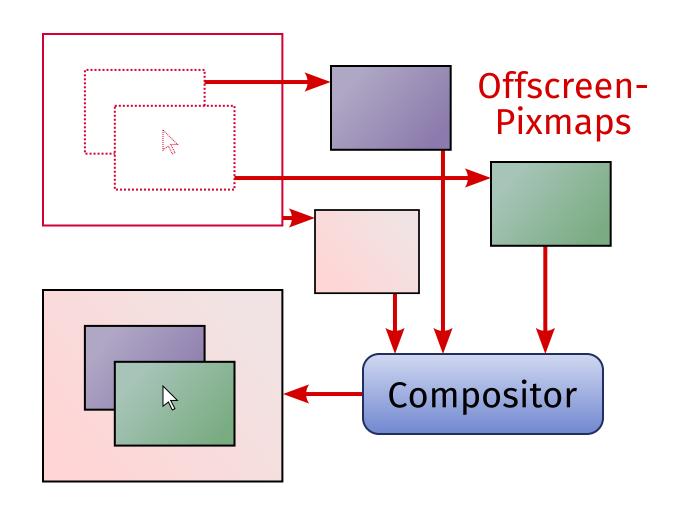
- xf86-video-modesetting: hardwareunabhängiger DDX-Treiber für X.Org auf Basis von KMS und Glamor
- KMSCON: Ablösung der Kernel-Framebuffer-Konsole durch eine Userspace-Terminalemulation
 - ► Features: Hardwarebeschleunigung, Multi-Monitor-Support, vollständiger Unicode-Support, Antialiasing, ...
- Weiterentwicklung von KMS: ADF (»Atomic Display Framework«)
 - nützlich für Hardware mit mehreren Overlay-Ebenen
 - Standard-Feature im Embedded-Bereich und auf Mobilgeräten
 - Einstellungen aller Overlays können synchrom (»atomar«) geändert werden
 - Flackern und Tearing wird verhindert

Compositing

Compositing

- unter X11 sind Fenster normalerweise »verlustbehaftet«
 - wenn durch ein anderes Fenster verdeckte Inhalte aufgedeckt werden (»Expose«), wird neu gezeichnet
- Alternative: Redirection
 - Fenster wird nicht auf den Bildschirm gezeichnet, sondern »off-screen« in eine Pixmap
 - Verarbeitung von Eingaben weiterhin so, als ob das Fenster am üblichen Platz wäre
- ein Compositor zeichnet die Off-Screen-Pixmaps an die richtige Stelle
 - nur ein »echtes« Fenster ohne Redirection: das Compositor Root Window
- Compositor meist in den Windowmanager integriert
- Unredirection = Aufheben der Redirection für Vollbild-Fenster

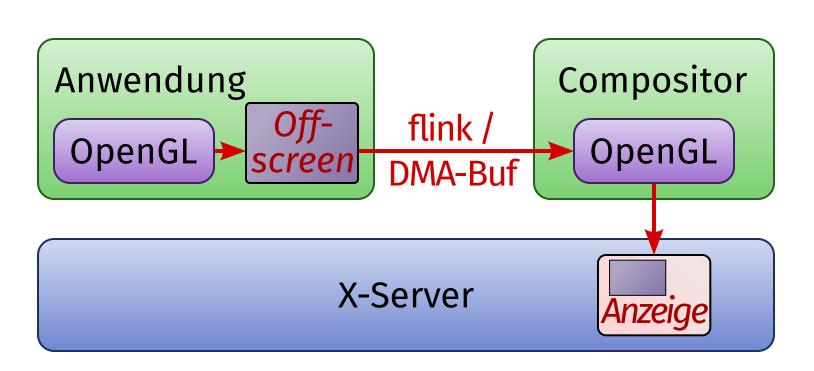




Compositing und OpenGL

Compositing mit OpenGL ist besonders interessant für »3D-Desktops« wie Compiz.

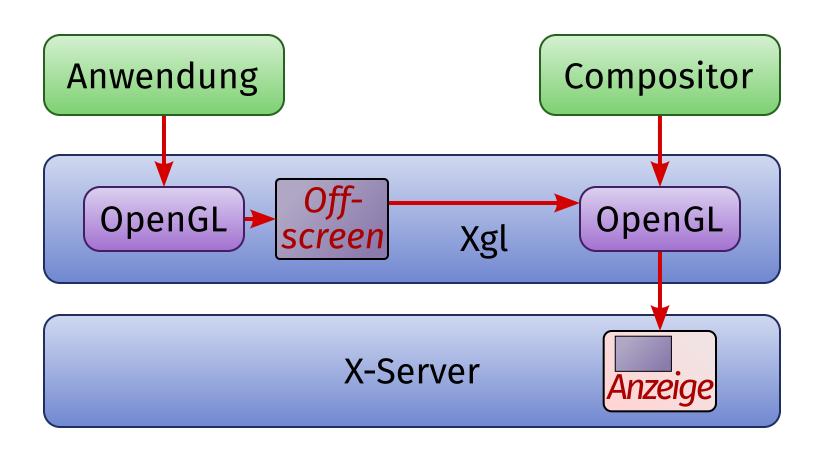
- aber: OpenGL kennt keine X11-Pixmaps, nur Texturen und Framebuffer
- Problem 1: Compositor benötigt Pixmaps als OpenGL-Texturen
 - ▶ Lösung: Extension GLX_EXT_texture_from_pixmap
- Problem 2: Compositor benötigt Zugriff auf Framebuffer anderer OpenGL-Kontexte (in anderen Prozessen!)
 - inzwischen leicht realisierbar mit DRI und Buffer Sharing
 - jeder OpenGL-Framebuffer ist ein DRI-Buffer
 - Compositor benutzt diese DRI-Buffer als Texturen





Erster Lösungsansatz für das OpenGL-Compositing-Problem: Xgl

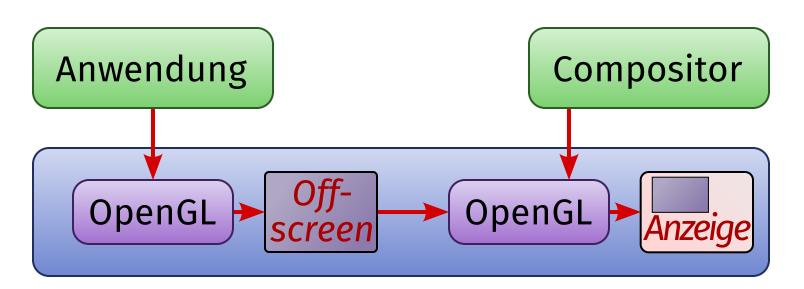
- Xgl = spezieller »virtueller« X-Server
- zeichnet ausschließlich mit OpenGL
 - X-Anwendungen mit der glitz-Library, einem Vorläufer von Glamor
 - OpenGL-Anwendungen mittels Indirect Rendering
 - alle OpenGL-Befehle gehen durch den Server hindurch
 - Ausgabe wird in OpenGL
 Frame Buffer Objects umgeleitet
 - Compositor erhält so Zugriff auf sämtliche Fensterinhalte
- Xgl selbst läuft wiederum auf einem »echten« X-Server



AIGLX

Zweiter Lösungsansatz für das OpenGL-Compositing-Problem: **AIGLX** (»Accelerated Indirect GLX«)

- ermöglicht hardwarebeschleunigtes
 Indirect Rendering für OpenGL
- erzwingt sogar Indirect Rendering:
 - OpenGL-Rendering findet ausschließlich im Server statt
 - Ausgabe wird in OpenGL
 Frame Buffer Objects umgeleitet
- Compositor erhält so Zugriff auf sämtliche Fensterinhalte



Treiber-Übersicht

Treiber für PC-Grafikhardware

- Treiber für DRI, X.Org (DDX), Mesa und Gallium3D heißen oft unterschiedlich
- teilweise verschiedene Kombinationsmöglichkeiten
- für nicht unterstützte Grafikhardware
 - Verwendung des VESA-BIOS oder der UEFI-Firmware für Grafikmoduseinstellung
 - OpenGL-Softwarerendering
 - früher Mesas Software-Renderer extrem langsam
 - heute Gallium3D llvmpipe: generiert Maschinencode, etwas schneller
- Intel-Chipsatz- oder Prozessorgrafik
 - vorbildlicher Treibersupport, ausschließlich Open Source
 - altmodisch: kein Gallium3D
 - lediglich experimenteller Treiber »ILO«
 - offizielle Treiber setzen direkt auf Mesa auf

Treiber für PC-Grafikhardware

- Grafikchips von ATI / AMD
 - proprietärer Closed-Source-Treiber: fglrx
 - ► AMD gibt Hardwaredokumentation heraus → guter Open-Source-Treibersupport
 - ▶ radeon-Treiberfamilie: Mesa für Radeon 7000 9250, Gallium3D ab Radeon 9500
 - ▶ radeonhd-Treiberfamilie: Mesa für Radeon X1000 HD4000, nicht weiterentwickelt
- Grafikchips von nVidia
 - proprietärer Closed-Source-Treiber: nvidia
 - ▶ spärliche Hardwaredokumentation → Open Source nur dank Reverse Engineering
 - ▶ **nv**-Treiber: alter Open-Source-2D-Treiber für Riva 128 und ältere GeForce
 - ▶ nouveau-Treiberfamilie: Gallium3D, ab GeForce FX
 - ▶ nouveau_vieux-Treiberfamilie: Mesa, Riva TNT bis GeForce 4

Typische Treiberstacks auf dem PC

Treiber	Fallback	Intel		AMD	nVidia	
Framebuffer	vesafb/ efifb	KMS	vesafb	KMS	vesafb	KMS
DRM/Kernel		i915	fglrx	radeon	nvidia	nouveau
X.Org-DDX	fbdev/vesa	intel	fglrx	radeon	nvidia	nouveau
2D-Beschl.		UXA / SNA	EXA	EXA / Glamor	EXA	EXA
OpenGL	Mesa	Mesa	fglrx	Mesa	nVidia	Mesa
Mesa	Gallium3D	i915/ i965		Gallium3D		Gallium3D
Gallium3D	llvmpipe			r300/r600/ radeonsi		nv30 / nv50 / nvc0 / nve0
OpenCL	Gallium3D	Beignet	fglrx	Gallium3D	nVidia	Gallium3D

Treiber für Embedded-GPUs

Die Treibersituation für GPUs in Smartphones, Tablets usw. ist deutlich komplizierter.

- GPU-, SoC- und Gerätehersteller liefert nur Closed-Source-Treiber
 - ▶ oft fürchterliche Qualität, viele Bugs
 - bisweilen nicht einmal Kerneltreiber in Source vorhanden
 - sogar die Weitergabe des Binärcodes(!) wird lizenzrechtlich eingeschränkt
- Ausnahme: Broadcom VideoCore IV (z.B. Raspberry Pi)
 - Dokumentation und Treiber im Februar 2014 veröffentlicht

Ansätze für Open-Source-Treiber mittels Reverse Engineering:

- Qualcomm Adreno Freedreno
- ARM Mali **Lima**
- Vivante Etna_viv
- nVidia Tegra Grate
- Imagination Technologies PowerVR ???

Andere Grafiksysteme

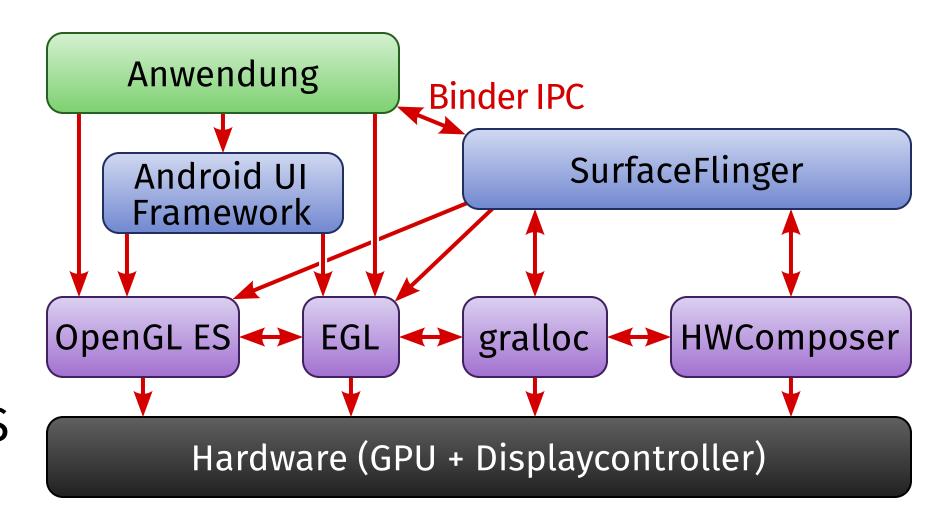
Blick über den Tellerrand

Bisher ging es ausschließlich um das X Window System, doch es gibt auch andere Grafiksysteme.

- die grundlegenden Konzepte sind aber immer ähnlich
- Beispiel: DirectFB
 - ▶ 1997 für Embedded-Systeme (Set-Top-Boxen) entwickelt
 - Ziel: Grafiksystem mit geringerem Ressourcenbedarf als X
 - setzt auf dem Linux-Framebuffer-Device auf
 - zusätzliche Hardwaretreiber für Beschleunigungsfunktionen
 - Grundlage: libdirectfb
 - verwaltet Grafik- und Soundausgabe sowie Eingabegeräte
 - eigener Fenstermanager, Portierungen von Toolkits, Kompatibilität zu X (dank speziellem X-Server), ...
 - dennoch: keine Relevanz auf dem Desktop

Android

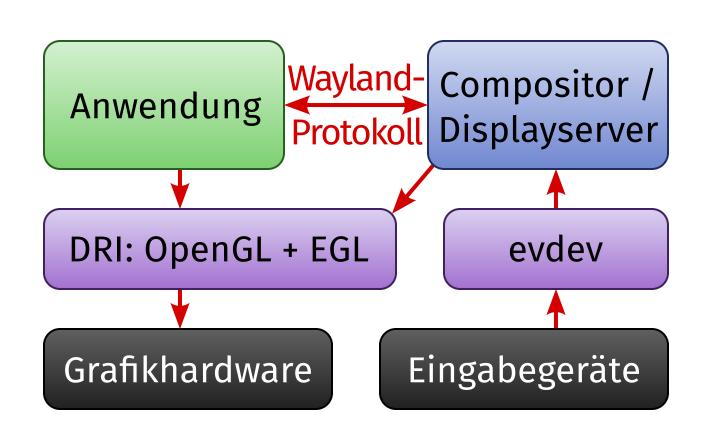
- Android nutzt von Linux (fast) nur den Kernel
 - kein GNU-Userland, kein X
 - eigene C-Bibliothek: **Bionic**
 - eigener IPC-Mechanismus: Binder
- Grafik-Grundlage: OpenGL ES und EGL
 - kein DRI, da meist proprietäre Treiber
- hardwarespezifische HWComposer-Library erfüllt ähnliche Aufgaben wie KMS
- gralloc zur Grafikspeicherverwaltung
 - ▶ in neueren Versionen Teil von HWComposer
- SurfaceFlinger als Compositor und Display-Server
- Anlegen der Grafikpuffer durch SurfaceFlinger



Wayland

Derzeit heißester Kandidat für die Ablösung des X Window Systems: Wayland

- Ziel: radikale Vereinfachung der Konzepte von X
- technisch gesehen ein Protokoll
 - vermittelt über Unix Domain Sockets
 - nicht netzwerktransparent
- Server-Teil ist kein eigenständiges Programm, sondern eine Library
 - ▶ wird vom Compositor benutzt
 → der Compositor ist der Display-Server
 - ► Referenzimplementierung: Weston
- Grundlagen sind EGL und DRI
- Anlegen der Grafikpuffer und Zeichnen findet im Client statt
- Eingabegeräte werden direkt über die Event-Schnittstelle angebunden



XWayland und Hybris

Wie kommen X-Anwendungen auf den Wayland-Schirm?

- XWayland = modifizierter »rootless« X.Org-Server, der alle X-Fenster als Wayland-Clients bereitstellt
- benötigt weiterhin hardwarespezifische DDX-Treiber, Ausnahmen:
 - xf86-video-wlshm (hardwareunabhängig, aber unbeschleunigt)
 - xf86-video-wlglamor (2D-Beschleunigung durch Glamor)

Wayland kann durch libhybris Android-Grafiktreiber benutzen:

- libhybris ist »Vermittler« zwischen der GNU-libc-Welt und der Bionic-Welt
 - ▶ libc-Programme können Bionic-Bibliotheken benutzen
 - ▶ insbesondere libGLESv2.so, den OpenGL-ES-Treiber
- setzt auch andere Android-Besonderheiten (z.B. gralloc, EGL-Spezialitäten) um

Mir

Canonicals Konkurrenz zu Wayland: Mir

- Grafiksystem für kommende Ubuntu-Versionen
 - noch nicht in 14.04, aber evtl. in 14.10
- konzeptionell sehr eng verwandt mit Wayland, aber andere, inkompatible Implementation
- benutzt weitere Teile von Android, z.B. das Eingabe-Subsystem
- mehr Fokus auf Datenaustausch zwischen Anwendungen als bei Wayland
- Anlegen der Grafikpuffer im Server, Zeichnen im Client
- XMir = XWayland für Mir
- benutzt ebenfalls libhybris für Android-Grafiktreiber
- großer Widerstand in der Community
 - technische Notwendigkeit angezweifelt

Videobeschleunigung

Videobeschleunigung

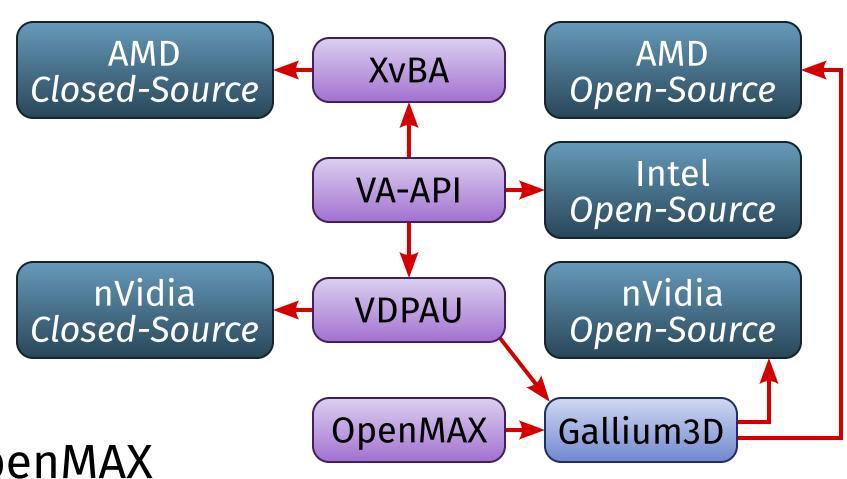
Es gibt mehrere Ansätze für hardwarebeschleunigte Videodarstellung unter X:

- **Xv** (X-Extension, 1991)
 - dient nur der Ausgabe, nicht der Decodierung
 - ► Funktionen: Skalierung, Farbraumkonvertierung
 - zwei typische Arten der Implementierung (auch gleichzeitig):
 - Overlay: blendet das Video direkt in die Bildschirmausgabe ein
 - **Textured Video**: zeichnet das Video mit der 3D-Hardware in den Framebuffer
- **XvMC** (X-Extension, 2000)
 - beschleunigt zwei spezielle Aspekte der MPEG-2-Decodierung: Motion Compensation (»MC«) und IDCT (8×8-Blocktransformation)
 - inzwischen obsolet
 - MPEG-2-spezifisch, nie für neuere Standards erweitert
 - nur von wenigen Treibern unterstützt

Hardwaredecodierung

Aktuelle GPUs enthalten Hardwaredecoder für einige Videostandards (u.a. H.264).

- mehrere konkurrierende APIs:
 - ▶ nVidia-proprietär: **VDPAU** (»Video Decode and Presentation API for Unix«)
 - sehr umfassend: Decodierung, Anzeige, Deinterlacing, ...
 - ► AMD-proprietär: **XvBA** (»Xv Bitstream Acceleration«)
 - nur Decodierung, Anzeige per OpenGL
 - ► Intel: VA-API (»Video Acceleration API«)
 - Decodierung in DRI-Buffer
 - Embedded-Bereich: OpenMAX
 - Industriestandard für De- und Encoding
- Situation verbessert sich jedoch langsam:
 - VA-API-Backends für VDPAU und XvBA
 - Gallium3D-State Tracker für VDPAU und OpenMAX
 - Gallium3D-Backends für nVidia- und AMD-Hardwaredecoder



Hybridgrafik

Hybridgrafik

- Viele aktuelle Notebooks haben zwei Grafikeinheiten:
 - Prozessorgrafik (»integrierte« GPU langsam, aber energiesparend)
 - zusätzliche (»dedizierte«) nVidia- oder AMD-GPU (schnell, aber stromfressend)
- vga_switcheroo: jeweils eine GPU wird abgeschaltet
 - Wechsel erfordert X-Server-Neustart
 - funktioniert nur bei Systemen mit »Video-Mux«, bei denen beide GPUs alle Videoaugänge bedienen können
 - Problem: neuere Modelle sind meist »muxless«
- proprietäre Treiber von AMD und nVidia haben inzwischen eigene Umschalter
 - basieren auf XRandR 1.4 (xrandr --setprovideroutputsource)
 - funktionieren auch mit »muxless« Systemen
 - ▶ aber: kopieren nur Ausgabe der dedizierten GPU auf integrierte GPU
 - keine Energiespareffekte (im Gegenteil beide GPUs sind aktiv!)

Bumblebee und PRIME

Für nVidia-Hybridsysteme (»Optimus«) mit proprietärem Treiber existiert eine »echte« Hybridgrafiklösung: **Bumblebee**

- zunächst läuft nur die integrierte Grafik
- wenn ein Programm über den Wrapper optirun gestartet wird, wird:
 - die dedizierte GPU aktiviert
 - ein zweiter (unsichtbarer) X-Server mit dem nVidia-Treiber gestartet
 - ▶ alle OpenGL-Zeichenbefehle mittels **primus** an den zweiten X-Server umgeleitet
 - ▶ nach jedem Frame das fertige Bild vom nVidia- zum Intel-X-Server zurückkopiert

Lösung aus dem Open-Source-Umfeld: PRIME

- derzeit in Entwicklung
- Erweiterung des DMA-Buf-APIs für GPU-übergreifendes Buffer Sharing
- volles dynamisches »Offloading« von OpenGL-Zeichenoperationen
- Aktivierung mit xrandr --setprovideroffloadsink

Vielen Dank für Ihr Interesse!