



Technical Brief

NVIDIA Unified Driver Architecture

An Elegant Approach for Lowering
Total Cost of Ownership



NVIDIA Unified Driver Architecture: Efficiently Uniting Hardware and Software

The NVIDIA® Unified Driver Architecture (UDA) serves as the foundation for the award-winning NVIDIA graphics driver for NVIDIA graphics processor units (GPUs) and the drivers for the NVIDIA nForce platform processors. A unique design and an unwavering corporate commitment to software stability and performance have positioned the UDA as the only driver solution offering both backwards and forwards compatibility—one driver can support all the current and previous versions of the related hardware. When a new version of an NVIDIA processor becomes available, the existing NVIDIA driver simplifies the integration of the new hardware or system into environments with many varied desktop PCs and operating systems. The NVIDIA UDA will not compromise performance for previous or emerging graphics and platform hardware, but rather reduces software overhead for driver functionality.

The success of the NVIDIA UDA continues to be cited as a major factor in graphics and platform selection by OEMs and systems vendors worldwide. With UDA, all end users enjoy the flexibility of easy upgrades when new drivers are available. The excellent track record for backwards and forwards compatibility also translate into significant reductions in total cost of ownership (TCO), savings that can be passed along to large enterprise customers and small/medium businesses. Many UDA characteristics contribute to the reductions in TCO:

- ❑ **Decreased maintenance time and cost:** With a single version of the NVIDIA drivers, only one software image has to be managed, configured, and installed for large-scale system deployments.
- ❑ **Investment protection:** NVIDIA continues to add more functionality and performance to the NVIDIA drivers. This allows older processors to enjoy more features and perform faster at no additional cost and without introducing complexities for end users or maintenance teams.
- ❑ **Reduced hardware/driver conflicts:** Every driver is thoroughly tested with all previous products, and new products are even tested with old drivers. The testing results in more stable operation during the life of the solution.
- ❑ **Increased scalability:** Upgrading products and adding new hardware is simplified. IT managers can add a full range of NVIDIA products to their configurations, all utilizing the same drivers.
- ❑ **Superior multi-platform support:** Since 90% of the NVIDIA driver code can be shared between operating systems, NVIDIA can provide full-featured, high-performance, and rock-solid stable driver support for all central processing units (CPUs) and operating systems.

This paper overviews traditional driver architectures, explains the technical and business challenges associated with the traditional architectures, and presents the NVIDIA UDA approach for overcoming these challenges.

The Challenge

To integrate hardware with the operating system (OS), drivers have traditionally been written in a layered fashion, where the hardware abstraction layer (HAL) encompasses the code that directly controls the hardware. OS dependencies make up the “back-end” of the driver, referred to as the “common” section since drivers for all hardware devices require some or all of this back-end code. (See Figure 1.)

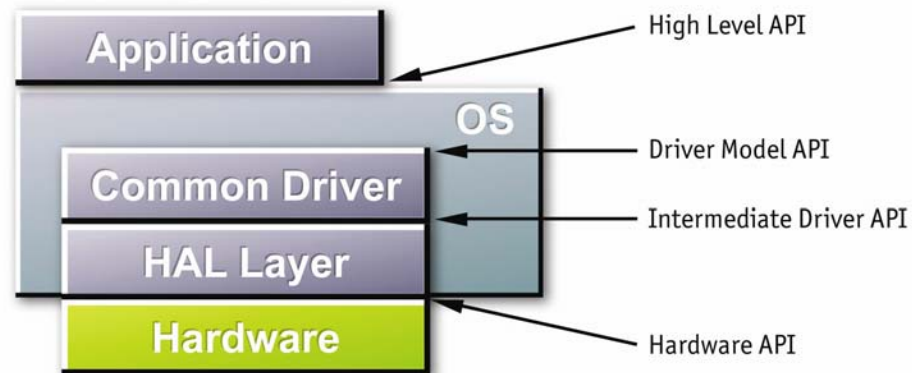


Figure 1. Common Driver Model Stack

Technical Issues

The common driver model stack, while straightforward, has introduced many challenges as systems and computing environments have increased in complexity. System drivers are affected by three aspects of the computing environment:

- ❑ **Operating systems versions** such as the various Microsoft® Windows® operating environments (Windows NT®, Windows 2000, Windows XP, Windows 9x), Linux, and Apple® Mac OS X
- ❑ **Application programming interfaces (APIs)** including GDI, VPE, WDM, OpenGL®, and Microsoft DirectX®
- ❑ **Hardware devices** supported by the numerous and varied NVIDIA graphics and platform solutions: past, present, and future

A single monolithic driver would grow to an unwieldy size, with very poor performance, if it were required to support all various offerings in each of these areas. Consequently, the traditional driver architecture has yielded numerous versions of drivers, each supporting some subset of OSs, APIs, and hardware.

The application of the traditional driver architecture and programming techniques result in many technical challenges:

- ❑ **Longer times to market**
All drivers must be enhanced to support a new or enhanced OS, API, or hardware solution. The many combinations translate into lengthy testing and certification cycles.
- ❑ **Increased support costs**
Different driver versions for the various OSs, APIs, and hardware solutions translate into increased IT maintenance time to manage stable hardware and software combinations.
- ❑ **Degraded performance**
It becomes virtually impossible to tune each possible execution path with multiple drivers. Using one, huge driver results in even slower performance.
- ❑ **Poor scalability**
A conventional programming approach does not scale well as the hardware and software become more complex. A monolithic driver will increase exponentially in complexity, making it impossible to maintain quality and performance.

Business Issues

A business analysis of the traditional driver architecture yields another list of challenges:

- ❑ **Software development costs**
Using the traditional driver model, an extremely large team of engineers is required to support driver engineering and bug fixes for all the various drivers. These costs must be passed along to customers.
- ❑ **Cumbersome release process**
Quality management efforts are complex and time consuming. To make matters worse, the testing efforts must be duplicated at each stage in the supply chain: the driver vendor, OEMs and system vendors, and large enterprises that certify all equipment before deployment.
- ❑ **Costly and slow certification testing phases**
Certification must be carried out for each driver-hardware combination, and for each system that integrates the driver. Large, monolithic drivers typically involve multiple testing passes and longer certification cycles.

The NVIDIA Solution

The NVIDIA architects set out to design and implement a driver architecture that would minimize TCO for NVIDIA processors, maximize performance, and provide unmatched stability and compatibility across the entire line of graphics and platform processors. The solution is the NVIDIA Unified Driver Architecture, which meets all the objectives and established the NVIDIA UDA as an industry-leading innovation (see Figure 2).

With the NVIDIA UDA approach, the NVIDIA driver software focuses on implementing the driver API functionality instead of tracking hardware differences. All hardware control operations are handled through the class-based object-oriented programming model. Performance-sensitive functionality (graphics and audio processing functions) takes advantage of a hardware-implemented HAL residing on NVIDIA chips. The hardware-implemented functionality minimizes the overhead associated with traditional drivers. The hardware/software combination uniquely balances performance and quality, a result that cannot be achieved with a software-only solution.

All NVIDIA drivers achieve excellent performance while delivering backward and forward compatibility. One driver is able to support all versions of the relevant hardware with highly abstracted class-based programming techniques that shield the software from the low-level hardware.

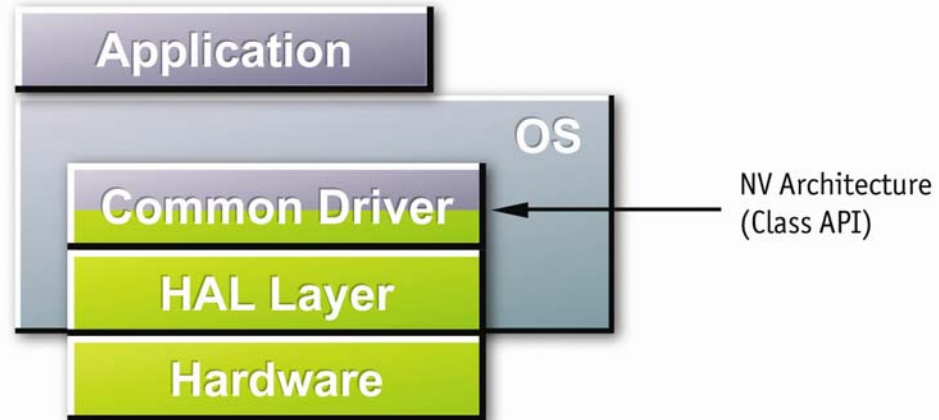


Figure 2. NVIDIA Driver Model Stack for GPUs and Performance-Sensitive Functionality

Object-Oriented Programming

The NVIDIA UDA includes an object-oriented abstraction of the hardware functionality, also referred to as a class-based model. *Objects* and *classes* provide mechanisms for efficiently controlling access to functionality and context information. By abstracting engine functionality, the NVIDIA architecture maintains engine independence while also reducing CPU, memory, and bus overhead for graphics and system functions.

A **class** is an abstract definition of a particular hardware function (to be implemented as a register mapping). An **object** is a single instance of a class (a specific register mapping), and therefore contains a particular context. A **method** is a write-only register within an object.

The NVIDIA UDA class structure and *methods* mimic current API models and usage. For example, the NVIDIA GPU driver provides classes that are designed to exactly match recent Microsoft Windows programming interfaces.

Resource Manager

Within the hardware layer of the NVIDIA UDA, the NVIDIA Resource Manager provides an intelligent solution for recognizing and managing classes and objects (see Figure 3). NVIDIA driver software provides a kernel-level library for recognizing and managing classes and objects. This NVIDIA innovation performs state and context management, allocating all architectural resources as needed for each driver client. Note that multiple clients are not aware of the other concurrent users of the chip in this context-management scheme.

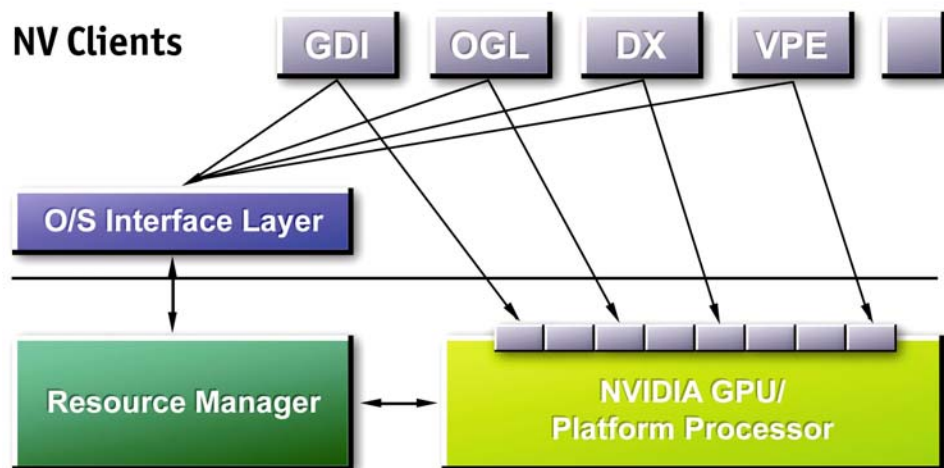


Figure 3. Architectural Driver Model Showing the Resource Manager

The allocation and management of architectural *objects* (hardware resources) provides several capabilities for compatibility management. As previously described, the Resource Manager establishes a channel between the hardware and a driver client in a manner that minimizes the driver overhead for ascertaining hardware resources.

By recognizing designated classes, the Resource Manager can handle hardware exception cases. Differences in hardware implementations are kept transparent to the user and handled in a manner that supports driver compatibility across all platforms. The Resource Manager also enables the use of legacy-class interfaces for maintaining backwards compatibility in subsequent hardware releases. Application engineers interface to all hardware using these standard conventions, which means that legacy code need not be changed to support new hardware.

Other functions handled by the Resource Manager include:

- ❑ **Error management:**
The class-recognition capabilities of the Resource Manager provide an excellent mechanism for implementing bug fixes.
- ❑ **OS-dependent portions of architecture**
The Resource Manager can also be used to manage release-to-release differences for OS memory management, process/thread control, plug-n-play, power management, and other capabilities.

UDA Benefits

The NVIDIA UDA has been proven over several generations of NVIDIA technologies and numerous NVIDIA product releases dating back to 1998 with the introduction of the NVIDIA RIVA128™. NVIDIA OEMs and customers report many advantages that have resulted from UDA:

- ❑ **A better design with more built-in features**
Since fewer engineers are required for version control and release management, more resources are allocated for performance optimizations and adding new features. The NVIDIA UDA approach also results in more freedom to change lower-level hardware, without any impact on OEMs and end users.
- ❑ **Excellent performance**
Driver functionality in hardware delivers the best possible performance and boosts all applications.
- ❑ **Stability and reliability**
NVIDIA UDA has passed the test of time, and contributes to the company's unmatched record for stable and reliable hardware and software.
- ❑ **Extremely short time to market**
NVIDIA releases are carried out in record time—less than 100 days from chip tape-out to production while the rest of the industry exceeds six months. OEMs gain fast access to new technology, and the UDA also means easy installation and support of new releases, with one driver supporting previous and new solutions.

❑ Lowered TCO

IT managers can generate a single software image with one version of the NVIDIA driver and deploy thousands of systems, each with different NVIDIA processors. This allows IT managers to qualify, manage, and support a single software architecture for multiple hardware configurations.

Conclusions

NVIDIA UDA has established a new tradition for driver architectures by delivering the best performance, unmatched compatibility, and significantly lower TCO. UDA goes beyond an elegant engineering design. The advantages of UDA have a positive impact on everyone that chooses NVIDIA technology:

- ❑ **Enterprises** enjoy IT cost and time savings from simplified support and upgrade procedures.
- ❑ **IT managers** can manage different hardware products with the same drivers, upgrade to new platforms without changing drivers, reduce system rollout times, and support flexible configurations of hardware platforms.
- ❑ **End users** gain fast access to new products and the ability to run existing games and applications on new platforms.
- ❑ **Software developers** can write one version of an application that is optimized for all NVIDIA solutions.
- ❑ **System vendors** can give customers the ability to generate a single software image supporting an entire hardware line, simplifying the introduction of new platforms into an installed base, streamlining certification and qualification cycles, and offering better overall TCO to their customers.

NVIDIA solutions will continue to rely on the UDA to enable rapid and consistent delivery of high-quality, high-performance software drivers. UDA overcomes the challenges associated with traditional driver architectures of the past, and allows hardware innovations to be introduced quickly and without impacting compatibility for existing systems and applications. No other graphics and platform processor vendor can enable a single software image across an entire company-wide offering.



Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

NVIDIA and the NVIDIA logo are registered trademarks and RIVA128 is a trademark of NVIDIA Corporation.

Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright NVIDIA Corporation 2003



NVIDIA.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com