DEVICE INDEPENDENT GRAPHICS SOFTWARE FOR THE HP3000

Peter Neuhaus
Hewlett Packard Company
Cupertino, California, USA

Summary

Occasionally, the requirements for displaying data graphically go beyond the capabilities of existing software packages. Consequently, it becomes necessary to begin the unpleasant task of developing custom code. Since this typically involves a significant investment of resources, it is important that the software be written so as to maximize its longevity and maintainability. To achieve these goals, it is necessary to understand the basic principles of graphic device independence and the concept of the Virtual Device Interface.

This paper will discuss these topics along with a discussion of how available graphics software tools can be used to assist in the development of device independent graphics software for the HP3000.

Background

In the early 1970s, the computer graphics industry realized that it needed to standardize some of the methods used in developing graphics software. The resulting conventions made it possible to create graphics in one environment (computer) and transport them to another with a minimum of recoding.

To date, only a few standards have been established, but others are under investigation. The Graphics Kernal System (GKS) has been adopted by the International Standards Organization and is being used extensively throughout Europe while the Siggragh CORE system, proposed in 1979, has not gained much acceptance. The debate continues, but GKS seems to be pulling ahead.

Regardless of whether or not one chooses to follow a strict standard, considerable improvements can be made in the writing of graphics software by following a few simple guidelines.

Frequently, companies plan to use only the specific graphics output devices that they already own, for example a HP7550 plotter or perhaps a non-HP graphics terminal. To support these devices, the specific commands required by the devices would be scattered throughout the application program (see figure 1). The result would be very efficient but would necessitate excessive modifications if new or additional output devices were acquired at a later date.

The First Step

Device independence is nothing new to the professional programmer. Common functions such as cursor control are often modularized into separate subroutines (device drivers) that can be easily modified or replaced to accommodate new output devices that require different commands for their proper operation. When it was necessary to drive more than one output device, a duplicate set of subroutines is written for each device (see figure 2).

In addition, if more than one device might be used simultaneously, it is necessary for subroutines with identical functions to have different names, such as LINE1, for drawing a line on device 1, or LINE2 for device 2. At this level, device independence was still not achieved since the LINE1 and LINE2 calls must be embedded in the application program.

Step Two

By inserting another level between the application program and the device drivers, the interface between the application program and the outside world is standardized. If this new level, perhaps a commercially available GKS package or CGL/3000 from the contributed library, contains a function that allows the application program to select which output device should be used, it is possible to remove the references to LINE1 and LINE2 and substitute a call to the new LINE function in the GKS/CGL package (see figure 3).

At this point, true device independence has been achieved since new devices can be supported without modifying the application program as long as someone writes a device driver for the new device. However, creating these new drivers can consume enormous amounts of programming effort because each device is unique in that it requires specific nonstandard "escape sequences" to perform a given task.

VDI - The Last Step

The graphics industry is attempting to standardize the hardware instructions required by graphic output devices through a concept called the Virtual Device Interface (now often called the Computer Graphics Interface). A VDI driver accepts all the command that a generic device might receive but only implements or emulates those that its device can perform.

If all graphic devices understood the same commands, the need for device drivers would be eliminated (see figure 4). Essentially, the device drivers would be implemented within the device's firmware. However, until the VDI concept becomes commonplace, it is necessary to employ the basic concepts of device independence when writing graphics applications. Several alternatives are possible.

Ways to be Independent

The most straightforward solution would be to obtain a graphics software library either from the computer manufacturer or from an independent third party. Such packages include a number of device drivers for the most popular graphic devices. The disadvantage to this solution becomes evident if it is necessary to change host computers at a later date.

Even switching between computer lines offered by the same manufacturer can cause significant problems. Therefore, when shopping for this type of software product, it is important to investigate the possibility of moving the product between systems. Packages written in standard languages such as Fortran or Pascal help simplify portability. But even standard languages often do not port well.

The ability to move to another CPU may sound like something that would not be done too often, but as desktop computers become as powerful as typical multi-user systems, many applications will be moved to smaller workstations. It's much like the user who feels he needs only 50 megabytes of disc storage, orders 100MB even though he "knows" it will never be needed, then runs out of disc space six months later. Applications and technologies change continuously. Investing the extra resources to implement a flexible solution often pays high dividends at a future date.

Sharing Graphics Data

Frequently, graphic databases created on one system need to be processed on another. To address this need, a standard format for exchanging databases, called the Initial Graphics Exchange Specification (IGES), has been established and is currently supported by a number of graphics packages. A similar newer standard, the Virtual Device Metafile (VDM), performs much the same functions.

Within HP, a standard called the Graphics Peripheral Interface Standard (GPIS) has been developed and is currently used by some HP150 graphics programs. Since it is similar to VDM, it can be easily modified to conform to the final version of VDM.

By simply using the IGES or VDM device driver, an application can store the resulting image or object description onto a transportable media such as magnetic tape, which can then be read by another IGES/VDM compatible system. Applications written in a device independent manner are able to utilize this useful feature.

Summary

The trade-offs involved in the decision to standardize the development of computer graphics software deals mainly with short term versus long term benefits. Projects that seem to be "one shot" programs may not appear to necessitate the features of device independence. But often, the programs are modified and used again, possibly for another "one shot" application. In general, establishing standards or guidelines in a programming environment leads to increases in productivity. The slight performance degradation created by the overhead of a graphics subroutine library can be offset by the ever decreasing costs of computer hardware.

Once standards have been implemented, applications can be developed faster since it becomes unnecessary to reinvent the wheel for each new project. In addition, program maintenance is simplified since each programmer understands the basic strategies used by his fellow graphics programmers. Overall, the need to be device independent will become increasingly important as the number and capabilities of systems and graphic devices expand.

Biography

Peter Neuhaus is a computer graphics specialist in the Information Systems and Networks Group, Marketing Communications Graphics Department. He is involved in exploring the use of computers by graphics artists to increase their productivity. During a leave from Hewlett-Packard, he taught computer graphics to engineers and artists at California State University, Long Beach.

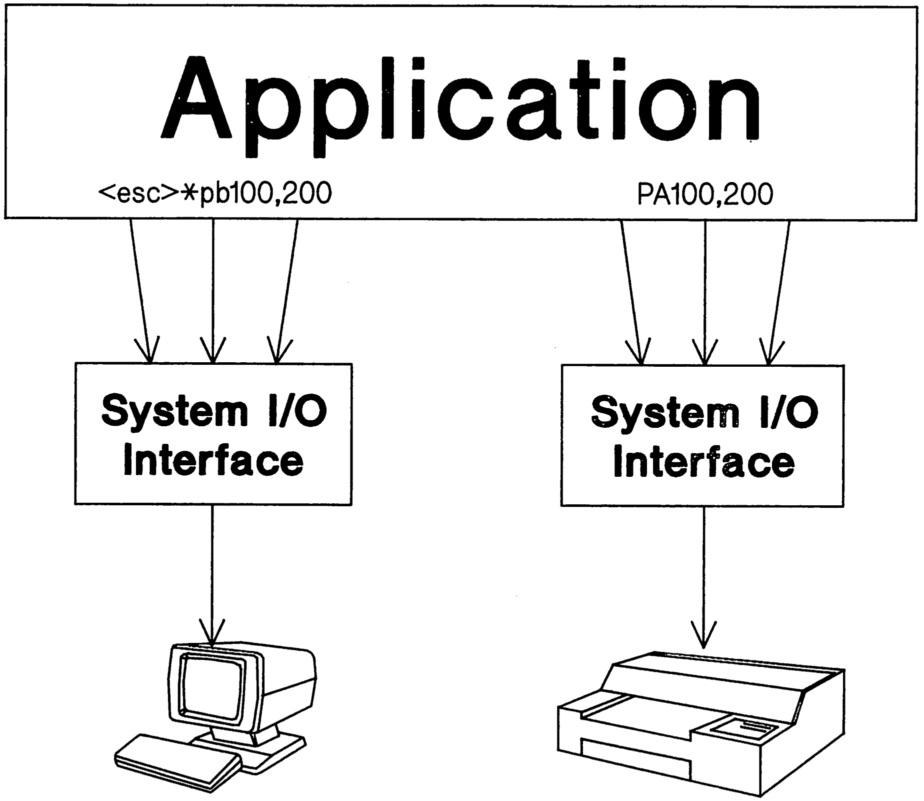

Figure 1 – Device dependent commands scattered within application program

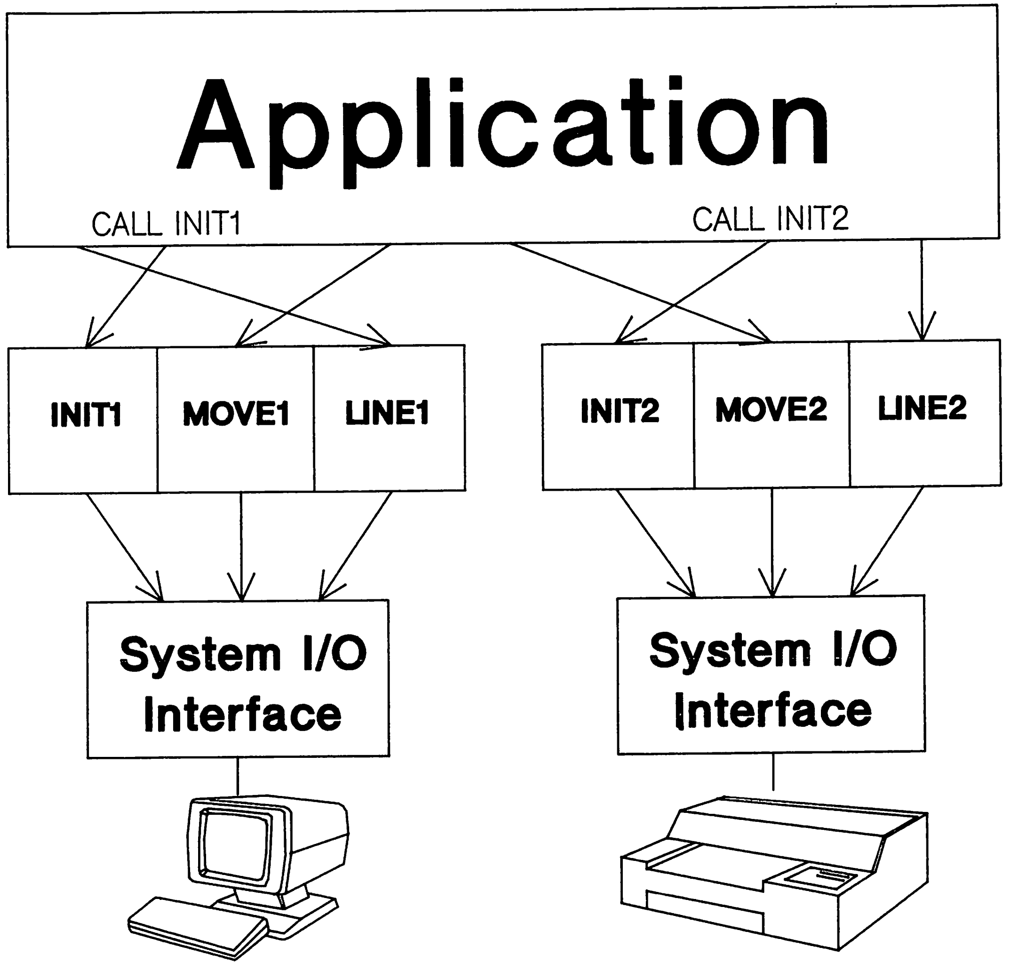

Figure 2 – One set of subroutines for each device

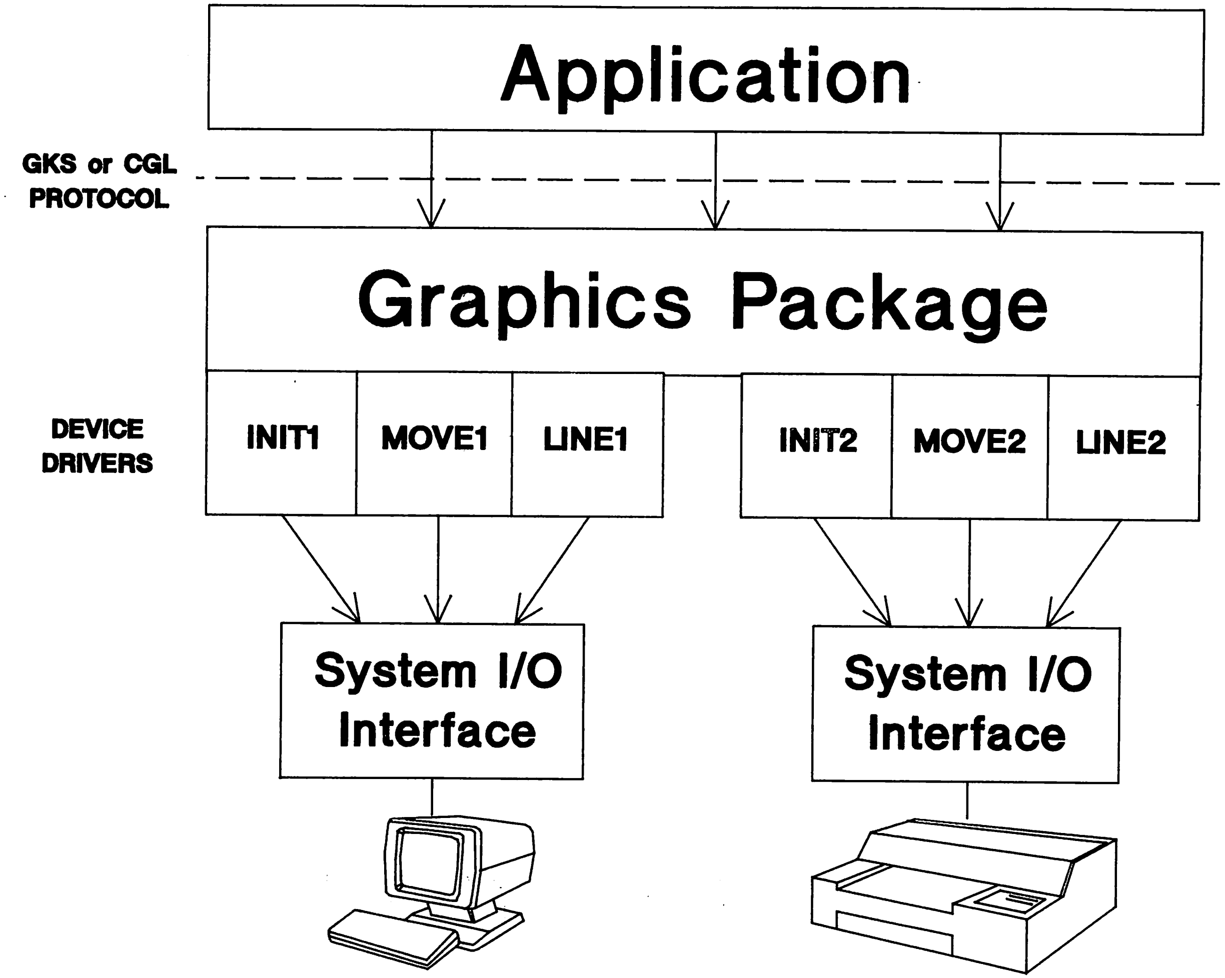

Figure 3 – Graphics package with its device drivers

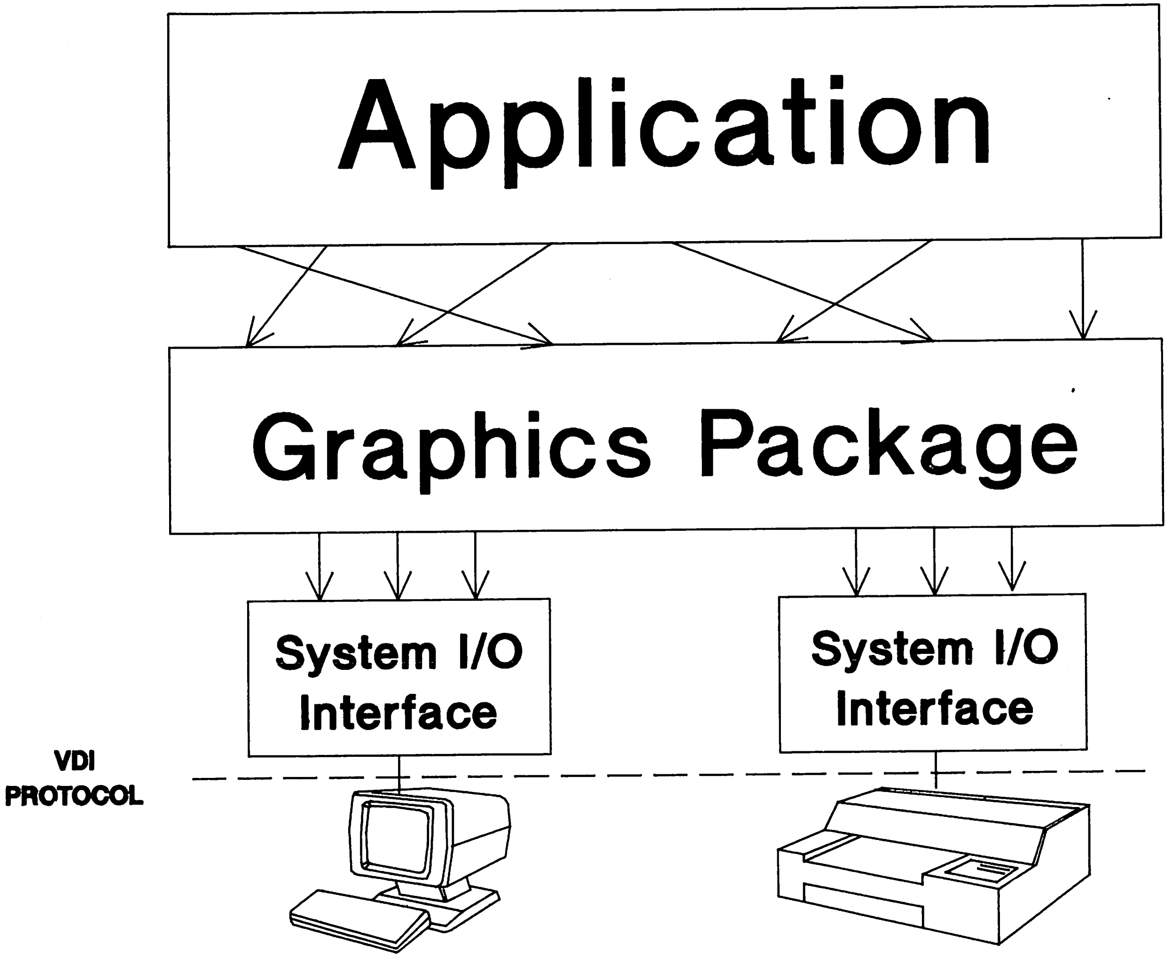

Figure 4 – Devices with internal VDI drivers