6-4-1982

Overview of the University of Pennsylvania CORE System Standard Graphics Package Implementation

Frederick P. Stluka

Brian F. Saunders

Paul M. Slayton

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-82-17.

Overview of the University of Pennsylvania CORE System Standard Graphics Package Implementation

Abstract

The CORE System is a proposed standard for a device-independent graphics system. The concept of a device-independent system was first developed in 1977 by the Graphics Standards Planning Committee (GSPC) of ACM Siggraph and later refined in 1979 [1,2]. The CORE System design has received favorable reviews and has been implemented by various vendors at several universities, and other computing facilities (e.g. [3,7]). The main objectives of the CORE System are to provide uniformity, compatibility, and flexibility in graphics software. Three advantages that the CORE system provides over non-standard graphics systems are device independence, program portability, and functional completeness.

A large number of different graphics hardware devices currently exist with a wide range of available functions. The CORE System provides device independence by shielding the applications programmer from specific hardware characteristics. The shielding is at the functional level: the device-independent (DI) system uses internal routines to convert the application programmer's functional commands to specific commands for the selected hardware device driver (DD). The progammer describes a graphical world to the CORE System in device-independent normalized device coordinates. The programmer also specifies the viewport on the logical view surface (output device) where a picture segment is to be placed.

As the CORE System becomes the accepted standard graphics package, program portability will become more feasible. Program portability means the ability to transport application programs between two sites without requiring structural modifications. The CORE System was designed for functional completeness so that any graphics function a programmer desires is either included within the system or can be easily built on top of CORE System routines.

Disciplines

Computer Engineering | Computer Sciences

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-82-17.

MS-CIS-82-17

OVERVIEW

of the

UNIVERSITY OF PENNSYLVANIA

C O R E S Y S T E M

STANDARD GRAPHICS PACKAGE

IMPLEMENTATION

Frederick P. Stluka

Brian F. Saunders

Paul M. Slayton

Norman I. Badler

Department of Computer and Information Science

Moore School D2

University of Pennsylvania

June 4, 1982

I. Introduction to the CORE System

The CORE System is a proposed standard for a device-independent graphics system. The concept of a device-independent system was first developed in 1977 by the Graphics Standards Planning Committee (GSPC) of ACM Siggraph and later refined in 1979 [1,2]. The CORE System design has received favorable reviews and has been implemented by various vendors at several universities, and other computing facilities (e.g. [3,7]). The main objectives of the CORE System are to provide uniformity, compatibility, and flexibility in graphics software. Three advantages that the CORE system provides over non-standard graphics systems are device independence, program portability, and functional completeness.

A large number of different graphics hardware devices currently exist with a wide range of available functions. The CORE System provides device independence by shielding the applications programmer from specific hardware characteristics. The shielding is at the functional level: the device-independent (DI) system uses internal routines to convert the application programmer's functional commands to specific commands for the selected hardware device driver (DD). The progammer describes a graphical world to the CORE System in device-independent normalized device coordinates. The programmer also specifies the viewport on the logical view surface (output device) where a picture segment is to be placed.

As the CORE System becomes the accepted standard graphics package, program portability will become more feasible. Program portability means the ability to transport application programs between two sites without requiring structural modifications. The CORE System was designed for functional completeness so that any graphics function a programmer desires is either included within the system or can be easily built on top of CORE System routines.

II. Characteristics of the University of Pennsylvania CORE System

In 1980, four senior students at the University of Glasgow implemented a CORE System as their senior project [3]. Like our system, theirs is written in Pascal with one device driver. Our CORE System is fully 3D while they created a 2D system. Our environment used a Ramtek GX-100B as its graphics device. We followed the guidelines of the 1979 GSPC design document while they followed the 1977 GSPC version. Both implementations used similar but

independently-derived linked list data structures to store primitives and segments.

According to the GSPC 1979 design document [1], we implemented a CORE system with Output Level 3C (including retained segments, highlighting, visibility, pick detection, and full image transformation: 3D scale, rotation and translation), and Input Level 2 (synchronous input). Due to the time constraints of a single year senior project, several features were not implemented: text primitives, logical input device echoing, pixel arrays for polygon fill, patterned polygon edge style, hidden surface removal, and the Metafile. It was determined that these features could be added to the system at a later data and still permit an acceptably functional graphics system. The system has been designed so that these features may be easily added. A device dependent text primitive may be used via the ESCAPE feature.

III. Pascal Implementation

III.A. The Pascal Language

This CORE System was implemented in Pascal on the Moore School's UNIVAC 1100/61. It is standard Pascal, but has some very nice enhancements. Pascal was chosen as the implementation language because of its excellent data structuring facilities, because it is the most widely used language at the Moore School, and because one of the authors (Stluka) had spent the previous summer installing Pascal on the UNIVAC 1100, and was therefore the local expert in the language and its implementation.

III.B. External Procedures

The CORE System physically consists of two libraries containing a relocatable element for each routine. One library contains all the DI routines, and the other library contains the DD routines for the Ramtek. The application programmer links to these two libraries before running a program.

Pascal 1100 supports the creation of relocatable Pascal elements. This feature gives Pascal several advantages over languages that do not allow creation of relocatables. One is that the implementers of the system are able to compile each routine separately into the library. The only way to create a library without separate relocatables is to make all the procedures internal to a large dummy routine and then just compile the dummy routine. This adds significantly to the development time if all routines

must be compiled together for a change in only one routine. Another advantage is that source code modules are small and easily handled by the text editor. Load modules created by an application program are also much smaller since they contain only the code for those library routines which are needed. This is much better than having to put a large dummy routine around all the procedures in the library. In that case, each application program load module would be huge, containing the code for every routine in the library, whether it is called or not.

It should be obvious therefore that the ability to create relocatable elements is necessary to any graphics development project that is building Pascal code libraries. This feature saved many hours of work and helped to make application programs as small and efficient as possible.

III.C. INCLUDE Facility

The use of INCLUDE statements made the system very modular and flexible. Each routine used INCLUDEs to declare all the global data types, global variables, and procedure definitions that are directly callable by the particular routine. The application program need INCLUDE only one file element to declare all the global data types, global variables, and definitions of all the user-callable routines.

III.D. Common Data Area

III.D.1. Input record

The input record contains all the characteristics for the six logical input devices. It is a linked list structure created and maintained by the DI routines which deal specifically with input. It is loaded by the device driver with initial values specific to the available physical input devices attached to the selected output device. Each record contains a fixed part and a variant part with a section for characteristics particular to each logical input device.

III.D.2. Escape record

The escape record is created by the application programmer who wishes to use the ESCAPE facility. The record contains arrays for integer, real, and character parameters. The record was designed in such a manner to allow for any possible type of parameter passing to occur. The application program fills the three arrays with the desired parameters of

each type and specifies the number of each in the record. The ESCAPE procedure is then called with an option code to select from up to 11 (an arbitrary limit) user-accessible DD features. The selected routine within the ESCAPE interprets the parameter fields of the record accordingly.

III.D.3. Control variables

The control variables are used throughout the CORE System and are part of the global export variables. Examples of control variables are system flags and characteristics, and the current operating position.

III.D.4. View State record

This record contains viewing parameters that are used in the 3D viewing pipeline. Typical variables in this record are flags and matrices, as well as viewing vectors. These variables are not known to the application programmer and are updated only through DI routines.

III.D.5. Color Table record

The color table is actually three arrays of color values representing amounts of Red, Green, and Blue. The record also contains the number of increments of each color array, and the low and high values. Color values are stored normalized in the range 0.0 to 1.0. It is the responsibility of each device driver to convert this color value into an appropriate color value for the selected device. The record is initialized and maintained by DI routines.

III.D.6. Device Driver record

This record contains specific information for each device driver to be included in the CORE System. The fields in the record are fixed, but their values are initialized at run time by the selected device driver. For example, each driver will set the variable VSPIXX to the maximum size of its horizontal screen resolution.

III.E. Pseudo-Display File

III.E.1. Segment records

The pseudo-display file is a two level Pascal linked list structure that contains both characteristics and coordinates of graphical data. The file is a segmented dynamic display structure which is composed of a variable number of segments of

dynamic size which describe the picture stored by the CORE System. The two levels of the structure are segments and primitives.

Each segment is a logical portion of the picture defined by the application programmer. Each segment has a set of dynamic and static attributes associated with it. The one static attribute, the image transformation limit, determines which types of transformations can be applied to the segment's image on the screen. The four dynamic attributes associated with each segments are its name, its current image transformation, its highlighting, and its detectability. Each segment is a record (but not a variant) containing a pointer to the list of primitives contained within the segment as well as all the segment static and dynamic attributes (Fig. 1).

III.E.2. Primitive records

The six types of primitives are LINE, POLYLINE, TEXT, POLYGON, MARKER, and POLYMARKER. Each primitive type has a different set of attributes associated with it. Each primitive is a record containing the necessary information to display the primitive, in addition to its pick identifier and a pointer to the next primitive on the linked list. Since each type of primitive has its own unique set of attributes, Pascal variant records are used. Primitives are stored in LIFO order in the linked list. Thus, when a newframe action takes place, the last primitive added to the pseudo-display file is the first to be drawn on the device. This, as discovered later, is the opposite of the intended temporal priority.

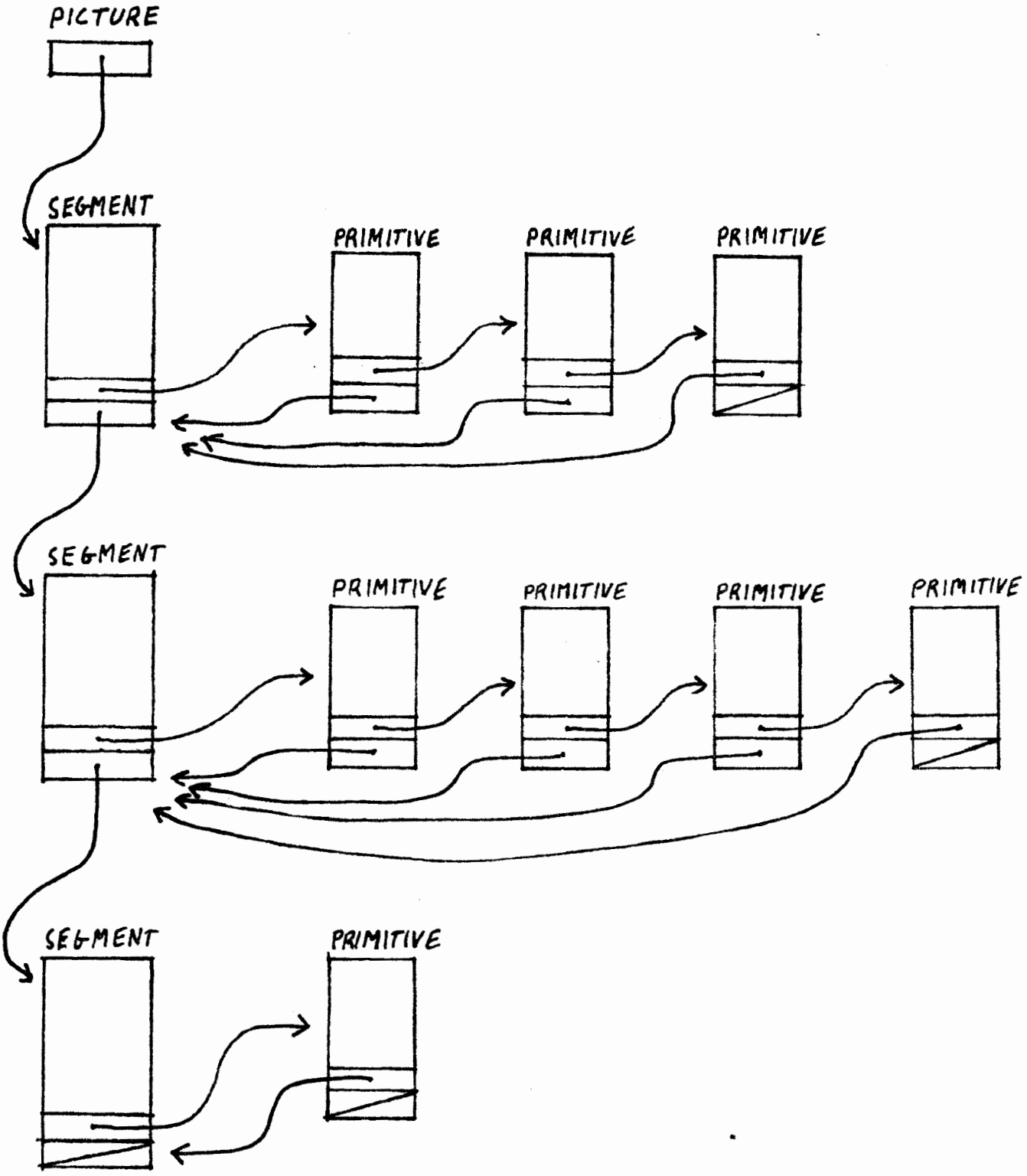
III.F. Viewing Pipeline

The application programmer specifies the picture in any appropriate 3D world coordinate system. To store objects in normalized device coordinates which are directly accessible by the device driver the following picture generation process is used [2,5,6].

First, the world coordinates are converted to normalized clipping coordinates by passing each point through two 4x4 composite viewing matrices. Now all the points are oriented with respect to either a truncated, normalized pyramid for perspective viewing, or a unit cube for parallel viewing.

Clipping is performed by comparing each point to the six planes of the view volume. Points found to be outside the view volume are either clipped to the

Fig. 1 SEGMENTED DYNAMIC DISPLAY FILE



planes or marked as clipped out.

If perspective viewing is desired, the truncated pyramid view volume is converted to a cube view volume ranging from (0,0,-1) to (0,0,1).

All coordinates are then mapped from the normalized view volume to the application specified viewport. In order to simplify the pick operation, an extent box surrounding the primitive is calculated by comparing the fully converted primitive coordinates to determine the maximum and minimum values along each axis [4,6].

III.G. Ramtek Device Driver

Currently, the only graphics device connected to the UNIVAC 1100 is the Ramtek. The Ramtek's current driver, a set of FORTRAN routines, was first converted to Pascal. Starting with this layer of low-level software, a CORE System driver was written. A set of approximately 25 routines was developed to interface the DI CORE system with this low-level software. The driver handles all output primitives and all six logical input devices.

IV. Interactive CORE

An interactive DI CORE routine tester was implemented which allows the application programmer to write graphics programs at the terminal and immediately see the results of specific DI routines. The user can experiment with the CORE system interactively without having to invest time and effort in writing PASCAL batch programs. The Interactive CORE facility is completely menu-driven and prompts the user for all routine parameters individually. In addition to the standard DI CORE routines, two features are included. First, a 3D cube is pre-defined in world coordinates and its viewing parameters can be varied. Second, a debugging feature allows the operator to directly access CORE System global variables. Overall the Interactive CORE facility should be a great aid for understanding the CORE System and writing graphics programs.

V. Effort

The design team consisted of Fred Stluka, Brian Saunders, and Paul Slayton. The implementation team also consisted of Fred Stluka, Brian Saunders, and Paul Slayton. The implementation team worked much harder. Because of the size of the project, the work had to be properly allocated to each member of the

design/implementation team. Stluka was responsible for the design of the pseudo-display file and partial implementation of the viewing pipeline. Saunders was responsible for the design of the input facilities and partial implementation of the viewing pipeline. Slayton was responsible for the design of the device driver format. Throughout the project, each member of the team worked with the others to aid in design and implementation of the other allocated sections of the project, thus this was really a team project. Each one knew what the others were doing and helped out whenever possible.

VI. Conclusion

The project was judged successful both as a educational exercise and as a means of developing portable graphics software in PASCAL, particularly targeted at a UNIVAC 1100. The system is presently being converted to run on a VAX-11/780 and little difficulty is expected. This computer will also permit development of the important asynchronous input level.

VII. Acknowledgments

The team would like to thank Norman Badler for all his help and aid in understanding what the CORE System is all about.

We would also like to thank the Moore School Computing Facility and Skip Dane for making the UNIVAC 1100 and the Ramtek talk to each other so we could draw pretty pictures.

VIII. References

[1] Graphics Standards Planning Committee Report, Computer Graphics vol. 13, No. 3, Aug. 1979.

[2] ACM SIGGRAPH Tutorial on Graphics Standards, Aug. 6-7, 1979, Chicago, IL.

[3] C. J. Nichol and A. C. Kilgour, "A Pascal Implementation of the GSPC CORE Graphics Package," Computer Graphics Vol. 15, No. 4, Dec. 1981, pp. 327-335.

[4] M. T. Garrett, "Logical Pick Device Algorithms for the CORE System," Computer Graphics Vol. 13, No. 4, Feb. 1980, pp. 303-313.

[5] W. M. Newman and R. F. Sproull, Principles of Interactive Computer Graphics, 2nd ed., McGraw-Hill, New York, NY, 1979.

[6] J. D. Foley and A. van Dam, Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading, MA, 1982.

[7] George Washington University Implementation of the 1979 GSPC CORE System, 27 May 1980.