Advantages of Using C Language in Graphics Programming

A.B. Kanawati
North Carolina A&T State University

Abstract

Graphics programming involves, perhaps more than nongraphic programming, a good deal of memory management. This paper emphasizes the use of structures and unions creatively in declaring memory for use in graphics programs, which will help in keeping track of information that, in graphics programming, is often not easily comprehended.

Introduction:

In graphics programming, memory management is an important and sometimes crucial part of the programming process. C gets its power by carefully including the right control, structures, and data types, and allowing their uses to be nearly unrestricted where meaningfully used. Moreover, C places no straight jacket on the programmer's access to the machine. It is impossible to present a thorough review of all aspects of C programming; however, headers, definitions, declarations, and other structural elements will be discussed as they pertain to the graphics programming process.

Declarations:

Declarations serve two purposes: first, they tell the compiler to set aside an appropriate amount of space in memory to hold values associated with variables; and second, they enable the compiler to instruct the machine to perform a specified operation correctly. In general, on a 16-bit processor, 16-bit integers are processed very efficiently. If you declare your graphics variables as integers (type int) you will be using the fastest method. There are two general rules regarding the placement of declarations: first, avoid global variables if possible, because it is difficult to track values changed globally; and second, avoid passing local variables as arguments because of the time involved in pushing and popping variables to service a stack.

Definitions:

typedef is used to abbreviate long declarations. Furthermore, if there are system-sensitive declarations, such as an int that is four-bytes on one system and two-bytes on another, these differences are critical to graphics programs — then the use of typedef may make the porting of the software easier. The programmer achieves a high degree of modularity and portability by using typedef to name derived types and by storing them in header files. Usage of defined macros is faster than calling functions with arguments.

Structures and Unions:

Structures and Unions are very powerful tools for graphics applications. You can use them to take apart and put together all kinds of complex data structures. Memory is allocated to a structure member-by-member. The compiler may need to align certain data on even word boundaries. Declaring the structure members wisely would save a lot of memory as illustrated in the following example:
struct hi {
    char one;
    int two;
    char three;
};

struct ho {
    int two;
    char one;
    char three;
};

main ( )
{
    printf("The size of hi: %d\n", sizeof (struct hi));
    printf("The size of ho: %d\n", sizeof (struct ho));
}

The following output was produced:
The size of hi: 6
The size of ho: 4

A savings of two bytes is accomplished by rearranging members. The savings in memory is rather substantial for an array of structures.

Headers:
The body of your C code is important, but much of the power comes from intelligently designed data structures. One of the earliest considerations in your design should be the design of the data format. The header file helps in such a task. The following file, EGA.h, is an example that shows the EGA Mode 6 mapping of bits onto a byte in memory. If you wish to examine the contents of the entire color specification byte, you can use it as follows:

    color_byte = EGA.all.pixels;
/* file EGA.h */

/* EGA header contains display structures */
typedef unsigned char byte;
typedef unsigned char bits;
#define EGAMODE 6
#define EGASEG 0xB800

union Display {
    struct {
        byte pixels;
    } all;
    struct {
        bits bit0 : 1;
        bits bit1 : 1;
        bits bit2 : 1;
        bits bit3 : 1;
        bits bit4 : 1;
        bits bit5 : 1;
        bits bit6 : 1;
        bits bit7 : 1;
    } bit;
};

As you can see in this example, an integer or unsigned member of a structure or union can be declared to consist of a specified number of bits. The width is at most the number of bits in a machine word. Typically, bit fields are declared as consecutive members of structure, and the compiler packs them into a minimal number of machine words. Whether the compiler assigns the bits in left-to-right or right-to-left order is machine dependent. Although some machines assign bit fields across word boundaries, most of them do not. Thus on a machine with four-byte words, the declaration

    struct abc {
        int a : 1, b : 16, c : 16;
    } x;

would cause x to be stored in two words, with a and b in the first word and c stored in the second. The chief reason for using bit fields is to conserve memory. On machines with four-
byte words we can store 32 one-bit variables in a single word. Alternatively, we could use the 32 char variables. Clearly the amount of memory saved by using bit fields can be substantial. There are some restrictions, however: arrays of bit fields are not allowed and the address operator "&" cannot be applied to bit fields. Unnamed bit fields can be used for padding and word alignment purposes. The unnamed bit field of width 0 is a special bit; it causes immediate alignment on the next word.

In graphics programs the individual elements of the C language do not stand alone, but rather, in conjunction with one another to accomplish efficiency and speed. C combines subtlety and elegance with raw power and flexibility, and firmly puts the graphics programmer in charge.

References:

(1) The C Programming Language, Brian W. Kernighan and Dennis M. Ritchie.