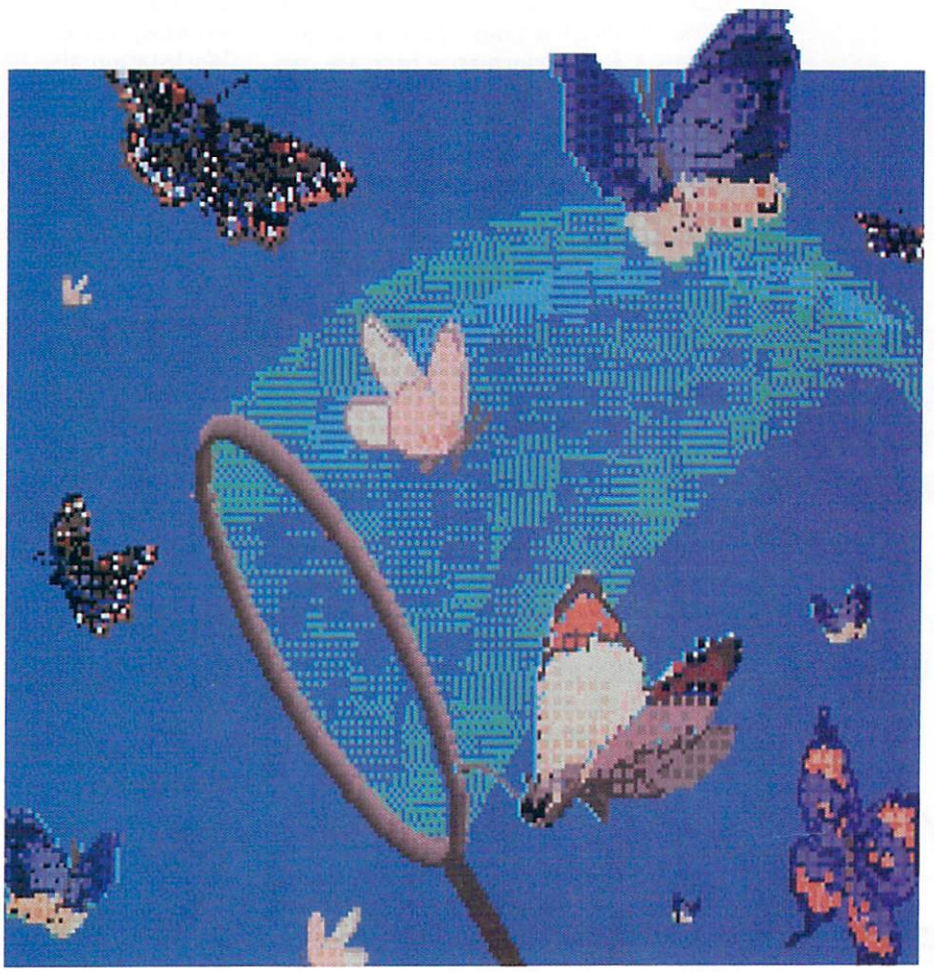
GRAPHICS

that won't stand still



Part I
of a programming
tutorial
on learning
to animate
BOBs and Virtual
Sprites with C.

By David T. McClellan

ILLUSTRATED BY ROGER GOODE

In the article Creating Menus with Intuition in the Jan./Feb. '87 issue of *AmigaWorld*, Vincent Hopson showed you how to program basic Intuition operations—set up Screens, Windows and Menus—and how to handle user interactions through them. This three-part series will build on what you were shown in that article and will show you how to program animation on the Amiga in C. I wrote a program, *pigs1.c* (Listing 1), that I will refer to throughout this article to explain the basic techniques I used.

Due to the amount of information I have to cover, this tutorial will appear in three parts: This part covers the layout of the images; the second will cover initialization and shutdown routines; the final article will discuss object movement, screen update, interaction, and extensions. The corresponding listing sections will be presented with each article.

The *pigs1.c* program builds and animates a simple picture: two pigs running along and hopping over a wall, a flying butterfly and a drifting balloon—all passing across a simple landscape. The pigs and butterfly are built with *Virtual Sprites*, or *VSprites*; the balloon is a *BOB (Blitter Object)*. Both are described below. These animated images move in different ways, and can collide with the sides of the picture. Using them, I am able to demonstrate the major features of Amiga animation: image rendering, movement, screen updating and collision handling.

Using Lattice C (version 3.03), the total source code compiles down to about 24K. I ran it on the AmigaDOS version 1.1 kernel and operating system; changes should be minimal for version 1.2 or for Manx C.

Animation Data Structures

First, I'll discuss the data structures I use in the program. All my static data structures are shown with initialization values in the listing of file *pigs1.c*; most of the #define constants used are contained in the accompanying *pigs.h* include file. File *pigs.h* is included in each of the source files, and includes all required Lattice and Amiga include files.

My NewScreen and NewWindow structures are similar to those in Hopson's article with a few modifications. First, my screen's Type is CUSTOMBITMAP and CUSTOMSCREEN; the window's Type is CUSTOMSCREEN. I need my own BitMap to handle the eight colors and to control updating. The only window Flags I set are WINDOWCLOSE, BORDERLESS and ACTIVATE, so that I get a close gadget, no borders and the window comes up active. To match those, I set the IDCMPFlags CLOSEWINDOW and REQCLEAR, so that I can receive CLOSEWINDOW messages for exiting the program. The screen has no menus attached, and its colors are given early in the *scrcolors*[ ] array.

In the listing, I initialized the NewScreen struct in the static data section; I did the NewWindow struct in my setup() routine (described in the next article). From now on, when I give a routine name in the article I'll follow it with () so you know it's a specific routine. The names of routines in my tutorial program all begin with lowercase letters; Amiga library routines begin in uppercase. Following the screen and window structure declaration, I laid out the VSprites and the BOB; after discussing these in general I'll show you what I did.

Graphic Elements

The Amiga uses two basic types of animation: sprite animation and blitter animation. Collectively, objects from both are called *Graphic Elements*, or *Gels*. Sprites are Gels supported by special hardware; they move independent of, and freely across, the background image (the *playfield*) without affecting it. The Amiga supports eight sprites, each of which is produced by one of eight hardware sprite Direct Memory Access (DMA) channels. (The Workbench pointer is a sprite.) The ►

width of a sprite is limited to 16 pixels; their height is not limited. They can have only three colors, plus "transparent." Sprite images can be specified and changed (like the cloud of Z's the pointer turns into), and are very easy to move. Sprites can optionally be managed by the system software and assigned as VSprites.

VSprites are system software Gels closely related to hardware sprites in data structure and are used by the software graphics kernel to expand on sprites. VSprites are temporarily assigned to sprites as needed. More than eight VSprites can be active at one time because, as the screen is refreshed, the Amiga can reuse hardware sprites to carry VSprite images. Each VSprite has its own special color set, the colors of which do not have to be part of the screen's set. VSprites also allow you to do double buffering and collision detection (running into borders or other VSprites or BOBs), which hardware sprites can't do on their own. Certain restrictions apply to VSprite use: More than four VSprites on one horizontal screen line (scanline) can cause problems with color interaction, and there can't be more VSprites on a scanline than there are sprites. VSprites are easier to use and track than sprites, so I used them for my demo.

Blitter animation offers more freedom than VSprite animation. BOBs can be as wide as you want and can have as many colors as the screen, up to 32, but drawing them costs more time, in blitter work. Unlike VSprites, which are drawn over the background, BOBs are drawn *into* the background scene. Therefore, each time a BOB moves, the Amiga blitter has to save a copy of the area it is moving the BOB to, restore the copy it saved of the area the BOB was sitting in, and then redraw the BOB. This takes time when several BOBs are hopping about, each of which can potentially overlay not only background, but other BOBs as well! I have one eight-color BOB I move around to show how setup and movement of this differs from VSprite usage.

Figure 1. The butterfly VSprite image with its bit and word mappings.

```
Line    Image Colors (blank = trans., B = black, R = red, G = green)
        Bit (from left) 0123456789012345
----    ----------------------------------------------
   0                         B    B
   1                      RR    B  B    RR
   2                      RGRGR  BB  RGRGR
   3                       GRRGRGBBGRGRRG
   4                        GGBGRBBRGBGG
   5                         RBRGBBGRBR
   6                        RGRG BB GRGR
   7                       RGRG  BB  GRGR
   8                      R      BB      R
```

```
Bitmap  Data: Bits 0-15 low                                  Word array
Line    order color bits,                                      (from Bitmap)
                                                               in hexadecimal
         Low bit of color     High bit of color
        0123 4567 8901 2345  6789 0123 4567 8901    Low, then High
        -------------------  -------------------    --------------
    0   0000 0100 0010 0000  0000 0000 0000 0000    0x0420, 0x0000
    1   0000 0010 0100 0000  1100 0000 0000 0011    0x0240, 0xC003
    2   0101 0001 1000 1010  1111 1000 0001 1111    0x518A, 0xF81F
    3   0100 1011 1101 0010  0111 1110 0111 1110    0x4BD2, 0x7E7E
    4   0011 1101 1011 1100  0011 0110 0110 1100    0x3DBC, 0x366C
    5   0000 1011 1101 0000  0001 0110 0110 1000    0x0BD0, 0x1668
    6   0001 0101 1010 1000  0011 1100 0011 1100    0x15A8, 0x3C3C
    7   0010 1001 1001 0100  0111 1000 0001 1110    0x2994, 0x781E
    8   0000 0001 1000 0000  1000 0000 0000 0001    0x0180, 0x8001
```

Pigs, Butterflies and Balloons

I use six VSprite structures: two apiece for each of the two pigs, one for the butterfly, and one for the BOB. (Each BOB has an associated VSprite structure for movement and image control, though it doesn't actually use a hardware sprite.) I draw each pig with two side-by-side VSprites, allowing me 32 pixels of image width; I just have to be careful not to visually "dissect" the pigs on-screen when I move them. The pigs are 11 pixels tall, and colored with pink, red and dark blue (and, of course, transparent). The butterfly is only 16 pixels wide, uses one VSprite, and is nine pixels tall. Its colors are red, green and black. A close look at the butterfly will illustrate how to lay out a VSprite.

In the include file *pigs.h*, I first define some constants to use later when allocating the VSprite: height, width, etc. (Some of the defines are for my VUserStuff VSprite field, which I will get into in the articles on setup and movement.) First, I set the butterfly's colors in the array *BflyColors*[ ], giving the red-green-blue values for colors 1, 2, and 3 of the VSprite. Transparent is always color 0. Color 1 is black; color 2 is red, and color 3 is green. Using these indices I describe the butterfly's image.

A VSprite's image is described one screen line at a time, two 16-bit "words" per line (two bits of color by 16 pixels of width). The first word of each line contains the low-order 16 bits of its color indices; the second word contains the high-order bits. I drew the butterfly on graph paper first, picked colors for each pixel, and then used the indices to get the word values. See Figure 1 for the image and the bit and word mappings.

Using the image in Figure 1, for example, the first four pixels of line 1 are red, red, transparent, transparent. The corresponding VSprite color indices are 2, 2, 0, 0 (or 10, 10, 00, 00 in binary). Splitting each index's two bits in two to get the two color planes gives 0000 and 1100, 0 and C in hex. Doing this for an entire image takes time, but isn't too difficult. I drew the pigs using a paint program and used the magnify option to blow up the pixels and transfer them to graph paper for translation into hex numbers.

Besides its image and color, each VSprite has a position (X,Y), and Flags (mostly used for BOB VSprites). The only Flags bit of interest is VSPRITE; if set, it indicates the VSprite is an independent VSprite; if not set, the VSprite belongs to a BOB. The other fields of the VSprite pertain more to movement and collision detection, so I will delay describing them until next issue.

BOBs are not much more complicated than VSprites; they have a few more flags and elements to be initialized, and their image array is laid out a little differently. They also have an associated VSprite ►

(BOBVSprite) for their VSprite-similar data (image, X,Y, collision information, etc.). The BOB image is laid out one whole bit plane at a time as opposed to the alternating plane method used for VSprites. I have three bit planes so I had three sections of my array—24 lines each and two words wide (the BOB is 32 pixels wide—two words—and 24 pixels high).

Dissecting an image in this form is similar to doing it for a VSprite. I first drew the image in eight colors (including transparent, see the OVERLAY flag below). I then transferred the color index number for each pixel to its corresponding location on graph paper. Then, scanline by scanline, I converted each number to three-bit binary and split the three bits into three separate parts of the image array. For example, on the fourth line of the BOB, the first six pixels were transparent (color 0), the next four yellow (color 3), then one red (color 6), one black (color 1), two more reds, three dark blues (color 7), three greens (color 4), one white (color 2), one black, two more whites, two more yellows and six more transparent pixels for a total of 32. The 32 color numbers, separated into three planes to become six sixteen-bit hexadecimal numbers, were thus:

Plane 0 (lowest bit):	0x03D3 0x84C0
Plane 1 (middle bit):	0x03CF 0x8BC0
Plane 2 (high bit):	0x002F 0xF0C0

If I do much more animation, I will either write or buy a program to generate the C data structures from an image; it takes a while to translate the bits!

Other BOB fields allow controlled variations of the image displayed. Two of the BOB's VSprite's fields control which of these color bit planes are drawn at runtime (PlanePick and PlaneOnOff); I set mine to draw all my bit planes, all the time. By changing these fields before you update the screen you can cause a single BOB to display several images.

A BOB keeps a few other arrays, too: for saving the background (SaveBuffer), and for use with double-buffered screens (DBuffer, DBuffer->BufBuffer); more on that in the third article. The BOB has flags for both its own Flags field (for use with AnimComps), and two special VSprite flags I use, SAVEBACK and OVERLAY. SAVEBACK causes the Amiga to save the background pixels the BOB overlays in the BOB's SaveBuffer; if it is off, the BOB acts like a paintbrush and smears across the background. OVERLAY causes the BOB to be drawn like a VSprite with transparency (color 0), so the background image shows through.

The memory allocated for these structures *must* be in the lower 512K of the Amiga's memory—the chip memory. My Amiga has only 512K, so I know my statically allocated image arrays are in chip memory. If the program were to be loaded into memory above 512K then I would have had to allocate chunks of chip memory the size of the images and then copy the static image arrays into those chunks.

That's it, for now. In the next article I'll cover program initialization and linking these VSprite and BOB data structures into the Amiga's runtime screen display, along with the relevant parts of the Pigs program.■

David McClellan is a Contributing Editor to AmigaWorld. *Address correspondence to him at 104 Chevron Circle, Cary, NC 27511.*

Listing 1. Pigs1.c

```
/* Program: pigs.c, part 1 by David Mclellan
This program creates a simple animation demo with
several VSprites and a Bob moving across a
background scene. The moving objects are: 2 pigs,
each consisting of 2 VSprites side-by-side; 1
butterfly which uses 1 VSprite, and one 8-color Bob.
Note:  This program was written under version 1.1
File: pigsl.c--data structures for tutorial program.
Include Files. Note - pigs.h includes all required
                Amiga Include files.  */
#include "pigs.h"
/* Global Data Structures */
/* Library Pointers */
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
/* Screen Colors */
struct color4 scrcolor[8] = {       /* Screen colors */
        0x00, 0x00, 0x0F,          /* Color 0 - Blue */
        0x00, 0x00, 0x00,         /* Color 1 - Black */
        0x0F, 0x0F, 0x0F,         /* Color 2 - White */
        0x0F, 0x0F, 0x00,        /* Color 3 - Yellow */
        0x08, 0x0F, 0x08,  /* Color 4 - Light Green */
        0x08, 0x08, 0x08,  /* Color 5 - Medium Grey */
        0x0F, 0x00, 0x00,           /* Color 6 - Red */
        0x0F, 0x0C, 0x00         /* Color 7 - Orange */
};
/* Screen Miscellanea */
struct TextAttr ffont = {
   "topaz.font", TOPAZ_SIXTY,
   FS_NORMAL,
   FPF_ROMFONT
};
struct NewScreen newscr = {
   0,0, WIDTH, HEIGHT, /* Left, Top, Width, Height */
   3,                                    /* Depth */
   BLACK, BLUE,              /* DetailPen, BlockPen */
   0,                                 /* ViewModes */
   CUSTOMSCREEN | CUSTOMBITMAP,            /* Type */
   &ffont,                                 /* Font */
   "Animated Pigs",                       /* Title */
   NULL,                               /* No Gadgets */
   NULL             /* Until Allocate CustomBitMap */
};
struct Screen *scr; /* Real screen from OpenScreen */
struct BitMap *bitm = NULL;
                        /* Custom BitMap storage */
/* Window Info */
struct NewWindow newwin;
                  /* Initialized in setup routine */
struct Window *win;/* Real window, from OpenWindow */
struct IntuiMessage *message; /* For getting messages
              from Intuition, such as CLOSEWINDOW */
unsigned long close_mask = 0;          /* Mask used by
           close_up_shop to decide what to close. */
/* Gel Information; tracks VSprites and Bobs */
struct GelsInfo *GInfo;              /* For InitGels */
/* VSprites and Related Info */
/* Butterfly Info */
short bfly_ys[BFLY_CYCLE] = { 4, 8, 14, -6,  0, -10,
             2, -20,  4, 8, -16, 2,  -6, 14, 0, 2};
/* Butterfly has cyclical up/down moves */
/* Colors */
WORD PigColors[] = { 0x0F44,                 /* Pink */
                     0x0F00,                 /* Red  */
                     0x000C             /* Dark Blue */
```

Listing continued on p. 98