Collision Detection Between Animation Objects

by Ken Farinsky

Collision detection allows the system to manage Gel-to-Border and Gel-to-Gel collision handling for an application. "Gels" are VSprite-based objects that may be displayed and moved about the Amiga display. These objects include VSprites (Virtual Sprites), Bobs (Blitter Objects), and AnimObs (Animation Objects). For a detailed discussion of collision detection between simple Gels (such as VSprites and Bobs), see the "Graphics: Sprites, Bobs and Animation" chapter in the *ROM Kernel Manual: Libraries and Devices*.

The basic premise of Amiga collision detection is to call a user supplied collision routine when two objects overlap. This is complicated by two further enhancements. First, borders are counted as objects and special-case processing of collisions is provided between Gels and the border. Second, each Gel has two masks that indicate how pairs of objects will react when they collide.

Collision detection between objects such as VSprites and Bobs is conceptually very simple. Each object is built upon a single underlying structure, the VSprite structure. After the Gel has been added to the system, the DoCollision() routine scans the list of VSprites for overlapping objects and calls the appropriate user-supplied collision routine in the event of a collision. Since there is a one-to-one correspondence between VSprites and objects, it is obvious which VSprite structure belongs to which object on the screen.

Animation Objects Are Not Simple Gels

Animation Objects contain multiple VSprite structures which are linked to form a larger object. When the object is being animated, the system cycles through these underlying components, displaying each at the appropriate moment. At any given point in time the system may be in the process of removing the last displayed component and adding in the next one. During such a component switch, the system will have two VSprites added for a single object: one is in the process of being removed and one is in the process of being displayed.

Special Initialization Steps for Complex Gel Collision Detection

When setting-up a multi-component Gel, each underlying VSprite structure must be initialized for collision, as the system will not look to the others for more information. An AnimOb's collision detection information need not be static. The collision information can change from frame to frame in the same manner image data can change. The example shows a special case, where all frames are similar enough that they have identical collision information.

The following fields in the VSprite structure must be properly initialized for system collision detection: HitMask, MeMask, BorderLine and CollMask. HitMask and MeMask tell the system which collision routine to call when a collision is detected. BorderLine and CollMask are "shadow masks" that are used to determine if this component overlaps any other object. These values are set just as they would be for a single VSprite or a Bob.

Note that it is impossible for components of a single AnimOb's sequence to collide with one another. However, in an AnimOb that contains multiple sequences, it is possible for a component in one sequence to collide with a component from a different sequence. This can be prevented through careful selection of HitMask and MeMask values, or can be detected in the individual collision routines.

What Happens When Complex Gels Collide

Collisions between Animation Objects *should* be fairly simple to process. The DoCollision() routine *should* examine all displayed objects for both overlap of objects and for correct interaction between the HitMask and MeMask, then call the appropriate collision routine.

Unfortunately, the DoCollision() routine does not ignore the previous AnimOb component, which is in the process of being removed from the display. These components are moved by the system to a position well above and to the left of the visible display. When the Gels are next drawn, the removed component will be moved off-screen, causing it to disappear from the display. At the same time, the new component will be drawn on-screen. This would not be a problem if the removed components did not overlap, however, all components that are being removed are positioned at the same off-screen location. This means that they all overlap and collisions will occur between all pairs of components that are in the process of being removed.

The `BOBSAWAY` Flag Comes to the Rescue

To handle collisions between pairs of AnimObs, the Gel-to-Gel collision routines must ignore the objects that are being removed. There are two ways to handle this. The first is to check for large negative position values in the X and Y variables of the VSprite structure. This works, but is not the best way to check for removal. The second, and preferred method, is to look at the flags of the associated Bob. If the `BOBSAWAY` flag is set, then the bob is being removed and should not be used in the collision routines. Note that this has been changed for 2.0, which will only look at active VSprites for collision detection.

The example code is made up of four files:

1) animtools.h - include file necessary to use animtools.
2) animtools_proto.h - prototypes for animtools functions.
3) animtools.c - animtools functions. These simplify the use of the animation sub-system.
4) collide.c - example code.

The example code was published in the latest release of the *ROM Kernel Manual: Libraries & Devices*. It has been updated and reprinted here. The example can be compiled and linked using Lattice C with these commands:

```
lc -b1 -cfist -v -y -ocollide.o collide.c
lc -b1 -cfist -v -y -oanimtools.o animtools.c
blink from lib:c.o collide.o animtools.o library
                           lib:lc.lib lib:amiga.lib to collide
```

1) /* File: animtools.h */

```
#ifndef GELTOOLS_H
#define GELTOOLS_H

/* these data structures are used by the functions in animtools.c to
** allow for an easier interface to the animation system.
*/

/* data structure to hold information for a new vsprite.
** note that:
**     NEWVSPRITE myNVS;
** is equivalent to:
**     struct newVSprite myNVS;
*/
typedef struct newVSprite
    {
    WORD           *nvs_Image;      /* image data for the vsprite   */
    WORD           *nvs_ColorSet;   /* color array for the vsprite  */
    SHORT           nvs_WordWidth;  /* width in words               */
    SHORT           nvs_LineHeight; /* height in lines              */
    SHORT           nvs_ImageDepth; /* depth of the image           */
    SHORT           nvs_X;          /* initial x position           */
    SHORT           nvs_Y;          /* initial y position           */
    SHORT           nvs_Flags;      /* vsprite flags                */

/* THIS HAS CHANGED for amiga mail. **********************************
**
** Add two lines to the NEWVSPRITE structure.
** This allows the created VSprite structures to be initialized with
** HitMask and MeMask information.
*/
    USHORT          nvs_HitMask;    /* Hit mask.                    */
    USHORT          nvs_MeMask;     /* Me mask.                     */
/*
** END OF CHANGE for amiga mail.    **********************************
*/
    } NEWVSPRITE;

/* data structure to hold information for a new bob.
** note that:
**     NEWBOB myNBob;
** is equivalent to:
**     struct newBob myNBob;
*/
typedef struct newBob
    {
    WORD       *nb_Image;       /* image data for the bob       */
    SHORT       nb_WordWidth;   /* width in words               */
    SHORT       nb_LineHeight;  /* height in lines              */
    SHORT       nb_ImageDepth;  /* depth of the image           */
    SHORT       nb_PlanePick;   /* planes that get image data   */
    SHORT       nb_PlaneOnOff;  /* unused planes to turn on     */
    SHORT       nb_BFlags;      /* bob flags                    */
    SHORT       nb_DBuf;        /* 1=double buf, 0=not          */
    SHORT       nb_RasDepth;    /* depth of the raster          */
    SHORT       nb_X;           /* initial x position           */
    SHORT       nb_Y;           /* initial y position           */
```

```
/* THIS HAS CHANGED for amiga mail. **********************************
**
** Add two lines to the NEWBOB structure.
** This is only used to pass the information to the VSprite structure
** in the makeBob() routine.
*/
    USHORT      nb_HitMask;     /* Hit mask.                    */
    USHORT      nb_MeMask;      /* Me mask.                     */
/*
** END OF CHANGE for amiga mail.    **********************************
*/
    } NEWBOB ;

/* data structure to hold information for a new animation component.
** note that:
**     NEWANIMCOMP myNAC;
** is equivalent to:
**     struct newAnimComp myNAC;
*/
typedef struct newAnimComp
    {
    WORD  (*nac_Routine)(); /* routine called when Comp is displayed.   */
    SHORT   nac_Xt;         /* initial delta offset position.           */
    SHORT   nac_Yt;         /* initial delta offset position.           */
    SHORT   nac_Time;       /* Initial Timer value.                     */
    SHORT   nac_CFlags;     /* Flags for the Component.                 */
    } NEWANIMCOMP;

/* data structure to hold information for a new animation sequence.
** note that:
**     NEWANIMSEQ myNAS;
** is equivalent to:
**     struct newAnimSeq myNAS;
*/
typedef struct newAnimSeq
    {
    struct AnimOb  *nas_HeadOb; /* common Head of Object.               */
    WORD   *nas_Images;         /* array of Comp image data             */
    SHORT  *nas_Xt;             /* arrays of initial offsets.           */
    SHORT  *nas_Yt;             /* arrays of initial offsets.           */
    SHORT  *nas_Times;          /* array of Initial Timer value.        */
    WORD (**nas_Routines)();    /* Array of fns called when comp drawn  */
    SHORT   nas_CFlags;         /* Flags for the Component.             */
    SHORT   nas_Count;          /* Num Comps in seq (= arrays size)     */

/* THIS HAS CHANGED for amiga mail. **********************************
**
** Delete two lines from the NEWANIMSEQ structure.
** These are removed becaues the information is now contained in the
** NEWBOB and NEWVSPRITE structures.
**
**  SHORT   nas_HitMask;
**  SHORT   nas_MeMask;
**
** END OF CHANGE for amiga mail.    **********************************
*/
    SHORT   nas_SingleImage;    /* one (or count) images.               */
    } NEWANIMSEQ;

#endif
```

2) /* File: animtools_proto.h */

```
struct GelsInfo *setupGelSys(struct RastPort *rPort, BYTE reserved);
VOID cleanupGelSys(struct GelsInfo *gInfo, struct RastPort *rPort);

struct VSprite *makeVSprite(NEWVSPRITE *nVSprite);
struct Bob *makeBob(NEWBOB *nBob);
struct AnimComp *makeComp(NEWBOB *nBob, NEWANIMCOMP *nAnimComp);
struct AnimComp *makeSeq(NEWBOB *nBob, NEWANIMSEQ *nAnimSeq);

VOID freeVSprite(struct VSprite *vsprite);
VOID freeBob(struct Bob *bob, LONG rasdepth);
VOID freeComp(struct AnimComp *myComp, LONG rasdepth);
VOID freeSeq(struct AnimComp *headComp, LONG rasdepth);
VOID freeOb(struct AnimOb *headOb, LONG rasdepth);
```

3) /* animtools.c 19oct89 original code by Dave Lucas.

```
** rework by (CATS)
**
** This file is a collection of tools which are used with the VSprite, Bob
** and Animation system software. It is intended as a useful EXAMPLE, and
** while it shows what must be done, it is not the only way to do it.
** If Not Enough Memory, or error return, each cleans up after itself
** before returning.
**
** NOTE:  these routines assume a very specific structure to the
** gel lists.  make sure that you use the correct pairs together
** (i.e. makeOb()/freeOb(), etc.)
**
** lattice c 5.04
** lc -b1 -cfist -v -y -oanimtools.o animtools.c
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <graphics/gfx.h>
#include <graphics/gels.h>
#include <graphics/clip.h>
#include <graphics/rastport.h>
#include <graphics/view.h>
#include <graphics/gfxbase.h>

/* THIS HAS CHANGED for amiga mail. **********************************
**
** In the RKM it was assumed that animtools stuff was in a different
** directory from the example program.
**
**      #include "/animtools/animtools.h"
**      #include "/animtools/animtools_proto.h"
**
** Now they are in the same directory as the example:
**
** END OF CHANGE for amiga mail.    **********************************
*/
#include "animtools.h"
#include "animtools_proto.h"

#include <proto/all.h>
```

```
/*-------------------------------------------------------------
** setup the gels system.  After this call is made you can use
** vsprites, bobs, anim comps, and anim obs.
**
** note that this links the GelsInfo structure into the rast port,
** and calls InitGels().
**
** all resources are properly freed on failure.
**
** It uses information in your RastPort structure to establish
** boundary collision defaults at the outer edges of the raster.
**
** This routine sets up for everything - collision detection and all.
**
** You must already have run LoadView before ReadyGelSys is called.
*/
struct GelsInfo *setupGelSys(struct RastPort *rPort, BYTE reserved)
{
struct GelsInfo *gInfo;
struct VSprite  *vsHead;
struct VSprite  *vsTail;

if (NULL != (gInfo =
     (struct GelsInfo *)AllocMem((LONG)sizeof(struct GelsInfo), MEMF_CLEAR)))
     {
     if (NULL != (gInfo->nextLine =
         (WORD *)AllocMem((LONG)sizeof(WORD) * 8, MEMF_CLEAR)))
         {
         if (NULL != (gInfo->lastColor =
             (WORD **)AllocMem((LONG)sizeof(LONG) * 8, MEMF_CLEAR)))
             {
             if (NULL != (gInfo->collHandler =
                 (struct collTable *)AllocMem((LONG)sizeof(struct
                                                       collTable), MEMF_CLEAR)))
                 {
                 if (NULL != (vsHead = (struct VSprite *)AllocMem(
                     (LONG)sizeof(struct VSprite), MEMF_CLEAR)))
                     {
                     if (NULL != (vsTail = (struct VSprite *)AllocMem(
                         (LONG)sizeof(struct VSprite), MEMF_CLEAR)))
                         {
                         gInfo->sprRsrvd   = reserved;

/* THIS HAS CHANGED for amiga mail. **********************************
**
** In the RKM leftmost and topmost were set to zero.
**
**                       gInfo->leftmost   = 0;
**                       gInfo->topmost    = 0;
**
** This has been changed to one to better keep items inside the
** boundaries of the display.
*/
                         gInfo->leftmost   = 1;
                         gInfo->topmost    = 1;
/*
** END OF CHANGE for amiga mail.    **********************************
*/

                         gInfo->rightmost  =
                             (rPort->BitMap->BytesPerRow << 3) - 1;
                         gInfo->bottommost = rPort->BitMap->Rows - 1;

                         rPort->GelsInfo = gInfo;
```

```
                        InitGels(vsHead, vsTail, gInfo);

                        return(gInfo);
                        }
                    FreeMem(vsHead, (LONG)sizeof(*vsHead));
                    }
                FreeMem(gInfo->collHandler,
                                  (LONG)sizeof(struct collTable));
                }
            FreeMem(gInfo->lastColor, (LONG)sizeof(LONG) * 8);
            }
        FreeMem(gInfo->nextLine, (LONG)sizeof(WORD) * 8);
        }
    FreeMem(gInfo, (LONG)sizeof(*gInfo));
    }
return(NULL);
}

/*-------------------------------------------------------------
** free all of the stuff allocated by setupGelSys().
** only call this routine if setupGelSys() returned successfully.
** the GelsInfo structure is the one returned by setupGelSys().
**
** It also unlinks the GelsInfo from the RastPort.
*/
VOID cleanupGelSys(struct GelsInfo *gInfo, struct RastPort *rPort)
{
rPort->GelsInfo = NULL;

FreeMem(gInfo->collHandler, (LONG)sizeof(struct collTable));
FreeMem(gInfo->lastColor, (LONG)sizeof(LONG) * 8);
FreeMem(gInfo->nextLine, (LONG)sizeof(WORD) * 8);
FreeMem(gInfo->gelHead, (LONG)sizeof(struct VSprite));
FreeMem(gInfo->gelTail, (LONG)sizeof(struct VSprite));
FreeMem(gInfo, (LONG)sizeof(*gInfo));
}

/*-------------------------------------------------------------
** create a VSprite from the information given in nVSprite.
** use freeVSprite() to free this gel.
*/
struct VSprite *makeVSprite(NEWVSPRITE *nVSprite)
{
struct VSprite *vsprite;
LONG            line_size;
LONG            plane_size;

line_size = (LONG)sizeof(WORD) * nVSprite->nvs_WordWidth;
plane_size = line_size * nVSprite->nvs_LineHeight;

if (NULL != (vsprite =
    (struct VSprite *)AllocMem((LONG)sizeof(struct VSprite), MEMF_CLEAR)))
    {
    if (NULL != (vsprite->BorderLine =
        (WORD *)AllocMem(line_size, MEMF_CHIP)))
        {
        if (NULL != (vsprite->CollMask =
            (WORD *)AllocMem(plane_size, MEMF_CHIP)))
            {
            vsprite->Y          = nVSprite->nvs_Y;
            vsprite->X          = nVSprite->nvs_X;
            vsprite->Flags      = nVSprite->nvs_Flags;
            vsprite->Width      = nVSprite->nvs_WordWidth;
```

```
            vsprite->Depth      = nVSprite->nvs_ImageDepth;
            vsprite->Height     = nVSprite->nvs_LineHeight;

/* THIS HAS CHANGED for amiga mail. *********************************
**
** In the RKM, the vsprite HitMask and MeMask were set to an arbitrary
** value (it was assumed that the programmer would later patch these
** values if collision detection was required!)
**
**          vsprite->MeMask     = 1;
**          vsprite->HitMask    = 1;
**
** Here, we have changed the NEWVSPRITE structure to contain these
** values, so this routine can set-up both of them.
*/
            vsprite->MeMask     = nVSprite->nvs_MeMask;
            vsprite->HitMask    = nVSprite->nvs_HitMask;
/*
** END OF CHANGE for amiga mail.    *********************************
*/

            vsprite->ImageData  = nVSprite->nvs_Image;
            vsprite->SprColors  = nVSprite->nvs_ColorSet;
            vsprite->PlanePick  = 0x00;
            vsprite->PlaneOnOff = 0x00;

            InitMasks(vsprite);
            return(vsprite);
            }
        FreeMem(vsprite->BorderLine, line_size);
        }
    FreeMem(vsprite, (LONG)sizeof(*vsprite));
    }
return(NULL);
}

/*-------------------------------------------------------------
** create a Bob from the information given in nBob.
** use freeBob() to free this gel.
**
** A VSprite is created for this bob.
** This routine properly allocates all double buffered information
** if it is required.
*/
struct Bob *makeBob(NEWBOB *nBob)
{
struct Bob         *bob;
struct VSprite     *vsprite;
NEWVSPRITE          nVSprite ;
LONG                rassize;

rassize = (LONG)sizeof(UWORD) *
            nBob->nb_WordWidth * nBob->nb_LineHeight * nBob->nb_RasDepth;

if (NULL != (bob =
    (struct Bob *)AllocMem((LONG)sizeof(struct Bob), MEMF_CLEAR)))
    {
    if (NULL != (bob->SaveBuffer = (WORD *)AllocMem(rassize, MEMF_CHIP)))
        {
        nVSprite.nvs_WordWidth  = nBob->nb_WordWidth;
        nVSprite.nvs_LineHeight = nBob->nb_LineHeight;
        nVSprite.nvs_ImageDepth = nBob->nb_ImageDepth;
        nVSprite.nvs_Image      = nBob->nb_Image;
        nVSprite.nvs_X          = nBob->nb_X;
```

```
        nVSprite.nvs_Y          = nBob->nb_Y;
        nVSprite.nvs_ColorSet   = NULL;
        nVSprite.nvs_Flags      = nBob->nb_BFlags;

/* THIS HAS CHANGED for amiga mail. **********************************
**
** The structure of the program has been changed to pass the values
** of the HitMask and the MeMask down to the VSprite structure.
** Push the values into the NEWVSPRITE structure for use in
** makeVSprite().
*/
        nVSprite.nvs_MeMask     = nBob->nb_MeMask;
        nVSprite.nvs_HitMask    = nBob->nb_HitMask;
/*
** END OF CHANGE for amiga mail.    **********************************
*/

        if ((vsprite = makeVSprite(&nVSprite)) != NULL)
            {
            vsprite->PlanePick = nBob->nb_PlanePick;
            vsprite->PlaneOnOff = nBob->nb_PlaneOnOff;

            vsprite->VSBob   = bob;
            bob->BobVSprite  = vsprite;
            bob->ImageShadow = vsprite->CollMask;
            bob->Flags       = 0;
            bob->Before      = NULL;
            bob->After       = NULL;
            bob->BobComp     = NULL;

            if (nBob->nb_DBuf)
                {
                if (NULL != (bob->DBuffer = (struct DBufPacket *)AllocMem(
                    (LONG)sizeof(struct DBufPacket), MEMF_CLEAR)))
                    {
                    if (NULL != (bob->DBuffer->BufBuffer =
                        (WORD *)AllocMem(rassize, MEMF_CHIP)))
                        {
                        return(bob);
                        }
                    FreeMem(bob->DBuffer,
                                  (LONG)sizeof(struct DBufPacket));
                    }
                }
            else
                {
                bob->DBuffer = NULL;
                return(bob);
                }

            freeVSprite(vsprite);
            }
        FreeMem(bob->SaveBuffer, rassize);
        }
    FreeMem(bob, (LONG)sizeof(*bob));
    }
return(NULL);
}
```

```
/*-------------------------------------------------------------
** create a Animation Component from the information given in nAnimComp
** and nBob.
** use freeComp() to free this gel.
**
** makeComp calls makeBob(), and links the bob into a AnimComp.
*/
struct AnimComp *makeComp(NEWBOB *nBob, NEWANIMCOMP *nAnimComp)
{
struct Bob      *compBob;
struct AnimComp *aComp;

if ((aComp = AllocMem((LONG)sizeof(struct AnimComp),MEMF_CLEAR)) != NULL)
    {
    if ((compBob = makeBob(nBob)) != NULL)
        {
        compBob->After   = NULL;  /* Caller can deal with these later. */
        compBob->Before  = NULL;
        compBob->BobComp = aComp;   /* Link 'em up. */

        aComp->AnimBob      = compBob;
        aComp->TimeSet      = nAnimComp->nac_Time; /* Num ticks active. */
        aComp->YTrans       = nAnimComp->nac_Yt; /* Offset rel to HeadOb */
        aComp->XTrans       = nAnimComp->nac_Xt;
        aComp->AnimCRoutine = nAnimComp->nac_Routine;
        aComp->Flags        = nAnimComp->nac_CFlags;
        aComp->Timer        = 0;
        aComp->NextSeq      = NULL;
        aComp->PrevSeq      = NULL;
        aComp->NextComp     = NULL;
        aComp->PrevComp     = NULL;
        aComp->HeadOb       = NULL;

        return(aComp);
        }
    FreeMem(aComp, (LONG)sizeof(struct AnimComp));
    }
return(NULL);
}

/*-------------------------------------------------------------
** create an Animation Sequence from the information given in nAnimSeq
** and nBob.
** use freeSeq() to free this gel.
**
** this routine creates a linked list of animation components which
** make up the animation sequence.
**
** It links them all up, making a circular list of the PrevSeq
** and NextSeq pointers. That is to say, the first component of the
** sequences' PrevSeq points to the last component; the last component of
** the sequences' NextSeq points back to the first component.
**
** If dbuf is on, the underlying Bobs'll be set up for double buffering.
** If singleImage is non-zero, the pImages pointer is assumed to point to
** an array of only one image, instead of an array of 'count' images, and
** all Bobs will use the same image.
*/
struct AnimComp *makeSeq(NEWBOB *nBob, NEWANIMSEQ *nAnimSeq)
{
int seq;
struct AnimComp *firstCompInSeq = NULL;
struct AnimComp *seqComp = NULL;
struct AnimComp *lastCompMade = NULL;
```

```
LONG image_size;
NEWANIMCOMP nAnimComp;

/* get the initial image.  this is the only image that is used
** if nAnimSeq->nas_SingleImage is non-zero.
*/
nBob->nb_Image = nAnimSeq->nas_Images;
image_size = nBob->nb_LineHeight * nBob->nb_ImageDepth * nBob->nb_WordWidth;

/* for each comp in the sequence */
for (seq = 0; seq < nAnimSeq->nas_Count; seq++)
    {
    nAnimComp.nac_Xt        = *(nAnimSeq->nas_Xt + seq);
    nAnimComp.nac_Yt        = *(nAnimSeq->nas_Yt + seq);
    nAnimComp.nac_Time      = *(nAnimSeq->nas_Times + seq);
    nAnimComp.nac_Routine   = nAnimSeq->nas_Routines[seq];
    nAnimComp.nac_CFlags    = nAnimSeq->nas_CFlags;

    if ((seqComp = makeComp(nBob, &nAnimComp)) == NULL)
        {
        if (firstCompInSeq != NULL)
            freeSeq(firstCompInSeq, (LONG)nBob->nb_RasDepth);
        return(NULL);
        }

/* THIS HAS CHANGED for amiga mail. *********************************
**
** Remove these two lines from the RKM.  This is because we are now
** passing the HitMask and MeMask directly down to the VSprite structure.
** NOTE that the deleted technique only sets up the two values for one
** VSprite in the AnimComp.  The new technique (used in this example)
** correctly does the set-up of all the VSprites in the AnimComp.
**
**   seqComp->AnimBob->BobVSprite->HitMask = nAnimSeq->nas_HitMask;
**   seqComp->AnimBob->BobVSprite->MeMask = nAnimSeq->nas_MeMask;
**
** END OF CHANGE for amiga mail.    *********************************
*/
    seqComp->HeadOb = nAnimSeq->nas_HeadOb;

    /* Make a note of where the first component is. */
    if (firstCompInSeq == NULL)
        firstCompInSeq = seqComp;

    /* link the component into the list */
    if (lastCompMade != NULL)
        lastCompMade->NextSeq = seqComp;

    seqComp->NextSeq = NULL;
    seqComp->PrevSeq = lastCompMade;
    lastCompMade = seqComp;

    /* If nAnimSeq->nas_SingleImage is zero,
    ** the image array has nAnimSeq->nas_Count images.
    */
    if (!nAnimSeq->nas_SingleImage)
        nBob->nb_Image += image_size;
    }
/* On The last component in the sequence, set Next/Prev to make
** the linked list a loop of components.
*/
lastCompMade->NextSeq = firstCompInSeq;
firstCompInSeq->PrevSeq = lastCompMade;

return(firstCompInSeq);
}
```

```
/*-------------------------------------------------------------
** free the data created by makeVSprite()
**
** assumes images deallocated elsewhere.
*/
VOID freeVSprite(struct VSprite *vsprite)
{
LONG    line_size;
LONG    plane_size;

line_size = (LONG)sizeof(WORD) * vsprite->Width;
plane_size = line_size * vsprite->Height;

FreeMem(vsprite->BorderLine, line_size);
FreeMem(vsprite->CollMask, plane_size);

FreeMem(vsprite, (LONG)sizeof(*vsprite));
}

/*-------------------------------------------------------------
** free the data created by makeBob()
**
** it's important that rasdepth match the depth you
** passed to makeBob() when this gel was made.
** assumes images deallocated elsewhere.
*/
VOID freeBob(struct Bob *bob, LONG rasdepth)
{
LONG    rassize;

rassize =  (LONG)sizeof(UWORD) *
        bob->BobVSprite->Width * bob->BobVSprite->Height * rasdepth;

if (bob->DBuffer != NULL)
    {
    FreeMem(bob->DBuffer->BufBuffer, rassize);
    FreeMem(bob->DBuffer, (LONG)sizeof(struct DBufPacket));
    }
FreeMem(bob->SaveBuffer, rassize);
freeVSprite(bob->BobVSprite);
FreeMem(bob, (LONG)sizeof(*bob));
}

/*-------------------------------------------------------------
** free the data created by makeComp()
**
** it's important that rasdepth match the depth you
** passed to makeComp() when this gel was made.
** assumes images deallocated elsewhere.
*/
VOID freeComp(struct AnimComp *myComp, LONG rasdepth)
{
freeBob(myComp->AnimBob, rasdepth);
FreeMem(myComp, (LONG)sizeof(struct AnimComp));
}

/*-------------------------------------------------------------
** free the data created by makeSeq()
**
** Complimentary to makeSeq(), this routine goes through the NextSeq
** pointers and frees the Components
**
** This routine only goes forward through the list, and so
** it must be passed the first component in the sequence, or the sequence
```

```
** must be circular (which is guaranteed if you use makeSeq()).
**
** it's important that rasdepth match the depth you
** passed to makeSeq() when this gel was made.
** assumes images deallocated elsewhere.
*/
VOID freeSeq(struct AnimComp *headComp, LONG rasdepth)
{
struct AnimComp *curComp;
struct AnimComp *nextComp;

/* this is freeing a loop of AnimComps, hooked together by the
** NextSeq and PrevSeq pointers.
*/

/* break the NextSeq loop, so we get a NULL at the end of the list. */
headComp->PrevSeq->NextSeq = NULL;

curComp = headComp;         /* get the start of the list */
while (curComp != NULL)
     {
     nextComp = curComp->NextSeq;
     freeComp(curComp, rasdepth);
     curComp = nextComp;
     }
}

/*-------------------------------------------------------------
** free an animation object (list of sequences).
**
** freeOb() goes through the NextComp pointers, starting at the AnimObs'
** HeadComp, and frees every sequence.
** it only goes forward. It then frees the Object itself.
** assumes images deallocated elsewhere.
*/
VOID freeOb(struct AnimOb *headOb, LONG rasdepth)
{
struct AnimComp *curSeq;
struct AnimComp *nextSeq;

curSeq = headOb->HeadComp;          /* get the start of the list */
while (curSeq != NULL)
     {
     nextSeq = curSeq->NextComp;
     freeSeq(curSeq, rasdepth);
     curSeq = nextSeq;
     }

FreeMem(headOb, (LONG)sizeof(struct AnimOb));
}
```

4) /* Collide.c */

```
/* This is an example that shows how to do collision detection between
** multiple animation objects and between animation objects and the border.
**
** Lattice C 5.04
** lc -b1 -cfist -v -y -ocollide.o collide.c
** lc -b1 -cfist -v -y -oanimtools.o animtools.c
** blink from lib:c.o collide.o animtools.o
**       library lib:lc.lib lib:amiga.lib to collide
*/
#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/gels.h>
#include <graphics/collide.h>
#include <exec/memory.h>
#include <libraries/dos.h>

#include "animtools.h"
#include "animtools_proto.h"

#include <stdlib.h>
#include <stdio.h>
#include <proto/all.h>
int CXBRK(void) { return(0); }

/* prototypes for functions in this file */
VOID __stdargs __saveds hit_routine(struct VSprite *vs1,struct VSprite *vs2);
VOID __stdargs __saveds bounceWall(struct VSprite *vs1,LONG borderflags);
struct AnimOb *setupBoing(SHORT dbufing);
VOID runAnimation(struct Window *win, SHORT dbufing,
                  struct AnimOb **animKey, struct BitMap **myBitMaps);
LONG setupPlanes(struct BitMap *bitMap, LONG depth, LONG width, LONG height);
struct BitMap **setupBitMaps(LONG depth, LONG width, LONG height);
VOID freePlanes(struct BitMap *bitMap, LONG depth, LONG width, LONG height);
VOID freeBitMaps(struct BitMap **myBitMaps,
                    LONG depth, LONG width, LONG height);
struct GelsInfo *setupDisplay(struct Window **win, SHORT dbufing,
                              struct BitMap **myBitMaps);
VOID drawGels(struct Window *win, struct AnimOb **animKey,
              SHORT dbufing, WORD *toggleFrame, struct BitMap **myBitMaps);

#define RBMWIDTH  320
#define RBMHEIGHT 200
#define RBMDEPTH    4

struct NewScreen ns =
    {
    0, 0, 320, 200, 2, 0, 1, NULL,
    CUSTOMSCREEN, NULL, "Collision With AnimObs", NULL, NULL
    };
struct NewWindow nw =
    {
    50, 50, 220, 100, -1, -1, CLOSEWINDOW,
    WINDOWCLOSE | RMBTRAP, NULL, NULL, "Close Window to Stop", NULL,
    NULL, 150, 100, 150, 100, CUSTOMSCREEN
    };

struct IntuitionBase *IntuitionBase = NULL;
struct GfxBase       *GfxBase       = NULL;

int return_code;

/* these give the number of frames (COUNT), size (HEIGHT, WIDTH, DEPTH),
** and word width (WWIDTH) of the animated object.
*/
```

```
#define BOING_COUNT     6
#define BOING_HEIGHT    25
#define BOING_WIDTH     32
#define BOING_DEPTH      1
#define BOING_WWIDTH    ((BOING_WIDTH + 15) / 16)

/* these are the IDs for the system to use for the three objects.
** these numbers will be used for the collision detection system.
**
** Do not use zero (0), as it is reserved by the collision system
** for border hits (see graphics/collide.h, BORDERHIT.)
*/
#define BOING_1     2
#define BOING_2     3
#define BOING_3     4

/* these are the number of counts that each frame is displayed.
** they are all one, so each frame is displayed once then the
** animation system will move on to the next in the sequence
*/
SHORT boing3Times[BOING_COUNT] = { 1, 1, 1, 1, 1, 1 };

/* these are all set to zero as we do not want anything added
** to the X and Y positions using ring motion control.
** all movement is done using the acceleration and velocity
** values.
*/
SHORT boing3YTranses[BOING_COUNT] = { 0, 0, 0, 0, 0, 0 };
SHORT boing3XTranses[BOING_COUNT] = { 0, 0, 0, 0, 0, 0 };

/* no special routines to call when each anim comp is displayed */
WORD (*boing3CRoutines[BOING_COUNT])(struct AnimComp *) =
    { NULL, NULL, NULL, NULL, NULL, NULL };

UWORD __chip boing3Image[BOING_COUNT][BOING_WWIDTH * BOING_HEIGHT * BOING_DEPTH] =
    {
    /*----- bitmap Boing, frame 0:  w = 32, h = 25 ------ */
        {
        0x0023, 0x0000,  0x004E, 0x3000,  0x00E3, 0x3A00,  0x03C3, 0xC900,
        0x0787, 0x8780,  0x108F, 0x8700,  0x31F7, 0x8790,  0x61F0, 0x4790,
        0x63E0, 0xFB90,  0x43E0, 0xF848,  0x3BC0, 0xF870,  0x3801, 0xF870,
        0x383D, 0xF070,  0x387E, 0x1070,  0x387C, 0x0EE0,  0xD87C, 0x1F10,
        0x467C, 0x1E10,  0x479C, 0x1E30,  0x6787, 0x3E20,  0x0787, 0xCC60,
        0x0F0F, 0x8700,  0x048F, 0x0E00,  0x0277, 0x1C00,  0x0161, 0xD800,
        0x0027, 0x2000,
        },
    /*----- bitmap Boing, frame 1:  w = 32, h = 25 ------ */
        {
        0x0031, 0x8000,  0x0107, 0x1800,  0x00F0, 0x1900,  0x09E1, 0xEC80,
        0x13C1, 0xE340,  0x1803, 0xE380,  0x387B, 0xC390,  0x30F8, 0x01D0,
        0x70F8, 0x3DC0,  0xE1F0, 0x3E08,  0x9DF0, 0x7C30,  0x9E30, 0x7C30,
        0x9E1C, 0x7C30,  0x1C1F, 0x9C30,  0x1C1F, 0x0630,  0x7C1F, 0x0780,
        0x623F, 0x0798,  0x63DE, 0x0F10,  0x23C1, 0x0F20,  0x33C3, 0xEE20,
        0x0BC3, 0xC380,  0x0647, 0xC700,  0x023F, 0x8E00,  0x0130, 0xF800,
        0x0033, 0x8000,
        },
    /*----- bitmap Boing, frame 2:  w = 32, h = 25 ------ */
        {
        0x0019, 0xC000,  0x0103, 0x8800,  0x0278, 0x8D00,  0x0CF0, 0xFE80,
        0x11F0, 0xF140,  0x0E60, 0xF1E0,  0x1C39, 0xF0C0,  0x1C3E, 0x30C0,
        0x387E, 0x0CE0,  0xF87C, 0x1F28,  0x8C7C, 0x1F18,  0x8F3C, 0x1F18,
        0x8F06, 0x1E18,  0x8F07, 0xDE18,  0x8F07, 0xC018,  0x6E0F, 0xC1C0,
        0x300F, 0x83C8,  0x31EF, 0x8390,  0x31F0, 0x8780,  0x11E0, 0xF720,
        0x11E1, 0xF1C0,  0x0B61, 0xE300,  0x071B, 0xC600,  0x0138, 0x6C00,
        0x0031, 0x8000,
        },
```

```
    /*----- bitmap Boing, frame 3:  w = 32, h = 25 ------ */
        {
        0x001C, 0xE000,  0x01B1, 0xCC00,  0x031C, 0xC500,  0x0C3C, 0x3680,
        0x1878, 0x7840,  0x2F70, 0x78E0,  0x0E08, 0x7860,  0x1E0F, 0xB860,
        0x1C1F, 0x0460,  0xBC1F, 0x07B0,  0xC43F, 0x0788,  0xC7FE, 0x0788,
        0xC7C0, 0x0F88,  0xC781, 0xEF88,  0xC783, 0xF118,  0x2783, 0xE0E8,
        0x3983, 0xE1E0,  0x3863, 0xE1C0,  0x1878, 0xC1C0,  0x3878, 0x3380,
        0x10F0, 0x78C0,  0x0B70, 0xF180,  0x0588, 0xE200,  0x009E, 0x2400,
        0x0018, 0xC000,
        },
    /*----- bitmap Boing, frame 4:  w = 32, h = 25 ------ */
        {
        0x000E, 0x6000,  0x00F8, 0xE400,  0x030F, 0xE600,  0x061E, 0x1300,
        0x0C3E, 0x1C80,  0x27FC, 0x1C60,  0x0784, 0x3C60,  0x4F07, 0xFE20,
        0x8F07, 0xC230,  0x1E0F, 0xC1F0,  0x620F, 0x83C8,  0x61CF, 0x83C8,
        0x61E1, 0x83C8,  0x63E0, 0x63C8,  0x63E0, 0xF9C8,  0x03E0, 0xF878,
        0x1DC0, 0xF860,  0x1C21, 0xF0E0,  0x5C3E, 0xF0C0,  0x0C3C, 0x11C0,
        0x143C, 0x3C40,  0x09B8, 0x3880,  0x05C0, 0x7000,  0x00CF, 0x0400,
        0x000C, 0x6000,
        },
    /*----- bitmap Boing, frame 5:  w = 32, h = 25 ------ */
        {
        0x0026, 0x2000,  0x00FC, 0x7400,  0x0187, 0x7200,  0x030F, 0x0100,
        0x0E0F, 0x0E80,  0x319F, 0x0E00,  0x23C6, 0x0F30,  0x63C1, 0xCF30,
        0x4781, 0xF310,  0x0783, 0xE0D0,  0x7383, 0xE0E0,  0x70C3, 0xE0E0,
        0x70F9, 0xE1E0,  0x70F8, 0x21E0,  0x70F8, 0x3FE0,  0x91F0, 0x3E38,
        0x4FF0, 0x7C30,  0x4E10, 0x7C60,  0x4E0F, 0x7860,  0x2E1F, 0x08C0,
        0x0E1E, 0x0E00,  0x049E, 0x1C80,  0x00E4, 0x3800,  0x00C7, 0x9000,
        0x000E, 0x6000,
        }
    };

/* these structures contain the initialization data for the animation
** sequence.
*/
NEWBOB newBoingBob =
    {
    NULL, BOING_WWIDTH, BOING_HEIGHT, BOING_DEPTH, 0x2, 0x0,
    SAVEBACK | OVERLAY, 0, RBMDEPTH, 0,0,0,0,
    };
NEWANIMSEQ newBoingSeq =
    {
    NULL, (WORD *)boing3Image, boing3XTranses, boing3YTranses,
    boing3Times, boing3CRoutines, 0, BOING_COUNT, 0,
    };

/*-------------------------------------------------------------------------
** setupBoing()
**
** make a new animation object.  since all of the boing balls use the same
** underlying data, the initalization structures are hard-coded into the
** routine (newBoingBob and newBoingSeq.)
**
** return a pointer to the object if successful, NULL if failure.
** set a global error code on failure.
*/
struct AnimOb *setupBoing(SHORT dbufing)
{
struct AnimOb   *bngOb;
struct AnimComp *bngComp;

if (NULL != (bngOb = AllocMem((LONG)sizeof(struct AnimOb), MEMF_CLEAR)))
    {
    newBoingBob.nb_DBuf    = dbufing;     /* double-buffer status  */
```

```
    newBoingSeq.nas_HeadOb = bngOb;       /* pass down head object */

    if (NULL != (bngComp = makeSeq(&newBoingBob, &newBoingSeq)))
        {
        bngOb->HeadComp = bngComp;  /* the head comp is the one that */
                                    /* is returned by makeSeq()      */
        return(bngOb);
        }
    FreeMem(bngOb, (LONG)sizeof(struct AnimOb));
    }
return_code = RETURN_WARN;
return(NULL);
}

/*-------------------------------------------------------------------------
** runAnimation()
**
** a simple message handling loop that also draws the successive frames.
*/
VOID runAnimation(struct Window *win, SHORT dbufing,
                  struct AnimOb **animKey, struct BitMap **myBitMaps)
{
struct IntuiMessage  *intuiMsg;
WORD toggleFrame;

/* toggleFrame is used to keep track of which frame of the double buffered
** screen we are currently displaying.  the variable must exist for the life
** of the displayed objects, so it is defined here.
*/
toggleFrame = 0;

/* end the loop on a CLOSEWINDOW event */
for (;;)
    {
    /* draw the gels, then check for messages.
    ** check the messages after each display so we get a quick response.
    */
    drawGels(win, animKey, dbufing, &toggleFrame, myBitMaps);

    /* quit on a control_c */
    if (SetSignal(0L,0L) & SIGBREAKF_CTRL_C)
        return;

    /* check for a closewindow event, die if found */
    while (intuiMsg = (struct IntuiMessage *)GetMsg(win->UserPort))
        {
        if (intuiMsg->Class == CLOSEWINDOW)
            {
            ReplyMsg((struct Message *)intuiMsg);
            return;
            }
        ReplyMsg((struct Message *)intuiMsg);
        }
    }
}

/*-------------------------------------------------------------------------
** setupPlanes()
**
** called only for double-buffered displays.
** allocate and clear each bit-plane in a bitmap structure.
** clean-up on failure.
*/
LONG setupPlanes(struct BitMap *bitMap, LONG depth, LONG width, LONG height)
```

```
{
SHORT plane_num ;

for (plane_num = 0; plane_num < depth; plane_num++)
    {
    if (NULL != (bitMap->Planes[plane_num] =
                (PLANEPTR)AllocRaster(width, height)))
        BltClear((APTR)(bitMap->Planes[plane_num]), (width / 8) * height, 1);
    else
        {
        freePlanes(bitMap, depth, width, height);
        return_code = RETURN_WARN;
        return(NULL);
        }
    }
return(TRUE);
}

/*--------------------------------------------------------------------------
** setupBitMaps()
**
** allocate the two bitmaps for a double-buffered display.
** routine only called when the display is double-buffered.
*/
struct BitMap **setupBitMaps(LONG depth, LONG width, LONG height)
{
static struct BitMap *myBitMaps[2];

/* allocate the two bit-map structures. these do not have to be in CHIP */
if (NULL != (myBitMaps[0] =
        (struct BitMap *)AllocMem((LONG)sizeof(struct BitMap), MEMF_CLEAR)))
    {
    if (NULL != (myBitMaps[1] =
        (struct BitMap *)AllocMem((LONG)sizeof(struct BitMap), MEMF_CLEAR)))
        {
        /* initialize the bit maps to the correct size. */
        InitBitMap(myBitMaps[0], (BYTE)depth, (SHORT)width, (SHORT)height);
        InitBitMap(myBitMaps[1], (BYTE)depth, (SHORT)width, (SHORT)height);

        /* allocate and initialize the bit-planes for the bit-maps. */
        if (NULL != setupPlanes(myBitMaps[0], depth, width, height))
            {
            if (NULL != setupPlanes(myBitMaps[1], depth, width, height))
                return(myBitMaps);

            freePlanes(myBitMaps[0], depth, width, height);
            }
        FreeMem(myBitMaps[1], (LONG)sizeof(struct BitMap));
        }
    FreeMem(myBitMaps[0], (LONG)sizeof(struct BitMap));
    }
/* on failure, everything is freed and a global return code is set. */
return_code = RETURN_WARN;
return(NULL);
}

/*--------------------------------------------------------------------------
** freePlanes()
**
** free all of the bit-planes in a bit-map structure.
*/
VOID freePlanes(struct BitMap *bitMap, LONG depth, LONG width, LONG height)
{
SHORT plane_num ;
```

```
for (plane_num = 0; plane_num < depth; plane_num++)
     {
     if (NULL != bitMap->Planes[plane_num])
         FreeRaster(bitMap->Planes[plane_num], (USHORT)width, (USHORT)height);
     }
}

/*-----------------------------------------------------------------------
** freeBitMaps()
**
** free the two bit-maps from the double buffered display.
** the bit-planes are freed first, then the bit-map structures.
*/
VOID freeBitMaps(struct BitMap **myBitMaps, LONG depth, LONG width, LONG height)
{
freePlanes(myBitMaps[0], depth, width, height);
freePlanes(myBitMaps[1], depth, width, height);

FreeMem(myBitMaps[0], (LONG)sizeof(struct BitMap));
FreeMem(myBitMaps[1], (LONG)sizeof(struct BitMap));
}

/*-----------------------------------------------------------------------
** setupDisplay()
**
** open the screen and the window for the display.
**
** if using double buffered display, assume the bit-maps have been opened
** and correctly set-up.
*/
struct GelsInfo *setupDisplay(struct Window **win, SHORT dbufing,
                              struct BitMap **myBitMaps)
{
struct GelsInfo    *gInfo;
struct Screen      *screen;

/* if double-buffered, set-up the new screen structure for custom bit map */
if (dbufing)
     {
     ns.Type |= CUSTOMBITMAP;
     ns.CustomBitMap = myBitMaps[0];
     }

/* open everything.  check for failure. */
if ((screen = (struct Screen *)OpenScreen(&ns)) != NULL)
     {
     nw.Screen = screen;
     if ((*win = (struct Window *)OpenWindow(&nw)) != NULL)
         {
         if (dbufing)
             {
             /* we are double buffered.  set the rastport for it. */
             (*win)->WScreen->RastPort.Flags = DBUFFER;

             /* this copies the intuition display (close gadget) to the
             ** second bit-map so the display does not flash when we change
             ** between them.
             */
             (*win)->WScreen->RastPort.BitMap = myBitMaps[1];
             BltBitMapRastPort(myBitMaps[0], 0,0, &(*win)->WScreen->RastPort,
                         0,0, RBMWIDTH, RBMHEIGHT, 0xC0);
             (*win)->WScreen->RastPort.BitMap = myBitMaps[0];
             }
```

```
        /* ready the gel system for accepting objects
        ** this is only done once for each rastport in use.
        */
        if (NULL != (gInfo = setupGelSys(&(*win)->WScreen->RastPort, 0xFC)))
            return(gInfo);

        CloseWindow(*win);
        }
    CloseScreen(screen);
    }
/* on an error everything is cleaned-up, and a global return code is set. */
return_code = RETURN_WARN;
return(NULL);
}

/*-------------------------------------------------------------------------
** drawGels()
**
** handle the update of the display.  Animate the simulation and check for
** collisions.  If the screen is double buffered, swap the bit map as
** required.
*/
VOID drawGels(struct Window *win, struct AnimOb **animKey, SHORT dbufing,
              WORD *toggleFrame, struct BitMap **myBitMaps)
{
/* do the required animation stuff.  you must sort both after the animate
** call and after the collision call.
*/
Animate((struct AnimOb *)animKey, &win->WScreen->RastPort);
SortGList(&win->WScreen->RastPort);

DoCollision(&win->WScreen->RastPort);
SortGList(&win->WScreen->RastPort);

/* toggle if double buffered */
if (dbufing)
    win->WScreen->ViewPort.RasInfo->BitMap = myBitMaps[*toggleFrame];

/* draw the new position of the gels into the screen. */
DrawGList(&win->WScreen->RastPort, &win->WScreen->ViewPort);

/* if using a double buffered display, you have a more complicated
** update procedure.
**
** if not, then simply use WaitTOF().
*/
if (dbufing)
    {
    MakeScreen(win->WScreen);
    RethinkDisplay();

    *toggleFrame ^= 1;
    win->WScreen->RastPort.BitMap = myBitMaps[*toggleFrame];
    }
else
    WaitTOF();
}

/*-------------------------------------------------------------------------
** bounceWall()
**
** handle bouncing the animation objects off of the walls.
**
```

```
** use __stdargs and __saveds because this routine is not directly called
** by this program.  The call to DoCollision() causes a call back to
** this routine when an animation object comes in contact with a wall.
** __stdargs says the arguments are passed on the stack.
** __saveds says to restore the data segment pointer on entry to the routine.
*/
VOID __stdargs __saveds bounceWall(struct VSprite *vs1,LONG borderflags)
{
struct AnimOb *ob;

/* get a pointer to the object from the sprite pointer. */
ob = vs1->VSBob->BobComp->HeadOb;

/* check for hits and act apropriately.
** for right and left, reverse the x velocity if the object is moving
** towards the wall (it may have already reversed but still be in contact
** with the wall.)
** for the bottom and top you also have to subtract out the acceleration.
*/
if (((borderflags & RIGHTHIT) && (ob->XVel > 0)) ||
     ((borderflags & LEFTHIT) && (ob->XVel < 0)))
     ob->XVel = -(ob->XVel);
else if (((borderflags & TOPHIT) && (ob->YVel < 0)) ||
          ((borderflags & BOTTOMHIT) && (ob->YVel > 0)))
     {
     ob->YVel -= ob->YAccel;
     ob->YVel = -(ob->YVel);
     }
}

/*------------------------------------------------------------------------
** hit_routine()
**
** handle the collision between two animation objects.
** this routine simulates objects bouncing off of each other.
** this does not do a very good job of it, it does not take into account
** the angle of the collision or real physics.
** if anyone wants to fix it, please feel free.
**
** use __stdargs and __saveds because this routine is not directly called
** by this program.  The call to DoCollision() causes a call back to
** this routine when two animation objects overlap.
** __stdargs says the arguments are passed on the stack.
** __saveds says to restore the data segment pointer on entry to the routine.
*/
VOID __stdargs __saveds hit_routine(struct VSprite *vs1,struct VSprite *vs2)
{
LONG vel1, vel2;

/* check that the bob is not being removed!  This is due to a 1.3 bug
** where all bobs are tested for collision, even the ones that are in
** the process of being removed.  See text for more information.
**
** bobs are moved to a very large negative position as they are being
** removed.  if the BOBSAWAY flag is set, then both bobs in the collision
** are in the process of being removed--don't do anything in the collision
** routine.
*/
if (!(vs1->VSBob->Flags & BOBSAWAY))
     {
     /* cache the velocity values
     ** Do the X values first (order is not important.)
     */
     vel1 = vs1->VSBob->BobComp->HeadOb->XVel;
```

```
    vel2 = vs2->VSBob->BobComp->HeadOb->XVel;

    /* if the two objects are moving in the opposite direction (X component)
    ** then negate the velocities.
    ** else swap the velocities.
    */
    if (((vel1 > 0) && (vel2 < 0)) || ((vel1 < 0) && (vel2 > 0)))
        {
        vs1->VSBob->BobComp->HeadOb->XVel = -vel1;
        vs2->VSBob->BobComp->HeadOb->XVel = -vel2;
        }
    else
        {
        vs1->VSBob->BobComp->HeadOb->XVel = vel2;
        vs2->VSBob->BobComp->HeadOb->XVel = vel1;
        }

    /* cache the velocity values
    ** Do the Y values second (order is not important.)
    */
    vel1 = vs1->VSBob->BobComp->HeadOb->YVel;
    vel2 = vs2->VSBob->BobComp->HeadOb->YVel;

    /* if the two objects are moving in the opposite direction (Y component)
    ** then negate the velocities.
    ** else swap the velocities.
    */
    if (((vel1 > 0) && (vel2 < 0)) || ((vel1 < 0) && (vel2 > 0)))
        {
        vs1->VSBob->BobComp->HeadOb->YVel = -vel1;
        vs2->VSBob->BobComp->HeadOb->YVel = -vel2;
        }
    else
        {
        vs1->VSBob->BobComp->HeadOb->YVel = vel2;
        vs2->VSBob->BobComp->HeadOb->YVel = vel1;
        }
    }
}

/*--------------------------------------------------------------------------
** main routine
**
** run a double buffered display if the user puts any arguments on the
** command line.
**
** open libraries, set-up the display, set-up the animation system and
** the objects, set-up collisions between objects and against walls,
** and run the thing.
**
** clean-up and close resources when done.
*/
VOID main(int argc, char **argv)
{
struct BitMap   **myBitMaps;
struct AnimOb    *boingOb;
struct AnimOb    *boing2Ob;
struct AnimOb    *boing3Ob;
struct Window    *win;
struct Screen    *screen;
struct GelsInfo  *gInfo;
struct AnimOb    *animKey;
SHORT dbufing;
```

```
return_code = RETURN_OK;

/* if any arguments, use double-buffering */
if (argc > 1)
    dbufing = 1;
else
    dbufing = 0;

/* open required libraries */
if (NULL == (IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library", 33L)))
    return_code = RETURN_FAIL;
else
    {
    if (NULL == (GfxBase = (struct GfxBase *)
            OpenLibrary("graphics.library", 33L)))
        return_code = RETURN_FAIL;
    else
        {
        /* note that setupBitMaps() will only be called if
        ** we are double buffering
        */
        if ((!dbufing) ||
            (NULL != (myBitMaps=setupBitMaps(RBMDEPTH,RBMWIDTH,RBMHEIGHT))))
            {
            if (NULL != (gInfo = setupDisplay(&win,dbufing,myBitMaps)))
                {
                /* you have to initialize the animation key
                ** before you use it.
                */
                InitAnimate(&animKey);

                /* set-up the first boing ball.
                ** all of these use the same data, hard coded into setupBoing().
                ** change the color by changing PlanePick.
                ** set the ID of the ball (MeMask) to BOING_1.
                ** HitMask = 0xFF means that it will collide with everything.
                 */
                newBoingBob.nb_PlanePick = 0x2;
                newBoingBob.nb_MeMask    = 1L<<BOING_1;
                newBoingBob.nb_HitMask   = 0xFF;
                if (NULL != (boingOb = setupBoing(dbufing)))
                    {
                    /* pick an initial position, velocity and acceleration
                    ** and add the object to the system.  NOTE that the
                    ** Y-velocity and X-acceleration are not set (they default
                    ** to zero.)  This means that the objects will maintain
                    ** a constant movement to the left or right, and will
                    ** rely on the Y accelleration for the downward movement.
                    ** The collision routines change these values, producing
                    ** bouncing off of walls and other objects.
                    **
                    ** NOTE:  ANFRACSIZE is a value that shifts animation
                    ** constants past an internal decimal point.  If you
                    ** do not do this, then the values will only be some
                    ** fraction of what you expect.  See the Rom Kernel
                    ** Manual:  Libraries and Devices.
                    */
                    boingOb->AnY    = 10 << ANFRACSIZE;
                    boingOb->AnX    = 250 << ANFRACSIZE;
                    boingOb->XVel   = -(4 << ANFRACSIZE);
                    boingOb->YAccel = 70;
                    AddAnimOb(boingOb, (APTR)&animKey,
                            &win->WScreen->RastPort);
```

```
            /* do the second object--see above comments. */
            newBoingBob.nb_PlanePick = 0x1;
            newBoingBob.nb_MeMask    = 1L<<BOING_2;
            newBoingBob.nb_HitMask   = 0xFF;
            if (NULL != (boing2Ob = setupBoing(dbufing)))
                {
                boing2Ob->AnY    = 50 << ANFRACSIZE;
                boing2Ob->AnX    = 50 << ANFRACSIZE;
                boing2Ob->XVel   = 3 << ANFRACSIZE;
                boing2Ob->YAccel = 70;
                AddAnimOb(boing2Ob, (APTR)&animKey,
                        &win->WScreen->RastPort);

                /* do the third object--see above comments.
                ** here we also use PlaneOnOff to change the color.
                */
                newBoingBob.nb_PlanePick = 0x1;
                newBoingBob.nb_PlaneOnOff= 0x2;
                newBoingBob.nb_MeMask    = 1L<<BOING_3;
                newBoingBob.nb_HitMask   = 0xFF;
                if (NULL != (boing3Ob = setupBoing(dbufing)))
                    {
                    boing3Ob->AnY    = 80 << ANFRACSIZE;
                    boing3Ob->AnX    = 150<< ANFRACSIZE;
                    boing3Ob->XVel   = 2 << ANFRACSIZE;
                    boing3Ob->YAccel = 70;
                    AddAnimOb(boing3Ob, (APTR)&animKey,
                            &win->WScreen->RastPort);

                    /* set up the collisions between boing balls.
                    ** NOTE that they all call the same routine.
                    */
                    SetCollision(BOING_1,hit_routine,gInfo);
                    SetCollision(BOING_2,hit_routine,gInfo);
                    SetCollision(BOING_3,hit_routine,gInfo);

                    /* set the collisions with the walls. */
                    SetCollision(BORDERHIT,bounceWall,gInfo);

                    /* everything set-up...run the animation. */
                    runAnimation(win, dbufing, &animKey, myBitMaps);

                    /* done...
                    ** free-up everything and clean up the mess.
                     */
                    freeOb(boing3Ob, RBMDEPTH);
                    }
                freeOb(boing2Ob, RBMDEPTH);
                }
            freeOb(boingOb, RBMDEPTH);
            }

        cleanupGelSys(gInfo, &win->WScreen->RastPort);
        screen = win->WScreen;
        CloseWindow(win);
        CloseScreen(screen);
        }

    if (dbufing)
        freeBitMaps(myBitMaps, RBMDEPTH, RBMWIDTH, RBMHEIGHT);
    }
CloseLibrary((struct Library *)GfxBase);
}
```

```
     CloseLibrary((struct Library *)IntuitionBase);
     }

/* return the global return code to the system. */
exit(return_code);
}
```