Blitter

Paul Manias

COLLABORATORS			
	TITLE : Blitter		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Paul Manias	July 26, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Blitter

1.1 Blitter.GPI

```
Name:          BLITTER.GPI AUTODOC
Version:       0.6 Beta.
Date:          2 May 1997
Author:        Paul Manias
Copyright:     DreamWorld Productions, 1996-1997.  All rights reserved.
Notes:         This  document is still being written and will contain errors
               in  a  number  of  places.  The information within cannot be
        treated as official until this autodoc reaches version 1.0.
```

1.2 Blitter.GPI Functions

```
 BLITTER.GPI
 AllocBlitMem()
 AllocBlitter()
 ClearBob()
 ClrScreen()
 CopyBuffer()
 DrawBob()
 DrawBobList()
 DrawLine()
 DrawUCLine()
 FreeBlitMem()
 FreeBlitter()
 FreeBob()
 GetBob()
 GetMBob()
 InitBob()

 Pixel Functions
 DrawPixel()
 DrawUCPixel()
 DrawPixelList()
 DrawUCPixelList()
 ReadPixel()
 ReadPixelList()
```

```
 RestoreList Functions
 InitRestore()
 FreeRestore()
 CleanUpRestore()
 ResetRestore()
 Restore()

 Map/Tile Functions
 InitMap()
 FreeMap()
 DrawMap()
 DrawMapArea()
 DrawMapEdge()
```

1.3 Blitter.GPI/AllocBlitter

```
NAME  AllocBlitter -- Allocate the blitter so that we may use it.

SYNOPSIS
  ErrorCode = AllocBlitter()
     d0

  ULONG AllocBlitter(void)

FUNCTION
  Allocates the blitter, and if necessary allows exclusive access for
  your  task only.  This function may also perform other tasks to get
  the  best  blitter performance possible, eg it may set up a blitter
  interrupt.

  You  must call this function before using any other blitter related
  functions, and then call FreeBlitter() before your program exits.

RESULT  ErrorCode - Returns NULL if successful.

SEE ALSO
  FreeBlitter
```

1.4 Blitter.GPI/FreeBlitter

```
NAME  FreeBlitter -- Free the blitter from our use.

SYNOPSIS
  FreeBlitter()

  void FreeBlitter(void)

FUNCTION
  Frees  the  blitter  if you successfully allocated it earlier on in
  your program.
```

```
  You   must   call   this   function   before   your   program   exits   if
  AllocBlitter() was successful.

SEE ALSO
  AllocBlitter
```

1.5 Blitter.GPI/InitBob

```
NAME  InitBob -- Initialise a Blitter OBject structure ready for blitting.

SYNOPSIS
  Bob = InitBob(GameScreen, Bob)
                  a0       a1

  APTR Bob InitBob(struct GameScreen *, APTR Bob)

  APTR Bob InitBobTags(struct GameScreen *, ULONG tag, ...)

FUNCTION
  Initialises a Blitter OBject.  All blitter structures are supported
  by  this  function,  so  you may pass it Bob or MBob (Multiple Bob)
  structure  types.   This  function  is fully supportive of List and
  Object  List  types,  just  pass  it a standard List or Object List
  rather than a bob structure.

  If  you  initialise  a  bob  or mbob using tags, make sure that you
  specify  the correct type of TAGS_BOB or TAGS_MBOB.  If the type is
  incorrect you will confuse the function and not get what you want.

  If  you  pass this function a bob that has already been initialised
  and  not  had  FreeBob() called on it, it will be ignored (no fatal
  error will be given).

  This  function  supports  garbage  protection  and  will  return an
  errorcode of ERR_STRUCT or ERR_DATA if problems are encountered.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Bob        - Pointer  to  a  Bob/MBob  structure,  tags, or a LIST/
         OBJECTLIST.

  Here  follows  a  description of all fields related to bob and mbob
  structures:

  BOB_VERSION (BB/MB)
  The  version  of the structure, currently BBV1 for standard bobs or
  MBV1 for multiple bobs.

  BOB_Stats (BB/MB)
  Private, expects to be zero on initialisation.

  BOB_GfxData, BOB_MaskData (BB/MB)
  Contain  pointers to the bob's graphics and mask origins, which are
  used  to  calculate  the  graphic and mask pointers for each frame.
  These  pointers  serve  as  the  offset for X/Y values in the bob's
  FrameList  (See  BOB_FrameList  for  more information).  You do not
```

need to supply these values if you have specified the DIRECT flag in BOB_Attrib.

BOB_Frame (BB)
The frame number of your bob, this field is ignored in the initialisation procedure but is used for the DrawBob() functions. The values for this field range from 0 to the maximum frame in your FrameList (below). The main use of this field is to allow you to give animation properties to a bob.

This field is only available for bob structures. For mbob's, the frame number is set in the frame list.

BOB_AmtEntries (MB)
Specifies the amount of images to be blitted when you call a drawing function. The value can be set dynamically and does not require initialisation by InitBob().

This field is only available for Mbob structures.

BOB_FrameList (BB/MB)
Points to a framelist that can contain a list of X and Y coordinates based on the GfxData and MaskData fields, or direct address pointers to the graphic and mask sources for each frame.

The X/Y format is the default. Note that the X values must be presented in offsets of 16, and that negative values are disallowed. The array for this format looks like this:

```
dc.w  <GfxX>,<GfxY>,<MaskX>,<MaskY>
dc.w  ...
dc.w  <Termination>
```

An example:

```
Frames:
    dc.w    0,10,16,10
    dc.w    0,26,16,26
    dc.w    -1
```

The direct graphics format is activated if you specify the DIRECT flag in BOB_Attrib. This array looks like this:

```
dc.l  <Graphic>,<Mask>
dc.l  ...
dc.l  <Termination>
```

An example:

```
Frames:
     dc.l    GFX_Sparkie+00,MSK_Sparkie+00
     dc.l    GFX_Sparkie+20,MSK_Sparkie+20
     dc.l    GFX_Sparkie+40,MSK_Sparkie+40
     dc.l    GFX_Sparkie+60,MSK_Sparkie+60
     dc.l    -1
```

The very first set of X/Y values / address pointers are considered

to be frame 0, then next are frame 1, and so on.

Note: If you specify the GENMASK (Generate Mask) flag as an attribute, all values entered in the Mask sections of the array are ignored.

BOB_SrcWidth (BB/MB)
The byte width of the bob's page area. This value is important for correctly pre-calculating the bob's modulo for the DrawBob() functions. If the bob was cut out of a picture (eg as a brush), this value will probably be the same as the one in BOB_Width. If you are blitting the bob directly from a picture, do not set this value, it will be inherited from Picture->Width.

BOB_Width (BB/MB)
The width of the bob in pixels. This field can be used to initialise the BOB_ByteWidth field, otherwise it is completely ignored by the other functions. There is no need to align this field to increments of 16. You may want to use it for making your collision detection routines as precise as possible.

BOB_ByteWidth (BB/MB)
The width of the bob in bytes. You are not allowed to use uneven byte values in this field, eg 1, 3, 5... only values of 2, 4, 6... The maximum width currently supported by this field is 128 bytes.

Note: You are asked not to write to this field as it will inherit its value from BOB_Width. You may read from it if necessary.

BOB_Height (BB/MB)
The height of the bob in pixels. Permission is given to make dynamic changes to your bob's height.

BOB_XCoord, BOB_YCoord (BB)
The X and Y positions of the sprite in relation to the screen's picture buffer. You are allowed to exceed the boundaries of your clip region, but only if you have specified the CLIP flag in BOB_Attrib (in which case the bob will be clipped appropriately).

If you omit the CLIP flag and then place the bob at the screen borders, you will risk blitting the bob to unknown video ram areas.

This field is only available for bob structures.

BOB_EntryList (MB)
If you are using multiple bob's, you will point to your list of images to be blitted here. Each entry is made up of type struct BobEntry, as <XCoord>,<YCoord>,<Frame>. There is no termination necessary for your list, but the BOB_AmtEntries field must be specified correctly.

If you specify SKIPIMAGE in an X Coordinate then that particular image will not be drawn to screen.

This field is only available for mbob structures.

BOB_ClipLX, BOB_ClipTY, BOB_ClipRX, BOB_ClipBY (BB/MB)

These fields set the clipping area for your bob. The values that you specify must be within the borders of your GameScreen's picture border, which means no negative values or values that exceed the Height and Width of the destination picture. X values must be specified in increments of 16 pixels.

Dynamic changes to these fields are not permitted.

BOB_FPlane (BB/MB)
Specifies the first plane that is going to be blitted to. Relevant for planar screen types only.

BOB_Planes (BB/MB)
The amount of planes used by this bob. This field will be initialised to the amount of planes in the GameScreen if set at zero.

BOB_PlaneSize (BB/MB)
The page size for this bob, calculated by PageWidth*PageHeight. this field is only relevant for planar bob's with more than 1 plane to be blitted.

BOB_PictureTags (BB/MB)
If this bob originates from a Picture structure, you may point to it here and InitBob() will use it to initialise certain areas of the bob structure. If the following fields are zero, they will be initialised to the equivalent Picture values:

BOB_GfxData, BOB_MaskData, BOB_SrcWidth, BOB_Planes, BOB_PlaneSize, BOB_Height, BOB_Width, BOB_ByteWidth.

If the picture has not already been initialised, InitBob() will attempt to initialise it with LoadPic().

BOB_EntrySize (MB)
Here you must specify the byte size that is taken up by each entry in your entrylist. This allows you to create mutant entry structures, which you can use to store extra data for each image in your entrylist. For more information see mutant structure types.

As a default this field should be set to BE_SIZEOF.

This field is only available for mbob structures.

BOB_Attrib (BB/MB)
The attributes used in the blitting of this bob, as well as some special flags that define the way the bob will be initialised. The current flags are:

CLIP - Allow clipping for this bob, to the regions given in ClipTX, ClipTY, ClipRX and ClipBY. Note that clipping does slow down the drawing procedure so omit this flag if possible.

MASK - Allow masking for this bob.

GENMASKS - Creates a mask for every graphic found in the FrameList

```
       of your bob.

  FILLMASK - Fills  any  "empty  holes"  in  the  mask,  so  that  no
       background  graphics  can show through the middle of the
       bob.   This  can be useful in certain situations, eg the
       Reko.GPI uses it so that only one mask is needed for all
       the cards.

  RESTORE  - Specified  if this bob is to be added to the RestoreList
       each time it is drawn.  This allows automatic background
       restoring  on  the  bob when the buffer returns from the
       display.  For more information see InitRestore().

  CLEAR  - Specified  if this bob is to be added to the RestoreList
       each time it is drawn.  This allows automatic background
       clearing  on  the  bob  when the buffer returns from the
       display.  For more information see InitRestore().

  CLRMASK  - A mask will be used if you clear this bob.

  CLRNOMASK- No mask will be used if you clear this bob (default).

RESULT  Bob - Returns the required structure if successful, NULL if error.

SEE ALSO
  FreeBob, games/games.i
```

1.6 Blitter.GPI/FreeBob

```
NAME  FreeBob -- Deallocate a Bob structure.

SYNOPSIS
  FreeBob(Bob)
          a1

  void FreeBob(APTR Bob)

FUNCTION
  Frees   all  the  allocations  made  for  an  initialised  bob/mbob
  structure.   You  need to call this function for every bob that you
  have initialised before your program exits.

  To  free  more  than  one  bob,  send this function a standard LIST
  containing a pointer to each bob that you want to free.

INPUTS  Bob - Pointer to an initialised Bob structure.

SEE ALSO
  InitBob, Lists
```

1.7 Blitter.GPI/ClearBob

```
NAME  ClearBob -- Clears a Bob image from a screen display.

SYNOPSIS
  ClearBob(GameScreen, Bob, Buffer)
        a0        a1    d0

  ClearBob(struct GameScreen *, APTR Bob, UWORD Buffer)

FUNCTION
  Clears  a  bob  image/s  from  the  screen.  This is a fast way for
  clearing  a bob as it is written for optimum blitter usage.  It can
  handle  mbob's,  but  for clearing many bob objects from screen you
  probably should be using a RestoreList.

  Note  that  there  is  no  need  to  set the CLEAR flag to use this
  function.   If  you need to clear with the bob's mask, set CLRMASK,
  otherwise set CLRNOMASK.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Bob        - Pointer to an initialised Bob/MBob structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.

SEE ALSO
  DrawBob
```

1.8 Blitter.GPI/DrawBob

```
NAME  DrawBob -- Draws a Blitter OBject directly to a screen buffer.

SYNOPSIS
  DrawBob(GameScreen, Bob, RestoreList, Buffer)
             a0        a1      a2   d0

  void DrawBob(struct GameScreen *, APTR Bob, UWORD Buffer,
          struct RestoreList *)

FUNCTION
  Draws  a  bob  to  screen  according  to the values in the bob/mbob
  structure.   If  the  draw  operation  specifies  for the bob to be
  CLEARed   or  RESTOREd,  then  you  must  supply  a  pointer  to  a
  RestoreList.  If not, this parameter will be ignored.

  The  methods  used to draw the bob will remain unknown to you:  the
  blitter,  CPU,  or  both  devices  may  be used to get the image on
  screen.   Keep  in mind that the primary objective of this function
  is  simply  to  get the image on screen as quickly as possible with
  whatever means available.

FEATURES
  The blitter functions have some special features that you should be
  aware of, if you are interested in obtaining maximum drawing speed.
  Where  possible,  the  CPU will be used to draw when the blitter is
  not available.  It will also assist the blitter by drawing parts of
  the  bob  while  the  blitter  draws other sections.  This parallel
```

```
  drawing gains considerable speed-up for 68020 machines and upwards.

  Blitting  images  at alignments of 16 pixels will be sped up due to
  the  fact  that  no shifting is required.  If you keep this in mind
  you  can  use  this  to  your advantage in certain situations.  One
  example  is  a  horizontal  shoot'em-up,  where you could align the
  bullets  of  your  ship  to  16 pixels.  This would give you a good
  speed advantage when blitting many of such objects.

  More  obvious features, such as blitting and clearing without masks
  will  also  give  a  natural speed up.  You can often use the CLEAR
  mode if you know that the background is empty.  Use Mbob's whenever
  possible,  and  always use Restorelist's as a fast way to redraw or
  clear your backgrounds.

NOTE  You must have initialised your bob before calling this function.

INPUTS  GameScreen  - Pointer to an initialised GameScreen structure.
  Bob         - Pointer to an initialised Bob/MBob structure.
  RestoreList - Pointer to a RestoreList if the Bob requires it.
  Buffer      - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.

SEE ALSO
  DrawBobList, InitBob
```

1.9 Blitter.GPI/DrawBobList

```
NAME  DrawBobList -- Draws a LIST of Blitter OBjects.

SYNOPSIS
  DrawBobList(GameScreen, BobList, RestoreList, Buffer)
                 a0         a1         a2    d0

  void DrawBobList(struct GameScreen *, LONG *BobList[],
        UWORD Buffer, struct RestoreList *)

FUNCTION
  This  is  a mass-drawing function that allows you to blit many Bobs
  from  a list onto a screen.  It handles all bob structure types and
  is  the  fastest way to process the drawing of many bobs at any one
  time.

  If any of the drawing operations specify for a bob to be CLEARed or
  RESTOREd, then you must supply a pointer to a RestoreList.  If not,
  this field will be ignored.

  The  methods  used to draw the bob will remain unknown to you:  the
  blitter,  CPU,  or  both  devices  may  be used to get the image on
  screen.   Keep  in mind that the primary objective of this function
  is  simply  to  get the image on screen as quickly as possible with
  whatever means available.

INPUTS  GameScreen  - Pointer to an initialised GameScreen structure.
  BobList     - Pointer to a LIST of Bob structures to draw.  Must be
          terminated by a LISTEND.
```

```
  RestoreList - Pointer to a RestoreList if any of the Bobs require it.
  Buffer      - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.

SEE ALSO
  DrawBob, InitBob
```

1.10 Blitter.GPI/ClrScreen

```
NAME  ClrScreen -- Clear a GameScreen's picture buffer.

SYNOPSIS
  ClrScreen(GameScreen, Buffer)
              a0         d0

  void ClrScreen(struct GameScreen *, UWORD Buffer)

FUNCTION
  Clears all of the data contained in a specific GameScreen's picture
  buffer.  The  method  used  to do this is largely dependent on the
  selection  made  from  GMSPrefs,  at  the  moment  there  are three
  available clear methods:

       Clear with Blitter.
       Clear with CPU.
       Clear with Blitter and CPU.

  The  default  is  the  Bliter  and  CPU  method,  which is the most
  efficient of the three.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.

SEE ALSO
  ClrArea
```

1.11 Blitter.GPI/CopyBuffer

```
NAME  CopyBuffer - Copy the contents of one buffer to another.

SYNOPSIS
  CopyBuffer(GameScreen, SrcBuffer, DestBuffer)
               a0          d0         d1

  void CopyBuffer(struct GameScreen *, UWORD SrcBuffer,
            UWORD DestBuffer)

FUNCTION
  Copies  the  contents from one screen buffer to another.  Note that
  this  copy  can  only  be  performed  within  the  same  GameScreen
  structure.

  It  will  use the CPU and blitter to perform this action as quickly
```

```
  as possible.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  SrcBuffer  - The buffer source ID, eg BUFFER1.
  DestBuffer - The buffer destination ID, eg BUFFER2.
```

1.12 Blitter.GPI/InitRestore

```
NAME  InitRestore -- Initialise a RestoreList for buffered clearing.

SYNOPSIS
  RestoreList = InitRestore(AmtBuffers, AmtEntries)
           d0           d0     d1

  struct RestoreList * InitRestore(UWORD AmtBuffers,
     UWORD AmtEntries)

FUNCTION
  Initialises  a restorelist, necessary for restoring the backgrounds
  of screen areas that have been blitted over by bobs.

  A RestoreList is required whenever you specify the CLEAR or RESTORE
  flags  in  a  bob structure.  This list can be used universally, so
  you  do  not  need  to allocate a RestoreList for each bob that you
  want to blit.

  RestoreList's  are  vital in double or triple buffered environments
  as  it  can be very difficult to keep track of the background areas
  that need to be replaced.  By using a RestoreList and the CLEAR and
  RESTORE flags, you will eliminate this problem completely.

  Each  RestoreList Entry currently takes up 28 bytes of memory.  How
  many  entries  you require is dependent on the type of game you are
  writing, but 50 entries will be more than sufficient in most cases.

  There  is  a  significant  speed  difference  between restoring and
  clearing  which  you  should be aware of.  CLEARing is fast because
  the  blitter  is  using fewer channels and the CPU can be used more
  effectively.   RESTOREing is slower as the blitter has to move data
  between  two  different  areas,  plus it has to perform this action
  twice  (once  to  save  the  background,  and  once  to  return the
  background).  Use the CLEAR option whenever possible (if background
  is black) otherwise you will probably have to use RESTORE.

NOTE  If  you  were  to  overload  your AmtEntries limit by blitting more
  images  than  what was specified, you will certainly corrupt system
  memory.   In  any  case,  you  should  always tend to allocate more
  entries than you require for absolute security.

INPUTS  AmtBuffers - The amount of buffers in the destination screen.
  AmtEntries - The  maximum  amount of images that will be blitted to
         the screen at any point in time.

RESULT  RestoreList - Pointer  to  the  allocated  RestoreList;  this  is a
          private structure and you may not touch its contents.
```

```
SEE ALSO
  FreeRestore, Restore
```

1.13 Blitter.GPI/FreeRestore

```
NAME  FreeRestore -- Deallocates and removes a RestoreList from memory.

SYNOPSIS
  FreeRestore(RestoreList)
      d0

  void FreeRestore(struct RestoreList *)

FUNCTION
  Deallocates  all memory associated with a RestoreList.  You may not
  continue use of this RestoreList after it has been freed.

INPUTS  RestoreList - Pointer to a previously allocated RestoreList.

SEE ALSO
  InitRestore
```

1.14 Blitter.GPI/ResetRestore

```
NAME  ResetRestore -- Resets a RestoreList by clearing old image history.

SYNOPSIS
  ResetRestore(RestoreList)
       a1

  void ResetRestore(struct RestoreList *)

FUNCTION
  Resets a RestoreList's image history so that any currently buffered
  images  are  no  longer  waiting to be restored.  This is useful in
  circumstances  such  as  completely changing the background imagery
  and no longer blitting the previous set of images.

INPUTS  RestoreList -- Pointer to an initialised RestoreList;

SEE ALSO
  CleanUpRestore
```

1.15 Blitter.GPI/Restore

```
NAME  Restore -- Restores all buffered images by clearing or replacing
       their backgrounds.

SYNOPSIS
  Restore(GameScreen, RestoreList)
```

```
       a0   a1

  void Restore(struct GameScreen *, struct RestoreList *)

FUNCTION
  This  function  will  restore  all  the  backgrounds that have been
  blitted  over  by bobs containing the RESTORE and CLEAR flags.  The
  positioning  of  this  function  is  quite important - it should go
  BEFORE  any Draw() functions to be effective, otherwise you may get
  graphical glitches or a system crash.

INPUTS  GameScreen  - Pointer to an initialised GameScreen structure.
  RestoreList - Pointer to an initialised RestoreList structure.

SEE ALSO
  InitRestore, ResetRestore, CleanUpRestore
```

1.16 Blitter.GPI/DrawPixel

```
NAME  DrawPixel -- Draw a single pixel to a GameScreen.

SYNOPSIS
  DrawPixel(GameScreen, Buffer, XCoord, YCoord, Colour)
         a0   d0  d1  d2  d3

  void DrawPixel(struct GameScreen *, UWORD Buffer, WORD XCoord,
       WORD YCoord, ULONG Colour)

FUNCTION
  Draws  a pixel to coordinates XCoord, YCoord on a GameScreen.  This
  function  will  check  the  given coordinates to make sure that the
  pixel  is  on  screen,  otherwise  it  is not drawn.  If you do not
  require clipping, use DrawUCPixel().

NOTES Never  supply a colour that is beyond the amount of colours for the
  screen.

  Chunky  pixels  are  drawn  many  times  faster than interleaved or
  planar pixels, due to its more convenient display format.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.
  XCoord     - X coordinate for pixel.
  YCoord     - Y coordinate for pixel.
  Colour     - Colour number to use for the pixel.

SEE ALSO
  DrawPixelList, DrawUCPixelList
```

1.17 Blitter.GPI/DrawUCPixel

```
NAME  DrawUCPixel -- Draw a pixel to screen without any clipping checks.

SYNOPSIS
  DrawUCPixel(GameScreen, Buffer, XCoord, YCoord, Colour)
              a0   d0    d1    d2    d3

  void DrawUCPixel(struct GameScreen *, UWORD Buffer, WORD XCoord,
        WORD YCoord, ULONG Colour)

FUNCTION
  Draws  a pixel to coordinates XCoord, YCoord on a GameScreen.  This
  function  does  not  perform  clipping of any sort, and expects the
  coordinates  to  be  within  the  limits of the GameScreen.  If you
  require  clipping,  you  will  need  to write the necessary routine
  yourself.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.
  XCoord     - X coordinate for pixel.
  YCoord     - Y coordinate for pixel.
  Colour     - Colour number to use for the pixel.

SEE ALSO
  DrawPixel, DrawPixelList, DrawUCPixelList
```

1.18 Blitter.GPI/DrawPixelList

```
NAME  DrawPixelList -- Draw a list of pixels to a screen buffer.

SYNOPSIS
  DrawPixelList(GameScreen, Buffer, PixelList)
       a0     d0       a1

  void DrawPixelList(struct GameScreen *, UWORD Buffer,
          WORD PixelList[])

FUNCTION
  Draws  an  entire  list  of  pixels to screen, with border clipping
  enabled.

  This  is  the second fastest way to draw many pixels without making
  multiple  library  calls.  For  even  faster  drawing, you may use
  DrawUCPixelList(),  but  be aware that that function has no active
  clipping for the pixels.

  The  Pixel  List  is  not the standard GMS "LIST" type.  Instead it
  looks like this:

    dc.w  <AmtEntries>,<EntrySize>
    dc.l  <*Array>

  Array:  dc.w  <XCoord>,<YCoord>
    dc.l  <Colour>
    dc.w  ...
```

```
    dc.l  ...

  Example  for  blitting  3 pixels to a 4 colour screen of dimensions
  320x256:

  PixelList:
     dc.w   3,PXL_SIZEOF
     dc.l   .Values

  .Values
     PIXEL  140,201,3
     PIXEL  036,165,1
     PIXEL  224,051,2

  The  PIXEL  macro  is  used to help you fit the three fields on one
  line.  Here is the C version:

  struct PixelList PixelList = { /* Definition of pixel list header */
    3,
    sizeof(struct PixelEntry),
    Pixels
  };

  struct PixelEntry Pixels[3] = { /* The list of pixel values */
    140,201,3
    036,165,1
    224,051,2
  };

  You  are  also  allowed  to  mutate each PixelEntry so that you can
  store  extra  data in the array.  For example, if you are writing a
  demo  with flashing lights/pixels, then it would be most convenient
  if  you  could  store  the  on/off  state of each pixel in the same
  array.  To do this you will need to increase the EntrySize field so
  that GMS knows the true size of each image entry.  Eg:

  LightList:
    dc.w  3,PXL_SIZEOF+2
    dc.l  .Values
  .Values PIXEL 140,201,3
    dc.w  0
    PIXEL 036,165,1
    dc.w  1
    PIXEL 224,051,2
    dc.w  0

  You  can  also  pull  out the PixelEntry structure from the include
  files and add extra fields to that if it is more convenient.

  A  flag  exists  for  conveniently skipping pixel entries.  Specify
  SKIPPIXEL  in the X coordinate if you do not wish for a pixel to be
  drawn from that entry.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.
  PixelList  - Points to a list of pixels, explained above.
```

```
SEE ALSO
   DrawUCPixelList, DrawPixel
```

1.19 Blitter.GPI/DrawUCPixelList

```
NAME  DrawUCPixelList -- Draw  an  unclipped  list of pixels to a screen
               buffer.

SYNOPSIS
  DrawUCPixelList(GameScreen, Buffer, PixelList)
         a0       d0  a1

  void DrawUCPixelList(struct GameScreen *, UWORD Buffer,
            struct PixelList *)

FUNCTION
  Draws  a  list  of  unclipped pixels to a screen buffer.  This is a
  special function that is provided only for situations where you are
  100%  certain  that  no  pixels  lie  outside  the picture borders.
  Because  there  is  no checking, any rogue pixels can cause illegal
  memory  over-writes, so be careful!  The advantage of this function
  is that it is very fast at mass pixel writes.

  See  DrawPixelList()  for  detailed  information  on  how to form a
  PixelList.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.
  PixelList  - Points to a list of pixels, explained above.

SEE ALSO
   DrawPixelList, DrawPixel
```

1.20 Blitter.GPI/ReadPixel

```
NAME  ReadPixel -- Reads a pixel colour from position X/Y.

SYNOPSIS
  Pixel = ReadPixel(GameScreen, Buffer, XCoord, YCoord)
   d0         a0   d0  d1  d2

  ULONG ReadPixel(struct GameScreen *, UWORD Buffer, WORD XCoord,
        WORD YCoord)

FUNCTION
  Reads  a pixel from a GameScreen buffer area and returns its colour
  number or RGB value depending on the screen type.  If you give this
  function coordinates that lie outside of the screen's picture area,
  it will return -1 in Pixel.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.
```

```
  XCoord     - The X coordinate to read the pixel from.
  YCoord     - The Y coordinate to read the pixel from.

RESULT  Pixel - The pixel colour number/RGB value or -1.

SEE ALSO
  ReadPixelList
```

1.21 Blitter.GPI/ReadPixelList

```
NAME  ReadPixelList -- Reads a list of pixels.

SYNOPSIS
  Pixel = ReadPixelList(GameScreen, Buffer, PixelList)
   d0      a0     d0       a1

  ULONG ReadPixelList(struct GameScreen *, UWORD Buffer,
            struct PixelList *)

FUNCTION
  Untested.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.
  PixelList  - Pointer to a list of pixels.

SEE ALSO
  ReadPixel
```

1.22 Blitter.GPI/DrawLine

```
NAME  DrawLine -- Draws a line between two points on a GameScreen.

SYNOPSIS
  DrawLine(GameScreen, Buffer, XStart, YStart, XEnd, YEnd, Colour)
              a0         d0      d1      d2     d3    d4     d5

  void DrawLine(struct GameScreen *, UWORD Buffer, WORD XStart,
                 WORD YStart, WORD XEnd, WORD YEnd, ULONG Colour)

FUNCTION
  Draws a line between (XStart,YStart) and (XEnd,YEnd).  Depending on
  selections  made  in  GMSPrefs this function may draw the line with
  the processor or blitter (or perhaps both).

  This  function  supports clipping for lines that are outside of the
  picture borders.  For faster line drawing, use DrawUCLine when you
  know that a line is within the screen borders.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.
  XStart     - X starting coordinate.
```

```
  YStart     - Y starting coordinate.
  XEnd       - X end coordinate.
  YEnd       - Y end coordiate.
  Colour     - Line colour.

SEE ALSO
  DrawUCLine
```

1.23 Blitter.GPI/DrawUCLine

```
NAME  DrawUCLine -- Draws  a  line  between  two  points on a GameScreen
         without clipping checks.

SYNOPSIS
  DrawUCLine(GameScreen, Buffer, XStart, YStart, XEnd, YEnd, Colour)
           a0   d0  d1  d2 d3    d4     d5

  void DrawUCLine(struct GameScreen *, UWORD Buffer, UWORD XStart,
                  UWORD YStart, UWORD XEnd, UWORD YEnd, ULONG Colour)

FUNCTION
  Draws a line between (XStart,YStart) and (XEnd,YEnd).  Depending on
  selections  made  in  GMSPrefs this function may draw the line with
  the processor or blitter (or perhaps both).

  The  function  does  not  perform  clipping  of  any sort, so it is
  imperative  that  you  keep  any  lines  that  you draw within your
  picture boundaries.  Otherwise use DrawLine().

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Buffer     - The Buffer ID, ie BUFFER1, BUFFER2 or BUFFER3.
  XStart     - X starting coordinate.
  YStart     - Y starting coordinate.
  XEnd       - X end coordinate.
  YEnd       - Y end coordiate.
  Colour     - Line colour.

SEE ALSO
  DrawLine
```

1.24 Blitter.GPI/AllocBlitMem

```
NAME  AllocBlitMem -- Allocate blitter memory.

SYNOPSIS
  Memory = AllocBlitMem(Size)
    d0           d0

  APTR AllocBlitMem(ULONG Size)

FUNCTION
  Allocates  a  block  of  memory  suitable  for the Blitter.GPI.  On
```

```
  current  Amiga's  it  will  only grab chip mem, but fast ram may be
  supported in the future (CPU only blitting).

INPUTS  Size - The Size of the memory to allocate.

RESULT  Memory - Pointer  to  the  allocated memory.  All blitter memory is
     formatted with 0's when allocated.  Returns NULL if error.

SEE ALSO
  FreeMemBlock
```

1.25 Blitter.GPI/GetBob

```
NAME  GetBob -- Gets the latest bob structure.

SYNOPSIS
  Bob = GetBob()
  d0

  struct Bob * GetBob(void);

FUNCTION
  Allocates  the latest version of a GMS bob structure and returns it
  back  to  you.   The structure fields will be empty so that you can
  fill them out to suit your requirements.  Before your program exits
  you  will need to free the structure, this is automatically done in
  the FreeBob() function.

  You  have  to use this function if you do not want to use tag lists
  to initialise your bobs (remember that it is illegal to compile and
  use pre-initialised structures in GMS programs).

RESULT  Bob - Points  to  the latest version of a GMS bob structure or NULL
        if failure.

SEE ALSO
  GetMBob, InitBob
```

1.26 Blitter.GPI/GetMBob

```
NAME  GetMBob -- Gets the latest mbob structure.

SYNOPSIS
  MBob = GetMBob()

  struct MBob * GetMBob(void);

FUNCTION
  Allocates the latest version of a GMS mbob structure and returns it
  back  to  you.   The structure fields will be empty so that you can
  fill them out to suit your requirements.  Before your program exits
  you  will need to free the structure, this is automatically done in
```

```
  the FreeBob() function.

  You  have  to use this function if you do not want to use tag lists
  to initialise your bobs (remember that it is illegal to compile and
  use pre-initialised structures in GMS programs).

RESULT  MBob - Points to the latest version of a GMS mbob structure or NULL
         if failure.

SEE ALSO
  GetBob, InitBob
```

1.27 Blitter.GPI/FreeBlitMem

```
NAME  FreeBlitMem -- Frees memory allocated by AllocBlitMem().

SYNOPSIS
  FreeBlitMem(MemBlock)
              d0

  void FreeBlitMem(APTR MemBlock)

FUNCTION
  Frees a memory block that has been allocated by AllocBlitMem().  If
  you  pass  this  function  a  pointer to NULL then the call will be
  ignored.

  Blitter  memory  that  is  allocated and not freed will stay in the
  system  resource  list,  which will present you with a message when
  your program exits.

INPUT MemBlock - MemBlock allocated from AllocBlitMem().

SEE ALSO
  AllocBlitMem
```

1.28 Blitter.GPI/

```
NAME

SYNOPSIS

FUNCTION

INPUTS

RESULT

SEE ALSO
```