

Blitter

Paul Manias

COLLABORATORS

	<i>TITLE :</i> Blitter		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Paul Manias	June 4, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Blitter	1
1.1	Blitter.GPI	1
1.2	Blitter.GPI Functions	1
1.3	games.library/AllocBlitter	2
1.4	games.library/FreeBlitter	2
1.5	games.library/InitBOB	3
1.6	games.library/FreeBOB	7
1.7	games.library/ClearBOB	7
1.8	games.library/DrawBOB	8
1.9	games.library/DrawBOBList	9
1.10	games.library/ClrScreen	10
1.11	games.library/CopyBuffer	10
1.12	games.library/InitRestore	11
1.13	games.library/FreeRestore	12
1.14	games.library/ResetRestore	12
1.15	games.library/Restore	12
1.16	games.library/DrawPixel	13
1.17	games.library/DrawUCPixel	14
1.18	games.library/DrawPixelList	14
1.19	games.library/DrawUCPixelList	16
1.20	games.library/ReadPixel	17
1.21	games.library/ReadPixelList	18
1.22	games.library/DrawLine	18
1.23	games.library/DrawUCLine	19
1.24	games.library/AllocBlitMem	20

Chapter 1

Blitter

1.1 Blitter.GPI

Name: BLITTER.GPI AUTODOC
Version: 0.5 Beta.
Date: 15 February 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
AllocBlitter()
AllocBlitMem()
ClearBOB()
ClrScreen()
CopyBuffer()
DrawBOB()
DrawBOBList()
DrawLine()
DrawUCLine()
FreeBlitter()
FreeBOB()
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 games.library/AllocBlitter

games.library/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 games.library/FreeBlitter

games.library/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 games.library/InitBOB

games.library/InitBOB

NAME InitBOB -- Initialise a Blitter OBject structure ready for blitting.

SYNOPSIS

```
ErrorCode = InitBOB(GameScreen, BOB)
                a0          a1
```

```
ULONG InitBOB(struct GameScreen *, APTR BOB)
```

FUNCTION

Initialises a Blitter OBject. All blitter structures are supported by this function, so you may pass it BB (standard BOB) or MB (multiple BOB) structure types. This function is fully supportive of LIST and OLST types, just pass it a standard LIST or OLST rather than a BOB structure.

INPUTS GameScreen - Pointer to an initialised GameScreen structure.
BOB - Pointer to a BOB/MBOB structure or a LIST/OLST.

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.

If you are using the default X/Y standard, then your X/Y values will be destroyed after initialisation and replaced with the direct format. This is done to cut down on memory usage, and on the likely-hood that you only need to initialise a BOB once.

Note: If you specify the GENMASK (Generate Mask) flag as an attribute, all values entered in the Mask sections of the array are ignored and replaced with pointers to the allocated mask areas.

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, this value will be identical to that pictures byte 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: The value given here is used in precalculations, so you cannot make dynamic changes to this field after initialisation.

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_Picture (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_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_SIZEEOF.

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 `ErrorCode` - Returns `NULL` if successful.

SEE ALSO

`FreeBOB`, `games/games.i`

1.6 games.library/FreeBOB

`games.library/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 games.library/ClearBOB

`games.library/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 games.library/DrawBOB

games.library/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, struct RestoreList *,
             UWORD Buffer)
```

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 until the blitter is available. 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 games.library/DrawBOBList

games.library/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[],
                 struct RestoreList *, UWORD Buffer)
```

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 games.library/ClrScreen

games.library/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 Blitter 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 games.library/CopyBuffer

games.library/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 games.library/InitRestore

games.library/InitRestore

NAME InitRestore -- Initialise a RestoreList for buffered clearing.

SYNOPSIS

```
RestoreList = InitRestore(AmtBuffers, AmtEntries)
                   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 games.library/FreeRestore

games.library/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 games.library/ResetRestore

games.library/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 games.library/Restore

games.library/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 games.library/DrawPixel

games.library/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 to use for the pixel.

SEE ALSO

DrawPixelList, DrawUCPixelList

1.17 games.library/DrawUCPixel

games.library/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 to use for the pixel.

SEE ALSO

DrawPixel, DrawPixelList, DrawUCPixelList

1.18 games.library/DrawPixelList

games.library/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.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
    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:

```
PIXELLIST(3) = {
    3,sizeof(struct PixelEntry),
    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
    PIXEL   140,201,3
    dc.w    0
    PIXEL   036,165,1
    dc.w    1
    PIXEL   224,051,2
    dc.w    0
```

The equivalent mutant structure in C might look like this:

```
struct MutantEntry {
    struct PixelEntry Data;
    UWORD State;
};
```

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.

NOTE Clipping is done to the screen borders and not to the picture borders.

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 games.library/DrawUCPixelList

games.library/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.

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

```
dc.w <AmtEntries>,<EntrySize>
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
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:

```
PIXELLIST(3) = {
    3,sizeof(struct PixelEntry),
    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 to the new size for each entry. Eg:

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

The equivalent mutant structure in C might look like this:

```
struct MutantEntry {
    struct PixelEntry Data;
    UWORD State;
};
```

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

DrawPixelList, DrawPixel

1.20 games.library/ReadPixel

games.library/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 games.library/ReadPixelList

games.library/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 games.library/DrawLine

games.library/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 coordinate.
Colour      - Line colour.
```

SEE ALSO

DrawUCLine

1.23 games.library/DrawUCLine

games.library/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 coordinate.
Colour      - Line colour.
```

SEE ALSO
DrawLine

1.24 games.library/AllocBlitMem

games.library/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
