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 Documentation

```
MODULE DOCUMENTATION
Name:      BLITTER
Version:   0.9 Beta
Date:      December 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.


CHANGES VERSION 0.9B
Removed: AllocBlitter()
         FreeBlitter()

Edited:  AllocBlitMem()
         FreeBlitMem()
         DrawBob()
         DrawBobList()
         DrawPixelList()
         CreateMasks()
```

1.2 Blitter Information

```
 INTRODUCTION

 OBJECTS
 Bitmap
 Bob
 MBob
 Restore

 FUNCTIONS
 AllocBlitMem()
 ClearBob()
```

```
ClearBitmap()
CopyBuffer()
CreateMasks()
DrawBob()
DrawBobList()
DrawLine()
DrawPixel()
DrawPixelList()
DrawUCLine()
DrawUCPixel()
DrawUCPixelList()
FreeBlitMem()
ReadPixel()
ReadPixelList()
SetBobDimensions()
SetBobDrawMode()
SetBobFrames()
```

1.3 Blitter Module Overview

BLITTER SUPPORT OVERVIEW

The blitter support is designed so that it can perform the task of drawing images to screen as quickly and efficiently as possible. The best way of doing this is to provide you with a large amount of drawing options, so that you can specify exactly how you want an image to be drawn. For experienced programmers this level of functionality is extremely useful in providing fast and optimised drawing speeds. If you're a beginner it may take some time before you learn what methods to use in each situation, but with a little practice you will know how to use the available options to your advantage. Currently, options include blitting in lists, clipping on/off, restore and clear modes, masking on/off, mask generation, multiple Bobs, setting clip areas, and others.

To make the Blitter module as effective as possible, special rules are in place that are ideal for games applications only. If you have come from OS programming then the ideas may sound a little unusual, but for games programming they make sense. The nature of any game is not to multi-task with other games, as it is impossible for a player to play two different games on the same screen at once. If two different games try to share resources, the result can be a catastrophe. Since GMS only allows one game to be using the display at any given time (ie no windows or screen dragging), it made sense that the only task allowed to use the drawing functions is the one at the front. This means that at any time when your task is active, you know that you have 100% available blitter time. Your drawing cycles will not be stolen by hidden tasks!

It was suggested to me in the past that I could use QBlit() or similar interrupt driven blitting. The advantage of this is that it is easier for the processor to do things while the blitter is active, and there is no blitter waiting involved. This sounded like a good idea at the time, but after trying it I found the results to be disappointing, so I dropped it. Why? Because this method did not recognise the fact that the blitter is so

SLOW! Using the blitter only, you would be lucky to get 7 32x32x32 BOBs on a 50fps screen with clear modes on. It doesn't matter how fast your routines are, the blitter will not move data any faster. This a sorry speed for any arcade game to be using.

Instead I am now in the process of implementing high-speed CPU assisted blitter drawing routines. These work extremely well in mass drawing operations with about 20-30% speed up on my '020 A1200+Fast in comparison to blitter-only drawing. On a '030 I would expect at least 40% faster operations, while on '040 and '060 we are probably looking at the CPU drawing 2 bobs while the blitter draws 1. I think Amiga owners with fast CPU's will appreciate this feature, while '000 users will not suffer because the blitter will take most of the load for slow CPU's.

Enjoy the fast drawing, and if you have any good ideas for advancements then send them all in.

1.4 Blitter: AllocBlitMem()

```
FUNCTION
Name:     AllocBlitMem()
Short:    Allocates a block of blitter memory.
Synopsis: APTR AllocBlitMem(LONG Size [d0], LONG Flags [d1])

DESCRIPTION
This function allocates a block of memory suitable for the Blitter module.
On current Amiga's it will only grab chip mem, but fast ram may be
supported in the future (for CPU blitting).

You can free the memory by calling FreeBlitMem() or FreeMemBlock().

NOTE
It is possible to use video memory as blitter memory. However the reverse
is not true, because as mentioned above blitter memory can be in fast or
"program" ram, which is not good for video displays.

INPUTS
Size  - The Size of the memory to allocate.
Flags - Special memory flags such as MEM_UNTRACKED.

RESULT
Returns a pointer to the allocated memory. All blitter memory is formatted
with 0's when allocated. If an error occurred, a NULL pointer is returned.

SEE ALSO
Blitter: FreeBlitMem()
Kernel:  AllocMemBlock()
         FreeMemBlock()
```

1.5 Blitter: ClearBob()

```
FUNCTION
Name:     ClearBob()
Short:    Clears a Bob image from a Bitmap.
Synopsis: ClearBob(APTR Bob [a1])

DESCRIPTION
Clears  a Bob's image from a bitmap.  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  a Bitmap you probably should be using a
Restore object.

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

INPUTS
Bob - Pointer to an initialised Bob/MBob structure.

SEE ALSO
Blitter: DrawBob()
```

1.6 Blitter: ClearBitmap()

```
FUNCTION
Name:     ClearBitmap()
Short:    Clear a bitmap.
Synopsis: void ClearBitmap(*Bitmap [a0]);

DESCRIPTION
Clears  all  of the data contained in a Bitmap.  The method used to do this
is  largely  dependent  on  the  selection  the user has made in the system
preferences.  At the moment there are three available clearing 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
Bitmap - Pointer to an initialised Bitmap structure.  The Bitmap memory
   must reside in Video RAM.
```

1.7 Blitter: CopyBuffer()

```
FUNCTION
Name:     CopyBuffer()
Short:    Copy the contents from one screen buffer to another.
Synopsis: void CopyBuffer(*Screen [a0], WORD SrcBuffer [d0],
           WORD DestBuffer [d1])
```

DESCRIPTION
Copies the contents from one screen buffer to another. Note that this copy can only be performed within the same Screen structure, so you cannot copy from one Screen to another.

This function will use both the CPU and blitter to perform this action as quickly as possible.

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

1.8 Blitter: CreateMasks()

FUNCTION
Name: CreateMasks()
Short: Creates or recreates all the masks for a Bob.
Synopsis: LONG CreateMasks(APTR Bob [a1])

DESCRIPTION
This function creates all the masks for a Bob or MBob. If the Bob has already been initialised, the old masks will be freed and a new set will be created.

Masks are created by looking at the graphics data of the Bob in question. Colour 0 is always considered to be the transparent colour, so the background pixels will always show through this colour.

If you set the FILLMASK option in Bob->Attrib, masked pixels will be created where there are "holes" in the graphic. For example if you draw a character with black eyes, normally you would end up seeing straight through them and into the background. With FILLMASK the eyes would be filled in. The filling routine is a simple linear filler only, so if you want to generate more complex masks it is best to draw the mask yourself until a better algorithm is implemented.

INPUT
Bob - Pointer to a Bob or MBob structure.

RESULT
Returns ERR_OK on success. This call can fail for a number of reasons, but more likely only if there is little memory left for mask allocation.

1.9 Blitter: DrawBob()

FUNCTION
Name: DrawBob()
Short: Draws a Blitter OBject directly to a bitmap.
Synopsis: void DrawBob(APTR Bob [a1]);

DESCRIPTION
This function draws a Bob or MBob to a Bitmap, according to the values in the Bob/MBob structure.

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 or 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 Restore objects as a fast way to redraw or clear your backgrounds.

INPUT
Bob - Pointer to an initialised Bob/MBob structure.

SEE ALSO
Blitter: DrawBobList()
Kernel: Draw()

1.10 Blitter: DrawBobList()

FUNCTION
Name: DrawBobList()
Short: Draws a list of Blitter OBjects onto a Bitmap.
Synopsis: void DrawBobList(APTR *BobList [a1]);

DESCRIPTION
This is a mass-drawing function that allows you to blit many Bobs from a list onto their respective Bitmaps. It handles all Bob structure types and is the fastest way to process the drawing of many Bobs at any one time.

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
BobList - Pointer to a LIST of Bob structures to draw. Must be terminated

```
    by a LISTEND.

SEE ALSO
Blitter: DrawBob()
```

1.11 Blitter: DrawLine()

```
FUNCTION
Name:     DrawLine()
Short:    Draws a line between two points on a Bitmap.
Synopsis: void DrawLine(*Bitmap [a0], WORD XStart [d1], WORD YStart [d2],
     WORD XEnd [d3], WORD YEnd [d4], LONG Colour [d5])

DESCRIPTION
Draws a line between (XStart,YStart) and (XEnd,YEnd). Depending on
selections made in the system preferences, 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 Bitmap's borders.

INPUTS
Bitmap  - Pointer to an initialised Bitmap structure.
XStart  - X starting coordinate.
YStart  - Y starting coordinate.
XEnd    - X end coordinate.
YEnd    - Y end coordiate.
Colour  - Line colour value.

SEE ALSO
Blitter: DrawUCLine()
```

1.12 Blitter: DrawPixel()

```
FUNCTION
Name:     DrawPixel()
Short:    Draw a single pixel to a Bitmap.
Synopsis: void DrawPixel(*Bitmap [a0], WORD XCoord [d1], WORD YCoord [d2],
     LONG Colour [d3])

DESCRIPTION
Draws a pixel to coordinates XCoord, YCoord on a Bitmap. This function
will check the given coordinates to make sure that the pixel is inside the
Bitmap area, 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 Bitmap.

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

```
INPUTS
Bitmap - Pointer to an initialised Bitmap structure.
XCoord - X coordinate for pixel.
YCoord - Y coordinate for pixel.
Colour - Colour number to use for the pixel.

SEE ALSO
Blitter: DrawPixelList()
         DrawUCPixelList()
```

1.13 Blitter: DrawUCLine()

```
FUNCTION
Name:     DrawUCLine()
Short:    Draws a line between two points on a Bitmap, without clipping
     checks.
Synopsis: void DrawLine(*Bitmap [a0], WORD XStart [d1], WORD YStart [d2],
     WORD XEnd [d3], WORD YEnd [d4], LONG Colour [d5])

DESCRIPTION
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
Bitmap - Pointer to an initialised Bitmap structure.
XStart - X starting coordinate.
YStart - Y starting coordinate.
XEnd   - X end coordinate.
YEnd   - Y end coordiate.
Colour - Line colour.

SEE ALSO
Blitter: DrawLine()
```

1.14 Blitter/DrawUCPixel

```
FUNCTION
Name:     DrawUCPixel()
Short:    Draw a pixel to Bitmap without any clipping checks.
Synopsis: void DrawUCPixel(*Bitmap [a0], WORD XCoord [d1], WORD YCoord [d2],
      LONG Colour [d3])

DESCRIPTION
Draws a pixel to coordinates (XCoord, YCoord) on a Bitmap. This function
does not perform clipping of any sort, and expects the coordinates to be
within the limits of the Bitmap. If you require clipping, you will need to
```

```
use DrawPixel().

INPUTS
Bitmap - Pointer to an initialised Bitmap structure.
XCoord - X coordinate for pixel.
YCoord - Y coordinate for pixel.
Colour - Colour number to use for the pixel.

SEE ALSO
Blitter: DrawPixel()
         DrawPixelList()
         DrawUCPixelList()
```

1.15 Blitter: DrawPixelList()

```
FUNCTION
Name:     DrawPixelList()
Short:    Draw a list of pixels to a Bitmap buffer.
Synopsis: void DrawPixelList(*Bitmap [a0], WORD *PixelList [a1]);

DESCRIPTION
This  function  draws  an  entire  list  of pixels to a Bitmap, 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(), at
the risk of losing active clipping for the pixels.

The  Pixel  List  is  not  the standard "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  ...

Here is an example for blitting 3 pixels to a 4 colour Bitmap of dimensions
320x256.  Note the use of the PIXEL macro that helps to fit the three entry
fields on one line:

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

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

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 DrawPixelList() 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 for that entry.

```
INPUTS
Bitmap    - Pointer to an initialised Bitmap structure.
PixelList - Points to a list of pixels, explained above.

SEE ALSO
Blitter: DrawPixel()
         DrawUCPixelList()
```

1.16 Blitter: DrawUCPixelList()

```
FUNCTION
Name:     DrawUCPixelList()
Short:    Draw an unclipped list of pixels to a Bitmap.
Synopsis: void DrawUCPixelList(*Bitmap [a0], *PixelList [a1])

DESCRIPTION
Draws a list of unclipped pixels to a Bitmap. 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
Bitmap - Pointer to an initialised Bitmap structure.
PixelList - Points to a list of pixels, explained above.

SEE ALSO
Blitter: DrawPixel
Blitter: DrawPixelList

1.17 Blitter: FreeBlitMem()

FUNCTION
Name: FreeBlitMem()
Short: Frees memory allocated by AllocBlitMem()
Synopsis: void FreeBlitMem(APTR MemBlock [d0]);

DESCRIPTION
This 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.

Any blitter memory that is allocated and not freed, will stay in the system resource list. Such bugs will cause the system to present you with an error message when your program exits.

INPUT
MemBlock - MemBlock allocated from AllocBlitMem().

SEE ALSO
Blitter: AllocBlitMem()
Kernel: FreeMemBlock()

1.18 Blitter: ReadPixel()

FUNCTION
Name: ReadPixel()
Short: Reads a pixel colour from position X/Y on a Bitmap.
Synopsis: LONG ReadPixel(*Bitmap [a0], WORD XCoord [d1], WORD YCoord [d2])

DESCRIPTION
This function reads a pixel from a Bitmap area and returns its colour number or RGB value, depending on the Bitmap->Type. If you give this function coordinates that lie outside of the Bitmap's area, it will return -1.

INPUTS
Bitmap - Pointer to an initialised Bitmap structure.
XCoord - The X coordinate to read the pixel from.

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

RESULT
The pixel number/RGB value or -1.

SEE ALSO
Blitter: ReadPixelList()
```

1.19 Blitter: ReadPixelList()

```
FUNCTION
Name:     ReadPixelList()
Short:    Reads a list of pixels from a Bitmap.
Synopsis: void ReadPixelList(*Bitmap [a0], *PixelList [a1])

DESCRIPTION
This function will take a given pixel list, analyse each set of coordinates
and write the pixel values from the Bitmap out to each colour field.
Anyy coordinates that lie outside of the Bitmap's boundary will receive a
colour value of -1.

NOTE
This function has not been tested yet.

INPUTS
Bitmap    - Pointer to an initialised Bitmap structure.
PixelList - Pointer to a pixel list array.

SEE ALSO
Blitter: ReadPixel()
```

1.20 Blitter: SetBobDimensions()

```
FUNCTION
Name:     SetBobDimensions()
Short:    Re-calculates the dimension fields for a Bob after one or more of
           them have changed.
Synopsis: void SetBobDimensions(*Bob [a1]);

DESCRIPTION
This function will re-calculate the dimension fields for a Bob after one or
more of them have changed.  By dimensions, this specifically means the
Width and Height values of the Bob.

If you change the Bob's Width or Height fields then you must call this
function to prepare the Bob for the next blit.  Otherwise you will most
likely not get the wanted result.

NOTE
If you are using generated masks and you change the size of your Bob, this
function will call CreateMasks() to cater for this change.  This may slow
the function down slightly.  If the Bob is using the RESTORE mode then the
```

function will be slowed down more considerably as all of the restore buffers will need to be reallocated.

INPUT
Bob - Pointer to the Bob to be updated.

1.21 Blitter: SetBobDrawMode()

FUNCTION
Name: SetBobDrawMode()
Short: Sets the drawing mode of a Bob.
Synopsis: LONG SetBobDrawMode(*Bob [a1], LONG Flags [d0]);

DESCRIPTION
If you want to change the drawing mode of a Bob that has already been initialised, you will need to call this function.

INPUT
Bob - Pointer to the Bob that is to be updated.
Flags - New settings for the Bob->Attrib field.

RESULT
Returns ERR_OK if the new setting was successful.

1.22 Blitter: SetBobFrames()

FUNCTION
Name: SetBobFrames()
Short: Recreates a Bob's direct lists after altering its frames.
Synopsis: LONG SetBobFrames(*Bob [a1]);

DESCRIPTION
This function recreates the Bob's GfxDirect and MaskDirect lists from the GfxCoords and MaskCoords lists. You need to call this function whenever you change the coordinates of one or more of the graphics or masks of your Bob. You may also need to call CreateMasks() if the masks go out of whack due to the changing of the coordinates.

If you are not using the GfxCoords or MaskCoords fields and instead are only using the *Direct fields, you must still call this function if you make changes to the direct lists.

NOTE
You cannot make attempts to expand the amount of frames in your Bob with this function.

INPUT
Bob - Pointer to the Bob to be updated.

RESULT
Returns ERR_OK on success.

1.23 Blitter:

```
FUNCTION
Name:
Short:
Assembler:
C/C++:

DESCRIPTION

NOTE

INPUT

RESULT

SEE ALSO
```