Plananarama 2.0

The Planar Graphics Engine for Hollywood

Andreas Falkenhahn

Table of Contents

1 General information

1.1 Introduction

The Plananarama plugin allows Hollywood to run on planar (palette-based) screens. This finally makes it possible to run Hollywood scripts on plain AGA or ECS systems with no graphics board installed - for the first time since Hollywood 1.93 (released in March 2005)! Starting with version 2.0, Hollywood required CyberGraphX or Picasso96 to run. Since then, many people have asked for a revival of Hollywood's planar engine, so here it is, a real blast from the past! Thanks to Hollywood 6's greatly extended plugin API this feature could be implemented completely in plugin space. Once Plananarama is installed, all Hollywood scripts will 'automagically' run on palette screens again! All resolutions are supported - from 8-bit LowRes to 1-bit productivity SuperHighRes Interlace.

Starting with Plananarama 2.0 it is also possible to get direct access to the screen's palette pens when it is running in fullscreen mode. This allows you to draw directly using the screen's palette without the need for any color remapping so drawing will become much faster. Also, it's possible to create nice palette effects by changing pens, e.g. it's possible to fade or cycle colors with almost no CPU load because in palette mode, Plananarama can directly modify the hardware's color register instead of having to redraw everything when colors change (as it needs to be done for real TrueColour graphics).

Also, Plananarama 2.0 introduces support for real hardware sprites. When using hardware sprites, the Amiga's custom chip hardware is used to draw the sprites which allows your script to draw sprites in next to no time with almost no CPU load.

All of this makes Plananarama the ultimate plugin for targetting classic Amiga systems that don't have a graphics board installed.

1.2 Terms and conditions

WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1.3 Requirements

This plugin requires at least Hollywood 6.0 since it uses the display and bitmap adapter APIs introduced with Hollywood 6.0. Some features, however, require newer Hollywood versions. Here's an overview of the features only available with newer Hollywood versions:

- Unicode support: requires Hollywood 7
- Support for `VanillaKey` and `OnDropFile` event handler: requires Hollywood 7
- Support for `OnRawKey` event handler: requires Hollywood 7.1
- Support for menus: requires Hollywood 9
- Support for Plananarama's special palette mode: requires Hollywood 9
- Hardware sprite support: requires Hollywood 9

The following optional components might be required as well:

- guigfx.library and render.library: These two libraries are required when the `PaletteMode` tag in the `@REQUIRE` call has been set to `False` (which is also the default). In that case, Plananarama will remap all graphics to the current screen's palette. This is done using guigfx.library and render.library. If you set `PaletteMode` to `True`, however, Plananarama won't require guigfx.library and render.library because graphics won't be remapped to the screen palette but all graphics will be drawn using the screen's pens.

  Note that render.library is available in two flavours: There is a v40 and a v30 branch. It is recommended to use the latest version of the v30 branch because all the v40 versions are C ports of the v30 branch versions which are all written in 68k assembler. The assembler version (= v30 branch) of render.library is much faster than the C version which is why you might want to use the assembler version instead. The latest assembler version of render.library is v32.0. So if you care about performance you should use this one. Also, the v40 branch of render.library requires an FPU which is another reason to use the latest version from the v32 branch instead.
- FBlit: When the `PaletteMode` tag in the `@REQUIRE` call has been set to `False` (which is also the default), it is highly recommended to use FBlit because otherwise your chip memory will be gone in no time and Hollywood will run out of memory. So make sure you install FBlit first and you add Hollywood to its "Include" list in the "FAllocBitMap" tab. If Hollywood is included in this list, it will be able to place

graphics in fast memory which is absolutely needed since 2 MB of chip memory certainly won't be enough for Hollywood. Using FBlit also has the advantage that Hollywood can use the CPU for blitting which is much faster than the blitter on higher 68k CPUs or WinUAE anyway. If `PaletteMode` is set to `True`, FBlit isn't necessary because Hollywood can store its graphics data in fast memory. In that case, however, you might want to use BlazeWCP (see below).

– BlazeWCP: When the `PaletteMode` tag in the `@REQUIRE` call has been set to `True`, Hollywood will store all graphics in fast memory and draw them using the `WriteChunkyPixels()` command from graphics.library. `WriteChunkyPixels()`, however, is very slow on OS3.1-3.9 so you might want to use BlazeWCP from Aminet to speed up things. Rumour has it that the implementation of `WriteChunkyPixels()` in the new classic AmigaOS releases by Hyperion (3.1.4 and up) has seen some optimizations so BlazeWCP might not be necessary on 3.1.4 and up any longer but I haven't done any benchmarks so I can't tell whether BlazeWCP is still necessary with the classic AmigaOS releases by Hyperion. It's definitely recommended with classic AmigaOS 3.1, 3.5 and 3.9.

1.4 Installation

To install Plananarama, just use the included installer or copy the file `plananarama.hwp` to `LIBS:Hollywood`.

2 About Plananarama

2.1 Credits

Plananarama was written by Andreas Falkenhahn. This plugin was first created during the development of Hollywood 6.0 as a proof-of-concept for the flexibility of Hollywood 6.0's powerful display and bitmap adapters. This plugin completely replaces Hollywood's default display adapter and installs a custom one that is able to run on planar screens.

If you want to contact me, you can either send an e-mail to andreas@airsoftsoftwair.de or use the contact form on http://www.hollywood-mal.com.

2.2 Frequently asked questions

This section covers some frequently asked questions. Please read them first before asking on the forum because your problem might have been covered here.

Q: Plananarama says "Cannot open guigfx.library!" but I have guigfx.library installed.

A: guigfx.library requires render.library and will fail to open in case render.library isn't present so make sure you have render.library as well. If you do, make sure you have the right version of render.library. Some require an FPU and will fail to open if no FPU is present. See Section 1.3 [Requirements], page 2, for details.

Q: My system crashes with a 8000000B software failure.

A: You are using a version of render.library or guigfx.library that needs an FPU on a system without an FPU. Make sure to use the appropriate version for your system because some versions of render.library and guigfx.library don't check if an FPU is present, they will just crash if there is none. See Section 1.3 [Requirements], page 2, for details.

Q: I'm getting an "Out of memory!" error but I have lots of fast memory.

A: First make sure you have installed FBlit. Then make sure that you've configured FBlit correctly. You have to add Hollywood to FBlit's "Include" list in the "FAllocBitMap" tab. If Hollywood is included in this list, it will be able to place graphics in fast memory which is absolutely needed since 2 MB of chip memory certainly won't be enough for Hollywood. So make sure you configure FBlit correctly.

Q: Plananarama is very slow.

A: This can have many reasons. If you've set the `PaletteMode` tag to `True` in `@REQUIRE`, make sure to install the BlazeWCP patch from Aminet to speed up drawing. In palette mode, you can also speed up drawing significantly if you only use palette graphics, i.e. don't use brushes, BGPics, anims etc. that are stored as hi/true colour RGB images but use palette images only. Ideally, they should use the same palette as the screen you want to draw them to so that Plananarama doesn't have to do any remapping but can just draw them.

Another reason might be that your script is drawing many images with alpha channel graphics, e.g. anti-aliased text/shapes or images with variable levels of transparency. These

things do not make much sense on palette-based screens and they are very, very expensive on the CPU because Plananarama needs to remap these images from true colour chunky pixels to planar graphics all the time.

Thus, if you care about performance you should not use any alpha channel images and transparency should be limited to monochrome transparency (i.e. visible and invisible pixels). The best idea is to use a fixed set of prerendered graphics. When just drawing prerendered graphics with no changes concerning the colors, Plananarama should perform quite well on planar screens and scripts should also be usable on slower systems, though you still need a fast CPU and some fast memory of course.

Q: Is there a Hollywood forum where I can get in touch with other users?

A: Yes, please check out the "Community" section of the official Hollywood Portal online at `http://www.hollywood-mal.com`.

Q: Where can I ask for help?

A: There's an active forum at `http://forums.hollywood-mal.com`. You're welcome to join it and ask your question there.

Q: I have found a bug.

A: Please post about it in the "Bugs" section of the forum.

2.3 Future

Here are some things that are on my to do list:

- add support for palette mode on public screens like Workbench; this requires a shared pen management system because Plananarama apps must share their pens with other apps on the same screen
- add support for hardware scrolling
- add support for hardware doublebuffers
- add support for bobs

Don't hesitate to contact me if Plananarama lacks a certain feature that is important for your project.

2.4 History

Please see the file `history.txt` for a complete change log of Plananarama.

3 Usage

3.1 Getting started

After you have installed Plananarama, Hollywood will automatically use it if CyberGraphX or Picasso96 are unavailable because your system doesn't have a graphics board. Thus, as soon as Plananarama is installed, all your Hollywood scripts will "automagically" run on palette screens because Hollywood will route them through Plananarama. If you leave everything to Hollywood, your scripts will run in remapping mode. This mode guarantees maximum compatibility but also needs lots of memory and can be slow, depending on what the script does.

It's also possible to write scripts that are specifically designed for Plananarama. This typically allows you to achieve a better performance because you're working within the restrictions of palette-based screens. Scripts specifically designed for Plananarama should request palette mode from Plananarama by setting the `PaletteMode` tag when `@REQUIRE-ing` the plugin, e.g.

```
@REQUIRE "plananarama", {PaletteMode = True}
```

The line above will tell Plananarama to run in palette mode. The difference between palette and remapping mode is that in palette mode your script has direct access to the screen's palette pens. For example, when calling Hollywood's `SetPen()` function in palette mode, you can directly change the color of a screen pen and using Hollywood's `SetPalette()` command you can set a whole new palette. This can be used for several effects like color cycling or fading and since changing screen palette pens is managed by the Amiga's custom chip hardware, the result will be visible instantly with almost no delay.

Furthermore, since you have full control over the screen's palette in palette mode, you can also store all your graphics in a way so that the palette of your images matches the palette of the screen. If that is the case, graphics can be drawn really quickly because no color remapping needs to be done and drawing graphics is just a matter of copying raw pixels. Thus, to get the best performance with Plananarama you should put the plugin in palette mode and then design your script in a way that avoids color remapping as much as possible, e.g. by making all images use the same, global palette. This also means that you shouldn't pass RGB colors when drawing primitives like rectangles, lines, circles etc. but that you should draw using palette pens instead. This is possible by setting the palette mode to `#PALETTE_PEN`, like so:

```
SetPaletteMode(#PALETTEMODE_PEN)
```

Palette mode will also allow you to use hardware sprites which can speed up things further because these sprites can be drawn in next to no time since they're completely handled by the Amiga's custom chip hardware. See Section 3.5 [Hardware sprites], page 11, for details. Another advantage of using palette mode is that your script won't require guigfx.library and render.library. To speed up drawing in palette mode, it's advised to install BlazeWCP though. See Section 1.3 [Requirements], page 2, for details.

Note that in palette mode, Plananarama will always open its own screen (even if the Hollywood script explicitly requests window mode). The reason for this is that full control over the screen palette is only possible in case Plananarama runs on its own screen. When running on Workbench or other screens shared with other applications, palette pens need

to be shared as well which makes things more complicated. See Section 3.4 [Palette mode], page 10, for details.

In remapping mode, which is Plananarama's default mode, Plananarama can run on its own screen or on other screens like Workbench. In case remapping mode is active, Plananarama won't give you access to the screen's palette pens but it will remap the colors of all graphics it draws to match the screen's palette. This is of course much slower than drawing graphics whose colors match the screen's palette in palette mode (see above) but it is very flexible and allows you to make any Hollywood script run on a palette screen as long as you have enough free memory. See Section 3.3 [Remapping mode], page 9, for details.

Note that palette mode requires Hollywood 9 or better. Remapping mode needs at least Hollywood 6.

3.2 Configuring Plananarama

When `@REQUIRE-ing` Plananarama, you can pass the following tags to Hollywood's `@REQUIRE` preprocessor command:

`PaletteMode:`
: This tag can be used to set whether or not Plananarama should run in palette mode. This defaults to `False` which means remapping mode (see above for details). (V2.0)

`NoBlackBackground:`
: If this is set to `True`, Plananarama won't set color 0 to black when opening in fullscreen mode. This is only handled when Plananarama is in remapping mode. In palette mode, Plananarama will use the display's palette. Defaults to `False`. (V2.0)

`SpriteResolution:`
: This tag can be used to force a specific sprite resolution for the hardware sprites created by Plananarama. By default, Plananarama will use the system's sprite resolution which might not be what you want. E.g. if the system's sprite resolution is hires, your sprites will appear in hires as well which might not be what you want. The system's sprite resolution is typically identical with the mouse pointer resolution set in the system's "Pointer" preferences because AmigaOS implements the mouse pointer using a hardware sprite. So if the user has configured a hires pointer here, then all your sprites will use hires by default as well. If you don't want that, set this tag to 1 to force lores sprite resolution. To force hires sprites, set the tag to 2. This tag defaults to 0 which means use the system's sprite resolution. Note that this tag is only ever useful on AGA systems because on ECS systems sprites are always lores. (V2.0)

`DitherMode:`
: When Plananarama is in remapping mode, this can be used to configure the dithering mode. This can be set to `None`, `FS` (the default), `Random`, or `Edd`. Here is a description of the different dither modes:

  `None` No dithering at all.

  `FS` Floyd-Steinberg dithering. This is the default.

`Random` Random dithering. This mode is significantly slower than Floyd-Steinberg dithering.

`Edd` EDD dithering. This mode is faster than Floyd-Steinberg dithering.

This tag is ignored in palette mode.

`DitherAmount:`
When Plananarama is in remapping mode, this can be used to set the dither amount. This must be between 0 and 255. Currently this value is of any use only for the `Random` dither mode. Defaults to 40. This tag is ignored in palette mode.

`AutoDither:`
When Plananarama is in remapping mode, this can be used to enable automatic dithering. If set to `True`, dithering is automatically activated for drawing a particular picture to a particular environment, when the loss of color information would exceed a certain threshold (see below). Defaults to `True`. This tag is ignored in palette mode.

`DitherThreshold:`
When Plananarama is in remapping mode, this can be used to set the threshold for automatic dithering. The lower, the earlier automatic dithering is activated. Useful thresholds range between 10 and 10000. Refer to render.library/RGBArrayDiversityA() for further details. Better you do not use this tag unless you have a good reason to. Let the user customize it with the environment variable `AUTODITHERTHRESHOLD`. Defaults to 250. This tag is ignored in palette mode.

`Precision:`
When Plananarama is in remapping mode, this can be used to set the precision for pen allocations. This can be `Exact`, `Image` (the default), `Icon`, or `GUI`. See graphics.library/ObtainBestPenA() for details. Note that the default precision suffices for almost every application. Pens are obtained in an extremely effective way. You get excellent results even with lower precisions. Commodore's idea with ObtainBestPenA() was to create a fair and effective pen-sharing mechanism, and Plananarama behaves in perfect accordance to this intention. Never use insane patches for ObtainBestPenA(). This tag is ignored in palette mode.

3.3 Remapping mode

By default, Plananarama will run in remapping mode. Remapping mode is the most convenient of Plananarama's display modes because it allows you to simply make any Hollywood script run on all kinds of palette screens. You don't have to adapt your code in any way, it will just work because all graphics are remapped to match the target screen's palette. However, this means lots of work for the CPU so it will be very slow. Also, it means that all images will be stored as RGB graphics which will consume a lot of memory. For example, a 640x480 image will require 1.2 megabytes of memory when stored as RGB but only 300kb of memory when stored as a palette image.

To speed things up, you could try using palette mode instead but this will require you to tailor your script specifically to the constraints of a palette-based display. See Section 3.4 [Palette mode], page 10, for details.

Note that when using Plananarama in remapping mode, it's advised to install FBlit. See Section 1.3 [Requirements], page 2, for details.

3.4 Palette mode

When using palette mode with Plananarama, you can achieve a better performance than in remapping mode but it comes at the cost that you must design your script specifically for Plananarama's palette mode. For example, you must make sure that your Hollywood display is a palette display. Otherwise you obviously won't gain any performance improvement because if the Hollywood display doesn't use a palette but RGB graphics, all graphics still have to be remapped just like in remapping mode.

Here is some example code that sets up a palette display and puts Plananarama in palette mode:

```
@REQUIRE "plananarama", {PaletteMode = True}
@DISPLAY {Palette = #PALETTE_AGA}

SetPaletteMode(#PALETTEMODE_PEN)
SetFillStyle(#FILLCOLOR)
SetDrawPen(2)
Box(#CENTER, #CENTER, 320, 240)
```

The code does several very important steps that are necessary to take full advantage of Plananarama's palette mode: First, it creates a palette display by using the `Palette` tag to assign the inbuilt palette `#PALETTE_AGA` to the display. Alternatively, you could also create a palette display by simply assigning a palette BGPic to it, e.g. like so:

```
@REQUIRE "plananarama", {PaletteMode = True}
@BGPIC 1, "background.iff", {LoadPalette = True}
```

Since we set `LoadPalette` to `True` in the code above, your display will automatically become a palette display because its BGPic is a palette one.

The second very important thing the first code snippet does is calling `SetPaletteMode()` with `#PALETTEMODE_PEN` passed to it. This is very important because if you don't do that, Hollywood will still remap all graphics to your display's palette which is slow. Only by setting palette mode to `#PALETTEMODE_PEN` can you tell Hollywood to not do any remapping but just copy the raw pixels. Of course, this means that if you draw images their palette must match the display palette or you'll get wrong colors.

Finally, the code snippet calls `SetDrawPen()` to set a drawing pen. This step is very important if you want to draw graphics primitives like lines, rectangles, circles, and so on. If the palette mode has been set to `#PALETTEMODE_PEN`, Hollywood functions like `Box()`, `Line()`, `Circle()`, etc. will ignore the RGB color that is passed to them. Instead, they will draw using the pen that has been set using `SetDrawPen()`. This is why the code above will draw a white rectangle and not a black one, even though the color argument in the call to `Box()` defaults to black because it has been left out.

Since we have full control over the hardware color registers, we could now easily turn the white rectangle into a red one by just changing the color of palette pen 2. This can be done like this:

```
SetPen(2, #RED)
```

Then we could smoothly fade out the red rectangle to black by doing something like this:

```
For Local k = 32 To 0 Step -1
   SetPen(2, RGB(255 * (k/32), 0, 0))
   VWait
Next
```

Of course you could also cycle the palette colors and apply a completely new palette using Hollywood's `SetPalette()` function. Lots of things are possible in palette mode.

Another advantage of using palette mode is that your script won't require guigfx.library and render.library. To speed up drawing in palette mode, it's advised to install BlazeWCP though. See Section 1.3 [Requirements], page 2, for details.

3.5 Hardware sprites

Starting with Plananarama 2.0 the plugin also supports Amiga hardware sprites. Since these are managed by the Amiga's custom chipset hardware, they can be drawn extremely efficiently without any performance penalties. However, sprites have always been the Amiga's Achilles heel since they are quite limited in comparison to other systems (especially in comparison to gaming consoles).

Specifically, there are the following limitations when it comes to hardware sprites on the Amiga:

- there are only 8 sprite DMA channels so you can only have a maximum of 8 sprites
- each sprite DMA channel can only handle 4 color graphics
- luckily, two sprite DMA channels can be combined to create a 16 color sprite but this means that if you use 16 color sprites, you can only have 4 of them because each 16 color sprite will occupy two sprite DMA channels
- on OCS/ECS the maximum sprite width is 16 pixels
- on AGA the maximum sprite width is 64 pixels
- there is no maximum sprite height
- the individual sprite DMA channels are tied to certain color registers and one color is always reserved for transparency. See Section 4.1 [planar.CreateSprite], page 13, for details.

Due to all these limitations you won't be able to move mountains with Amiga hardware sprites but if you only need to have a few sprites, they can still be quite useful because they can be drawn so quickly since they are completely handled on the hardware level.

Note that when using hardware sprites you should use Plananarama in palette mode because in remapping mode you won't have control over the screen's palette pens so there is no way to set the sprite colors. See Section 3.4 [Palette mode], page 10, for details.

Also note that you might want to set the `SpriteResolution` tag when using hardware sprites. Otherwise your sprites will use the system's sprite resolution which might not be

what you want. E.g. if the system's sprite resolution is hires, your sprites will appear in hires as well which might not be what you want. The system's sprite resolution is typically identical with the mouse pointer resolution set in the system's "Pointer" preferences because AmigaOS implements the mouse pointer using a hardware sprite. So if the user has configured a hires pointer here, then all your sprites will use hires by default as well. If you don't want that, set the `SpriteResolution` tag to 1 to force lores sprites. See Section 3.2 [Configuring Plananarama], page 8, for details. Note that the `SpriteResolution` tag is only really needed on AGA systems because on ECS systems sprites are always lores.

3.6 RapaGUI and MUI Royale support

Plananarama also supports the RapaGUI and MUI Royale plugins. When Plananarama is installed, both RapaGUI and MUI Royale will automatically run on palette screens as well. Note, however, that you must not use Plananarama's palette mode when using RapaGUI and MUI Royale. With those plugins, Plananarama must always be used in remapping mode.

4 Function reference

4.1 planar.CreateSprite

NAME

planar.CreateSprite – create hardware sprite from brush (V2.0)

SYNOPSIS

```
[id] = planar.CreateSprite(id, brushid)
```

FUNCTION

This function converts the brush specified by `brushid` into a hardware sprite and assigns the identifier `id` to it. If you specify `Nil` in the `id` argument, `planar.CreateSprite()` will automatically choose a vacant identifier for this sprite and return it to you.

Note that the brush you pass to this function must respect the Amiga hardware sprite limitations. This means that it must adhere to the following rules:

- it must be a palette brush
- on OCS/ECS systems the maximum sprite width is 16 pixels
- on AGA systems the maximum sprite width is 64 pixels
- the palette brush must use either 4 or 16 colors

Also note that the Amiga hardware supports only 8 sprite DMA channels. Each channel can have a 4 color sprite. The 8 sprite DMA channels are tied to the following color registers:

- Channels 0 and 1: Color registers 16 to 19 (color 16 is transparent). Note that channel 0 is reserved for use by Intuition for the mouse pointer.
- Channels 2 and 3: Color registers 20 to 23 (color 20 is transparent).
- Channels 4 and 5: Color registers 24 to 27 (color 24 is transparent).
- Channels 6 and 7: Color registers 28 to 31 (color 28 is transparent).

Two channels can be combined to create a 16 color sprite. This means that if you use 16 color sprites, you can only have 4 instead of 8 because a 16 color sprite will block two sprite DMA channels. 16 color sprites are tied to the color registers 16 to 31 (color 16 is transparent).

Note that `planar.CreateSprite()` won't map the sprite to a sprite DMA channel immediately. This is the job of `planar.MapSprite()`. Thus, you can create more hardware sprites with `planar.CreateSprite()` than there are sprite DMA channels. This is useful if you want to animate sprites, for example. In that case, you could first convert all the animation frames to hardware sprites using `planar.CreateSprite()` and then map and unmap the single animation frames before/after displaying them with `planar.MoveSprite()`.

INPUTS

`id`	identifier for the hardware sprite or `Nil` for auto id selection
`brushid`	identifier of the palette brush to be converted to a hardware sprite

RESULTS

`id` optional: identifier of the hardware sprite; will only be returned if you pass `Nil` as argument 1 (see above)

EXAMPLE

```
@REQUIRE "plananarama", {PaletteMode = True}
@DISPLAY {Palette = #PALETTE_AGA}
SetPaletteMode(#PALETTEMODE_PEN)
SetFillStyle(#FILLCOLOR)
CreateBrush(1, 64, 64, {Palette = #PALETTE_GRAY4})
SelectBrush(1)
For Local k = 0 To 2
   SetDrawPen(k + 1)
   Box(k * 21, 0, 21, 64)
Next
EndSelect
planar.CreateSprite(1, 1)
planar.MapSprite(1)
Repeat
   planar.MoveSprite(1, MouseX(), MouseY())
   planar.VWait()
Forever
```

The code above will create a 4 color 64x64 sprite, map it to a sprite DMA channel and then move it to where the mouse pointer is. Note that it doesn't matter that we pass `#PALETTE_GRAY4` to `CreateBrush()` since the hardware sprite will use the screen's palette so we just use `#PALETTE_GRAY4` as a dummy to tell `CreateBrush()` to give us a 16 color sprite.

4.2 planar.FreeSprite

NAME

planar.FreeSprite – free hardware sprite (V2.0)

SYNOPSIS

```
planar.FreeSprite(id)
```

FUNCTION

This function will free the hardware sprite specified by `id`. If the sprite is currently mapped to a sprite DMA channel, it will be unmapped before it is freed.

INPUTS

`id` identifier of hardware sprite to free

4.3 planar.GetSpriteType

NAME

planar.GetSpriteType – get hardware sprite object type (V2.0)

SYNOPSIS

```
type = planar.GetSpriteType()
```

FUNCTION

This function returns the hardware sprite object type registered by Plananarama. You can then pass this object type to Hollywood's `GetAttribute()` function to query the following attributes of hardware sprites:

`#ATTRWIDTH:`
: The hardware sprite width.

`#ATTRHEIGHT:`
: The hardware sprite height.

`#ATTRDEPTH:`
: The hardware sprite depth.

`#ATTRXPOS:`
: The hardware sprite x position.

`#ATTRYPOS:`
: The hardware sprite y position.

`#ATTRSTATE:`
: The sprite DMA channel the hardware sprite has been mapped to. For unmapped hardware sprites, this will be -1.

INPUTS

none

RESULTS

`type` hardware sprite object type registered by Plananarama

EXAMPLE

```
DMASPRITE_TYPE = planar.GetSpriteType()
w = GetAttribute(DMASPRITE_TYPE, 1, #ATTRWIDTH)
h = GetAttribute(DMASPRITE_TYPE, 1, #ATTRHEIGHT)
```

The code above queries the width and height of hardware sprite 1.

4.4 planar.HaveAGA

NAME

planar.HaveAGA – check if AGA chipset is present (V2.0)

SYNOPSIS

```
ok = planar.HaveAGA()
```

FUNCTION

This function returns `True` if the AGA chipset is present, `False` otherwise.

INPUTS

none

RESULTS

`ok` `True` or `False` to indicate if the AGA chipset is present

4.5 planar.MapSprite

NAME

planar.MapSprite – map hardware sprite to DMA channel (V2.0)

SYNOPSIS

```
planar.MapSprite(id[, t])
```

FUNCTION

This function maps the hardware sprite specified by `id` to a free sprite DMA channel. The hardware sprite specified by `id` must have been created using `planar.CreateSprite()` first.

Further parameters can be specified in the optional table argument. The following tags are currently recognized:

`Channel:` By default, `planar.MapSprite()` will choose a sprite DMA channel automatically. If you want to use a specific sprite DMA channel, you can set it using this tag. This must be a number between 1 and 7 (channel 0 is reserved for use by Intuition for the mouse pointer sprite). Note that if the sprite uses 16 colors you can only use channels 2, 4 or 6 because 16 color sprites occupy two adjacent sprite DMA channels.

`Display:` This can be set to the identifier of a Hollywood display the sprite should appear on. This is normally not necessary and the sprite will just use the current Hollywood display.

Note that if the sprite DMA channel is automatically chosen by `planar.MapSprite()`, you can query the `#ATTRSTATE` attribute to find out the DMA channel your sprite has been mapped to after calling `planar.MapSprite()`.

To unmap a hardware sprite from a DMA channel, use `planar.UnmapSprite()` See Section 4.7 [planar.UnmapSprite], page 17, for details.

INPUTS

`id` identifier of hardware sprite to map to DMA channel

`t` optional: table containing further arguments (see above)

EXAMPLE

See Section 4.1 [planar.CreateSprite], page 13.

4.6 planar.MoveSprite

NAME

planar.MoveSprite – set sprite position (V2.0)

SYNOPSIS

```
planar.MoveSprite(id, x, y)
```

FUNCTION
This function moves the hardware sprite specified by `id` to the position specified by `x` and `y`. This is only possible for sprites that have been mapped to a sprite DMA channel before so you need to call `planar.MapSprite()` before using this function.

INPUTS

`id`	identifier of hardware sprite to move
`x`	desired new x position
`y`	desired new y position

EXAMPLE
See Section 4.1 [planar.CreateSprite], page 13.

4.7 planar.UnmapSprite

NAME
planar.UnmapSprite – unmap hardware sprite from DMA channel (V2.0)

SYNOPSIS
```
planar.UnmapSprite(id)
```

FUNCTION
This function unmaps the hardware sprite specified by `id` from the sprite DMA channel it is currently attached to. The sprite must have been mapped to a DMA channel using `planar.MapSprite()` before.

INPUTS

`id`	identifier of hardware sprite to unmap from DMA channel

4.8 planar.VWait

NAME
planar.VWait – wait for vertical blank interrupt (V2.0)

SYNOPSIS
```
planar.VWait()
```

FUNCTION
This will wait for the vertical blank interrupt. When using Plananarama, it's better to use this function instead of Hollywood's own `VWait()` function because Plananarama's `planar.VWait()` function operates on a level that is closer to the Amiga's custom chip hardware.

INPUTS
none

Index