GFXStuff

COLLABORATORS			
	TITLE : GFXStuff		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

GFXStuff

1.1 GFXStuff V 1.3

GFXStuff V1.3

Table of contents

1.2 GFXStuff V1.3

GFXStuff V1.3

The History of GfxStuff V1.3

1.0 : First Release (EGS only)

1.1 : Fixed a bug in RtgScreenAtFront (EGS only)

1.2 : Rewrote LoadRGBRtg EGS-Code, so it works now

100% the same like LoadRGBRtg of rtgmaster.library

Added support for CyberGFX (up to now only OpenLibs,

CloseLibs and GetScreenmodes...)

Added support for the EGS System for the Commodore A2410 Board

1.3 : changed rs_Locks from Byte to Word length.

changed LoadRGBRtg another time to adapt it to the

way this is done in the latest version of rtg.library.

(now 32 Bit color values are used...)

Updated includes.

Implemented CyberGraphX stuff (no *real* Double Buffering,

as this currently is not possible on CyberGraphX, only a "MoveScreen")
Converted GFXStuff to Amigaguide Format
Changed rs_... to rsEGS_... and rsCGX_...

1.3 GFXStuff V1.3

GFXStuff V1.3
What is this / Copyright
This is a collection of code that helps to support
Graphics Boards for Amiga games and Amiga demos especially.
It contains well documented sources, that explain how to use
the libraries of the WB-Emulations and it even shows some tricks,
how you can access the graphics hardware as low-level as possible.
The function calls are comaptible with the upcoming rtgmaster.library
(in fact, this is stuff i myself wrote for rtgmaster.library), so
you are quite easily able to switch to rtgmaster.library later.
The Copyright about this stuff remains MINE. So if you want to use
this code in your game/demo, you will have to add a notice to the
documentation of your code that you use these code done by me.
If you want, you may pay me for this code or give me a free copy
of your game :))) but you do not NEED to. But it would be fine ...
Sorry, if this document has mistakes... i did a BIG deal testing
the functions, but a mistake may be in it anytimes... (but i hope
it is bug-free... and as to my function tests, it seems to be bugfree...
i used this functions in a project of my own...) But IF you find a
bug anyways, please send a bug-report to haeuser@tick.informatik.uni-stuttgart.de

1.4 GFXStuff V1.3

GFXStuff V1.3
The Code
Following is the documented source code for the different functions
in assembly language. Some of the functions need egs.library,
graphics.library, cybergraphics.library, dos.library, intuition.library
and/or utility.library to be opened. As Library bases the functions
use EgsBase, GfxBase, CGXBase, DosBase, IntBase and UtilityBase, so be
sure to have those variables set to valid values (of course egs.library
is only needed for EGS, cybergraphics.library only for CyberGraphX).
The EGS code runs with OS 2.0 and upwards, the CyberGraphX code runs

(like CyberGraphX itself) with OS 3.0 and upwards.
The needed libraries are NOT ALL opened and closed using the Initialisation
and Close functions that are provided by this code. Only egs.library
and cybergraphics.library are handled there. For the rest, you will
have to provide your own code. :)
The Egs OpenLib opens ONLY egs.library, and the CyberGraphX one ONLY
cybergraphics.library !!!
The functions basically will restore a3-a7 and d3-d7 to the values
they contained BEFORE the function call.
This code does not CARE at all, if you have enough Video RAM on your
Board to open a 4096x4096 Triple Buffering 24 Bit Screen or something
like that, so be CAREFUL :)
Also used in the code are the following definitions :
DO NOT EXPECT ANY OF THE USED DATA STRUCTURES NOT TO CHANGE IN
RTGMASTER.LIBRARY !!! Use the provided interfaces, do NOT
look at for example the RtgScreen structures DIRECTLY !!!
_LVOOpenLibrary EQU -552
_LVOCloseLibrary EQU -414
_LVOAllocMem EQU -198
_LVOFreeMem EQU -210
CALLSYS MACRO
jsr _LVO\1(a6)
ENDM
XLIB MACRO
XREF _LVO\1
ENDM
To get information about EGS, click here
To get information about CyberGraphX, click here

1.5 GFXStuff V1.3

GFXStuff V1.3
EGS Support
OpenLibs is some initialization code, that opens
the needed libraries and additional checks,
if an EGS System using an EGS Graphics Board
is provided (it does not start the EGS Support
code if an EGS system running on ECS/AGA is
provided). This system does not react, if only the libraries are

installed. It only reacts, if EGS is correctly installed, otherwise
it fails.
GetScreenmodes gets you a list of the EGS Screenmodes. It is
explained there which format the Screenmodestructure
has.
FreeScreenmodes frees the Screenmodelist again and
returns the memory used by it to the system.
SwitchScreens performs Double and/or Triplebuffering.
GetBufAdr returns you the base address of the Video RAM.
As this is not REALLY possible under EGS, it returns the base address of
the lowest possible video driver of EGS, you can write directly to it, as
IF this WOULD be the base address of the Video RAM and it is about nearly
100% as fast as a direct access to the Video RAM using CyberGraphX on the
same system.
LockRtgScreen locks one of our screens for private usage
and returns the base address of the currently displayed buffer (same note
as for GetBufAdr here...)
UnlockRtgScreen unlocks a screen again.
LoadRGBRtg is a LoadRGB32 clone and works exactly
like this OS 3.0 colortable setting function. It is only of use for
screens with 8 Bit or fewer colors. For 15/16/24 Bit direct color
access is used.
OpenRtgScreen is the function that IN FACT opens
one of our screens.
CloseRtgScreen closes an RtgScreen again.
ScreenAtFront finds out, if an RtgScreen is at
front currently. Very useful for a game that runs in Multitasking,
to stop action as soon as the user clicks the game to back, and
to reactivate it again later.
GetRtgScreenData finds out some useful stuff
about an RtgScreen, for example how the Video RAM is organized.
CloseLibs closes the Rtg System.
BlitRtg is used for blitting. The function waits
for the blitter before returning.
Data structures lists and explains the data
structures (EGS ones and my own ones) used by this code.
These EGS functions support 8 Bit and 24 Bit chunky EGS Screenmodes.
1 Bit Screens are supported too, using Planar EGS Screenmodes.
NOTE : ANY Reference to RtgScreen refers to RtgScreenEGS in fact !!!

1.6 GFXStuff V1.3

GFXStuff V1.3

OpenLibs EGS

This function examines the System, if there is an EGS running on
A GRAPHICS BOARD. If not, it returns 0. If this function does not fail,
it opens egs.library. This function (and so the rest of this whole
EGS Stuff) will work on : EGS Spectrum, EGS Piccolo, EGS Piccolo SD64,
EGS Rainbow 3, EGS G110, A2410 with EGS driver installed. It *may*
work on EGS Rainbow 2 and EGS Graffity (do not confuse with the Graffiti,
which is something else) too, but i do not know, as i have no documentation
about these two boards available.
You SHOULD NOT call other EGS functions of this package, if OpenLibs failed.
This function has no parameters and returns a Boolean in d0 (and it sets
EgsBase).
OpenLibs:

```
movem.l a0-a6/d0-d7,-(sp)
move.l #0,EgsBase
lea EgsName,a1
moveq #6,d0
move.l $4,a6
CALLSYS OpenLibrary
cmp.l #0,d0
beq .Exit
move.l d0,EgsBase
move.l EgsBase,a6
CALLSYS E_LockEGSVideo
CALLSYS E_GetHardInfo
move.l d0,a3
CALLSYS E_UnlockEGSVideo
move.l #1,d0
move.l ehi_Drivers(a3),a0
lea edd_Node(a0),a0
move.l LN_SUCC(a0),a0
cmp.l #0,LN_SUCC(a0)
beq .Exit
cmp.l #1,d0
bne.s .LookRB3a
move.l LN_NAME(a0),a1
```

```
cmp.l #'PICO',(a1)
bne.s .LookRB3a
bsr .Found
.LookRB3a:
cmp.l #1,d0
bne.s .LookRB3b
move.l LN_NAME(a0),a1
cmp.l #'RB3a',(a1)
bne.s .LookRB3b
bsr .Found
.LookRB3b:
cmp.l #1,d0
bne.s .LookA2410
move.l LN_NAME(a0),a1
cmp.l #'RB3b',(a1)
bne.s .LookA2410
bsr .Found
.LookA2410:
cmp.l #1,d0
bne.s .LookRB2a
move.l LN_NAME(a0),a1
cmp.l #'A241',(a1)
bne.s .LookRB2a
bsr .Found
.LookRB2a:
cmp.l #1,d0
bne.s .LookRB2b
move.l LN_NAME(a0),a1
cmp.l #'RB2a',(a1)
bne.s .LookRB2b
bsr .Found
.LookRB2b:
cmp.l #1,d0
bne.s .LookG110
move.l LN_NAME(a0),a1
cmp.l #'RB2b',(a1)
bne.s .LookG110
bsr .Found
.LookG110:
```

```
cmp.l #1,d0
bne.s .LookGVP
move.l LN_NAME(a0),a1
cmp.l #'G110',(a1)
bne.s .LookGVP
bsr .Found
.LookGVP:
cmp.l #1,d0
bne.s .LookLoop
move.l LN_NAME(a0),a1
cmp.l #'LEGS',(a1)
bne.s .LookLoop
bsr .Found
.LookLoop:
cmp.l #0,d0
beq .QuitLoop
cmp.l #0,LN_SUCC(a0)
beq .Exit
lea edd_Node(a0),a0
move.l LN_SUCC(a0),a0
cmp.l #1,d0
bne.s .SearchRB3a
move.l LN_NAME(a0),a1
cmp.l #'PICO',(a1)
bne.s .SearchRB3a
bsr .Found
.SearchRB3a:
cmp.l #1,d0
bne.s .SearchRB3b
move.l LN_NAME(a0),a1
cmp.l #'RB3a',(a1)
bne.s .SearchRB3b
bsr .Found
.SearchRB3b:
cmp.l #1,d0
bne.s .SearchA2410
move.l LN_NAME(a0),a1
cmp.l #'RB3b',(a1)
bne.s .SearchA2410
```

```
bsr.s .Found
.SearchA2410:
cmp.l #1,d0
bne.s .SearchRB2a
move.l LN_NAME(a0),a1
cmp.l #'A241',(a1)
bne.s .SearchRB2a
bsr .Found
.SearchRB2a:
cmp.l #1,d0
bne.s .SearchRB2b
move.l LN_NAME(a0),a1
cmp.l #'RB2a',(a1)
bne.s .SearchRB2b
bsr .Found
.SearchRB2b:
cmp.l #1,d0
bne.s .SearchG110
move.l LN_NAME(a0),a1
cmp.l #'RB2b',(a1)
bne.s .SearchG110
bsr .Found
.SearchG110:
cmp.l #1,d0
bne.s .SearchGVP
move.l LN_NAME(a0),a1
cmp.l #'G110',(a1)
bne.s .SearchGVP
bsr.s .Found
.SearchGVP:
cmp.l #1,d0
bne .LookLoop
move.l LN_NAME(a0),a1
cmp.l #'LEGS',(a1)
bne .LookLoop
bsr.s .Found
bra .LookLoop
.Found:
move.l #0,d0
```

```
rts
.QuitLoop:
movem.l (sp)+,a0-a6/d0-d7
move.l #1,d0
rts
.Exit:
movem.l (sp)+,a0-a6/d0-d7
.CloseEGS:
move.l $4,a6
move.l EgsBase,a1
CALLSYS CloseLibrary
move.l #0,d0
rts
EgsBase: dc.l 0
EgsName: dc.b 'egs.library',0
```

1.7 GFXStuff V1.3

```
GFXStuff V1.3
GetScreenModes EGS
This function gets the Screenmodes in the format of EGS (that
will be explained in the data structures chapter). It returns
a pointer to the EGS Screenmode system list, so you SHOULD NOT
CHANGE ANYTHING IN THERE, else you will change the already
installed Screenmodes in the system. This function has no
parameters and returns the Screenmodelist in d0.
GetScreenmodes:
movem.l a3/a6,-(sp)
move.l EgsBase,a6
CALLSYS E_LockEGSVideo
CALLSYS E_GetHardInfo
move.l d0,a3
CALLSYS E_UnlockEGSVideo
move.l ehi_Modes(a3),d0
movem.l (sp)+,a3/a6
rts
```

1.8 GFXStuff V1.3

GFXStuff V1.3

FreeScreenModes EGS

As you only operate with a pointer to the EGS System Screenmode list, you wont' have to free anything and this call is :

FreeScreenmodes:

rts

1.9 GFXStuff V1.3

GFXStuff V1.3

SwitchScreens EGS

SwitchScreens performs Double- and Triplebuffering. It is assumed that a0 contains a RtgScreen Screen Handle of an already opened RtgScreen, and that d0 contains the Buffer number between 0 and 2. Here 0 is the Buffer that is displayed after you opened the Screen using OpenRtgScreen.

SwitchScreens:

```
movem.l a2/a3/a6/d7,-(sp)
cmp.l rsEGS_ActiveMap(a0),d0
beq .Exit
move.l rsEGS_NumBuf(a0),d1
cmp.l d1,d0
bge .Exit
cmp.l #0,d0
beq.s .BufferOK
cmp.l #1,d0
beq.s .BufferOK
cmp.l #2,d0
bne .Exit
.BufferOK:
move.l a0,a3
move.l EgsBase,a6
move.l rsEGS_MyScreen(a3),a0
cmp.l #0,d0
bne.s .NotA
move.l rsEGS_MapA(a3),a1
bra .NotC
```

```
.NotA:
cmp.l #1,d0
bne.s .NotB
move.l rsEGS_MapB(a3),a1
bra .NotC
.NotB:
move.l rsEGS_MapC(a3),a1
.NotC:
move.l d0,d7
move.l a1,rsEGS_FrontMap(a3)
CALLSYS E_FlipMap
move.l d7,rsEGS_ActiveMap(a3)
.Exit:
movem.l (sp)+,a2/a3/a6/d7
rts
```

1.10 GFXStuff V1.3

```
GFXStuff V1.3
GetBufAdr EGS
This function assumes the RtgScreen Screen Handle of
an already opened RtgScreen being in a0, the number of the
buffer to be examined between 0 and 2 in d0 and it will return
the base address in d0.
GetBufAdr:
cmp.l #0,d0
beq.s .MapA
cmp.l #1,d0
beq.s .MapB
move.l rsEGS_MapC(a0),a0
bra .FindPlane
.MapA:
move.l rsEGS_MapA(a0),a0
bra .FindPlane
.MapB:
move.l rsEGS_MapB(a0),a0
.FindPlane:
move.l ebm_Plane(a0),d0
rts
```

1.11 GFXStuff V1.3

GFXStuff V1.3

LockRtgScreen EGS

This function assumes the RtgScreen Screen Handle of an already
opened RtgScreen in a0, and it returns the base address of the
first (= at the time of screen opening displayed buffer, the
Buffer with number 0) Buffer in d0. You should call UnlockRtgScreen
before closing the Screen one time for each call of LockRtgScreen.
You may nest these calls up to 65535 times.
You should NOT remove this Delay in this function, the system
NEEDS this time to lock the Screen, believe me ... and locking the
Screen is not speed critical anyways, as this i susually done
directly after opening the screen...

```
LockRtgScreen:
movem.l a4/a6,-(sp)
move.l rsEGS_MapA(a0),a1
add.b #1,ebm_Lock(a1)
add.w #1,rsEGS_Locks(a0)
move.l rsEGS_MapB(a0),a1
cmp.l #0,a1
beq.s .NoB
add.b #1,ebm_Lock(a1)
.NoB:
move.l rsEGS_MapC(a0),a1
cmp.l #0,a1
beq.s .NoC
add.b #1,ebm_Lock(a1)
.NoC:
move.l a0,a4
move.l DosBase,a6
move.l #5,d1
jsr -198(a6)
move.l a4,a0
movem.l (sp)+,a4/a6
move.l rsEGS_MapA(a0),a0
move.l ebm_Plane(a0),d0
rts
```

1.12 GFXStuff V1.3

GFXStuff V1.3

UnlockRtgScreen EGS

This function does the opposite to LockRtgScreen.

It assumes the RtgScreen Screen Handle of the Screen

to be unlocked in a0. It does not need some time to

work like the LockRtgScreen call.

```
UnlockRtgScreen:
cmp.w #0,rsEGS_Locks(a0)
beq.s .Exit
move.l rsEGS_MapA(a0),a1
sub.b #1,ebm_Lock(a1)
move.l rsEGS_MapB(a0),a1
cmp.l #0,a1
beq .NoB
sub.b #1,ebm_Lock(a1)
.NoB:
move.l rsEGS_MapC(a0),a1
cmp.l #0,a1
beq .NoC
sub.b #1,ebm_Lock(a1)
.NoC:
sub.w #1,rsEGS_Locks(a0)
.Exit:
rts
```

1.13 GFXStuff V1.3

GFXStuff V1.3

LoadRGBRtg EGS

This function behaves similar to LoadRGB32 of graphics.library,

that is, it sets a color table for a 256 color EGS Screen, in

this case.

An example for such a color table:

```
dc.w 2,0 ; set two colors starting with color 0
dc.l $FFFFFFFF,0,0 ; set color 0 to Red
dc.l 0,$C0C0C0C0,0 ; set color 1 to blue (color value $0C)
dc.w 1,3 ; another 1 color(s), starting with color 3
```

```
dc.l 0,0,$FFFFFFFF ; setting it to green
dc.w 0 ; finished
```

The function expects the RtgScreen Screen handle in a0, and a pointer to the color table in a1.

```
LoadRGBRtg:
movem.l d3-d7/a2-a6,-(sp)
move.l a6,a5
cmp.l #4,rsEGS_Bytes(a0)
beq .Exit
cmp.l #0,a1
beq .Exit
.NewStart:
move.l a0,a3
move.l rsEGS_Bytes(a0),d0
cmp.l #0,d0
beq .OneBit
cmp.l #1,d0
beq .OneByte
bra .Exit
.OneBit:
clr.l d3
clr.l d4
move.w (a1)+,d3
move.w (a1)+,d4
cmp.l #0,d3
beq .Exit
cmp.l #3,d3
bge .Exit
cmp.l #0,d4
beq .StartZero
cmp.l #1,d4
beq .StartOne
bra .Exit
.StartZero:
move.l a3,a0
move.l rsEGS_MyScreen(a0),a0
move.l #0,d0
move.l (a1)+,d1
move.l (a1)+,d2
```

```
move.l (a1)+,d3
swap d1
swap d2
swap d3
and.l #$ffff,d1
and.l #$ffff,d2
and.l #$ffff,d3
divu #257,d1
divu #257,d2
divu #257,d3
and.l #$ffff,d1
and.l #$ffff,d2
and.l #$ffff,d3
move.l a1,d7
move.l EgsBase(a5),a6
CALLSYS E_SetRGB8
move.l d7,a1
cmp.l #1,d3
beq .Exit
.StartOne:
move.l a3,a0
move.l rsEGS_MyScreen(a0),a0
move.l #1,d0
move.l (a1)+,d1
move.l (a1)+,d2
move.l (a1)+,d3
swap d1
swap d2
swap d3
and.l #$ffff,d1
and.l #$ffff,d2
and.l #$ffff,d3
divu #257,d1
divu #257,d2
divu #257,d3
and.l #$ffff,d1
and.l #$ffff,d2
and.l #$ffff,d3
CALLSYS E_SetRGB8
```

```
bra .Exit
.OneByte:
move.l a1,a4
clr.l d1
clr.l d0
move.w (a4)+,d1
cmp.l #0,d1
beq .Exit
cmp.l #257,d1
bge .Exit
move.w (a4)+,d0
cmp.l #256,d0
bge .Exit
move.l d1,d2
sub.l #1,d1
move.l d1,d6
move.l d0,d5
move.l EgsBase(a5),a6
.Loop:
move.l (a4)+,d1
move.l (a4)+,d2
move.l (a4)+,d3
swap d1
swap d2
swap d3
and.l #$ffff,d1
and.l #$ffff,d2
and.l #$ffff,d3
divu #257,d1
divu #257,d2
divu #257,d3
and.l #$ffff,d1
and.l #$ffff,d2
and.l #$ffff,d3
move.l d5,d0
move.l a3,a0
move.l rsEGS_MyScreen(a0),a0
CALLSYS E_SetRGB8
add.l #1,d5
```

```
dbra d6,.Loop
move.w (a4),d0
move.l a4,a1
move.l a3,a0
cmp.w #0,d0
bne .NewStart
.Exit:
move.l #0,d0
movem.l (sp)+,d3-d7/a2-a6
rts
```

1.14 GFXStuff V1.3

```
GFXStuff V1.3
OpenRtgScreen EGS
This function opens IN FACT an RtgScreen. It returns a RtgScreen
Screen Handle in d0. In a2 it assumes a ScreenQuery structure
to get the information which Width, Height, Depth... you need.
The way this function will be called, will change with
rtgmaster.library, as this library will handle this using a
Screenmoderequester.
OpenRtgScreen:
movem.l d2-d7/a2-a6,-(sp)
cmp.l #0,EgsBase
beq .Exit
move.l a2,a4
move.l sq_Buffers(a4),d4
cmp.l #0,d4
beq .Exit
move.l #3,d0
cmp.l d0,d4
bgt .Exit
.BufferDone:
move.w sq_Depth(a4),d0
move.w sq_Width,-(sp)
move.w sq_Height,-(sp)
cmp.l #24,d0
beq .TrueColor
cmp.l #8,d0
```

```
beq .Chunky
bra .Mask
.TrueColor:
move.l #E_PIXELMAP,d5
move.w #24,d6
bra.s .Continue
.Chunky:
move.l #E_PIXELMAP,d5
move.w #8,d6
bra.s .Continue
.Mask:
move.l #E_BITPLANEMAP,d5
move.w #1,d6
.Continue:
move.l #rsEGS_SIZEOF,d0
move.l #MEMF_CLEAR,d1
move.l $4,a6
CALLSYS AllocMem
cmp.l #0,d0
beq .Error1
move.l d0,a3
move.l EgsBase,a6
move.l #ens_SIZEOF,d0
move.l #MEMF_CLEAR,d1
move.l $4,a6
CALLSYS AllocMem
cmp.l #0,d0
beq .Error3
move.l d0,a0
move.l sq_ScreenMode(a4),a1
move.l a1,ens_Mode(a0)
move.w d6,ens_Depth(a0)
move.w #0,ens_Pad_1(a0)
move.l #0,ens_Colors(a0)
move.l #0,ens_Map(a0)
move.l #0,ens_Flags(a0)
move.l #0,ens_Flags(a0)
move.l #0,ens_Mouse(a0)
move.l sq_EdcmpFlags(a4),d0
```

```
move.l d0,ens_EdcmpFlags(a0)
move.l sq_Port(a4),d0
move.l d0,ens_Port(a0)
move.l EgsBase,a6
move.l a0,d7
CALLSYS E_OpenScreen
cmp.l #0,d0
beq .Error4
move.l d0,rsEGS_MyScreen(a3)
move.l d7,a1
move.l #ens_SIZEOF,d0
move.l $4,a6
CALLSYS FreeMem
move.l #0,rsEGS_ActiveMap(a3)
cmp.l #2,d4
bge.s .DBuff
move.l #0,rsEGS_MapB(a3)
move.l #0,rsEGS_MapC(a3)
tst.l (sp)+
bra .NoDBuff
.AnotherLabel:
sub.l d0,d0
sub.l d1,d1
move.w (sp)+,d1
move.w (sp)+,d0
sub.l d2,d2
move.w d6,d2
move.l d5,d3
move.l #0,d4
add.l #E_EB_DISPLAYABLE,d4
add.l #E_EB_BLITABLE,d4
add.l #E_EB_SWAPABLE,d4
add.l #E_EB_CLEARMAP,d4
move.l rsEGS_MyScreen(a3),a0
move.l esc_Map(a0),a0
move.l EgsBase,a6
CALLSYS E_AllocBitMap
cmp.l #0,d0
beq .Error5
```

```
move.l d0,rsEGS_MapC(a3)
move.l d0,a0
CALLSYS E_ClearBitMap
bra .NoDBuff
.DBuff:
move.w (sp)+,d1
move.w (sp)+,d0
sub.l d2,d2
move.w d6,d2
move.l d5,d3
move.l d4,-(sp)
move.l #0,d4
add.l #E_EB_DISPLAYABLE,d4
add.l #E_EB_BLITABLE,d4
add.l #E_EB_SWAPABLE,d4
add.l #E_EB_CLEARMAP,d4
move.l rsEGS_MyScreen(a3),a0
move.l esc_Map(a0),a0
move.l EgsBase,a6
move.w d0,-(sp)
move.w d1,-(sp)
CALLSYS E_AllocBitMap
move.w (sp)+,d3
move.w (sp)+,d2
move.l (sp)+,d4
cmp.l #0,d0
beq .Error5
move.w d2,-(sp)
move.w d3,-(sp)
move.l d0,rsEGS_MapB(a3)
move.l d0,a0
CALLSYS E_ClearBitMap
cmp.l #3,d4
beq .AnotherLabel
move.l #0,rsEGS_MapC(a3)
tst.l (sp)+
.NoDBuff:
move.l EgsBase,a6
move.l rsEGS_MyScreen(a3),a0
```

```
move.l esc_Map(a0),a0
move.l a0,rsEGS_MapA(a3)
move.l a0,rsEGS_FrontMap(a3)
CALLSYS E_ClearBitMap
move.l rsEGS_MapA(a3),a0
move.w ebm_Width(a0),d0
ext.l d0
move.l d0,rsEGS_Width(a3)
move.b ebm_Depth(a0),d0
move.l d5,rsEGS_Type(a3)
cmp.b #8,d0
beq.s .OneByte
cmp.b #1,d0
beq.s .OneBit
move.l #4,rsEGS_Bytes(a3)
bra.s .ALabel
.OneBit:
move.l #0,rsEGS_Bytes(a3)
bra .ALabel
.NoB:
move.l rsEGS_MapC(a3),d0
cmp.l #0,d0
bne .NoC
move.l #2,rsEGS_NumBuf(a3)
bra .SetLock
.NoC:
move.l #3,rsEGS_NumBuf(a3)
bra .SetLock
.OneByte:
move.l #1,rsEGS_Bytes(a3)
.ALabel:
move.l rsEGS_MapB(a3),d0
cmp.l #0,d0
bne .NoB
move.l #1,rsEGS_NumBuf(a3)
.SetLock:
move.w #0,rsEGS_Locks(a3)
move.l a3,d0
movem.l (sp)+,d2-d7/a2-a6
```

```
rts
.Exit:
move.l #0,d0
movem.l (sp)+,d2-d7/a2-a6
rts
.Error1:
tst.l (sp)+
.Raus:
movem.l (sp)+,d2-d7/a2-a6
move.l #0,d0
rts
.Error3:
tst.l (sp)+
move.l a3,a1
move.l #rsEGS_SIZEOF,d0
CALLSYS FreeMem
move.l (sp)+,a0
move.l EgsBase,a6
CALLSYS E_DisposeBitMap
movem.l (sp)+,d2-d7/a2-a6
move.l #0,d0
rts
.Error4:
tst.l (sp)+
move.l ens_Map(a4),a0
move.l EgsBase,a6
CALLSYS E_DisposeBitMap
move.l a4,a1
move.l #ens_SIZEOF,d0
move.l $4,a6
CALLSYS FreeMem
move.l a3,a1
move.l #rsEGS_SIZEOF,d0
CALLSYS FreeMem
movem.l (sp)+,d2-d7/a2-a6
move.l #0,d0
rts
.Error5:
move.l rsEGS_MyScreen(a3),a0
```

```
CALLSYS E_CloseScreen
move.l rsEGS_MapC(a3),a0
cmp.l #0,a0
beq .Error6
CALLSYS E_DisposeBitMap
.Error6:
move.l a3,a1
move.l #rsEGS_SIZEOF,d0
move.l $4,a6
CALLSYS FreeMem
movem.l (sp)+,d2-d7/a2-a6
move.l #0,d0
rts
.Error2:
tst.l (sp)+
tst.l (sp)+
move.l a3,a1
move.l #rsEGS_SIZEOF,d0
CALLSYS FreeMem
movem.l (sp)+,d2-d7/a2-a6
move.l #0,d0
rts
```

1.15 GFXStuff V1.3

```
GFXStuff V1.3
CloseRtgScreen EGS
This function closes an RtgScreen again. It assumes it RtgScreen
handle in a0. You do not need to change back to Buffer before
closing the Screen, the function handles this itself. :)
CloseRtgScreen:
movem.l a2/a3/a6,-(sp)
move.l a0,a3
move.l rsEGS_ActiveMap(a0),d0
cmp.l #0,d0
beq.s .NoFlip
move.l rsEGS_MyScreen(a0),a0
move.l EgsBase,a6
CALLSYS E_WaitTOF
```

```
move.l a3,a0
move.l rsEGS_MapA(a0),a1
move.l rsEGS_MyScreen(a0),a0
CALLSYS E_FlipMap
.NoFlip:
move.l EgsBase,a6
move.l rsEGS_MyScreen(a3),a0
CALLSYS E_CloseScreen
move.l rsEGS_MapB(a3),a0
cmp.l #0,a0
beq.s .NoB
CALLSYS E_DisposeBitMap
.NoB:
move.l rsEGS_MapC(a3),a0
cmp.l #0,a0
beq.s .NoC
CALLSYS E_DisposeBitMap
.NoC:
move.l $4,a6
move.l a3,a1
move.l #rsEGS_SIZEOF,d0
CALLSYS FreeMem
movem.l (sp)+,a2/a3/a6
rts
```

1.16 GFXStuff V1.3

```
GFXStuff V1.3
RtgScreenAtFront EGS
This function gets an RtgScreen Screen handle of an already opened
Screen in a0. If it returns -1 (#$FFFFFFFF), the Screen is at the
front, if it returns 0, it is at the back. It returns 0/-1 in d0.
RtgScreenAtFront:
movem.l a4-a6,-(sp)
move.l a6,a5
move.l EgsBase(a5),a6
move.l rsEGS_MyScreen(a0),a0
move.l a0,a4
CALLSYS E_WhichMonitor
```

move.l d0,a0

CALLSYS E_WhichScreen

move.l #0,d1

cmp.l a4,d0

bne .Not

move.l #$ffffffff,d1

.Not:

move.l d1,d0

movem.l (sp)+,a4-a6

rts

1.17 GFXStuff V1.3

GFXStuff V1.3

GetRtgScreenData EGS

This function takes the RtgScreen handle of an already opened

RtgScreen in a0, and a list of TagItems (standard Amiga DOS 2.0

construct) in a1. It will fill the valid information for this

Tags for this screens. There are :

grd_Buffers : Get the number of Buffers this Screen has (1-3)

grd_Width : Get the Width of the Screen

grd_Height : Get the Height of the Screen

grd_Depth : Get the Depth of the Screen.

grd_PixelLayout : The Pixellayout, see table below

grd_Colorspace : The Colorspace, see table below

As to Video RAM organization :

For grd_Depth returning 1 : One Bit after the other.

For grd_Depth returning 8 : Standard 1 Byte Chunky organization.

For grd_Depth returning 24 : %rrrrrrrr gggggggg bbbbbbbb xxxxxxxx

R is Red, G is Green, B is Blue, X is Alpha Channel, IGNORE X !!!

For 1 Bit, grd_PixelLayout = 0, grd_Colorspace = 0

For 8 Bit, grd_PixelLayout = 2, grd_Colorspace = 0

For 24 Bit, grd_PixelLayout = 7, grd_Colorspace = 1

This function will change with rtgmaster.library as there will

be more Video RAM organizations to be handled. Read the Autodocs

of rtgmaster.library as soon as you get them.

This function needs the Utility Library to be opened already.

GetRtgScreenData:

movem.l a2-a6,-(sp)

```
move.l a1,a4
cmp.l #0,a0
beq .Exit
cmp.l #0,a1
beq .Exit
move.l a0,a3
move.l a1,a0
move.l #grd_Buffers,d0
move.l UtilityBase,a6
CALLSYS FindTagItem
cmp.l #0,d0
beq .NoBuffers
move.l rsEGS_MapB(a3),d1
cmp.l #0,d1
beq .NoB
move.l rsEGS_MapC(a3),d1
cmp.l #0,d1
beq .NoC
move.l d0,a0
move.l #3,ti_Data(a0)
bra .NoBuffers
.NoC:
move.l d0,a0
move.l #2,ti_Data(a0)
bra .NoBuffers
.NoB:
move.l d0,a0
move.l #1,ti_Data(a0)
.NoBuffers:
move.l a4,a0
move.l #grd_Width,d0
CALLSYS FindTagItem
cmp.l #0,d0
beq .BranchMinWidth
move.l rsEGS_MyScreen(a3),a0
move.l esc_Map(a0),a0
move.l d0,a1
sub.l d0,d0
move.w ebm_Width(a0),d0
```

```
ext.l d0
move.l d0,ti_Data(a1)
.BranchMinWidth:
move.l a4,a0
move.l #grd_Height,d0
CALLSYS FindTagItem
cmp.l #0,d0
beq .BranchMinHeight
move.l rsEGS_MyScreen(a3),a0
move.l esc_Map(a0),a0
move.l d0,a1
move.w ebm_Height(a0),d0
ext.l d0
move.l d0,ti_Data(a1)
.BranchMinHeight:
move.l a4,a0
move.l #grd_Depth,d0
CALLSYS FindTagItem
cmp.l #0,d0
beq .BranchChunky
move.l rsEGS_MyScreen(a3),a0
move.l esc_Map(a0),a0
sub.l d1,d1
move.b ebm_Depth(a0),d1
ext.w d1
ext.l d1
move.l d0,a0
move.l d1,ti_Data(a0)
.BranchChunky:
move.l a4,a0
move.l #grd_PixelLayout,d0
CALLSYS FindTagItem
cmp.l #0,d0
beq .BranchColor
sub.l d1,d1
move.l rsEGS_MyScreen(a3),a0
move.l esc_Map(a0),a0
move.b ebm_Depth(a0),d1
ext.w d1
```

```
ext.l d1
cmp.b #1,d1
beq .TheOne
cmp.b #8,d1
beq .TheEight
.The24:
move.l #grd_TRUECOL32,d1
move.l d0,a0
move.l d1,ti_Data(a0)
bra .BranchColor
.TheEight:
move.l #grd_CHUNKY,d1
move.l d0,a0
move.l d1,ti_Data(a0)
bra .BranchColor
.TheOne:
move.l #grd_PLANAR,d1
move.l d0,a0
move.l d1,ti_Data(a0)
.BranchColor:
move.l a4,a0
move.l #grd_ColorSpace,d0
CALLSYS FindTagItem
cmp.l #0,d0
beq .Exit
sub.l d1,d1
move.l rsEGS_MyScreen(a3),a0
move.l esc_Map(a0),a0
move.b ebm_Depth(a0),d1
ext.w d1
ext.l d1
cmp.b #1,d1
beq .TheOne2
cmp.b #8,d1
beq .TheEight2
.The242:
move.l #grd_RGB,d1
move.l d0,a0
move.l d1,ti_Data(a0)
```

```
bra .Exit
.TheEight2:
move.l #grd_Palette,d1
move.l d0,a0
move.l d1,ti_Data(a0)
bra .Exit
.TheOne2:
move.l #grd_Palette,d1
move.l d0,a0
move.l d1,ti_Data(a0)
.Exit:
move.l a4,a1
movem.l (sp)+,a2-a6
rts
```

1.18 GFXStuff V1.3

GFXStuff V1.3

CloseLibs EGS

Simply closes egs.library. Call this after you closed all RtgScreens.

You will have to close the other Libraries yourselves.

```
CloseLibs:
cmp.l #0,EgsBase
beq .Exit
move.l $4,a6
.CloseEGS:
move.l $4,a6
move.l EgsBase,a1
CALLSYS CloseLibrary
.Exit:
move.l #0,d0
rts
```

1.19 GFXStuff V1.3

GFXStuff V1.3

Data Structures EGS

The EGS Screenmode structure (like used in GetScreenmodes)

```
STRUCTURE E_ScreenMode,0
```

STRUCT esm_Node,LN_SIZE ; in esmNode.ln_Name you will find the name of the Screenmode
UWORD esm_Horiz ; Horizontal resolution
UWORD esm_Vert ; Vertical resolution
UWORD esm_Pad
ULONG esm_Depths ; Color depths (this code supports 1, 8 and 24 Bit)
APTR esm_Driver ; Some infos, that you probably not need :)
STRUCT esm_Specs,24*4 ; More things that you probably do not need...
LABEL esm_SIZEOF
The RtgScreen structure for EGS. This is considered PRIVATE DATA for
the provided function. Use these functions to access it !!! This
structure probably will change a bit in rtgmaster.library.
STRUCTURE RtgScreenEGS,0
APTR rsEGS_MyScreen ; The EGS Screen handle
ULONG rsEGS_ActiveMap ; The Buffer number of the active Buffer
APTR rsEGS_MapA ; Buffer 0 E_EBitmap
APTR rsEGS_MapB ; Buffer 1 E_EBitmap
APTR rsEGS_MapC ; Buffer 2 E_EBitmap
APTR rsEGS_FrontMap ; Buffer address of Buffer rsEGS_ActiveMap
ULONG rsEGS_Bytes ; 0 = 1 Bit, 1 = 8 Bit, 4 = 24 Bit
ULONG rsEGS_Width ; Width of the RtgScreen for fast access on it
ULONG rsEGS_NumBuf ; Number of Buffers of this RtgScreens
UWORD rsEGS_Locks ; The Screen Lock Word
LABEL rsEGS_SIZEOF
STRUCTURE ScreenQuery,0
ULONG sq_ScreenMode ; Pointer to the *NAME* of the wanted Screenmode
UWORD sq_Width ; The Width
UWORD sq_Height ; The Height
UWORD sq_Depth ; The color depth
ULONG sq_Buffers ; How many Buffers...
ULONG sq_Port ; An eDCMP Messageport (if not used: 0)
ULONG sq_EdcmpFlags ; The used Edcmp-Flags of the Screen (if no eDCMP: 0)
LABEL sq_SIZEOF
NOTE: That Structure is different to the CyberGraphX version, as here
the Screenmode *NAME* is provided, not the ModeID !!!
You get a list of the Screenmodes with all that names and other stuff
using GetRtgScreenModes. sq_Port and sq_EdcmpFlags are usually set
to 0. See more information about it in Some additional notes .

1.20 GFXStuff V1.3

GFXStuff V1.3
CyberGraphX Support
OpenLibs is some initialization code, that opens
the needed libraries and additional checks,
if a CyberGraphX System is provided. This system does not react, if only
the libraries are installed. It only reacts, if CyberGraphX is correctly
installed, otherwise it fails.
GetScreenmodes gets you a list of the CyberGraphX Screenmodes.
It is explained there which format the Screenmodestructure has.
FreeScreenmodes frees the Screenmodelist again and
returns the memory used by it to the system.
SwitchScreens performs Double and/or Triplebuffering.
Currently this function is not implemented, as CyberGraphX up to now
does not support Double Buffering. You find some hints here, though,
what possibilities to *fake* DOuble Buffering exist.
GetBufAdr returns you the base address of the Video RAM.
LockRtgScreen locks one of our screens for private usage
and returns the base address of the currently displayed buffer.
UnlockRtgScreen unlocks a screen again.
LoadRGBRtg is a LoadRGB32 clone and works exactly
like this OS 3.0 colortable setting function. It is only of use for
screens with 8 Bit or fewer colors. For 15/16/24 Bit direct color
access is used.
OpenRtgScreen is the function that IN FACT opens
one of our screens. This function may change, as soon as CyberGraphX
supports real Double Buffering.
CloseRtgScreen closes an RtgScreen again.
ScreenAtFront finds out, if an RtgScreen is at
front currently. Very useful for a game that runs in Multitasking,
to stop action as soon as the user clicks the game to back, and
to reactivate it again later.
GetRtgScreenData finds out some useful stuff
about an RtgScreen, for example how the Video RAM is organized.
CloseLibs closes the Rtg System.
BlitRtg is used for blitting.
The function waits for the blitter before returning.
Data structures lists and explains the data
structures (CyberGraphX ones and my own ones) used by this code.
These CyberGraphX functions support 8, 15, 16 and 24 Bit CyberGraphX
Screenmodes.
NOTE: ANY reference to RtgScreen refers to RtgScreenCyber in fact !!!

1.21 GFXStuff V1.3

GFXStuff V1.3

OpenLibs Cyber

This function opens the Cybergraphics.library and it checks, if there
is a working Cybergraphics System installed. If yes, it return 1 in d0,
otherwise it returns 0 in d0. It does not take any parameters.
It needs graphics.library to be opened correctly.

OpenLibs:

```
movem.l a0-a6/d0-d7,-(sp)
move.l $4,a6
move.l #0,CGXBase
lea CyberName,a1
moveq #40,d0
jsr -552(a6)
move.l d0,CGXBase(a5)
beq.s .Error
move.l #-1,d6
.Next Move.l d6,d0
move.l a6,-(sp)
move.l GfxBase(a5),a6
Jsr -732(a6)
move.l (sp)+,a6
Move.l d0,d6
move.l a6,-(sp)
move.l CGXBase(a5),a6
move.l d6,d0
jsr -54(a6)
move.l (sp)+,a6
cmp.l #0,d0
bne .CyberFound
Cmp.l #-1,d6
Beq .Error
bra .Next
.CyberFound
Move.l a5,d0
Bra.s .Ok
.Error movem.l (sp)+,a0-a6/d0-d7
Moveq #0,d0
```

```
rts
.Ok Movem.l (sp)+,a0-a6/d0-d7
moveq #1,d0
Rts
CyberName: dc.b 'cybergraphics.library',0
even
CGXBase: dc.l 0
```

1.22 GFXStuff V1.3

GFXStuff V1.3
GetScreenmodes Cyber
This function returns a list containing all Cybergraphics
Screenmodes in d0. You will have to set the tags to the
desired values. The Cybergraphics Screenmode structure
will be discussed in the Chapter about Cybergraphics
data structures.

```
GetScreenmodes:
move.l a6,-(sp)
move.l CGXBase,a6
lea tags,a1
jsr -72(a1) ; AllocCModeListTagList
move.l (sp)+,a6
rts
tags:
dc.l CYBRMREQ_MinWidth,MinWidth
dc.l CYBRMREQ_MaxWidth,MaxWidth
dc.l CYBRMREQ_MinHeight,MinHeight
dc.l CYBRMREQ_MaxHeight,MaxHeight
dc.l CYBRMREQ_MinDepth,MinDepth
dc.l CYBRMREQ_MaxDepth,MaxDepth
```

It is possible to access the Cybergraphics screenmodes using
the Systemlist of Screenmodes too. To find out, if a Mode is
a CyberGFX mode or not, call -54(a6) from the CyberGraphics Library
(IsCyberModeID). It will return 0, if it is *NO*
Cybergraphics Mode. Using CyberGFX it is even possible to get
>8 Bit Modes that way (unusual for an Amiga WB-Emulation). Of course,
if you use the above function, ALL returned Screenmodes are
Cybergraphics ones.

Usually supported depths are :
- 8 Bit chunky
- 15 Bit Hi-Color
- 16 Bit Hi-Color
- 24 Bit True-Color with or without Alpha-Channel
(Note : The workbench will think 24 Bit colour depth Screens
have 0 colors, but it works anyways...)
But anyways, it is better to use the above given function than to
examine the Screenmode systemlist.
About, how the screenmemory is organized, read the chapter about
GetRtgScreenData.

1.23 GFXStuff V1.3

GFXStuff V1.3
FreeScreenmodes Cyber
This function frees a Screenmode list again that was allocated
the CyberGraphX Native way (AllocCModeListTagList, like explained
in GetScreenModes). It expects the list in a0.

```
FreeScreenmodes:
move.l a6,-(sp)
move.l CGXBase,a6
move.l Modes,a0
jsr -78(a6) ; FreeCModeList
move.l (sp)+,a6
rts
```

1.24 GFXStuff V1.3

GFXStuff V1.3
GetBufAdr Cyber
This function is used to access the video RAM directly.
Primarily it reads out the variables set by OpenRtgScreen.
You may then write directly to the Video RAM like you wish.
The function expects the RtgScreen Handle of an opened RtgScreen
in a0 and the number of the buffer to examine in d0 (between
0 and 2, where 0 is the buffer, that is displayed after
OpenRtgScreen). It returns the requested value in d0.

```
GetBufAdr:
Lsl.w #2,d0
Move.l rsCGX_MapA(a0,d0.w),d0
Rts
```

1.25 GFXStuff V1.3

GFXStuff V1.3

LockRtgScreen Cyber

This function mainly is provided for compatibility
to rtgmaster.library. It does not do much, but it
returns the Video RAM base address of Buffer 0 (the
one that is displayed after opening the Screen)
in d0. In a0, the RtgScreen Handle of an opened
RtgScreen is assumed. You may nest calls of this
function up to 65535 times. Usually you call it
once after a OpenRtgScreen call.

```
LockRtgScreen:
add.w #1,rsCGX_Locks(a0)
move.l rsCGX_MapA(a0),d0
rts
```

1.26 GFXStuff V1.3

GFXStuff V1.3

UnlockRtgScreen Cyber

This function unlocks a RtgScreen again. It does not do
much, but it is provided as to compatibility to rtgmaster.library.
It expects the RtgScreen Handle of an opened RtgScreen in a0.

```
UnlockRtgScreen:
clr.l d0
move.w rsCGX_Locks(a0),d0
cmp.w #0,d0
beq .Exit
subq.w #1,rsCGX_Locks(a0)
.Exit
rts
```

1.27 GFXStuff V1.3

GFXStuff V1.3

LoadRGBRtg Cyber

This function provides a LoadRGB32 clone.
It is used to set the colorregisters for

indirect color modes (that are 8 Bit chunky
modes for this function collection).
The function expects a RtgScreen Handle of
an opened RtgScreen in a0 and a pointer to
a Colortable in a1. The colortable looks
exactly the same as for LoadRGBRtg EGS .
This function needs graphics.library to be
opened.
LoadRGBRtg:
Movem.l d2-d5/a2-a3/a6,-(sp)
Move.l GfxBase(a6),a6
Move.l rsCGX_MyScreen(a0),a2
Lea 44(a2),a2
Move.l a1,a3
.Loop2 Move.w (a3)+,d4
Beq.s .Exit
Move.w (a3)+,d5
Bra.s .Wend
.Loop Move.l a2,a0
Move.w d5,d0
Addq.w #1,d5
Move.l (a3)+,d1
Move.l (a3)+,d2
Move.l (a3)+,d3
Jsr -852(a6) Graphics - SetRGB32
.Wend Dbra d4,.Loop
Bra.s .Loop2
.Exit Movem.l (sp)+,d2-d5/a2-a3/a6
Rts

1.28 GFXStuff V1.3

GFXStuff V1.3
CloseRtgScreen Cyber
This function closes an RtgScreen again. It is NOT needed
to change the Buffer to the starting buffer again before doing
this (at least up to now, with this MoveScreen trick, but probably
won't change with CyberGraphX 3, after what i heard about how
DoubleBuffering will work with CyberGraphX3 ...) You should call

UnlockRtgScreen as much as you called LockRtgScreen before closing
the Screen (for compatibility issues only...)
This function assumes the RtgScreen Screen handle being in a0.
It uses the intuition.library, so it should be opened.

```
CloseRtgScreen:
movem.l a6,-(sp)
move.l IntBase,a6
move.l rsCGX_MyScreen(a0),d0
beq .ScreenClosed
move.l d0,a0
jsr -66(a6)
.ScreenClosed:
movem.l (sp)+,a6
rts
```

1.29 GFXStuff V1.3

GFXStuff V1.3
ScreenAtFront Cyber
This function will return -1 (#$FFFFFFFF), if the
RtgScreen which RtgScreen handle is currently in a0,
is in FRONT, else it will return 0. It will return its
values in d0. This function can be used for OS conform
games. It needs the intuition.library already opened.

```
ScreenAtFront:
Move.l IntBase,a1
Move.l 60(a1),d0
Cmp.l rsCGX_MyScreen(a0),d0
Beq.s .AtFront
Moveq #0,d0
Rts
.AtFront
Moveq #-1,d0
Rts
```

1.30 GFXStuff V1.3

GFXStuff V1.3

CloseLibs Cyber

This function closes the cybergraphics.library again...

CloseLibs:

cmp.l #0,CGXBase

beq .Exit

move.l $4,a6

move.l CGXBase,a1

CALLSYS CloseLibrary

.Exit:

move.l #0,d0

rts

1.31 GFXStuff V1.3

GFXStuff V1.3

GetRtgScreenData Cyber

With this function you may get some information about
the used Screenmode of a RtgScreen, for example, how
the video memory is organized (mainly useful for >8 Bit,
as on 8 Bit it is the same for all CyberGraphX and EGS
Boards...).
The function expects The RtgScreen Screen handle is
expected to be in a0, in a1 there is expected a pointer
to an array of TagItems, where the correct values will be
filled in by this functions. The function has no return
value and expects Utility Library to be open.
This function probably will be changed a bit in rtgmaster.library.
Valid Tags are :
grd_Buffers ; Returns the number of used Buffers
; (But remember, no real Double Buffering up to now
; in CyberGraphX !!!)
grd_Width ; The Width
grd_Height ; The Height
grd_Depth ; The Depth
grd_PixelLayout ; The Pixellayout, look at table below
grd_ColorSpace ; The Colorspace, look at table below

PixelLayout :

grd_PLANAR EQU 0 ; Non interleaved planar layout [X bitplanes/pixel]

grd_PLANARI EQU 1 ; Interleaved planar layout [X bitplanes/pixel]

grd_CHUNKY EQU 2 ; 8-bit Chunky layout [BYTE/pixel]

grd_HICOL15 EQU 3 ; 15-bit Chunky layout [WORD/pixel]

grd_HICOL16 EQU 4 ; 16-bit Chunky layout [WORD/pixel]

grd_TRUECOL24 EQU 5 ; 24-bit Chunky layout [3 BYTES/pixel]

grd_TRUECOL24P EQU 6 ; 24-bit Chunky layout [3 BYTEPLANES/pixel]

grd_TRUECOL32 EQU 7 ; 24-bit Chunky layout [LONG/pixel] (RGBx or BGRx)

grd_GRAFFITI EQU 8 ; 8-bit Graffiti-type Chunky layout (very special...)

grd_TRUECOL32B EQU 9 ; 24-bit Chunky layout [LONG/pixel] (xRGB or xBGR)

; the grd_GRAFFITI is only for compatibility with rtgmaster.library (Graffiti sublibrary)

; Similar to grd_PLANAR/grd_PLANARI ...

ColorSpace :

grd_Palette EQU 0 ; Mode uses a Color Look-Up Table (CLUT)

grd_RGB EQU 1 ; standard RGB color space

grd_BGR EQU 2 ; high-endian RGB color space, BGR

grd_RGBPC EQU 3 ; RGB with lowbyte and highbyte swapped

grd_BGRPC EQU 4 ; BGR with lowbyte and highbyte swapped

Some examples :

grd_PixelLayout = 9 and grd_ColorSpace = 4

=> xBGR with low and high word swapped...

grd_PixelLayout = 2 and grd_Colorspace = 0

=> 8 Bit chunky with colortable

grd_PixelLayout = 4 and grd_Colorspace = 1

=> 16 Bit RGB

(In the sourcefiles there is a list of ALL

currently supported Color Layouts, including there

settings for these Tags...)

GetRtgScreenData:

Move.l a0,a2

Move.l a1,a3

Move.l rsCGX_MyScreen(a2),a4

move.l a6,a5

Move.l UtilityBase,a6

Move.l #grd_Width,d0

Move.l a3,a0

Jsr -30(a6) Utility - FindTagItem

Tst.l d0

```
Beq.s .NoWidth
move.l d0,a1
move.l rsCGX_RealMapA(a4),a0
move.l #CYBRMATTR_WIDTH,d1
move.l CGXBase,a6
move.l a1,-(sp)
jsr -96(a6); GetCyberMapAttr
move.l (sp)+,a1
move.l UtilityBase,a6
move.l d0,ti_Data(a1)
.NoWidth
Move.l #grd_Height,d0
Move.l a3,a0
Jsr -30(a6) Utility - FindTagItem
Tst.l d0
Beq.s .NoHeight
move.l d0,a1
move.l rsCGX_RealMapA(a4),a0
move.l #CYBRMATTR_HEIGHT,d1
move.l CGXBase,a6
move.l a1,-(sp)
jsr -96(a6); GetCyberMapAttr
move.l (sp)+,a1
move.l UtilityBase,a6
move.l d0,ti_Data(a1)
.NoHeight
Move.l #grd_PixelLayout,d0
Move.l a3,a0
Jsr -30(a6) Utility - FindTagItem
Tst.l d0
Beq .NoPixelLayout
move.l d0,a1
move.l rsCGX_RealMapA(a4),a0
move.l #CYBRMATTR_PIXFMT,d1
move.l CGXBase,a6
move.l a1,-(sp)
jsr -96(a6) ; GetCyberMapAttr
move.l (sp)+,a1
move.l UtilityBase,a6
```

```
cmp.l #0,d0
beq .FmtChunky
cmp.l #5,d0
blt .FmtHi15
cmp.l #9,d0
blt .FmtHi16
cmp.l #11,d0
blt .FmtTrue24
cmp.l #11,d0
beq .FmtTrue32B
cmp.l #12,d0
beq .FmtTrue32
cmp.l #13,d0
beq .FmtTrue32
bra .NoPixelLayout ; Unsupported Pixellayout
.FmtChunky:
move.l #grd_CHUNKY,d0
bra .ValueReturn
.FmtHi15:
move.l #grd_HICOL15,d0
bra .ValueReturn
.FmtHi16:
move.l #grd_HICOL16,d0
bra .ValueReturn
.FmtTrue24:
move.l #grd_TRUECOL24,d0
bra .ValueReturn
.FmtTrue32:
move.l #grd_TRUECOL32,d0
bra .ValueReturn
.FmtTrue32B:
move.l #grd_TRUECOL32B,d0
bra .ValueReturn
.ValueReturn:
move.l d0,ti_Data(a1)
.NoPixelLayout
Move.l #grd_ColorSpace,d0
Move.l a3,a0
Jsr -30(a6) Utility - FindTagItem
```

```
Tst.l d0
Beq .NoColorSpace
move.l d0,a1
move.l rsCGX_RealMapA(a4),a0
move.l #CYBRMATTR_PIXFMT,d1
move.l CGXBase,a6
move.l a1,-(sp)
jsr -96(a6) ; GetCyberMapAttr
move.l (sp)+,a1
move.l UtilityBase,a6
cmp.l #0,d0
beq .FmtPalette
cmp.l #1,d0
beq .FmtRGB
cmp.l #5,d0
beq .FmtRGB
cmp.l #9,d0
beq .FmtRGB
cmp.l #11,d0
beq .FmtRGB
cmp.l #2,d0
beq .FmtBGR
cmp.l #6,d0
beq .FmtBGR
cmp.l #10,d0
beq .FmtBGR
cmp.l #12,d0
beq .FmtBGR
cmp.l #3,d0
beq .FmtRGBPC
cmp.l #7,d0
beq .FmtRGBPC
cmp.l #4,d0
beq .FmtBGRPC
cmp.l #8,d0
beq .FmtBGRPC
cmp.l #13,d0
beq .FmtRGB
bra .NoColorSpace ; Unknown Format
```

```
.FmtPalette
move.l #grd_Palette,d0
bra .ReturnValue
.FmtRGB
move.l #grd_RGB,d0
bra .ReturnValue
.FmtBGR
move.l #grd_BGR,d0
bra .ReturnValue
.FmtRGBPC
move.l #grd_RGBPC,d0
bra .ReturnValue
.FmtBGRPC
move.l #grd_BGRPC,d0
bra .ReturnValue
.ReturnValue
move.l d0,ti_Data(a1)
.NoColorSpace
Move.l #grd_Depth,d0
Move.l a3,a0
Jsr -30(a6) Utility - FindTagItem
Tst.l d0
Beq.s .NoDepth
move.l d0,a1
move.l rsCGX_RealMapA(a4),a0
move.l #CYBRMATTR_DEPTH,d1
move.l CGXBase,a6
move.l a1,-(sp)
jsr -96(a6); GetCyberMapAttr
move.l (sp)+,a1
move.l UtilityBase,a6
move.l d0,ti_Data(a1)
.NoDepth
Move.l #grd_Buffers,d0
Move.l a3,a0
Jsr -30(a6) Utility - FindTagItem
Tst.l d0
Beq.s .NoBuffers
move.l d0,a1
```

```
move.l rsCGX_NumBuf(a4),d0
move.l d0,ti_Data(a1)
.NoBuffers
Movem.l (sp)+,a2-a5/a6
rts
```

1.32 GFXStuff V1.3

GFXStuff V1.3
BlitRtg Cyber
This function can be used to blit parts of ONE Buffer of a screen
to a part of ANOTHER Buffer (or the same one) of a screen. This
function uses the graphics board processor of the Board. Up to
now, it is NOT in rtgmaster.library, but maybe this will change in the future.
For compatibility to versions of this function of OTHER graphics boards
(for example EGS...) it is NOT ALLOWED that source and destination
rectangle OVERLAP !!!
This function expects The RtgScreen Screen Handle of the Screen, whose
Buffers have to be blitted, in a0, the number of the source Buffer in d6,
the number of the destination Buffer in d7, SrcX and SrxY in d0 and d1,
DstX and DstY in d2 and d3, and width and height of the Blit in d4 and
d5.
NOTE: I did not give that function a test run up to now (but i already
tested some stuff with BltBitmap together with CyberGraphX, and i assume
this function should be correct...)
This function needs the Graphics.library to be opened.
The function returns, after the blit is REALLY finished, so it
WAITS for the Graphics Board blitter...
BlitRtg:

```
movem.l d6-d7/a3/a6,-(sp)
move.l a0,a3
move.l rsCGX_RealMapA(a0),a1
move.l #0,a2
cmp.l #0,d6
beq .Weiter
cmp.l #1,d6
beq .Buffer1Src
bra .Buffer2Src
.Buffer1Src:
```

```
movem.l d4/d5,-(sp)
move.l rsCGX_Width(a3),d4
clr.l d4
clr.l d5
move.w rsCGX_Height(a3),d5
mulu d5,d4
add.l d4,d1
movem.l (sp)+,d4/d5
bra .Weiter
.Buffer2Src:
movem.l d4/d5,-(sp)
move.l rsCGX_Width(a3),d4
clr.l d4
clr.l d5
move.w rsCGX_Height(a3),d5
mulu d5,d4
add.l d4,d1
add.l d4,d1
movem.l (sp)+,d4/d5
.Weiter:
clr.l d6
cmp.l #0,d7
beq .WeiterD
cmp.l #1,d7
beq .Buffer1Dest
bra .Buffer2Dest
.Buffer1Dest:
movem.l d4/d5,-(sp)
move.l rsCGX_Width(a3),d4
clr.l d4
clr.l d5
move.w rsCGX_Height(a3),d5
mulu d5,d4
add.l d4,d3
movem.l (sp)+,d4/d5
bra .WeiterD
.Buffer2Dest:
movem.l d4/d5,-(sp)
move.l rsCGX_Width(a3),d4
```

```
clr.l d4
clr.l d5
move.w rsCGX_Height(a3),d5
mulu d5,d4
add.l d4,d3
add.l d4,d3
movem.l (sp)+,d4/d5
.WeiterD:
clr.l d7
move.b #$C0,d6
move.b #$FF,d7
move.l GfxBase,a6
jsr -30(a6) ; BltBitMap
jsr -228(a6) ; WaitBlit
movem.l (sp)+,d6-d7/a3/a6
rts
```

1.33 GFXStuff V1.3

GFXStuff V1.3
BlitRtg EGS
This function can be used to blit parts of ONE Buffer of a screen
to a part of ANOTHER Buffer (or the same one) of a screen. This
function uses the graphics board processor of the Board. Up to
now, it is NOT in rtgmaster.library, but maybe this will change in the future.
It is NOT ALLOWED that source and destination
rectangle OVERLAP !!!
This function expects The RtgScreen Screen Handle of the Screen, whose
Buffers have to be blitted, in a0, the number of the source Buffer in d6,
the number of the destination Buffer in d7, SrcX and SrxY in d0 and d1,
DstX and DstY in d2 and d3, and width and height of the Blit in d4 and
d5.
NOTE: I did not give that function a test run up to now (but i already
used the underlying EGS function...)
This function needs the Egsblit.library to be opened. This library up to now
won’t be opened of the Initialization function of this code.
This function WAITS for the blitter before returning.
BlitRtg:

```
movem.l d6/a3/a6,-(sp)
```

```
move.l a0,a3
move.l rsEGS_MyScreen(a0),a1
movem.l d2/d3,-(sp)
move.l d4,d2
move.l d5,d3
movem.l (sp)+,d4/d5
cmp.l #0,d6
beq .Buffer0Src
cmp.l #1,d6
beq .Buffer1Src
bra .Buffer2Src
.Buffer0Src:
move.l rsEGS_MapA(a3),a0
bra .Weiter
.Buffer1Src:
move.l rsEGS_MapB(a3),a0
bra .Weiter
.Buffer2Src:
move.l rsEGS_MapC(a3),a0
.Weiter:
move.l #0,d6
cmp.l #0,d7
beq .Buffer0Dest
cmp.l #1,d7
beq .Buffer1Dest
bra .Buffer2Dest
.Buffer0Dest:
move.l rsEGS_MapA(a3),a1
bra .WeiterD
.Buffer1Dest:
move.l rsEGS_MapB(a3),a1
bra .WeiterD
.Buffer2Dest:
move.l rsEGS_MapC(a3),d1
move.l EgsBlitBase,a6
CALLSYS EB_CopyBitMap
movem.l (sp)+,d6/a3/a6
rts
```

1.34 GFXStuff V1.3

GFXStuff V1.3

Data structures Cyber

The screenmode structure (like used in GetScreenmodes) :

```
STRUCTURE CyberModeNode,0
STRUCT cmn_Node,LN_SIZE
STRUCT cmn_ModeText,DISPLAYNAMELEN ; Screenmodename
ULONG cmn_DisplayID ; ModeID
UWORD cmn_Width ; Width
UWORD cmn_Height ; Height
UWORD cmn_Depth ; Depth
APTR cmn_DisplayTagList ; Ignore this at the moment
LABEL cmn_SIZEOF
```

The Depth is 8, 15, 16 or 24. And remember : Contrary to
the EGS version of GetScreenmodes this function returns
a pointer to a list that has to be FREED at the end.

```
STRUCTURE RtgScreenCGX,0
APTR rsCGX_MyScreen ; An Intuition Screen
ULONG rsCGX_ActiveMap ; Number of active buffer
APTR rsCGX_MapA ; Video Mem Adress of Buffer 0 (at front at the beginning...)
APTR rsCGX_MapB ; The same for buffer 1
APTR rsCGX_MapC ; The same for buffer 2
APTR rsCGX_FrontMap ; The address of the Buffer in front...
ULONG rsCGX_Bytes ; How many bytes one pixel fills (2 for 16 Bit for example)
ULONG rsCGX_Width ; The Width (Caution! Longword!)
UWORD rsCGX_Height; The Height (Caution! Now word... sorry for that :) )
ULONG rsCGX_NumBuf ; Number of buffers of that screen...
UWORD rsCGX_Locks ; The Rtg Locks...
APTR rsCGX_ModeID ; The ModeID of this screen ...
ULONG rsCGX_RealMapA ; The graphics.library Bitmap structure (modified by Cyber to support Chunky)
STRUCT rsCGX_Tags,16 ; Some place for OpenRtgScreen to put something there :)
LABEL rsCGX_SIZEOF
```

Note : Like RtgScreenEGS this structure will change in
rtgmaster.library... but it is considered PRIVATE to
rtgCGX.library anyways...

```
STRUCTURE ScreenQuery,0
ULONG sq_ModeID ; ModeID of the to be used Screenmode
ULONG sq_Width ; The Width
```

ULONG sq_Height ; The Height
UWORD sq_Depth ; The color depth
ULONG sq_Buffers ; How many Buffers...
LABEL sq_SIZEOF
NOTE : This sturcture is different to the EGS version, as the
MODEID and not the Screenmodename is provided.
There are no Messageport parameters like for EGS... look at
the Notes for more details on Message stuff...

1.35 GFXStuff V1.3

GFXStuff V1.3
Source
In this section you may load the Source of
this document as text file (all source without
documentation, only the function listed one
after another, and the data structures
in addition...)
Click here for the EGS Source.
Click here for the Cyber Source.

1.36 GFXStuff V1.3

GFXStuff V1.3
Include
Click here for the EGS Includes.
Click here for the Cyber Includes.

1.37 GFXStuff V1.3

GFXStuff V1.3
Includes EGS
Some notes about these Includes :
These are CUSTOM includes, as there are no EGS
ASM Includes for EGS V6 (and this code needs
EGS V6 at least). So you can't use other includes
than those in this package !!!
Additional... these includes may be incomplete,
as i only patched that things in the V5 EGS ASM

includes, that i NEEDED from V6... but for this code
they are enough :)
egs.i
lvo_egs.i
egsblit.i
lvo_egsblit.i

1.38 GFXStuff V1.3

GFXStuff V1.3
Include Cyber
cybergraphics.i

1.39 GFXStuff V1.3

GFXStuff V1.3
Some Notes...
Things to look at for compatibility...
Keyboard/Mouse code ...
Screenmode frequencies ...

1.40 GFXStuff V1.3

GFXStuff V1.3
Things to look for compatibility ...
- Be *careful* with playing around with Copper, Blitter,...
(Do NOT use anything of this stuff with EGS, using Blitter
with the OS is okay for CyberGraphX, but *NOT* using the
hardware registers...)
- Be careful, if you modify hardware registers, switch of
Multitasking, switch to supervisor mode, play around with
the interrupts, Disable Multitasking and such stuff...
Some stuff MIGHT work, some things NOT... better do not
use ANYTHING of this stuff okay (reading out $DFF006 is
okay, though :))
- Even if some of this codes looks SILLY (for example
that PixelLayout/Colorspace stuff) USE IT... it will
make it easier to switch to rtgmaster.library later...
- But do not assume the RtgScreen structure to stay the same

with rtgmaster.library !!!
- Do not access EGS Screens with RtgScreenCyber or
CyberGraphX Screens with RtgScreensEGS !!!
- Do *NOT* use system Double Buffering (may change for
CyberGraphX only for CyberGraphX 3.0)
- Do *NOT* use IDCMP for EGS Screens... does not work !!!
- Stay OS compatible !!!

1.41 GFXStuff V1.3

GFXStuff V1.3
Keyboard/Mouse Code ...
For CyberGraphX :
- Use IDCMP by accessing the MyScreen structure
of RtgScreenCyber, which is a simple Intuition Screen
(You have to open a window on it, then...)
- Use according devices that work WITHOUT an opened
Window. This is the *best* method, as it works with
ALL known Workbench emulations, especially for EGS
and CyberGraphX.
- PROBABLY accessing the keyboard interrupt will work
too, but i do not know...
For EGS :
- Use according devices that work WITHOUT an opened
Window. This is the *best* method, as it works with
ALL known Workbench emulations, especially for EGS
and CyberGraphX.
- Use the eDCMP Method (eDCMP is similar to IDCMP for
EGS, but you need no window for it (as you can’t
open a window on an E_EScreen anyways...), it is available
on a screen). It is described in the following...
1. How do i get the Message Port ?
You have to provide your own MessagePort to OpenRtgScreen (struct MessagePort of
exec.library, or a pointer to it, to be exact) using the ScreenQuery structure.
Following is the data for sq_EdcmpFlags :
* Corresponding EDCMPFlagSet has 32 bits !
E_eMOUSEBUTTONS EQU 1
E_eMOUSEMOVE EQU 2
E_eRAWKEY EQU 4

E_eTIMETICK EQU 8

E_eDISKINSERTED EQU 16

E_eDISKREMOVED EQU 32

E_eNEWPREFS EQU 64

(like that defined in egs.i. There is *NO* VanillaKey, so all has be done using RAWKEY !!!)

2. Getting the Messages from the Port

A. A usual Wait/WaitPort with the appropriate Parameters

B. A usual GetMsg that returns a pointer to a

E_EGSMessage.

STRUCTURE E_EGSMessage,0

STRUCT ems_Msg,MN_SIZE ; has long word size

ULONG ems_Class

UWORD ems_Code

UWORD ems_Qualifier

APTR ems_IAddress

WORD ems_MouseX

WORD ems_MouseY

ULONG ems_Seconds

ULONG ems_Micros

APTR ems_EdcmpScreen

LABEL ems_SIZEOF

(looks well-known, doesn't it ? :))

The rest should be no problem...

1.42 GFXStuff V1.3

GFXStuff V1.3

Screenmode frequencies ...

As with Amiga graphics Board everyone can

define his own Screenmodes, you MIGHT want

to check what frequencies a choosen Screenmode has,

so it is possible to check, if that Screenmode is okay

for the connected monitor. Of course you might say the

user should look after it, if all Screenmodes are okay,

but if you WANT to check, this is how to do it : (But most

people simply just don't check for the frequencies anyways :))

For Cybergraphics, simply examine the ModeID of a Screenmode

with the functions that are supported by graphics.library (NOT

using any Cybergraphics stuff, NOT using any Cybergraphics Screenmode

structures !!!) Only examine the graphics.library Screenmode structures,
like you do for, let's say, a PAL Screenmode !!!
So for Cybergraphics, this check is quite easy... for EGS it is more
complicated.
For EGS you CAN'T use the system list of Screenmodes, EVEN if the EGS
Modes are in the systemlist, as there are no correct values in the
system Screenmode list for the frequencies. In the following there
is explained how you can get the frequency values for EGS :
A. Find the EGS Screenmode structure in the stuff that GetScreenmodes returns
B. If the structure is (for example) in a0, look at the Array esm_Specs
(24 ScreenParamPtr of EGS). Look for *THE* pointer that is for the
color depth for that you want to know the frequencies (the frequencies
are color depth dependent for EGS). Therefor, the first pointer is
at the one for color depth 1, the 8th for 256 colors, the 24th for
24 Bit.
C. Get the values out of the following structure :

```
struct E_ScreenParam {
struct MinNode MinNode; /* internal chaining, the list is found
* in the associated screen param struct.
*/
WORD PixFreq; /* pixel frequency in units of 1/100 MHz */
WORD LineFreq; /* line frequency in units of 1/100 KHz */
WORD FrameFreq; /* frame frequency in units of 1/100 Hz */
WORD Pad0;
ULONG Flags; /* ScreenParamFlagSet */
WORD Hfront; /* horizontal front porch in ns */
WORD Hsync; /* horizontal sync in ns */
WORD Hback; /* horizontal back porch in ns */
WORD Vfront; /* vertical front porch in µs */
WORD Vsync; /* vertical sync in µs */
WORD Vback; /* vertical back porch in µs */
WORD Vpre; /* vertical preequalisation in µs */
WORD Vpost; /* vertical postequalisation in µs */
E_ScreenSpecPtr Screen; /* the ruling screen param */
};
```

(Sorry, i do not have ASM-Includes for that, but should be easy to convert...)
As to what each value is, i snapped the following out of the EGS includes :

```
/*
* Some notes about display and syncing...
```

```
*
*
*
* +---------------------------------------------+
* |################################### HH |\
* |################################### HH | |
* |################################### HH | |
* |################################### HH | |
* |################################### HH | |
* |################################### HH | \ active vertical
* |################################### HH | /
* |################################### HH | |
* |################################### HH | |
* |################################### HH | |
* |################################### HH | |
* |################################### HH | |
* |################################### HH | |
* |################################### HH |/
* | HH |\_ vertical \
* | HH | | front |
* | HH |/ porch |
* |VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV|\_ vertical | vertical
* |VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV|/ sync \ blank
* | HH |\ /
* | HH | | |
* | HH | | vertical |
* | HH | \ back |
* | HH | / porch |
* | HH | | |
* | HH |/ /
* +---------------------------------------------+
* \_____________ _____________________/\ _/\/\_ _/
* \/ \/ | \/
* active horizontal | | horizontal back porch
* | |
* | \ horizontal sync
* |
* \ horizontal front porch
*
```

* _ ________/

* \/

* horizontal blank

*

* Pixel timing diagramm:

*

* . . . .

* ._ ._ ._ ._ ._ ._ ._ ._ ._ ._ ._ ._ ._ ._

* PClock : | | | | | | | | | | | | | | | | | | | | | | | | | | | |

* _|.|_|.|_|.|_|.|_|.|_|.|_|.|_|.|_|.|_|.|_|.|_|.|_|.|_|.|_|.

* . . . . .___________ . . .

* RGB : . . . . . . .___/ . . \ . . .

* . . . . . .___/ . . . | . ._____

* . . .___. . / . . . . | . / .

* . .___/ \ . | . . . . | . | .

* black ...../........_____/....................._____/......

* . | . . . .

* blank ____/. . . . .

* . .____

* DispEn : . | . . . .

* _____|..

* . . . .

*

* PClock or pixel frequency determines the speed at which pixels are sampled

* and sent to the DACs.

*

*

*

*

* Horizontal timinig diagram:

*

* . . . . .

* RGB : ############## ##############. . . ######

* _____##############...##############_____________######

* ._______________________________. . . ._____

* DispEn: | | . . |

* _____|.............................|____________|.....

* . . ._. .

* HSync : . . | | .

```
* _____|.|_____
* . . . . .
* Sync on : ############## ###############. . . ######
* green _____##############...###############_____..._____######
* . . |_| .
* . . . . .
*
* | - active video ---------- | hf | | hb |
* | | hs |
* | | |
* | | - hblank - |
* | |
* | - linetime ------------------------------ |
*
* hf : Horizontal front porch, the time between the end of the
* active display to the start of the horizontal sync pulse.
* This value controls the size of the right border of a display.
*
* hs : Horizontal sync, this signal starts the horizontal flyback.
*
* hb : Horizontal back porch, the time between the and of the
* horizontal sync pulse to the start of the next display line.
* This value contorls the size of the left border of a display.
*
* hblank : The full blanking period, from the end of line n to the start
* of line n+1. This value controlls the width of the display.
* If hf and hb are changed, but hblank kept the same, the display
* can be moved to the left or right.
*
* hfreq : The horizontal line frequency equals 1/linetime
*
* Vertical timing diagramm:
*
* . . . . . .
* RGB : ####. ####. ####. . . . . #
* ####._####__####_____________________________________#
* ___ . ___ . ___ . ___ . . . . . _
* DispEn : |.| |.| |.| |. . . . .|
* ...|_|...|_|...|_|...|_____________________________|.
```

```
* . . . . . .
* HSync : | | | | | | | | | | | |
* ____|_____|_____|_____|__|__|__|__|__|__|_____|_____|__
* . . . . ._____. . . .
* VSync : . . . . | | . . .
* _____________________________________|.....|_____________________
* . . . . ._____. . . .
* CSync : | | | | | | . . |
* ____|_____|_____|_____|_____|_____|.....|_____________________|__
* . . . . .__ __. . . .
* CSync : | | | | | | | | | | | |
* tesselated____|_____|_____|_____|__|__|..|..|__|__|_____|_____|__
* . . . . . .
* CSync on: ####. ####. ####. . . . . #
* green ####._####._####._____.__ __.......__ __._____._____._#
* | | | | | |__|__| | | | |
* . . . . . .
*
* | |vpre | |vpost| |
* | |
* | --- vf --- | vs | ----- vb -------- |
* | |
* <-- active --- | - vertical blanking period vblank - |->
* |
* <------------- | - frametime -------------------------->
*
* vf : Vertical front porch, the time from the end of the last
* active line to the start of the sync signal, controls the
* the size of the bottom border of the display
*
* vs : Vertical sync pulse, start the vertical flyback
*
* vb : Vertical back porch, the time from the end of the vertical
* sync to the start of the next display frame. This value
* controls the size of the top border
*
* vpre,
* vpost : Vertical ??serration?? pulses, needed in interlace mode to
* sync on a half line
```

```
*
* vblank : The full vertical blanking period this value controls
* the vertical size of the display
*
* vfreq : The vertical frame frequency equals 1/frametime
*
*
*/
```

D. You are able to do the frequency check automatically :

- Get the display driver address out of the EGS Screenmode address (which you get using GetScreenmodes)

```
struct E_DisplayDriver {
/* E_EGSObject Object; */
/* it's an object !!! */
struct Node Node; /* internal chaining, the list
* is found in the hardinfo
* structure
*/
WORD Pad0;
char *Prefix; /* the prefix name of all the
* drivers screenmodes like
* "LEGSa:" or "RB3a:"
*/
ULONG Depths; /* supported colordepths, each
* bit represents one depth, so
* 1<<8 stands for eight biz
*/
struct E_MinMax Freqs[24]; /* supported pixel frequency
* intervalls for specific color
* depths. (as C does not support
* array starting at one, the
* index is colordepth minus one)
*/
struct MinList Monitors; /* supported specs of the attached
* monitor
*/
ULONG Flags; /* DisplayDriverFlagSet; */
char *Default; /* the name of the drivers defaults
* file
```

```
*/
ULONG MaxPixel[24]; /* The maxmimum number of pixels
* the graphics device can
* in a given color depth. (As C
* does not support arrays
* starting at one, the index is
* colordepth minus one)
*/
ULONG MemSize; /* The size of the onboard memory in
* bytes
*/
UWORD MouseSize; /* The maximum mouse size, may be
* less for some screenmodes
*/
UWORD Pad1;
char *description; /* A small text describing the
* graphics device
*/
};
```

- get the address of the Monitor there (struct MinList Monitors).

```
struct E_MonitorSpec {
struct Node Node; /* internal chaining, the list is found in
* the hard info "monitors", the lh_name
* field contains the monitor name
*/
WORD Pad0;
struct List Screens; /* list of the supported screenmodes; the
* nodes are of type "E_ScreenSpec"
*/
WORD Pad1;
ULONG Sync; /* Syncing flags */
WORD MinHoriz; /* minimum horizontal frequency * 1/100KHz */
WORD MaxHoriz; /* maximum horizontal frequency * 1/100KHz */
WORD MinVerti; /* minimum vertical frequency   * 1/100Hz */
WORD MaxVerti; /* maximum vertical frequency * 1/100Hz */
};
```

- And now you are able to compare, if the frequencies for the selected Screenmode are okay for the connected monitor :)

1.43 GFXStuff V1.3

GFXStuff V1.3

OpenRtgScreen Cyber

This function opens IN FACT an RtgScreen. It returns a RtgScreen
Screen Handle in d0. In a2 it assumes a ScreenQuery structure
to get the information which Width, Height, Depth... you need.
The way this function will be called, will change with
rtgmaster.library, as this library will handle this using a
Screenmoderequester.
NOTE: Up to now, there was no possibility to get the Video RAM
base address under CyberGraphX OS-friendly. The only possibility
up to now was to put the function that should draw into video
RAM in a Hook function and then call a special Cybergraphics.library
function (but that function did not RETURN the base address). Inside
this Hook it was forbidden to use OS Calls. I simply
returned the base address in d7 and hoped the function that
calls the Hook won't modify d7 :) And in fact it worked...
of course this is not the fine way to do it... as you never
know if this still works with later versions of CyberGraphX...
Starting with CyberGraphX V40.60, there is a Lockbitmap function
that is better for what we want (but it is still forbidden to use
OS functions between Lock and Unlock, so you should Lock, save
the base address and Unlock then... sadly the value
for LBMI_BASEADDRESS is missing in the V40.60 Includes
of Cybergraphics... so up to now i used in this
piece of code this d7 method... as i said, with the up to
date version of Cybergraphics it works... i supplied the
Lockbitmap function too, commented out, but until i know
what value LBMI_BASEADDRESS has, i can't use it this way...
Of course, another possibility would be to do the
COMPLETE drawing code inside the Hook and not returning
the base address in this code...
NOTE: This function requires intuition.library to be opened
already...
OpenRtgScreen:
movem.l d3-d7/a2-a6,-(sp)
cmp.l #0,CGXBase
beq .Exit

```
move.l a2,a4
move.w sq_Width(a2),-(sp)
move.w sq_Height(a2),-(sp)
move.l a2,a4
move.l a3,a0
move.l sq_Buffers(a2),d4
move.l #3,d0
cmp.l d0,d4
bgt .Error1
.BufferDone:
move.l #rsCGX_SIZEOF,d0
move.l #MEMF_CLEAR,d1
move.l $4,a6
CALLSYS AllocMem
cmp.l #0,d0
beq .Error1
move.l d0,a3
move.l #136,d0
move.l #MEMF_CLEAR,d1
CALLSYS AllocMem
cmp.l #0,d0
beq .Error2
move.l d0,d7
move.l d0,a0
move.l #SA_Left,(a0)+
move.l #0,(a0)+
move.l #SA_Top,(a0)+
move.l #0,(a0)+
move.l #SA_Height,(a0)+
clr.l d0
move.w (sp)+,d0
clr.l d5
move.w d0,d5
move.l d0,d1
cmp.l #1,d4
beq .OneBuffer
cmp.l #2,d4
beq .TwoBuffer
cmp.l #3,d4
```

```
beq .ThreeBuffer
bra .Error3
.ThreeBuffer:
add.l d1,d0
add.l d1,d5
add.l d1,d5
add.l d1,d0
.TwoBuffer:
add.l d1,d0
add.l d1,d5
.OneBuffer:
move.l d0,(a0)+
move.l #SA_Width,(a0)+
clr.l d0
move.w (sp)+,d0
clr.l d3
move.w d0,d3
move.l d3,(a0)+
clr.l d0
move.w sq_Depth(a4),d0
move.l #SA_Depth,(a0)+
move.l d0,(a0)+
move.l #SA_Title,(a0)+
move.l #0,(a0)+
move.l #SA_DisplayID,(a0)+
move.l sq_ModeID(a4),a1
move.l a1,(a0)+
; No Autoscroll for CyberGraphX,
; so the user can't watch the
; DBuffering Buffers...
; No overscan, too...
; And no interleaved...
; Draggable HAS to be enabled...
; as it is per default...
move.l #SA_Quiet,(a0)+
move.l #1,(a0)+
move.l #SA_DClip,(a0)+
move.l d7,a2
add.l #128,a2
```

```
move.l a2,a1
move.w #0,(a1)+
move.w #0,(a1)+
move.w #640,(a1)+
move.w #480,(a1)+
move.l a2,(a0)+
move.l #TAG_END,(a0)+
move.l #0,(a0)+
move.l d7,a1
move.l #0,a0
move.l IntBase,a6
jsr -612(a6) ; OpenScreenTagList
cmp.l #0,d0
beq .Error3
move.l d7,a1
move.l d0,rsCGX_MyScreen(a3)
move.l #136,d0
move.l $4,a6
CALLSYS FreeMem
move.l #0,rsCGX_ActiveMap(a3)
move.l rsCGX_MyScreen(a3),a0
clr.l d0
move.w sc_Width(a0),d0
ext.l d0
move.l d0,rsCGX_Width(a3)
move.w d5,rsCGX_Height(a3)
move.l d4,rsCGX_NumBuf(a3)
move.w #0,rsCGX_Locks(a3)
move.l rsCGX_MyScreen(a3),a0
lea sc_RastPort(a0),a0
move.l rp_BitMap(a0),a0
move.l a0,rsCGX_RealMapA(a3)
move.l sq_ModeID(a4),d1
move.l d1,rsCGX_ModeID(a3)
move.l #CYBRIDATTR_PIXFMT,d0
move.l CGXBase,a6
jsr -102(a6) ; GetCyberIDAttr
cmp.l #0,d0
beq .OneByte
```

```
cmp.l #9,d0
blt .TwoBytes
cmp.l #11,d0
blt .ThreeBytes
cmp.l #10,d0
bgt .FourBytes
bra .Error4 ; Unknown Pixelformat
.OneByte:
move.l #1,rsCGX_Bytes(a3)
bra .GetMap
.TwoBytes:
move.l #2,rsCGX_Bytes(a3)
bra .GetMap
.ThreeBytes:
move.l #3,rsCGX_Bytes(a3)
bra .GetMap
.FourBytes:
move.l #4,rsCGX_Bytes(a3)
.GetMap:
move.l #0,rsCGX_MapB(a3)
move.l #0,rsCGX_MapC(a3)
;move.l rsCGX_RealMapA(a3),a0
;lea rsCGX_Tags(a3),a1
;move.l a1,a2
;move.l #LBMI_BASEADDRESS,(a2)+
;move.l #0,(a2)+
;move.l #0,(a2)+
;move.l #0,(a2)
;move.l a6,-(sp)
;move.l CGXBase,a6
;jsr -168(a6) ; LockBitMapTagList
;move.l d0,a0
;cmp.l #0,d0
;beq .Error4
;lea rsCGX_Tags(a3),a1
;move.l 4(a1),d0
;move.l d0,rsCGX_MapA(a3)
;move.l d0,d7
;jsr -174(a6) ; UnLockBitmap
```

```
;move.l (sp)+,a6
lea Hook,a0
move.l rsCGX_MyScreen(a3),a1
lea sc_RastPort(a1),a1
move.l #0,a2
move.l a6,-(sp)
move.l CGXBase,a6
jsr -156(a6) ; DoCDrawMethodTagList
move.l (sp)+,a6
move.l d7,rsCGX_MapA(a3)
move.l d7,rsCGX_FrontMap(a3)
cmp.l #1,d4
beq .OnlyOne
move.l rsCGX_Width(a3),d0
clr.l d1
move.w rsCGX_Height(a3),d1
cmp.l #2,d4
beq .Zwoo
divu #3,d1
bra .Go
.Zwoo:
divu #2,d1
.Go:
move.w d1,rsCGX_Height(a3)
mulu d1,d0
add.l d0,d7
move.l d7,rsCGX_MapB(a3)
cmp.l #2,d4
beq .OnlyOne
move.l rsCGX_Width(a3),d0
mulu d1,d0
add.l d0,d7
move.l d7,rsCGX_MapC(a3)
.OnlyOne:
move.l a3,d0
clr.l d1
clr.l d2
move.w rsCGX_Height(a3),d1
move.l d1,d2
```

```
move.l #0,rsCGX_OffA(a3)
move.l d1,rsCGX_OffB(a3)
add.l d2,d1
cmp.l #2,d4
beq .SaveMap2
move.l d1,rsCGX_OffC(a3)
.SaveMap2:
movem.l (sp)+,d3-d7/a2-a6
rts
.Exit:
move.l #0,d0
movem.l (sp)+,d3-d7/a2-a6
rts
.Error1:
tst.l (sp)+
.Raus:
movem.l (sp)+,d3-d7/a2-a6
move.l #0,d0
rts
.Error2:
move.l $4,a6
move.l a3,a1
move.l #rsCGX_SIZEOF,d0
CALLSYS FreeMem
bra .Error1
.Error3:
move.l $4,a6
move.l d7,a1
move.l #136,d0
CALLSYS FreeMem
bra .Error2
.Error4:
move.l rsCGX_MyScreen(a3),a0
move.l IntBase,a6
jsr -66(a6)
bra .Error2
Hook:
dc.l 0,0,HookFunc,0,0
HookFunc:
```

```
movem.l d0-d6,-(sp)
move.l (a1),d7 ; Address
move.l 4(a1),d1 ; X-Offset
move.l 8(a1),d2 ; Y-Offset
clr.l d3
clr.l d4
move.w 20(a1),d3 ; BytesPerRow
move.w 22(a1),d4 ; BytesPerPixel
clr.l d6
mulu d4,d1
add.l d1,d6
mulu d2,d3
add.l d3,d6
ror.l #2,d6
add.l d6,d7
movem.l (sp)+,d0-d6
rts
```

1.44 GFXStuff V1.3

GFXStuff V1.3
SwitchScreens Cyber
Now, this is a SAD chapter up to now. For CyberGraphX 3.0
there is support for DoubleBuffering announced, though...
Up to now you have the following possibilities :
1. Use the blitting functions, and blit IN FRONT, as
FAST AS POSSIBLE...
2. Use "MoveScreen". The bad thing about this call is :
- It is slower than REAL Double Buffering (probably
faster than Method 1. on most boards, though... maybe
not on ones with REAL FAST Blitters...)
- It does not return control about the program AT ONCE.
It waits, till the next VBlank happens (at least without
a WB-Emulation this Intuition.library call behaves that
way, and i assume it behaves the same way under CyberGraphX...)
- If you call TWO times MoveScreen in a short time, only the
FIRST one actually will be executed (but only if there
is a VERY short time between the two...) You might use RemakeDisplay
or something like that to FORCE the system to do the MoveScreen

(but i am not sure, if RemakeDisplay will work with CyberGraphX :))
I choose method 2. The code that i am providing does NOT call
RemakeDisplay or something like that, but it seems the time
where one MoveScreen will be ignored has to be VERY short, and
most games have to do some calculations between the MoveScreens,
so it should be no problem :)
Sure, as soon as CyberGraphX provides REAL DoubleBuffering, i will
update that section of this document...
This code requires the RtgScreen structure of the Screen being
in a0, and the buffer number between 0 and 2 in d0.
NOTE: For 640*480 the highest multiple-height Screen that is supported
by CyberGraphX, seems to be 640x1440, so do NOT try to do 4 Buffers,
3 Buffers is the Maximum, and with bigger Screenmodes maybe even
only 2 Buffers will work...
This code requires Intuition Library already opened.

```
SwitchScreens:
movem.l d2-d7/a4-a6,-(sp)
move.l IntBase,a6
move.l rsCGX_ActiveMap(a0),d1
move.l d0,rsCGX_ActiveMap(a0)
sub.l d7,d7
move.w rsCGX_Height(a0),d7
move.l rsCGX_MyScreen(a0),a0
cmp.l #0,d1
beq .BufOne
cmp.l #1,d1
beq .BufTwo
cmp.l #2,d1
beq .BufThree
bra .Exit
.BufOne:
sub.l d1,d1
cmp.l #0,d0
beq .Exit
cmp.l #1,d0
beq .ZO
cmp.l #2,d0
beq .ZT
bra .Exit
```

```
.BufTwo:
sub.l d1,d1
cmp.l #0,d0
beq .OZ
cmp.l #1,d0
beq .Exit
cmp.l #2,d0
beq .OT
bra .Exit
.BufThree:
sub.l d1,d1
cmp.l #0,d0
beq .TZ
cmp.l #1,d0
beq .TO
cmp.l #2,d0
beq .Exit
bra .Exit
.ZO:
sub.l d6,d6
sub.w d7,d6
move.w d6,d1
bra .Swap
.ZT:
sub.l d6,d6
sub.l d5,d5
move.w d7,d5
add.w d7,d5
sub.w d5,d6
move.w d6,d1
bra .Swap
.OZ:
move.w d7,d1
bra .Swap
.OT:
sub.l d6,d6
sub.w d7,d6
move.w d6,d1
bra .Swap
```

```
.TZ:
move.w d7,d1
add.w d7,d1
bra .Swap
.TO:
move.w d7,d1
.Swap:
sub.l d0,d0
move.w #0,d0
jsr -162(a6)
.Exit:
movem.l (sp)+,d2-d7/a4-a6
rts
```