

Graphics Synchronization Library

GAMEBOY Advance

黄炎中

(HuangYZ)

v2.6

Contents

1	简介	1
1.1	关于GsLIB	1
1.2	设计目标	1
1.3	联系	1
1.4	关于我	1
1.5	GsLib 简介	2
1.5.1	前言	2
1.5.2	命名由来	2
1.6	更新历史	2
1.7	感谢	3
1.8	本次更新	4
2	安装与使用	5
2.1	ARMSDT版	5
2.1.1	安装	5
2.1.2	如何开始	5
2.1.3	建议	5
2.2	GNUPRO版	6
2.2.1	安装	6
2.2.2	MAKEFILE说明	7
3	硬件相关	9
3.1	GBA硬件说明	9
4	函数参考	10
4.1	数据传送	10
4.1.1	DMA清除	10
	DmaClear	10
	DmaClearIf	10
	DmaArrayClear	11
	DmaArrayClearIf	11
4.1.2	DMA传输数据	12
	DmaCopy	12
	DmaCopyIf	12

	DmaArrayCopy	12
	DmaArrayCopyIf	13
4.1.3	DMA控制	13
	DmaWait	13
	DmaStop	14
4.2	中断控制	14
4.2.1	改写中断向量表	14
4.2.2	打开中断服务	15
4.3	系统调用	15
4.3.1	Halt	15
4.3.2	IntrWait	15
4.3.3	VBlankIntrWait	15
4.3.4	DivArm	15
4.3.5	DivRemArm	16
4.3.6	Sqrt	16
4.3.7	ArcTan	16
4.3.8	ArcTan2	16
4.3.9	LZ77UnCompWram	17
4.3.10	LZ77UnCompVram	17
4.3.11	HuffUnComp	17
4.3.12	RLUnCompWram	17
4.3.13	RLUnCompVram	18
4.3.14	Diff8bitUnFilterWram	18
4.3.15	Diff8bitUnFilterVram	18
4.3.16	Diff16bitUnFilter	18
4.4	输入控制	19
4.5	图形函数	19
4.5.1	宏	19
	GsSetMode	19
	GsGetTileSize	20
4.5.2	图像载入	20
	GsLoadBgPal	20
	GsLoadBgPal16	21
	GsLoadObjPal	21
	GsLoadObjPal16	21
4.5.3	初始化	22
	GsClearRamAll	22
	GsClearWorkRam	22
	GsClearGraphicRam	22
	GsClearVram	22
	GsClearOamRam	22
	GsClearPaletteRam	23
4.5.4	MODE 0~2专用函数	23
	GsInitBG	23
	GsLoadBgMap	23
	GsLoadBgTile	23

	GsSetBg	24
	GsBgPriority	24
	GsBgOffset	25
4.5.5	MODE1~2时BG2, BG3专用	25
	GsBgAffineInit	25
	GsBgAffine	25
	GsBgMakeAffine	26
	GsBgAffineRegInit	26
	GsSetBgLoop	26
4.5.6	MODE4专用函数	27
	GsSetPix	27
	GsWaitSync	27
	GsFlip	27
	GsSortImage	27
	GsClearBackBuffer	28
4.5.7	特效	28
	GsSetEffect	28
	GsSetMosaic	29
	GsSetAlpha	29
	GsSetBright	29
	GsCloseEffect	29
4.5.8	窗口	30
	GsCreateWin	30
	GsReleaseWin	30
4.6	文字显示	31
4.6.1	MODE4专用	31
	GsTextOut	31
4.6.2	TILE模式	31
	GsTileTextInit	31
	GsTextPix	31
	GsTileTextOut	32
	GsTileClearAll	32
4.7	精灵使用	32
4.7.1	概述	32
4.7.2	OBJ的结构	33
	GBA默认的OBJ结构	33
	GsLIB使用的OBJ结构	33
4.7.3	函数	34
	GsLoadObjChar	34
	GsLoadObjTile	34
	GsGetObjSize16	35
	GsGetObjSize256	35
	GsOamCopy	35
	GsSortSPRITE	36
	GsKillSPRITE	36
	GsAffineSPRITE	36

4.8	组合精灵	37
4.8.1	基本理论	37
	前言	37
	理论	37
	后记	39
4.8.2	结构参考	39
	GtSPRAnim_Frame	39
	GtSPRFrameList	40
	GtSPRITE	40
	GtSPRITERect	40
	GtSPRBitmap	41
4.8.3	相关函数	41
	GtLoadObjChar2D	41
	GtSPRLoadBitmap	41
	GtSPRCreate	42
	GtSPRSetPal	42
	GtSPRGetSize	43
	GtSPRSetAffine	43
	GtSPRAnimate	43
	GtSPRSetXY	44
	GtSPRMove	44
	GtSPRFlipH	45
	GtSPRRotate	45
4.9	声音调用	45
4.9.1	声音控制	45
	GsPlaySound	45
	GsSetDSoundVol	46
	GsSwitchSOUND	46
4.9.2	MusicPlayerAGB2000超快教程	46
	概念	46
	安装	47
	配置	48
	调用	48
	例子	49
4.10	计数器	49
4.10.1	硬件相关知识	49
4.10.2	相关函数	50
	GsOpenTimerIRQ	50
	GsSelectTM	50
	GsSetCount	50
	GsStartTIMER	51

5	GSLIB辅助开发工具	52
5.1	图象图形	52
5.1.1	Gfx2GBA	52
5.1.2	PalTrans	53
5.2	音乐音效	53
5.2.1	wav2gbac	53

Chapter 1

简介

1.1 关于GsLIB

本函数库内部包含2个部分:基本部分(Stand Library) 和扩展部分(Extend Library).

基本库主要是把GBA的硬件功能用函数的形式表现出来.力求正确和完整
扩展库处于基本库之上,主要强调对GBA的扩展应用.包括图象特效,调色板特效等等. 基本库和扩展库捆绑发布.

1.2 设计目标

尽可能地发挥GBA的硬件机能函数上的真正方便,简洁. 在方便的同时,可以对硬件的完全自由地操作. 支持各种模拟器,和烧录设备. 这些理念贯彻于整个GSLIB.

1.3 联系

如果有什么问题,可以和我联系

Email <mailto:hslyzd@online.sh.cn>

1.4 关于我

中文名字 黄炎中

通用文名 HuangYZ

生日 1981年9月2日

年龄 22岁

居住 上海浦东

经验 C语言 9年
C++ 4年
汇编 3年包括 (x86, mips, arm)

1.5 GsLib 简介

GsLib 是一个GameBoyAdvance的非官方开发函数库. 作者: [HuangYZ](#)

1.5.1 前言

1年前我开始着手研究GBA, 起初,看了水银的教程, 开始对GBA有了开发的兴趣.逐渐,发现GBA的硬件结构相对简单. 但是,所有函数必须自己写.然后用编译器对其交叉汇编,才能产生GBA能使用的软件.由于当时,几乎所有关于GBA开发的资料, 都是外国人写的,而且,各路函数都不同,标准也不一样(毕竟不是官方的),所以,给我们国家许多想对GBA开发软件的朋友造成了种种迷惑和障碍.

为了方便群众,使更多人加入GBA的开发阵营, 我把所掌握的关于GBA的知识写成了GsLib 希望可以帮助那些真正喜欢GBA的朋友.提高大家的编程水平.

1.5.2 命名由来

以前对PLAYSTATION开发游戏,使用PSYQ开发包,它是我所见过的最容易上手和方便的开发工具

其中有图形的函数,名为Graphics Synchronization 做的相当出色.所以,为了使我的GSLIB使用上尽

可能地和PSYQ,一样方便,所以,处于某种精神上的寄托,就命名为GsLIB for GBA.

1.6 更新历史

GsLIB 2.6	2003年4月9日 增加了摄像机的支持 增加了对非采样声音的支持 增加了对存盘文件的支持 修正了一些BUG
GsLIB 2.5.2	2003年2月9日 完善组合OBJ的若干操作。 加入了组合OBJ编辑器

GsLIB 2.5.1	2003年2月2日 修正播放mod音乐时的BUG
GsLIB 2.5.0	2003年2月1日 加入了对组合OBJ的控制 加入了调试操作的相关函数 加入了NMOD播放器,使MOD声音播放成为可能 增强了makefile的功能 修正了一些BUG
GsLIB 2.2.1	2003年1月4日 2.2版本的修正版,新增加了几个控制BG的函数 以及修正了几个BUG
GsLIB 2.2.0	2003年1月4日 加入了TILE模式下的中文显示支持 中文字模使用10X10,并且丢弃使用16X16 修正一些BUG
GsLIB 2.0.0	2002年12月16日 重新写了开发帮助文档. 重新写了MAKEFILE,以前版本用户请用新的MAKEFILE代替原来的文件 新增加了一堆函数用来辅助GBA软件的开发.请到函数使用里看详细帮助. 新增加了许多SAMPLES,展示GBA的硬件功能 修正了精灵使用的一个隐含BUG. 修正了TILE下的几个BUG
GsLIB 1.0.1	2002年11月26日 修正了烧卡的BUG,从此,使用GSLIB开发的游戏,都能烧到GBA卡里,在真正的GBA上运行了
GsLIB 1.0.0	2002年11月22日 GSLIB的第一个版本,初步把GBA的所有硬件功能都用软件表示. 尽最大努力,把函数简化.当然存在很多BUG.

1.7 感谢

水银	把我拉进了GBA的开发圈
EyesOnMe	我的好友,鼓励我开发GsLIB,给了我不少好的建议.
乐水	我的好友, BEAUTY的作者,和我经常讨论技术问题.
Mikeshi	我的好友,对MAKEFILE颇有研究,从他那里,我学到了很多.
宾宾	我最初认识的开发GBA的朋友.<特训>的作者.
ROC	很强的程序员,对GBA硬件的研究很深入.帮了我不少忙.
ES_ZETA	我的好友,对GSLIB做出了许多贡献.
TLP	感谢对GSLIB的测试和支持
NEIMOD	NMOD的开发者,为我们创造了这么好的MOD播放器
ARM公司的所有成员	没有他们,就没有GBA,更没有我的GSLIB.
还有... 你	

1.8 本次更新

1. 增加了摄像机技术,从而可以自由地使用大地图
2. 对SOUND 1~4的支持.
3. 增加了对存盘的支持

Chapter 2

安装与使用

2.1 ARMSDT版

2.1.1 安装

在我主页下载最新版本的GsLIB后,双击后自动安装.安装完了,程序会自动设置路径

2.1.2 如何开始

在你的安装目录下打

GSGO 回车

就可以开始使用了.

2.1.3 建议

看看SAMPLES目录下的几个演示程序.

Helloworld MODE4下写文字的演示

BGDEMO1 在MODE0下普通BG的操作演示

BGDEMO2 在MODE1下对旋转BG进行操作

SPRITE 演示如何操作OBJ

IRQ 展示如何操作中断(借用SPRITE的例子)

FADE 特效演示

MOSAIC 马塞克特效演示
BGPAL 在MODE0下,混合色深BG的DEMO
MODE3 MODE3下显示图片的演示
MODE4 MODE4下显示图片的演示
OBJ-MAP 展示精灵和MAP的同时显示

2.2 GNUPRO版

2.2.1 安装

把GNUPRO ¹ 和

ARMELF_000512

正确安装后, 就可以安装GSLIB GNU了.

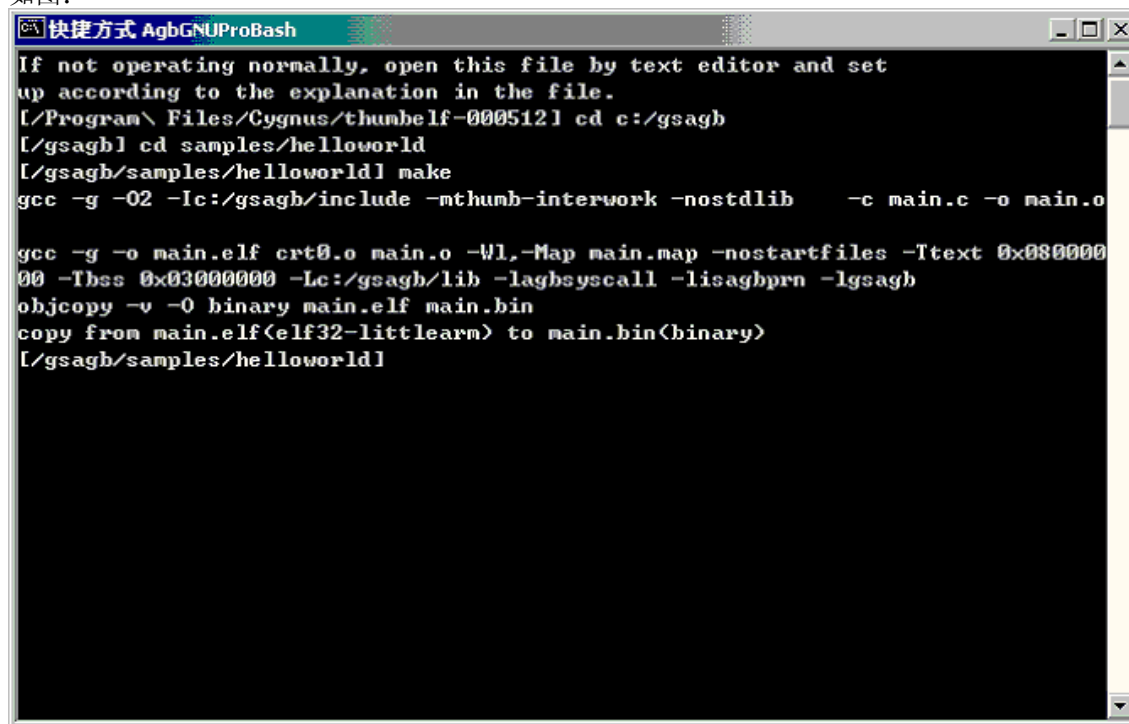
强烈建议将GSAGB装在C盘根目录下, 因为许多SAMPLE的MAKE文件里都是以C盘为默认路径的。

把GSAGB.ZIP解包到c:\gsagb 后, 进入GNU环境, 键入:

```
cd c:/gsgba/samples
cd helloworld
make
```

¹ GNUPRO是编译器, 而armelf.000512 是ARM芯片对应的插件, 默认安装路径是在 C:\Program Files\Cygnus

如图:



```
快捷方式 AgbGNUProBash
If not operating normally, open this file by text editor and set
up according to the explanation in the file.
[/Program\ Files\Cygnus\thumbelf-000512] cd c:/gsagb
[/gsagb] cd samples/helloworld
[/gsagb/samples/helloworld] make
gcc -g -O2 -Ic:/gsagb/include -mthumb-interwork -nostdlib -c main.c -o main.o

gcc -g -o main.elf crt0.o main.o -Wl,-Map main.map -nostartfiles -Ttext 0x080000
00 -Thss 0x03000000 -Lc:/gsagb/lib -lagbsyscall -lisagbprn -lgsagb
objcopy -v -O binary main.elf main.bin
copy from main.elf(elf32-littlearm) to main.bin(binary)
[/gsagb/samples/helloworld]
```

这样, 就可以编译演示程序helloworld 了。看到上面的图, 就说明正确编译了。

如果成功编译, 则表示编译器和GSLIB都被正确安装了。否则, 请检查是否安装正确。

2.2.2 MAKEFILE说明

GSLIB GNUPRO版本, 取消了原来ARMSDT版本中的MAKE系统。取而代之的是标准的makefile 这就意味着, 大家要自己写makefile

别怕, makefile文件很简单的

参照helloworld, 一共是4个文件, makefile, gasdepend crt0.s rom_header.s

其中, 我们只关心makefile而不用理会另外 3 个。但是这 4 个文件必须同时存在于一个目录下。

简单说明一下makefile 前 3 行, 是这样的,

```
.SFILES = crt0.s
.CFILES = main.c
.OFILES = $(.SFILES:.s=.o) $(.CFILES:.c=.o)
```

其中.SFILES 表示汇编程序文件序列。.CFILES 表示 C 语言源文件序列.OFILES 表示目标文件 O 序列我们一般只要编辑CFILES就可以了。举个例子，假设有make.c 在当前目录下gfx1.c gfx2.c gfx3.c 在当前目录,gfx目录下这 4 个文件需要一起编译。那么你可以把.CFLAG写成

```
.CFLAG = main.c gfx/gfx1.c gfx/gfx2.c gfx/gfx3.c
```

如果需要换行的话必须使用\

比如

```
.CFLAG =main.c gfx/gfx1.c gfx/gfx2.c \  
gfx/gfx3.c
```

最后存盘make即可编译。

Chapter 3

硬件相关

3.1 GBA硬件说明

注意！

这个文件不是GSLIB的文档资料。

这里只能给出一个连接。

<http://www.cs.rit.edu/~tjh8300/CowBite/CowBiteSpec.htm>

Chapter 4

函数参考

4.1 数据传送

4.1.1 DMA清除

DmaClear

DmaClear(DmaNo, Data, Destp, Size, Bit); **参数:**

DmaNo	DMA通道号0~3
Data	清除后数据
Destp	目标地址
Size	大小
Bit	宽度16 或32

说明:

使用DMA对Destp开始地址到Destp + size 之间的数据清成Data
比如说,使用

```
DmaClear(3 ,0 ,0x6000000,512,16);
```

可以把从0x6000000开始的512个字节清成0

DmaClearIf

DmaClearIf(DmaNo, Data, Destp, Size, Bit);

参数:

DmaNo	DMA通道号0~3
Data	清除后数据
Destp	目标地址
Size	大小
Bit	宽度16 或32

说明:

使用DMA对Destp开始地址到Destp + size 之间的数据清成Data
完成后向CPU发出中断请求

DmaArrayClear

DmaArrayClear(DmaNo, Data, Destp, Bit);

参数:

DmaNo	DMA通道号0~3
Data	清除后数据
Destp	目标地址
Bit	宽度16 或32

说明:

使用DMA对Destp数组清成Data

DmaArrayClearIf

DmaArrayClearIf(DmaNo, Data, Destp, Bit);

参数:

DmaNo	DMA通道号0~3
Data	清除后数据
Destp	目标数组
Bit	宽度16 或32

说明:

使用DMA对Destp数组清成Data
完成后向CPU发出中断请求

4.1.2 DMA传输数据

DmaCopy

DmaCopy(DmaNo, Srcp, Destp, Size, Bit);

参数:

DmaNo	DMA通道号0~3
Srcp	源地址
Destp	目标地址
Size	传送字节数
Bit	宽度16 或32

说明:

使用DMA把Srcp地址开始的数据传输Size个字节到Destp.

DmaCopyIf

DmaCopyIf(DmaNo, Srcp, Destp, Size, Bit);

参数:

DmaNo	DMA通道号0~3
Srcp	源地址
Destp	目标地址
Size	传送字节数
Bit	宽度16 或32

说明:

使用DMA把Srcp 地址开始的数据传输Size个字节到Destp
完成后向CPU发出中断请求

DmaArrayCopy

DmaArrayCopy(DmaNo, Srcp, Destp, Bit);

参数:

DmaNo	DMA通道号0~3
Srcp	源地址

Destp	目标地址
Bit	宽度16 或32

说明:

使用DMA把Srcp 数组传输到Destp地址
一般用于一个数组向内存传输数据.

DmaArrayCopyIf

DmaArrayCopyIf(DmaNo, Srcp, Destp, Bit);

参数:

DmaNo	DMA通道号0~3
Srcp	源地址
Destp	目标地址
Bit	宽度16 或32

说明:

使用DMA把Srcp数组传输到Destp地址
一般用于一个数组向内存传输数据.
完成后向CPU发出中断请求

4.1.3 DMA控制

DmaWait

DmaWait(DmaNo);

参数:

DmaNo 通道号0~3

说明:

等待上一次DMA 操作结束
这个是和一次DMA操作相对应的.

请注意,如果执行这句前,没有使用过任何DMA操作,那么,后果可能会导致死循环

DmaStop

```
DmaStop(DmaNo);
```

参数:

DmaNo 通道号0~3

说明:

强行停止上一次DMA操作

4.2 中断控制

使用中断包括以下2个步骤

4.2.1 改写中断向量表

在AgbMain的外面写中断向量表

提供个例子

```
const IntrFuncp IntrTable[14] =
{
    dummy,    // V Blank 中断
    dummy,    // H Blank中断
    dummy,    // V Counter中断
    dummy,    // Timer 0中断
    dummy,    // Timer 1中断
    dummy,    // Timer 2中断
    dummy,    // Timer 3中断
    dummy,    // Serial communication中断
    dummy,    // DMA 0中断
    dummy,    // DMA 1中断
    dummy,    // DMA 2中断
    dummy,    // DMA 3中断
    dummy,    // Key中断
    dummy,    // Cassette中断
};
```

4.2.2 打开中断服务

如果要使用响应中断的话，就在对应位置改写成你的中断服务程序。就完成第1步了。

在主程序体开始的时候，写入GsOpenIRQ()就可以打开：

V Blank 中断，Key 中断，DMA 0，DMA 1，DMA 2，DMA 3 中断。

4.3 系统调用

4.3.1 Halt

宏 Halt()

说明 CPU停机，直到有中断产生才继续

4.3.2 IntrWait

函数 void IntrWait(u8 InitCheckClear, u16 IntrFlags);

参数 **InitCheckClear** 中断检查位

IntrFlags 等待此中断的发生

说明 等待中断的发生，期间CPU处于等待状态

4.3.3 VBlankIntrWait

函数 void VBlankIntrWait(void)

说明 等待VBLANK中断的发生

4.3.4 DivArm

函数 s32 DivArm(s32 Denom, s32 Number);

说明 除法运算

返回值 =Number/Denom

寄存器r0 返回Number/Denom

寄存器r1 返回Number%Denom

参数 **Number** 被除数

Denom 除数

4.3.5 DivRemArm

函数 s32 DivRemArm(s32 Denom, s32 Number);

说明 取模运算

返回值 = Number % Denom

寄存器r0 返回Number/Denom

寄存器r1 返回Number%Denom

Number 被除数

Denom 除数

4.3.6 Sqrt

函数 u16 Sqrt(u32 X);

说明 求平方根运算

4.3.7 ArcTan

函数 u16 ArcTan(s16 Tan);

说明 求反三角函数正切TAN

参数 Tan 符号位1位
 整数部分1位
 小数部分14位

返回值 ($-\pi/2 < \text{角度} < \pi/2$)
 0xC000 至0x4000

4.3.8 ArcTan2

函数 u16 ArcTan2(s16 X, s16 Y);

说明 根据三角型的两条直角边求反三角函数正切TAN

参数 返回0x0000 ~ 0xFFFF
 ($0 < \text{角度} < 2\pi$)

4.3.9 LZ77UnCompWram

函数 void LZ77UnCompWram(void *Srcp, void *Destp);

说明 由Agbcomp -l 参数生成的LZ77压缩数据文件的解压缩函数，把从源地址开始的压缩2进制数据解压缩到WRAM内目标地址

参数 void *Srcp 源地址
void *Destp WRAM内目标地址

4.3.10 LZ77UnCompVram

函数 void LZ77UnCompVram(void *Srcp, void *Destp);

说明 由Agbcomp -l 参数生成的LZ77压缩数据文件的解压缩函数，把从源地址开始的压缩2进制数据解压缩到VRAM内目标地址

参数 void *Srcp 源地址
void *Destp VRAM内目标地址

4.3.11 HuffUnComp

函数 void HuffUnComp(void *Srcp, void *Destp);

说明 由Agbcomp -h 参数生成的哈夫曼树压缩数据文件的解压缩函数，把从源地址开始的压缩2进制数据解压缩到目标地址

参数 void *Srcp 源地址
void *Destp 目标地址

4.3.12 RLUnCompWram

函数 void RLUnCompWram(void *Srcp, void *Destp);

说明 由Agbcomp -r 参数生成的RUN LENGTH压缩数据文件的解压缩函数，把从源地址开始的压缩2进制数据解压缩到目标地址

参数 void *Srcp 源地址
void *Destp 目标地址 只能在WRAM内，不可在VRAM内

4.3.13 RUnCompVram

函数 void RUnCompVram(void *Srcp, void *Destp);

说明 由Agbcomp -r 参数生成的RUN LENGTH压缩数据文件的解压缩函数，把从源地址开始的压缩2进制数据解压缩到目标地址

参数 void *Srcp 源地址

void *Destp 目标地址 只能在VRAM内，不可在WRAM内

4.3.14 Diff8bitUnFilterWram

函数 void Diff8bitUnFilterWram(void *Srcp, void *Destp);

说明 由Agbcomp -d 参数生成的已过滤文件的解压缩函数，把从源地址开始的压缩2进制数据解压缩到目标地址

参数 void *Srcp 源地址

void *Destp 目标地址 只能在WRAM内，不可在VRAM内

4.3.15 Diff8bitUnFilterVram

函数 void Diff8bitUnFilterVram(void *Srcp, void *Destp);

说明 由Agbcomp -d 参数生成的已过滤文件的解压缩函数，把从源地址开始的压缩2进制数据解压缩到目标地址

参数 void *Srcp 源地址

void *Destp 目标地址 能在VRAM内，不可在WRAM内

4.3.16 Diff16bitUnFilter

函数 void Diff16bitUnFilter(void *Srcp, void *Destp);

说明 由Agbcomp -d 参数生成的已过滤文件的解压缩函数，把从源地址开始的压缩2进制数据解压缩到目标地址

参数 void *Srcp 源地址

void *Destp 目标地址

4.4 输入控制

包含2个部分，连续按键 和一次性按键

连续按键部分用Cont 检测, 按一个键不放，就会有连续的响应

一次性按键部分用Trg检测

标志:	U_KEY	方向上
	D_KEY	方向下
	L_KEY	方向左
	R_KEY	方向右
	START_BUTTON	按键START
	SELECT_BUTTON	按键SELECT
	L_BUTTON	按键L
	R_BUTTON	按键R
	A_BUTTON	按键A
	B_BUTTON	按键B

函数: KeyRead();

用途: 得到按下的键
先用KeyRead()得到按下的键
然后用Cont或Trg 与标志做与运算

比如

```
void querykey()
{
    KeyRead();
    If (Cont & U_KEY) // 连续检测是否有方向上按下
    .....;
    If (Trg & A_BUTTON) // 检测一次是否有A按键按下
    .....;
}
```

4.5 图形函数

4.5.1 宏

GsSetMode

GsSetMode(mode);

参数:

mode 有如下标志

显示模式

MODE_0
MODE_1
MODE_2
MODE_3
MODE_4
MODE_5

用来表示OBJ里活动块的排列方式

DISP_OBJ_CHAR_1D_MAP
DISP_OBJ_CHAR_2D_MAP

说明:

mode是显示模式| 排列方式

GsGetTileSize

GsGetTileSize(tile); **说明:**

得到tile数组的BLOCK数量（每个BLOCK是2KB）

4.5.2 图像载入

GsLoadBgPal

void GsLoadBgPal(const u16 *pal,int count);

参数:

pal 载入调色盘的地址

count 载入字节数

说明:

调色盘函数

载入BG调色盘256色

GsLoadBgPal16

```
void GsLoadBgPal16(const u16 *pal,int no);
```

参数:

pal 载入调色盘的地址

no 目标段0~15

说明:

载入BG调色盘16色到目标段

GsLoadObjPal

```
void GsLoadObjPal(const u16 *pal,int count);
```

参数:

pal 载入调色盘的地址

count 载入字节数最多256

说明:

载入OBJ调色盘256色

GsLoadObjPal16

```
void GsLoadObjPal16(const u16 *pal,int no);
```

参数:

pal 载入调色盘的地址

no 目标段0~15

说明:

载入OBJ调色盘16色到目标段

4.5.3 初始化

GsClearRamAll

```
void GsClearRamAll( void );
```

说明:

初始化内存

GsClearWorkRam

```
void GsClearWorkRam( void );
```

说明:

初始化WRAM内存

GsClearGraphicRam

```
void GsClearGraphicRam( void );
```

说明:

初始化图象内存

GsClearVram

```
void GsClearVram( void );
```

说明:

初始化VRAM内存

GsClearOamRam

```
void GsClearOamRam( void );
```

说明:

初始化OAM内存

GsClearPaletteRam

```
void GsClearPaletteRam( void );
```

说明:

初始化调色盘内存

4.5.4 MODE 0~2专用函数

GsInitBG

```
void GsInitBG();
```

说明:

初步始化BG,在开始使用MAP前必须做的一步。

GsLoadBgMap

```
void GsLoadBgMap(const u16 *map,u16 size,u8 locate);
```

参数:

const u16 *map MAP数据的首地址

u16 size MAP数据的大小

u8 locate 载入的目标地址, 0~31

说明:

把MAP 载入VRAM, locate 是0~31

GsLoadBgTile

```
void GsLoadBgTile(const u8 *tile,u16 size,u8 locate);
```

参数:

const u8 *tile tile数据的首地址, 一般在ROM内

u16 size tile数据的大小

u8 locate 载入的目标地址, 0~3

说明:

把tile载入VRAM的locate区

GsSetBg

```
void GsSetBg(int bgno,int size,int colormode,int charbase,int scbase,int mosaic);
```

参数:

bgno BG层号0~3

size 大小有以下几个标志

对于TEXT的BG:

Gs_BG_TEXT_SIZE_256x256

Gs_BG_TEXT_SIZE_512x256

Gs_BG_TEXT_SIZE_256x512

Gs_BG_TEXT_SIZE_512x512

对于旋转BG:

Gs_BG_AFFINE_SIZE_128x128

Gs_BG_AFFINE_SIZE_256x256

Gs_BG_AFFINE_SIZE_512x512

Gs_BG_AFFINE_SIZE_1024x1024

Colormode 色彩模式, 以下几个标志

Gs_BG_COLOR16 表示此BG是16色的

Gs_BG_COLOR256 表示此BG是256色的

Charbase 活动块的开始位置0~3

Scbase MAP数据的开始位置0~31

Mosaic 是否有马塞克

说明:

设置BG模式

使用MODE 0~2时对BG进行设置。

GsBgPriority

```
void GsBgPriority(int bgno,int prio);
```

参数:

bgno 层号0~3
prio 优先级（0为最高）

说明:

设置BG优先级

GsBgOffset

```
void GsBgOffset(int bgno, u16 x, u16 y);
```

参数:

bgno 层号0~3
x 横向卷轴移动量
y 纵向卷轴移动量

说明:

对bgno背景进行卷轴操作

4.5.5 MODE1~2时BG2，BG3专用

GsBgAffineInit

```
void GsBgAffineInit(GsBGAffine *aff);
```

参数:

GsBGAffine结构的指针aff，此指针指向一个BG变化对象

说明:

旋转BG专用，初始化这个BG变化对象，一般将此函数放于主程序开始

GsBgAffine

```
void GsBgAffine(int bgno, GsBGAffine *aff);
```

参数:

int bgno 选择一个旋转BG，只能是2或3
GsBGAffine结构的指针aff 此指针指向一个BG变化对象

说明:

aff里的数据改变aff里的系统PA，PB，PC，PD 创建变化数据

GsBgMakeAffine

```
void GsBgMakeAffine(int bgno,GsBGAffine *aff);
```

参数:

int bgno 选择一个旋转BG，只能是2或3
GsBGAffine结构的指针aff 此指针指向一个BG变化对象

说明:

使aff的对象影响硬件的一些积存器，从而使变化体现在屏幕上

GsBgAffineRegInit

```
void GsBgAffineRegInit(int bgno);
```

参数:

int bgno 旋转BG，只能是2或3

说明:

使aff的对象影响硬件的一些积存器，从而使变化体现在屏幕上

GsSetBgLoop

```
void GsSetBgLoop(int bgno,u8 flag);
```

参数:

bgno 旋转BG号
flag 允许/不允许

说明:

使aff的对象影响硬件的一些积存器，从而使变化体现在屏幕上

4.5.6 MODE4专用函数

GsSetPix

```
void GsSetPix(u8 x,u8 y,u16 color);
```

参数:

x,y 屏幕上的位置
u16 color 调色盘中的编号

说明:

把调色盘中编号为color的颜色显示到屏幕上画点,在屏幕外的点不会被显示: void GsSetPixLimit(u8 x,u8 y,u16 color);

GsWaitSync

```
void GsWaitSync(void);
```

说明:

等待同步信号, MODE4里必须把这句话放在主循环里

GsFlip

```
void GsFlip();
```

说明:

翻转前后缓冲, MODE4里必须把这句放在主循环里

GsSortImage

```
void GsSortImage1(const u16 *IndexData, unsigned long size);
```

参数:

const u16 *IndexData 图象的开始地址
unsigned long size 图象的大小

¹MODE3中也可使用

说明:

显示图象，在MODE4里，如果要连续显示一张图片的话，必须把这句话放在主循环里

GsClearBackBuffer

```
void GsClearBackBuffer(void);
```

说明:

清除后背缓冲

4.5.7 特效

GsSetEffect

```
void GsSetEffect(u16 tg1,u16 tg2);
```

参数:

tg1 是第一个目标,有以下标志

BLD_BG0_1ST	// Select BG0 1st Pixel
BLD_BG1_1ST	// Select BG1 1st Pixel
BLD_BG2_1ST	// Select BG2 1st Pixel
BLD_BG3_1ST	// Select BG3 1st Pixel
BLD_OBJ_1ST	// Select OBJ 1st Pixel
BLD_BD_1ST	// Select Background Color 1st Pixel
BLD_1ST_ALL	// Select All 1st Pixel

tg2 是第二个目标,有以下标志:

BLD_BG0_2ND	// Select BG0 2nd Pixel
BLD_BG1_2ND	// Select BG1 2nd Pixel
BLD_BG2_2ND	// Select BG2 2nd Pixel
BLD_BG3_2ND	// Select BG3 2nd Pixel
BLD_OBJ_2ND	// Select OBJ 2nd Pixel
BLD_BD_2ND	// Select Background color 2nd Pixel
BLD_2ND_ALL	// Select All 2nd Pixel

说明:

GBA硬件上把tg1和tg2作逻辑运算操作.从而使达到某种效果.
比如: 选择BG1 和OBJ做特效的话,可以执行以下操作

GsSetEffect(BLD_BG0_1ST, BLD_OBJ_2ND);

GsSetMosaic

void GsSetMosaic(u8 bh,u8 bv,u8 oh,u8 ov);

参数:

bh,bv 背景H方向和V方向的MOSAIC参数0~31

oh,ov OBJ在H方向和V方向的MOSAIC参数0~31

说明:

设置马赛克特效

GsSetAlpha

void GsSetAlpha(u8 ap_eva,u8 ap_evb);

参数:

ap_eva

ap_evb

说明:

设置ALPHA混合度,可以修改这2个数,得到ALPHA的效果

GsSetBright

void GsSetBright(s16 level);

说明:

level是亮度, 暗到亮为-15 ~ 15

-15是黑色, 15是白色

GsCloseEffect

void GsCloseEffect();

说明:

关闭所有特效

4.5.8 窗口

GsCreateWin

```
void GsCreateWin(u8 winno,u8 x1,u8 y1,u8 x2,u8 y2,u8 OBJ_WINDOW,u8  
WINDOW01,u8 fx0_on, u8 fx1_on);
```

参数:

winno=0 表示窗口0

winno=1 表示窗口1

x1,y1,x2,y2 表示窗口大小0~255

OBJ_WINDOW 表示哪些OBJ窗口的有效范围.

WINDOW01 表示哪些WINDOW0,1的有效范围

有以下标志

WIN_BG0_ON	BG0 ON
WIN_BG1_ON	BG1 ON
WIN_BG2_ON	BG2 ON
WIN_BG3_ON	BG3 ON
WIN_OBJ_ON	OBJ ON

Fx0_on 表示是否允许WINDOW0特效ON/OFF

Fx1_on 表示是否允许WINDOW1特效ON/OFF

说明:

创建一个窗口

GsReleaseWin

```
void GsReleaseWin();
```

说明:

释放所有窗口

4.6 文字显示

4.6.1 MODE4专用

GSLIB包括ASCII码的128个标准字符，和GB2312的所有中文字因为字库比较大，所以，在需要时，在头部加上#include“GsTEXT.h”即可使用文字输出。

GsTextOut

函数 GsTextOut(int x,int y,u16 color,char *s)

参数 x,y 在LCD的x, y位置开始显示

color BG调色板中的颜色编号。

***s** 字符串，可以是中文，也可以是字符。用\n表示换行

说明 文字输出，在XY位置输出*S里的文字，颜色是COLOR

4.6.2 TILE模式

GsTileTextInit

函数 void GsTileTextInit(u8 bg, u8 dist,u8 blockLoc,u8 PalNo);

参数 u8 bg 表示初始化的BG

u8 dist VRAM段地址(0,1,2,3) 最终结果是导致0x6000000 + 4000*dist

u8 blockLoc 表示MAP数据的载入位置(0~31)

u8 PalNo (选择16色调色盘的组号 (0~16))

说明 在TILE模式下显示文字的初始化函数，决定了VRAM的分配方式。完成后，这个BG是256X256X16模式。文字颜色为16色调色盘中的一个颜色，

可以用PALNO来选择16组调色版里的一组。而且，这句话必须放在GsInitBG()之后。切记

EXAMPLE GsTileTextInit(3,3,6,0);

表示，BG3为文字层，对应的是第3区

0x6000000+dict*0x4000=0x600c000 这里强烈推荐第2个参数为3

把MAP数据放在第6个块内使用第0组调色盘。

GsTextPix

函数 void GsTextPix(int x,int y,u16 color);

参数 int x,y坐标

说明 在刚才定义的BG上画点

GsTileTextOut

函数 void GsTileTextOut(int x,int y,const char *str,u16 color);

参数 int x,y 坐标 注意, y必须在0~120之内

const char *str 字符串

u16 color 颜色

说明 显示文字, 可以是ASCII字符或中文可以用'\n'来换行

GsTileClearAll

函数 void GsTileClearAll();

说明 清除这个BG的所有数据

4.7 精灵使用

4.7.1 概述

GBA里根据屏幕模式不同,在VRAM里的OBJ内存大小也不一样

在MODE 0,1,2 下OBJRAM是32KB

在MODE 3,4,5 下OBJRAM是16KB

在MODE 0,1,2下

如果载入的图形是16色的,那么OBJRAM就可以被分割成班1024个区

如果载入的图形是256色的,那么OBJRAM就可以被分割成班512个区

在MODE 3,4,5下

如果载入的图形是16色的,那么OBJRAM就可以被分割成班512个区

如果载入的图形是256色的,那么OBJRAM就可以被分割成班256个区

可以用GsLoadObjTile函数来载入

4.7.2 OBJ的结构

GBA默认的OBJ结构

```
typedef struct s_OamData{

    u32 VPos:8;                //Y方向坐标
    u32 AffineMode:2;          // 旋转放缩方式
    u32 ObjMode:2;             //OBJ模式
    u32 Mosaic:1;              //是否允许马赛克
    u32 ColorMode:1;           //16 colors/256 colors选择
    u32 Shape:2;               //OBJ形状,包括以下标志
                                Gs_OAM_SQUARE      OBJ为方形
                                Gs_OAM_HRECT        OBJ为横长方形
                                Gs_OAM_VRECT        OBJ为竖长方形

    u32 HPos:9;                //Y方向坐标
    u32 AffineParamNo_L:3;     //选择一个变换参数号0~31
    u32 HFlip:1;               //H方向翻转0 无效, 1 翻转
    u32 VFlip:1;               //V方向翻转0 无效, 1 翻转
    u32 Size:2;                //OBJ大小包括以下标志:
                                Gs_OAM_SMALL
                                Gs_OAM_MIDDLE
                                Gs_OAM_BIG
                                Gs_OAM_LARGE

    u32 CharNo:10;             //CHAR号,在VRAM中OBJRAM里的区号
    u32 Priority:2;             //优先级0~3
    u32 Pltt:4;                //在OBJ 调色盘里选择一个调色盘编号0~16
    u16 AffineParam;           //仿射变换参数系统使用

}s_OamData;
typedef volatile s_OamData SPRITE;
```

这里列出SHAPE和SIZE的关系

	Gs_OAM_SMALL	Gs_OAM_MIDDLE	Gs_OAM_BIG	Gs_OAM_LARGE
Gs_OAM_SQUARE	8 x 8	16 x 16	32 x 32	64 x 64
Gs_OAM_HRECT	16 x 8	32 x 8	32 x 16	64 x 32
Gs_OAM_VRECT	8 x 16	8 x 32	16 x 32	32 x 64

GsLIB使用的OBJ结构

```
typedef struct GsSPRITE
{

    u8 ID:7;                  //在OAM中的编号
```

```

        u8 Flag:1; //是否已经被画到屏幕上
        SPRITE obj; //引用OAM的结构

    }GsSPRITE;

typedef struct GsSPRITEAffine
{
    s16 RatioX; //x方向上的放缩
    s16 RatioY; //y方向上的放缩
    u16 Theta; //旋转角度(0 - 255) << 8

}GsSPRITEAffine;

```

4.7.3 函数

GsLoadObjChar

```
GsLoadObjChar(src,mode);
```

参数:

src 源数组名地址,是用GFX2GBA或其他转换软件生成的数组

mode 传输位置, 包括以下几个标志

```

    OBJ_MODE0_VRAM
    OBJ_MODE1_VRAM
    OBJ_MODE2_VRAM
    OBJ_MODE3_VRAM
    OBJ_MODE4_VRAM
    OBJ_MODE5_VRAM

```

这是和显示方式有关的,几个默认载入位置, 当然,也可以是其他地址

说明:

把OBJ的数据传输到OBJRAM的mode 位置。最最直接的方法.

GsLoadObjTile

```
void GsLoadObjTile(const u8 *src, u16 size, int loc,int COLORDEP);
```

参数:

const u8 *src 数组名地址
u16 size 这个数组的大小
int loc 目标区位置, 请注意限制!!
 MODE 0,1,2下 16色最多1024个区
 256色最多512个区
 MODE 3,4,5下 16色最多512个区
 256色最多256个区
int COLORDEP 表示要载入的色深有2个标志:
 Gs_OAM_COLOR16 表示载入数据为16色
 Gs_OAM_COLOR256 表示载入数据为16色

说明:

将在ROM里的OBJ数据传输到OBJRAM里,LOC位置

GsGetObjSize16

GsGetObjSize16(src) 宏

参数:

src 源TILE数据数组

说明:

得到16色模式的OBJ数据所占的BLOCK数量 (每个BLOCK占2KB)

GsGetObjSize256

GsGetObjSize256(src) 宏

说明:

得到256色模式的OBJ数据所占的BLOCK数量 (每个BLOCK占2KB)

GsOamCopy

void GsOamCopy();

说明:

更新OAM区

只要使用精灵,就必须把这句话放在循环中.否则所有对精灵的操作都无效

GsSortSPRITE

```
void GsSortSPRITE(GsSPRITE *p);
```

参数:

GsSPRITE *p 一个GsSPRITE结构的精灵

说明:

把p所指的OBJ复制到屏幕上,如果已经存在,则在原来的位置上更新数据

GsKillSPRITE

```
void GsKillSPRITE(GsSPRITE *p);
```

参数:

GsSPRITE *p GsSPRITE结构的指针

说明:

如果p所指的OBJ在OAM之内,则把p所指的OBJ在OAM里删除

GsAffineSPRITE

```
void GsAffineSPRITE(GsSPRITE *p,GsSPRITEAffine *affsrc);
```

参数:

GsSPRITE *p 一个精灵,必须有可选转/放缩属性

GsSPRITEAffine *affine 仿射结构可以在这个里面设置放缩和旋转

说明:

设置仿射变换参数使这个精灵放缩或旋转

4.8 组合精灵

4.8.1 基本理论

前言

在游戏业发展的20年里，2D游戏迅速发展，相对应的技术也不断提高，使我们现在的2D游戏十分地“眩”。游戏机上很早就硬件支持了“精灵”这种结构，通过硬件加速，使我们看到的游戏中的角色活动十分流畅，变化丰富。但是，硬件资源相对的匮乏，存储器容量的限制，CPU的速度，无法装载更多的精灵动画帧数据。限制了发展。所以，2D中的组合精灵在这情况下出现。有效地解决了硬件和软件的矛盾。这个技术，在许多游戏中是很常见的。特别是KONAMI的恶魔城系列，把此技术发挥到极限。

理论

组合OBJ把原来的一个角色，分解为许多子“部分（part我认为应该这么叫）”而这些part都是一个硬件上独立的精灵，而且都有相对应的动画帧序列。也就是说，可以动画。所以，反过来说，组合OBJ就是把一些相互独立的精灵，有序地统一起来。成为一个整体。

因为这些独立的精灵一般都有硬件上的特效支持（旋转，放缩），所以一个组合OBJ也具有一般精灵的所有性质。下面简单地说一下。

1. 我认为的组合OBJ的基本结构

和一般的精灵一样，应该包含

x,y 位置，旋转角，V，H翻转,X, Y 放缩数据,PART部分数据

写成C语言结构就是：

```
typedef struct ComObj
{
    int x,y;
    int Theta;
    int V_FLIP;
    int H_FLIP;
    int ScaleX,ScaleY;
    SPRITE *part;           //子精灵
    int partCount;          //子精灵的数量
}ComObj;
```

2. 初始化

子精灵的X, Y位置是存放于一个偏移数据中, 描述的是与组合OBJ位置的相对距离。所以, 初始的时候, 这些子精灵的位置, 就是组合OBJ的位置+ 数据中的距离。(ox,oy)

3. 组合OBJ的平移

这个是最容易的了, 如果组合OBJ的需要移动的话, 最终体现在改变子精灵的位置上。比如组合OBJ的X增加了dx, 那么所有子精灵的X也要加dx。对Y也一样。强烈建议使用dx, dy来对真实坐标进行操作, 不要直接改写真实坐标。

4. 组合OBJ的反转。

为了实现这个功能, 必须得到子精灵的大小, 长sweight和高sheight。因为在偏移数据中存放的是相对于组合OBJ位置的相对偏移量, 所以, 如果做了FLIP后, 所有子精灵也要做FLIP而且偏移量也要取相反值

假设组合OBJ位置位X, Y那么原来的子精灵位置应该是

$$x + ox, y + oy$$

反转后, 这个位置应该是:

$$\text{HFLIP: } x - ox - \text{sweight}$$

$$\text{VFLIP: } y - oy - \text{sheight}$$

按照上面的公式, 就能完成反转。

5. 旋转/放缩

这个有点难度。

请先看如下公式:

$$\begin{Bmatrix} x_2 \\ y_2 \end{Bmatrix} = \begin{Bmatrix} A & B \\ C & D \end{Bmatrix} \begin{Bmatrix} x_1 - x_0 \\ y_1 - y_0 \end{Bmatrix} + \begin{Bmatrix} x_0 \\ y_0 \end{Bmatrix}$$

$$A = \frac{1}{\alpha} \cos \theta, B = \frac{1}{\alpha} \sin \theta, C = -\frac{1}{\beta} \sin \theta, D = \frac{1}{\beta} \cos \theta$$

$$x_2 = A(x_1 - x_0) + B(y_1 - y_0) + x_0$$

$$y_2 = C(x_1 - x_0) + D(y_1 - y_0) + y_0$$

其中,

α 表示在X方向上的放缩, 在结构是ScaleX成员

β 表示在Y方向上的放缩, 在结构是ScaleY成员

θ 旋转角度结构中是Theta

x_0, y_0 表示旋转放缩参考点。是对子精灵的, 一般需
要看实际情况指定

$x1, y1$ 表示旧位置

$x2, y2$ 表示新位置

从上面可以看出，旋转和放缩的效果是可以在一个公式里完成的。

一般地说，硬件已经支持了单个OBJ的旋转和放缩效果，所以对于组合OBJ来说，只要改变子精灵的位置就可以了。当然也是用这个公式。

以上就是对组合OBJ的一般整体操作。当然，也可以针对其中某一个子精灵单独做一些操作，（平移，旋转，放缩，动画等等）

后记

组合OBJ的出现就是为了解决内存空间和动画的矛盾。虽然复杂了点，但是带来的效果却是很不错的。日本的许多游戏厂商对这个技术玩的很转。而我没见到中国的游戏厂商使用过。（可能是我孤陋寡闻）。虽然3D游戏这几年横行，但是2D的许多技巧是3D的基础。

我们应该把这个技术学好。促进我国的游戏编程整体提高。²

4.8.2 结构参考

GtSPRAnim_Frame

组合OBJ帧数据

```
结构 typedef struct GtSPRAnim_Frame
{
    u16 shape:2;           //形状
    u16 size:2;            //大小
    u16 vflip:1;           //是否V翻转
    u16 hflip:1;           //是否H翻转
    u16 charno:10;         //CHAR地址
    s8 dx;                 //离开根结点的相对位置X
    s8 dy;                 //离开根结点的相对位置Y
}GtSPRAnim_Frame;
```

²此文撰写于GsLib 正式支持组合精灵技术之前，所以可能与目前的正式版本有所出入

GtSPRFrameList

帧列表数据，用来把各个部件的帧集中建立索引

```
结构 typedef struct GtSPRFrameList
{
    const GtSPRAnim_Frame *frames;    //帧序列
    u8 MaxFrame;                      //帧总数

}GtSPRFrameList;
```

参考 4.8.2GtSPRAnim_Frame

GtSPRITE

存在于ROM中的组合OBJ结构数据

```
结构 typedef struct GtSPRITE                //一个组合精灵
{
    u8 oam_locate;                          //OAM位置
    const GtSPRFrameList *SubObjFrame;      //帧列表
    u8 part;                                //一共包含的部分数

}GtSPRITE;
```

参考 4.8.2GtSPRFrameList

GtSPRITERect

描述整个组合OBJ的大小

```
结构 typedef struct GtSPRITERect
{
    u8 width;    //组合OBJ的长
    u8 height;   //组合OBJ的宽

}GtSPRITERect;
```

GtSPRBitmap

描述组合OBJ资源图片的数据

```
结构 typedef struct GtSPRBitmap
{
    const u16 *Paddr;    //调色盘地址
    const u8 *Daddr;    //数据地址
    u8 Width;           //图片的宽
    u8 Height;          //图片的长
    u8 ObjPalNo;         //选择的调色盘
    u16 StartChar;       //在VRAM中的开始位置
}GtSPRBitmap;
```

4.8.3 相关函数

GtLoadObjChar2D

以2D方式载入一张图片

```
函数 void GtLoadObjChar2D(
    const u8 *p,          //指向图片资料地址
    u8 hc,                //图片长度
    u8 vc,                //图片高度
    u16 offset
);
```

说明 将图片传输到以OBJVRAM的目标位置offset开始存放

返回值 无

参考 [4.8.3GtSPRSetPal](#)

GtSPRLoadBitmap

载入一个BMP图片到VRAM

```
函数 void GtSPRLoadBitmap(
    const GtSPRBitmap *bmp          //GtSPRBitmap结构的数据
);
```

说明 载入一个BMP图片 必须先设置好GtSPRBitmap结构参数后才可使用

GtSPRCreate

创建组合OBJ

函数 void GtSPRCreate(

```
    const GtSPRITE *p,    /*这个参数是由组合OBJ编辑器所创建的组合OBJ
                           的名称.存在于ROM中.*/
    u8 oam_location,      //存在于OAM中的开始位置0~127
    u8 OBJMODE,           //表示整个组合OBJ的属性,
    u8 PlttNo,            //表示选择一个在OBJ调色盘中的调色版号,0~15
    u8 prioi              //表示组合OBJ的优先级

);
```

说明 创建组合OBJ的方法.由于组合OBJ数据是在ROM中的,要创建的话必须用此函数,将数据再入内存.然后才可以使用. 其中, OBJMODE有如下标志:

Gs.OAM_NORMAL	正常状态
Gs.OAM_SEMI	ALPHA透明状态
Gs.OAM_WINDOW	表示组合OBJ受到窗口影响

返回值 无

参考 4.8.3GtLoadObjChar2D,
4.8.3GtSPRSetPal

GtSPRSetPal

设置OBJ部件的调色盘

函数 void GtSPRSetPal(

```
    const GtSPRITE *p,    //源图片资料地址
    u8 oam_location,      //存在于OAM中的位置
    u8 part,              //组合OBJ的第part个部件
    u8 PlttNo             //目标OBJRAM中的开始位置0~1024

)
```


说明 选择组合OBJ的第PART个部件的为第PlttNO个调色盘

返回值 无

参考 4.8.3GtLoadObjChar2D

GtSPRGetSize

得到组合OBJ大小

函数 void GtSPRGetSize (

```
    const GtSPRITE *p,    //存在于ROM中的组合OBJ指针
    GtSPRITERect *dec      //整个组合OBJ的矩形范围
);
```

说明 得到组合OBJ的大小,结果存在于GtSPRITERect内

返回值 无

参考 4.8.3GtSPRCreate

GtSPRSetAffine

设置一个部件的属性

函数 void GtSPRSetAffine(

```
    const GtSPRITE *p,    //存在于ROM中的组合OBJ指针
    u8 oam_location,       //在OAM中的开始位置u8 part
);                          //部件的序号
```

说明 为组合OBJ的第part个部件设置仿射参数组,使其能够旋转防缩

参考 4.8.3GtSPRCreate

4.8.3GtSPRSetXY

GtSPRAnimate

动画一个部件

函数 void GtSPRAnimate(

```

        const GtSPRITE *p,    //指向存在于ROM中的组合OBJ指针
        u8 oam_location,      //存在于OAM中的位置
        u8 part,              //组合OBJ的第part个部件
        u8 frame
    );                          帧号

```

说明 动画播放组合OBJ的第PART个部件,速度和帧数可以由frame控制(可以不用担心frame出界)

返回值 无

参考 4.8.3GtSPRCreate
4.8.3GtSPRSetPal

GtSPRSetXY

设置组合OBJ坐标

函数 void GtSPRSetXY(

```

        const GtSPRITE *p,    //存在于ROM中的组合OBJ指针
        u8 oam_location,      //存在于OAM中的位置
        int x,int y
    );                          设置在屏幕上的位置

```

说明 为组合OBJ设置在屏幕上的根坐标

参考 4.8.3GtSPRCreate

GtSPRMove

移动一个组合OBJ

函数 void GtSPRMove(

```

        const GtSPRITE *p,    //存在于ROM中的组合OBJ指针
        u8 oam_location,      //存在于OAM中的位置
        int dx,int dy          //离开原来坐标的偏移量
    );

```

说明 将在屏幕上的组合OBJ整个移动dx,dy

参考 4.8.3GtSPRSetXY

GtSPRFlipH

某个部件横向翻转

函数 void GtSPRFlipH(

```
    const GtSPRITE *p,    //存在于ROM中的组合OBJ指针
    u8 oam_location,      //在OAM中的开始位置
    u8 part
);                        部件号
```

说明 将某部件做H方向上的反转

GtSPRRotate

旋转一个部件

函数 void GtSPRRotate(

```
    const GtSPRITE *p,    //存在于ROM中的组合OBJ指针
    u8 oam_location,      //在OAM中的开始位置
    u8 part,              //部件号
    GsSPRITEAffine *affine, //这个部件的旋转参数
    u8 AffineNo,          //选择的旋转参数组0~31
    u8 centerx, u8 centery, //旋转准心位置
    s16 Theta              //旋转角度
);
```

说明 用来旋转组合OBJ的一个部件,让这个部件围绕centerx,centery旋转,旋转角度为Theta期间可以控制这个部件自身是否有旋转,其旋转参数在Affine中

参考 4.8.3GtSPRCreate
4.8.3GtSPRSetAffine

4.9 声音调用

4.9.1 声音控制

GsPlaySound

```
void GsPlaySound(u8 no,u8 lh,const u8 *sound, u16 sampleRate)
```

参数:

no	通道号
lh	高位低位
sound	采样数据开始地址
sampleRate	采样频率

说明:

播放采样声音

GsSetDSoundVol

```
void GsSetDSoundVol(u8 left, u8 right);
```

参数:

left	左声道	0~7
right	右声道	0~7

说明:

调节声音响度

GsSwitchSOUND

```
void GsSwitchSOUND(u8 no,u8 lr,u8 stat);
```

参数:

no	通道号	1~4
lr	左右声道	包括SOUND_LEFT 和SOUND_RIGHT 这2个标志。
Stat	开关	包括ON , OFF

说明:

打开关闭非线性硬件声源

4.9.2 MusicPlayerAGB2000超快教程

概念

使用MP2K的步骤一般包含以下几个

1. 录制采样并且转换成aiff格式(可以从wav文件转)
2. 利用录制的采样来编辑midi声音文件(这个不用说了吧)
3. 将aiff文件和midi文件放到对应的目录内.
4. 编辑mks4agb.ini
5. 编译.调试
6. 将生成的文件用于游戏中.

下面我们将详细介绍MP2K的安装和使用:

安装

创建一个目录,比如mp2k,将

- mks4agb.exe
- aif2agb.exe
- mid2agb.exe
- mks4agb.ini
- MusicPlayDef.s
- mks4agbLib.o
- mks4agbLib.h
- crt0.o
- SoundMon.o
- m4aLibOD1.o
- m4aLibUSC.o

拷贝到这个目录内,然后创建

```
aiff   : Waveform Data File Input Directory
midi   : Song Data File Input Directory
out    : Sound Objects Output Directory
src    : Source Files Output Directory
```

这几个目录.

然后把采样数据,AIF文件全部拷贝到AIFF目录内(比如说有aif0,aif1,aif2这3个采样)

把midi文件拷贝到midi目录内.(比如说mid0.mid)

配置

编辑mks4agb.ini, 在`agb_lib` =里
是设置AGBLIB的目录,比如(`agb_lib = c:\Agb\lib`)
`cmd_path =`
设置你的GCC目录,比如(`cmd_path = C:\Program files\Cygnus\thumbelf-000512\H-i686-cygwin32\bin;`)
其他目录,有E文解释,我不多说了.

在`vgroup = 0`
这里开始就是设置采样数据.我们这里设置成
`p000 = A, aif0, 99, 99, 99, 99`
`p001 = A, aif1, 99, 99, 99, 99`
`p002 = A, aif2, 99, 99, 99, 99`
⋮

`p127 = R, 1,c1`

这句话是一定要加的

即可

在;Setting Song这里写成
`s000 = mid0.mid, 3,0, 127, 10 ;midi sound 0`
最先的参数是在midi目录下的MIDI文件名字. 后面的参数以后解释,以后的参数暂时不需要改动

最后,到cmd下面,打mks4agb,然后你就可以看见你的“工程”被正确编译了.

调用

编译后,会生成几个文件:

SoundMon.elf 可以让你测试音乐效果的程序
一堆.o的文件 在out目录下

将这些文件拷贝到你的程序目录里
在你的程序里

```
#include "mks4agb.h"
```

将

```
m4aSoundInit();
```

加到主程序开始处,
将

```
m4aSoundMain();  
m4aSoundVSync();
```

加到VBL中断服务程序中即可.

在需要播放音乐的地方写上

```
m4aSongNumStart(0);    //其中,0是歌曲的编号.
```

在需要停止音乐的地方写上

```
m4aSongNumStop(0);    //其中,0是歌曲的编号.
```

最后在makefile里把Soundfiles一起编译,就可以了(Soundfiles是和那堆.o文件是在一个目录内的.)

具体看下面实例

例子

几个必须的目录和解释

aiff目录

最初的MP2K是不带任何采样数据的.为此,我准备了一个以前搞PSX时用的MIDI采样数据,一共是77个采样.可以满足90%的MID音乐需求.这些采样已经被我处理成aif格式,并且已经在aiff目录下面了.以方便大家使用.

Midi目录

这里面是放的你需要的MIDI歌曲.因为MID文件是指令的集合,所以不同的采样表,会有不同的效果.在GBA里面,所有音乐效果和背景音乐,都需要编辑MID来使用.在这个例子里,准备了4个MID演示

out目录

这是用来存放转换后的所有数据.

Demo目录

这是我写的一个使用MP2K播放MID的实例,附带源程序大家可以参考使用.

4.10 计数器

4.10.1 硬件相关知识

GBA包含4个硬件时钟,和4个记数积存器

设置完记数初值后,根据所选择的记数通道,根据频率每次跳变则减1,减完后发出中断请求。

硬件时钟号	每次跳变的间隔时间	
0	系统时钟	(59.595 ns)
1	64倍系统时钟	(3.814 μ s)
2	256倍系统时钟	(15.256 μ s)
3	1024倍系统时钟	(61.025 μ s)

4.10.2 相关函数

GsOpenTimerIRQ

```
void GsOpenTimerIRQ(int no);
```

参数:

int no 时钟积存器号(通道号) 0~3

说明:

打开no号时钟中断
这句话是很重要的,在每次执行其他TIMER操作的时候都要用上, 否则将无法得到其结束信息

GsSelectTM

```
void GsSelectTM(int no,int htno);
```

参数:

int no 时钟积存器号(通道号) 0~3

int htno 硬件时钟

说明:

为一个通道选择一个硬件时钟

GsSetCount

```
void GsSetCount(int no ,u16 num);
```

参数:

int no 时钟积存器号0~3

u16 num 记数初值

说明:

为一个通道分配一个记数初值

GsStartTIMER

`void GsStartTIMER(int no);`

参数:

int no 时钟积存器号0~3

说明:

开始一个通道的计时。

Chapter 5

GSLIB辅助开发工具

5.1 图象图形

5.1.1 Gfx2GBA

名称 Gfx2GBA.exe

	-p 调色盘名	-m 压缩MAP
	-o 输出路径	-M 非压缩MAP
	-f 输出格式src是以文本格式输出	-mm 压缩MAP
	-s 断截面	-MM 非压缩MAP数据
	-Sp 调色盘开始下标	-b 使MAP数据为0
	-Sm MAP数据开始下标	-mc save map in column order
	-St TILE数据开始下标	-F turn off check for flipped tiles
	-T 活动块大小	-rs 输出旋转BG格式的MAP数据
	-t 活动块尺寸	-P 不保留调色板数据
参数	-tc save tiles as columns	-G 不保留TILE数据
	-c 设置色深	-D 不保留MAP数据
	-C Color-Offset	-X 不保留任何东西
	-A force color offset add	-Z 压缩所有数据
	-a 设置透明色	-zt 压缩位图数据
	-v 设置VRAM偏移量	-zp 压缩调色盘数据
	-x 不合并调色盘	-zm 压缩MAP数据
	-q quiet mode	-ap use aPLib as compressor
	-B only optimize blank tiles	-aps use aPLib (safe) as compressor
	-align add alignment info for GCC	

- 例子
1. 把一张名为sample.bmp的图片转换成MODE0下的TEXT BG 16色
gfx2gba -fsrc -psample_pal -t8 -c16 -mm sample.bmp
 2. 把一张名为sample.bmp的图片转换成MODE0下的TEXT BG 256色

```
gfx2gba - fsrc - psample_pal - t8 - c256 - mm sample.bmp
```

3. 把一张名为sample.bmp的图片转换成MODE1下的旋转BG

```
gfx2gba - fsrc - psample_pal - t8 - mm - rs sample.bmp
```

因为旋转BG只可能是256色的所以不用加-c

4. 把一张名为sample.bmp的图片非压缩的GBA图片

```
gfx2gba - fsrc - psample_pal - t81 - M sample.bmp
```

5.1.2 PalTrans

名称 PalTrans²

说明 可以对pcx图象的调色盘进行高级操作的工具

使用方法

```
paltrans start end in .pcx out.pcx
```

参数 start 在BGPAL里的开始位置
 end 在BGPAL里的结束位置

解释

就是把一张256色的PCX图片，把调色盘压缩到(end-start)大小并从start位置开始存放。这个工具的用途就是解决旋转BG只能使用256色而产生的调色盘冲突问题的。如在MODE2 下面，需要2张不同的BMP，如果不经处理，直接加载2张调色盘的话，会得到错误效果。原因是调色盘不一致，从而导致其中有一层BG的调色盘使用了另外一层BG的调色盘。而没使用自己。所以就要对数据部分进行操作。如果用GBA来进行对数据部分进行转换的话，将是很浪费时间的。所以有了这个工具。就方便多了。

5.2 音乐音效

5.2.1 wav2gbac

名称 wav2gbac.exe

使用方法 wav2gbac input.wav output.c

说明 把采样波形文件INPUT转换成文本C语言数组。

¹-t8 因为GSLIB 比较接近硬件，所以只支持8X8大小的TILE

²作者：乐水，在此表示感谢