HAM Tutorial :: Introduction

by Aaron Rogers

Version 1.7.7

Don't speak English? Click HERE!

Want to view any or all of the tutorials OFFLINE?! Click HERE!

NOTE: Be sure to look at the History page for the latest changes.

Purpose:
The purpose of this document is to teach you how to use HAM to create your own video games for the Gameboy Advance. HAM is really quite simple to use and takes most of the work out of GBA programming.

Thanks:
I would especially like to thank tubooboo for creating this wonderful development environment.

I also need to thank The PERN Project for inspiration, ideas and information about the GBA as well as the many developers out there who open the source code to their demos and programs. Without them, this would not be possible.

Finally, I should mention a few sites out there that provide tons of resources related to GBA developement.
http://www.gbadev.org/
http://www.gbaemu.com/
http://www.devrs.com/gba/

Prerequisites:

NOTE: This tutorial is currently geared towards development in a Windows enviroment. Before HAM 2.50, there was no Linux support. I have not tested the tutorials in Linux yet.

Of course, you need to have HAM installed. If not, go here and get it. The current version is 2.50 and all of these tutorials currently DO NOT work with it without modification. I am working to fix this as soon as possible.

I expect that you have at least a beginner's knowledge of C/C++ programming.

You'll also need your favorite text editor or IDE.

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

Finally, you'll need a way to test your compiled code. I would recommend an emulator, a Flash Advance kit or if your demo is under 256 KB, you can use an MBV2 cable.

Emulators: VisualBoy Advance, Boycott Advance
Flash Advance: Visoly.com (homepage), EasyBuy2000.com (buy one)
MBV2 Cable: Lik-Sang.com (buy one), FAQ

One final note. To make my life easier, I created a file called *am.bat* in **\ham\tools**. The file contains the following line: **cls && make && hello.gba**
What this means is that when I want to compile my code all I have to do is type **am**. This will clear the screen, run make and if it compiles successfully, it will run hello.gba. If your output file has a different name, then of course you should change it to that.

Troubleshooting:
I will not cover the setup of HAM. It is very straightforward and any problems with it should be directed to the author tubooboo.

I would also highly recommend that you join hamdev on YahooGroups.com.

Feedback
If you have any questions, suggestions or comments about this tutorial, please email me. I would really appreciate it!

Also, if you wanna mention that you learned how to program your GBA demo/game/etc. from here, please do so!

<< HOME >>

HAM Tutorial :: Day 1 :: GBA Hardware

I won't go into too much detail here as there are already quite a few web pages and tutorials dedicated to the GBA hardware. I will give you a quick overview of the system, however, as it is necessary to understand some of the content covered in this tutorial.

- 32bit ARM7TDMI CPU running on RISC architecture (16.78 MHz)
- 96 KB of video memory, 32KB of internal ram and 256KB of external ram
- 6 screen modes (max resolution: 240x160); up to four backgrounds (or layers or planes); some backgrounds can be rotated and scaled; all can be scrolled, mosaic, faded in or out and alpha blended

Video Modes

- Mode 0 - tile mode; screen is composed of 8x8 pixel squares; all 4 backgrounds; no rotation or scaling
- Mode 1 - tile mode; screen is composed of 8x8 pixel squares; BGs 0,1, and 2; BG 2 can rotate/scale
- Mode 2 - tile mode; screen is composed of 8x8 pixel squares; BGs 2 and 3; both can rotate/scale
- Mode 3 - bitmap mode; linear buffer of 16 bit pixels; NO double-buffering
- Mode 4 - bitmap mode; linear buffer of 8 bit pixels with a 16 bit palette; double-buffering; palettized diplay mode
- Mode 5 - bitmap mode; linear buffer of 16 bit pixels; double-buffering; 160x128 pixels

NOTE: 8 bit = 256 colors, 16 bit = 65,536 colors.

When operating in tile mode, you can use up to four backgrounds each controlled independently using different maps and tiles. By default they are rendered 3,2,1,0 meaning that background 0 would be displayed over 3.

Sprites

The GBA supports hardware rendered sprites that can either share one 256 color palette or there can be up to 16 different 16 color palettes that a sprite can use. Sprite palettes are separate from the screen palette. There can be up to 128 different sprites ranging in size from 8x8 to 64x64 pixels. Modes 0,1,2 have 32 KB of sprite memory while Modes 3,4,5 have 16 KB.

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

Sound

The GBA has 6 sound channels. 4 are standard music channels similar to the GB/GBC. The other two are allow direct sound sampling.

I recommend you go to The Audio Advance for more info on the GBA sound system.

Well, that's it for Day 1. We didn't go into too much detail on the hardware. Again, I will cover what is necessary as we go through the tutorial.

For more information on GBA hardware, I would recommend you check out the following sites:
CowBite Virtual Hardware Specifications
Mappy VM SDK 0.1

Discuss Day 1

<< HOME >>

HAM Tutorial :: Day 2 :: "Hello, World!"

I would say it's about time to do some actual coding. So, without further ado, here we go!

NOTE: Remember to copy *makefile* from HAM into your source directory before running make or download it from here.

```
// The Main HAM Library
#include "mygba.h"

int main()
{
  // Initialize HAMlib
  ham_Init();

  // Initialize the text display system on the
  // background of your choice
  ham_InitText(0);

  // Draw some text
  ham_DrawText(0,0,"Hello, World!");

  while(1)
  {
    // Infinite loop to keep the program running
  }

  return 0;
}
```

Code Explanation

#include "mygba.h"
This is the main include that you'll need in all of your programs which use HAM.

ham_Init();
This is needed in every HAM program to initialize HAMlib and must be the ***first*** HAM function called.

ham_InitText(0);
You can pass it background 0 - 3. You can only run the HAM text system in background mode 0 - 2. Remember these are the tile modes.
NOTE: If backgrounds and background modes don't make sense to you right now, come back after Day 4 or Day 6.

ham_DrawText(0,0,"Hello, World!");
The first number passed is the column and the second is the row. For

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

example 0,3 would be the first column in the fourth row.
Horizontal Values: 0 - 29
Vertical Values: 0 - 19

while(1)
Without this, the program would just go into an infinite loop displaying the HAM logo.

Well, that's it for Day 2. I don't think there is anything difficult about the code. Feel free to tinker with it before moving on to Day 3.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day2_Hello_World.zip

HAM Version 2.30 And Lower
Necessary for all HAM programs: makefile
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day2_Hello_World.zip

View Demo Now

Discuss Day 2

<< HOME >>

HAM Tutorial :: Day 3 :: Input

Another easy thing to do with HAM is user input. Today we'll start with the code from the Day 2 tutorial and add some code to check for input.

```
// The Main HAM Library
#include "mygba.h"

int main()
{
 // Variables
 bool dpad_pressed = 0;

 // Initialize HAMlib
 ham_Init();

 // Initialize the text display system on the
 // background of your choice
 ham_InitText(0);

 // Draw some text
 ham_DrawText(0,0,"Hello, World!");
 ham_DrawText(11,18,"Press D-Pad To Reset");

 while(!dpad_pressed)
 {
  // Check for input and change key_pressed
  if (F_CTRLINPUT_UP_PRESSED ||
      F_CTRLINPUT_DOWN_PRESSED ||
      F_CTRLINPUT_LEFT_PRESSED ||
      F_CTRLINPUT_RIGHT_PRESSED)
  {
   dpad_pressed = 1;
  }
 }

 return 0;
}
```

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

Code Explanation

if (F_CTRLINPUT_UP_PRESSED ...)
During the while loop it will do a check to see if up, down, left or right is pressed. After changing dpad_pressed to false, it will quit the program and 'reset' the game.

Button	HAM Equivalent
UP	F_CTRLINPUT_UP_PRESSED
DOWN	F_CTRLINPUT_DOWN_PRESSED
LEFT	F_CTRLINPUT_LEFT_PRESSED
RIGHT	F_CTRLINPUT_RIGHT_PRESSED
START	F_CTRLINPUT_START_PRESSED
SELECT	F_CTRLINPUT_SELECT_PRESSED
B	F_CTRLINPUT_B_PRESSED
A	F_CTRLINPUT_A_PRESSED
L	F_CTRLINPUT_L_PRESSED
R	F_CTRLINPUT_R_PRESSED

NOTE: In order to do diagonal movement, you can use something like:
if (F_CTRLINPUT_UP_PRESSED && F_CTRLINPUT_LEFT_PRESSED)
There will be an example of this in a later project.

Well, another day, another tutorial. Again, nothing difficult here.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day3_Input.zip

HAM Version 2.30 And Lower
Necessary for all HAM programs: makefile
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day3_Input.zip

View Demo Now

Discuss Day 3

< < HOME > >

HAM Tutorial :: Day 4 :: Backgrounds - Bitmap Modes

Today we'll start learning about backgrounds (also known as layers or planes) and background modes. Remember there are six background modes and up to four backgrounds (layered on top of each other) per mode.

I am going to start with the background modes, Mode 3-5, because they are the easiest to understand.

The first thing you have to know about a background is that it requires a certain format. HAM comes with an easy to use program called *gfx2gba* that is used to convert BMPs or PCXs to the proper format.

NOTE: Please take the time to read the Readme that comes with gfx2gba, it will save you from many headaches! The latest version of the readme can be found here.

The image should be 240x160 pixels (or 160x128 if you are using Mode 5). BMPs can be 4 or 8 bits while PCXs should be 8 bits. Feel free to take a peek at the Graphics FAQ if you have questions about formats and palettes, etc.

For this tutorial we'll work exclusively with bitmaps. I've created a picture (which is quite ugly) called bg.bmp. Copy the file to your directory (or create your own BMP). For these tutorials we'll have a subdirectory **gfx** for all graphics used in our games. Change to the directory and type:
gfx2gba -D -fsrc -pbg.pal -t1 bg.bmp

This will create two files:
bg.pal.c
bg.raw.c

Go ahead and take a quick peek at the files. The important thing about bg.pal.c is the name of the array that is created. It should be **bg_Palette**. The array created in bg.raw.c is **bg_Bitmap**. You will see these names later in the tutorial program.

Speaking of which, let's go ahead and display that background.

```
// The Main HAM Library
#include "mygba.h"

// Graphics Includes
#include "gfx/bg.raw.c"
#include "gfx/bg.pal.c"

int main()
{
  // Initialize HAMlib
  ham_Init();
```

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

```
  // Setup the background mode
  ham_SetBgMode(4);

  // DMA the background picture
  TOOL_DMA1_SET(&bg_Bitmap,
                MEM_BG_PTR,
                SIZEOF_32BIT(bg_Bitmap),
                DMA_TRANSFER_32BIT,
                DMA_STARTAT_NOW)

  // Initialize the background palette
  ham_LoadBGPal(&bg_Palette,SIZEOF_16BIT(bg_Palette));

  while(1)
  {
    // Infinite loop to keep the program running
  }

  return 0;
}
```

Code Explanation

#include "gfx/bg.raw.c"
#include "gfx/bg.pal.c"
Again these are the palette and raw file created from our original image.

ham_SetBgMode(4);
For this example we need to use Mode 4. This mode is great for dealing with the screen as if it were a big bitmap. In this mode the pixels are 8 bit indexes in a 16 bit palette. What this means is that you can store 256 16 bit colors for the background into palette memory and then access the screen as an array. The index of the color is put into the array giving you the advantage of drawing the screen with half the memory, twice as fast and allows you to have two screens in memory. You can setup one screen while the other is being displayed. When finished, you just swap the screens. This is known as double-buffering.

TOOL_DMA1_SET(...)
TOOL_DMA1_SET is very straightforward: A DMA is a way to copy a block of memory from adress A to adress B, in our case from ROM to BG RAM. This will be explained in more detail later. You just need to know that this is a very fast way to copy data.

ham_LoadBGPal(...)
Load the palette for our background.

Well, that was a bit much for today. I think that's enough for now. Try creating your own background, convert the file with **gfx2gba** and see if you can display it.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Graphics: bg.bmp, bg.pal.c, bg.raw.c
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day4_BGs_Bitmap_Modes.zip

HAM Version 2.30 And Lower
Necessary for all HAM programs: makefile
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day4_BGs_Bitmap_Modes.zip

View Demo Now

Discuss Day 4

<< HOME >>

HAM Tutorial :: Day 5 :: Sprites

I think it's time to start learning about sprites, don't you? 'What is a sprite,' you ask. Well, I would think you would already know, but just in case you don't a sprite is just an image on the screen (except the background, although a sprite could be that big). Any character, coin, etc. on the screen is a sprite.

Just as with backgrounds, a sprite must be in the proper format. Again, a sprite can be from 8x8 up to 64x64 pixels. There can be up to 128 sprites on the screen at a time. They can share one 256 color palette or there can be up to 16 different 16 color palettes that a sprite can use.

As I mentioned in Day 4, I will work exclusively with bitmaps. For this tutorial I have created a sprite which is just a blue ball called blue_ball.bmp. Copy the file to your **gfx** subdirectory and type:
gfx2gba -D -fsrc -pblue_ball.pal -t8 blue_ball.bmp

This will create two files:
blue_ball.pal.c
blue_ball.raw.c

Again, two arrays will be created. The palette will be **blue_ball_Palette** and the image will be **blue_ball_Bitmap**.

For today's code we'll just display the sprite on the screen. Later we'll work on moving it around the screen. I hope that you all understand global variables.

NOTE: You'll notice that when the ball is displayed on the screen you see the white background that is part of the original image. Later on when we cover alpha blending, you'll learn how to show only the blue ball without the white background.

```
// The Main HAM Library
#include "mygba.h"

// Graphics Includes
#include "gfx/blue_ball.raw.c"
#include "gfx/blue_ball.pal.c"

// Global Variables
u8 theball[1]; // Sprite index for the ball

int main()
{
  // Initialize HAMlib
  ham_Init();

  // Setup the background mode
```

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

```
  ham_SetBgMode(4);

  // Initialize the sprite palette
  ham_LoadObjPal(&blue_ball_Palette, 256);

  // Setup the sprite
  theball[0] = ham_CreateObj(&blue_ball_Bitmap,0,2,
OBJ_MODE_NORMAL,1,0,0,0,0,0,0,110,50);

  // Draw the sprite
  ham_CopyObjToOAM();

  while(1)
  {
    // Infinite loop to keep the program running
  }

  return 0;
}
```

Code Explanation

u8 theball[1];
A u8 is defined in mygba.h as an unsigned char.
I will use indeces for sprites because if the sprite rotates or turns at all, it is easy to load the different frames into the array and use this for animation. This may seem somewhat confusing so I will explain it more when we do our first project.

ham_SetBgMode(4);
For this example we'll use Mode 4 although it doesn't really matter. Sprites are separate of BGMode EXCEPT that there is less sprite storage space in certain BG Modes. However, HAM handles the trouble there, no matter what BGMode you are in. You'll just have less available OBJ RAM in some modes.

theball[0] = ham_CreateObj(...);
ham_CreateObj is used to create a sprite in the HAM system and return the reference index for further operations on the sprite. It is not very difficult to understand but there is a lot of information that you must pass to the function. The first thing is the array associated with the image that you want to display. Next is the size and shape of the object. Check the chart to see what your choices are. In this case, I am using an 32x32 pixel image, so I pass it 0 and then 2. What this means is that there are 4x4 8 pixel blocks. Think about that for a minute to make sure you understand what I mean. Next is the object mode. For now, I am just using OBJ_MODE_NORMAL. Next is the color mode. For this sprite I have one 256 color palette, so I pass it 1. Next is the palette number. Because I am using color mode 1, you just pass it 0. If you were using 16 separate 16 color palettes, you would pass it the palette number which corresponds to your image (0-15). Next are mosaic, horizontal flipping and vertical flipping. I am not using any of

them, so they are all 0. Next is the priority of the object, in this case it's 0. Finally you pass it the X and Y position on the screen.

ham_CopyObjToOAM();
This function is used to load all of the objects to OAM. This basically means it draws the objects.

Well, that was a bit much for today. I think that's enough for now. Try creating your own sprite, convert the file with **gfx2gba** and see if you can display it.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Graphics: blue_ball.bmp, blue_ball.pal.c, blue_ball.raw.c
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day5_Sprites.zip

HAM Version 2.30 And Lower
Necessary for all HAM programs: makefile
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day5_Sprites.zip

View Demo Now

Discuss Day 5

<< HOME >>

HAM Tutorial :: Day 6 :: Backgrounds - Tile Modes

I think it's time to learn about the tile modes, Mode 0 - 2. Tile modes have been around for quite a while and they can be very useful. You start with a set of tiles, all of which are 8x8 pixel bitmaps. Then you have a map which is used to create a background out of those tiles.

For this tutorial, basically what I do is take a bitmap and break it into tiles and create a map with gfx2gba. These are then used to recreate the original bitmap. In most modes the screen is 240x160 pixels. This means there are 600 8x8 tiles (600 tiles * 8 pixels wide * 8 pixels high = 38400 pixels = 240 * 160 pixels). Think about that for a minute before you move on to make sure you understand what I mean.

For more information on tile-based graphics, check out these sites:
http://www.nonoche.com/imaging/en/
Tile Based Games FAQ
http://www.gamedev.net/reference/list.asp?categoryid=44

For this tutorial, I've created a picture called paradise.bmp (which I took from a recent trip to the Bahamas). Copy the file (or your own BMP) to the **gfx** directory.
Change to that directory and type:
gfx2gba -fsrc -m -pparadise.pal -t8 paradise.bmp
NOTE: This is different from what we used on Day 4.
-t8 means 8 pixel tiles, -m means map

This will create three files:
paradise.pal.c
paradise.raw.c
paradise.map.c

Again, go ahead and take a quick peek at the files. The important thing about paradise.map.c is the name of the array that is created. It should be **paradise_Map**. The array created in paradise.raw.c is **paradise_Tiles**. You will see these names later in the tutorial program.

Speaking of which, let's go ahead and display that (tile mode) background.

```
// The Main HAM Library
#include "mygba.h"

// Graphics Includes
#include "gfx/paradise.pal.c"
#include "gfx/paradise.raw.c"
#include "gfx/paradise.map.c"

int main()
{
  // Variables
```

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

```
  map_fragment_info_ptr bg_paradise; // Pointer to our background

  // Initialize HAMlib
  ham_Init();

  // Setup the background mode
  ham_SetBgMode(1);

  // Initialize the background palette
  ham_LoadBGPal(&paradise_Palette,256);

  // Setup the tileset for our image
  ham_bg[0].ti = ham_InitTileSet(&paradise_Tiles,
                                 SIZEOF_16BIT(paradise_Tiles),1,1);

  // Setup the map for our image
  ham_bg[0].mi = ham_InitMapEmptySet(3,0);
  bg_paradise = ham_InitMapFragment(&paradise_Map,
                                    30,20,0,0,30,20,0);
  ham_InsertMapFragment(bg_paradise,0,0,0);

  // Display the background
  ham_InitBg(0,1,0,0);

  while(1)
  {
    // Infinite loop to keep the program running
  }

  return 0;
}
```

Code Explanation

map_fragment_info_ptr bg_paradise;
This data type is used to store all the information about our background. If you want to know more about it, here's a code snippet from *mygba.h*.

```
typedef struct map_fragment_info_typ
{
  u16* src; // location of this map (usually ROM/EXWRAM)
  u16* src_ofs; // location of the data start IN the map currently set
  u8 map_rot; // designates this map to be rotation type map
  u16 map_total_x; // actual map size in tiles / x
  u16 map_total_y; // actual map size in tiles / y
  u16 map_ofs_x; // offset into maps location start in tiles / x
  u16 map_ofs_y; // offset into maps location start in tiles / y
  u16 map_tiles_x; // actual map size (from offset on) in tiles / x
  u16 map_tiles_y; // actual map size (from offset on) in tiles / y
  u16 map_line_ofs; // offset that needs to be added after reading 1 line
  // of the map to reach next lines map_x
} map_fragment_info,*map_fragment_info_ptr;
```

ham_SetBgMode(1);
For this example we'll use Mode 1 . In this tile mode the screen is composed of 8x8 pixel squares. Backgrounds 0, 1, and 2 are available. Background 2 can be rotated or scaled.

ham_bg[0].ti = ham_InitTileSet(...);
This loads our tileset to background 0. It is passed the address of the tileset, the size of the tiles to be copied in 16 bit chunks, the color mode, and cbb_only_mode. This last parameter is somewhat tricky - I recommend you use the default of 1 for now.

ham_bg[0].mi = ham_InitMapEmptySet(3,0);
This sets up an empty map linked to a background. Basically it just gets a map ready for the next function we'll call.

bg_paradise = ham_InitMapFragment(...)
This sets up a rectangular portion on our background into which we'll "paste" our data with the next function.

ham_InsertMapFragment(bg_paradise,0,0,0);
This actually copies the map from the fragment we just created into our empty background.

ham_InitBg(0,1,0,0);
This will actually draw the background. It is passed the background number you wish to initialize, whether the background is hidden or not, the priority of the background, and whether the background should be mosaic or not. If you haven't setup the .ti and .mi parameters, this will return an error.

Well, I don't think that was too difficult. Again, mess with the code and see what you can do before moving on to Day 7.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Graphics: paradise.bmp, paradise.pal.c, paradise.raw.c, paradise.map.c
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day6_BGs_Tile_Modes.zip

HAM Version 2.30 And Lower
Necessary for all HAM programs: makefile
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day6_BGs_Tile_Modes.zip

View Demo Now

Discuss Day 6

<< HOME >>

HAM Tutorial :: Day 7 :: Project 1 - Tetris

Well, we've finally gotten to the end of the first week. I think it's about time to put together the various things we've learned into a project. What we'll do now is work on a Tetris style game.

NOTE: We are not going to actually finish the game now (although you can feel free to try that if you really want to). We may come back to this for a later project.

I hope you all know about function prototypes at this point.

For this tutorial I've created one sprite image and two background images. Copy these files to the **gfx** directory. This is going to be a two step process. First, go to that directory and type:
gfx2gba -t8 -m -pmaster.pal -fsrc *.bmp

This will create the following files:
master.pal.c
block_L.map.c
block_L.raw.c
bubbles.map.c
bubbles.raw.c
outline_plain.map.c
outline_plain.raw.c

This will create the necessary background image files and it will setup one palette for all of our graphics. The problem is, we don't want the sprite to be in a tile/map format, so we'll type the following command:
gfx2gba -D -fsrc -pblock_L.pal -t8 block_L.bmp

This will create the proper **blocks_L.raw.c**

NOTE: We will not need **block_L.map.c** or **block_L.pal** so you should delete those now.

There is one thing you'll need to know before we start that I didn't explain fully earlier - transparency (or alpha blending). The first color in the palette array is the alpha color. What this means is that any part of your image which has that color is transparent. The system palette in Windows has black in the first spot. Therefore, any part of my image which is black will not be drawn to the screen. You can tell this if you look at outline_plain.bmp and compare it to the "game" created.

Well, here we go. Pay attention now, this may get tricky!

```
// The Main HAM Library
#include "mygba.h"

// Graphics Includes
```

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

```
#include "gfx/master.pal.c"
#include "gfx/bubbles.raw.c"
#include "gfx/bubbles.map.c"
#include "gfx/outline_plain.raw.c"
#include "gfx/outline_plain.map.c"
#include "gfx/block_L.raw.c"

// Global Variables
u8 block; // Sprite index for blocks
u8 block_x = 61; // X position of block (column)
u8 block_y = 0; // Y position of block (row)

// Function Prototypes
void vblFunc(); // VBL function
void query_keys(); // Query the keyboard
void redraw_block(); // Redraws the falling block

// Main function
int main()
{
 // Variables
 map_fragment_info_ptr bg_bubbles;
 map_fragment_info_ptr bg_outline_plain;

 // Initialize HAMlib
 ham_Init();


 // Initialize the text display system
 // on the background of your choice
 ham_InitText(2);

 // Setup the background mode
 ham_SetBgMode(1);

 // Initialize the palettes
 ham_LoadBGPal(&master_Palette,256);
 ham_LoadObjPal(&master_Palette,256);

 // Setup the tileset for our image
 ham_bg[0].ti = ham_InitTileSet(&bubbles_Tiles,
                                SIZEOF_16BIT(bubbles_Tiles),1,1);
 ham_bg[1].ti = ham_InitTileSet(&outline_plain_Tiles,
                                SIZEOF_16BIT(outline_plain_Tiles),1,1);

 // Setup the map for our image
 ham_bg[0].mi = ham_InitMapEmptySet(3,0);
 ham_bg[1].mi = ham_InitMapEmptySet(3,0);

 bg_bubbles = ham_InitMapFragment(&bubbles_Map,30,20,0,0,30,20,0);
 bg_outline_plain = ham_InitMapFragment(&outline_plain_Map,
                          30,20,0,0,30,20,0);

 ham_InsertMapFragment(bg_bubbles,0,0,0);
 ham_InsertMapFragment(bg_outline_plain,1,0,0);
```

```
  // Display the background
  ham_InitBg(0,1,1,0);
  ham_InitBg(1,1,0,0);

  // Draw some text
  ham_DrawText(21,8,"Next Up");

  // Setup the blocks
  block = ham_CreateObj(&block_L_Bitmap,0,2,OBJ_MODE_NORMAL,
                        1,0,0,0,0,0,0,block_x,block_y);

  // Start the VBL interrupt handler
  ham_StartIntHandler(INT_TYPE_VBL,&vblFunc);

  while(1)
  {
   // Infinite loop to keep the program running
  }

  return 0;
} // End of main()

// VBL function
void vblFunc()
{
  ham_CopyObjToOAM(); // Copy block sprite to hardware
  query_keys(); // Check for movement
  redraw_block(); // Then redraw the block

   return;
} // End of vblFunc()

// Query the keyboard
void query_keys()
{
  if(F_CTRLINPUT_UP_PRESSED)
  {
   if (block_y < 117) block_y = 117;
   ham_DrawText(1,18,"Up ");
  }

  if(F_CTRLINPUT_DOWN_PRESSED)
  {
   if (block_y < 117) block_y++;
   ham_DrawText(1,18,"Down ");
  }

  if(F_CTRLINPUT_LEFT_PRESSED)
  {
   if (block_x > 61) block_x--;
   ham_DrawText(1,18,"Left ");
  }
```

```
 if(F_CTRLINPUT_RIGHT_PRESSED)
 {
  if (block_x < 120) block_x++;
  ham_DrawText(1,18,"Right");
 }


 return;
} // End of query_keys()

// Redraws the falling block
void redraw_block()
{
 ham_SetObjX(block,block_x);
 ham_SetObjY(block,block_y);


 return;
} // End of redraw_block()
```

Code Explanation

// Global Variables
I think these should all be self explanatory.

// Function Protoypes
I will explain each function after I explain what's in main()

map_fragment_info_ptr bg_bubbles;
A map_fragment_info_ptr is a pointer pointer to a structure of type map_fragment_info that contains all the neccessary data for further operations, such as copying this part of the ROM map onto an existing screen map.

ham_InitText(2)
For this project we will do our text output on background 2. Backgrounds 0 and 1 will be used for the playing field.

block = ham_CreateObj(...)
If you don't remember what this call is, take a look at Day 5.

ham_StartIntHandler(INT_TYPE_VBL,&vblFunc);
This is a very important function. Basically what it does is tells the CPU in the GBA that every vertical blank it should run the function we specifiy. In this case, we will create a function called vblFunc(). You may ask, what is a vertical blank? A vertical blank occurs every time the GBA finishes writing everything to the screen which is usually 60 times a second.

NOTE: For more info on GBA interrupts, etc, click here.
(*8. Hardware Interrupts*)

So, let's talk about the the other functions.

// VBL function
void vblFunc()
{
ham_CopyObjToOAM();
Again, you should remember this from Day 5.

The query_keys() and redraw_block() functions are then called. See below.

} *// End of vblFunc()*

// Query the keyboard
void query_keys(){...}
All this function does is check if the D-Pad is pressed, outputs the direction that was pressed, and if it's legal, it moves the piece. I don't think there's anything difficult about this, but feel free to tinker with it to see how/why it works.

Let's move on the last function.

// Redraws the falling block
void redraw_block()
{
ham_SetObjX(block,block_x);
ham_SetObjY(block,block_y);

These two calls set the new location of the block based on what happened in the query_keys() function.

} *// End of redraw_block()*

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Graphics: block_L.bmp, block_L.raw.c, bubbles.bmp, bubbles.map.c, bubbles.raw.c, master.pal.c, outline_plain.bmp, outline_plain.map.c, outline_plain.raw.c
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day7_Project1_Tetris.zip

HAM Version 2.30 And Lower
Necessary for all HAM programs: makefile

Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day7_Project1_Tetris.zip

View Demo Now

Discuss Day 7

<< HOME >>

HAM Tutorial :: Day 8 :: Sprites #2 - Animation

Well, we covered a lot the first week but, of course, there is so much more to learn. The next thing we'll cover is animating a sprite. What we'll do is create one image that is actually made up of a bunch of smaller images. (Click here to see what I mean.) Then we'll load in each respective piece as the plane flies around the screen.

I downloaded the free graphic off the Internet and then used Photoshop to manipulate it and to create the final image. I decided to make the sprite 64x64 pixels. Since there are 8 different angles, the final image is 64x512 pixels.

Today's example will again be created using a tile mode and there is only one background. Hopefully you all know what command to use to convert the image, but just in case:
gfx2gba -fsrc -m -pbackground.pal -t8 water.bmp

Again, the sprite will not be in a tile/map format, so type:
gfx2gba -D -fsrc -pobject.pal -t8 red_plane_anim.bmp

Your **gfx** directory should have the following files:
background.pal.c
object.pal.c
red_plane_anim.raw.c
water.map.c
water.raw.c

On to the code:

```
// The Main HAM Library
#include "mygba.h"

// Graphics Includes
#include "gfx/background.pal.c"
#include "gfx/object.pal.c"
#include "gfx/water.raw.c"
#include "gfx/water.map.c"
#include "gfx/red_plane_anim.raw.c"

// Defines
#define ANIM_UP         0
#define ANIM_UPRIGHT    1
#define ANIM_RIGHT      2
#define ANIM_DOWNRIGHT  3
#define ANIM_DOWN       4
#define ANIM_DOWNLEFT   5
#define ANIM_LEFT       6
```

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

```
#define ANIM_UPLEFT          7

// Global Variables
u16 dir_plane = 2; // Direction the plane is facing
u8 plane[1]; // Sprite index for the plane
u8 plane_x = 110; // X position of the plane
u8 plane_y = 50; // Y position of the plane

// Function Prototypes
void vblFunc(); // VBL function
void query_keys(); // Query the keyboard
void redraw_plane(); // Redraws the falling block

// Main function
int main()
{
  // Variables
  map_fragment_info_ptr bg_water; // Pointer to our background

  // Initialize HAMlib
  ham_Init();

  // Initialize the text display system
  // on the background of your choice
  ham_InitText(2);

  // Setup the background mode
  ham_SetBgMode(1);

  // Initialize the palettes
  ham_LoadBGPal(&background_Palette,256); // Background palette
  ham_LoadObjPal(&object_Palette,256); // Sprite palette

  // Setup the tileset for our image
  ham_bg[0].ti = ham_InitTileSet(&water_Tiles,
                                 SIZEOF_16BIT(water_Tiles),1,1);

  // Setup the map for our image
  ham_bg[0].mi = ham_InitMapEmptySet(3,0);
  bg_water = ham_InitMapFragment(&water_Map,30,20,0,0,30,20,0);
  ham_InsertMapFragment(bg_water,0,0,0);

  // Display the background
  ham_InitBg(0,1,0,0);

  // Setup the plane
  plane[0] = ham_CreateObj(&red_plane_anim_Bitmap,0,3,
                     OBJ_MODE_NORMAL,1,0,0,0,0,0,0,plane_x,plane_y);

  // Start the VBL interrupt handler
  ham_StartIntHandler(INT_TYPE_VBL,&vblFunc);
```

```
  while(1)
  {
    // Infinite loop to keep the program running
  }


  return 0;
} // End of main()

// VBL function
void vblFunc()
{
  ham_UpdateObjGfx(plane[0],
                   (void*)&red_plane_anim_Bitmap[4096*dir_plane]);
  ham_CopyObjToOAM(); // Copy plane sprite to hardware
  query_keys(); // Check for movement
  redraw_plane(); // Then redraw the plane


  return;
} // End of vblFunc()

// Query the keyboard
void query_keys()
{
  // UP only
  if(F_CTRLINPUT_UP_PRESSED && !F_CTRLINPUT_RIGHT_PRESSED &&
    !F_CTRLINPUT_LEFT_PRESSED)
  {
    if (plane_y > 0) plane_y--;
    dir_plane = ANIM_UP;
  }


  // UP + RIGHT
  if(F_CTRLINPUT_UP_PRESSED && F_CTRLINPUT_RIGHT_PRESSED)
  {
    if (plane_y > 0) plane_y--;
    if (plane_x < 176) plane_x++;
    dir_plane = ANIM_UPRIGHT;
  }


  // RIGHT only
  if(F_CTRLINPUT_RIGHT_PRESSED && !F_CTRLINPUT_UP_PRESSED &&
     !F_CTRLINPUT_DOWN_PRESSED)
  {
    if (plane_x < 176) plane_x++;
    dir_plane = ANIM_RIGHT;
  }


  // DOWN + RIGHT
  if(F_CTRLINPUT_DOWN_PRESSED && F_CTRLINPUT_RIGHT_PRESSED)
  {
    if (plane_y < 96) plane_y++;
    if (plane_x < 176) plane_x++;
```

```
    dir_plane = ANIM_DOWNRIGHT;
   }

   // DOWN only
   if(F_CTRLINPUT_DOWN_PRESSED && !F_CTRLINPUT_RIGHT_PRESSED &&
     !F_CTRLINPUT_LEFT_PRESSED)
   {
    if (plane_y < 96) plane_y++;
    dir_plane = ANIM_DOWN;
   }

   // DOWN + LEFT
   if(F_CTRLINPUT_DOWN_PRESSED && F_CTRLINPUT_LEFT_PRESSED)
   {
    if (plane_y < 96) plane_y++;
    if (plane_x > 0) plane_x--;
    dir_plane = ANIM_DOWNLEFT;
   }

   // LEFT only
   if(F_CTRLINPUT_LEFT_PRESSED && !F_CTRLINPUT_UP_PRESSED &&
   !F_CTRLINPUT_DOWN_PRESSED)
   {
     if (plane_x > 0) plane_x--;
    dir_plane = ANIM_LEFT;
   }

   // UP + LEFT
   if(F_CTRLINPUT_UP_PRESSED && F_CTRLINPUT_LEFT_PRESSED)
   {
    if (plane_y > 0) plane_y--;
    if (plane_x > 0) plane_x--;
    dir_plane = ANIM_UPLEFT;
    }

   return;
  } // End of query_keys()

  // Redraws the plane
  void redraw_plane()
  {
   ham_SetObjX(plane[0],plane_x);
   ham_SetObjY(plane[0],plane_y);

   return;
  } // End of redraw_plane()
```

Code Explanation

```
// Defines
#define ANIM_UP          0
#define ANIM_UPRIGHT     1
#define ANIM_RIGHT       2
#define ANIM_DOWNRIGHT   3
#define ANIM_DOWN        4
#define ANIM_DOWNLEFT    5
#define ANIM_LEFT        6
#define ANIM_UPLEFT      7
```

(I'll assume you know what defines are.) What I am doing is setting up a way to keep track of direction and we'll use this when loading the image that corresponds to the direction.

// Global Variables
These should all make sense to you.

// Function Prototypes
These are almost identical to those from Day 7.

// Main function
int main()
You should notice that almost everything is the same as Day 7 as well except for the following code.

// Setup the plane
plane[0] = ham_CreateObj(...)
This call is almost identical to the one used in Day 5 except that the shape and size are 0,3 this time. That corresponds to and 8x8 8 pixel square, which just means a 64x64 pixel square, and that is what I said the plane sprite is.

// VBL function
void vblFunc()
Again, this is the same as Day 7 except for the first line, which is very important.

ham_UpdateObjGfx(...);
This call is used to update a sprite. Basically our plane sprite index will be updated with the part of the image that corresponds to the direction in which the plane is flying. For example, I defined *dir_plane = 2* at the beginning of the code because I want the plane to start out flying to the right. (Notice *ANIM_RIGHT = 2*). Now, 'where does the 4096 come from?' you ask. Remember that the sprite is 64x64 pixel square and 64 * 64 = 4096. That means that every 4096 spots in the array corresponds to one direction for the plane. 'Huh?' you ask. Well, take a look at the following:

red_plane_anim_Bitmap[0] corresponds to the plane flying straight up.
red_plane_anim_Bitmap[4096] corresponds to the plane flying diagonally up and to the right.
red_plane_anim_Bitmap[8192] corresponds to the plane flying to the right.
... and so on ...
So, *dir_plane* = *2* and 4096 * 2 = 8192. Therefore we update *plane[]* with the data starting at *red_plane_anim_Bitmap[8192]*.

One final note on this function call. You'll notice the *(void *)* before *&red_plane_anim_Bitmap[...]*. HAM will handle the function call without *(void *)*, and your code will compile, it just makes the error messages disappear.

// Query the keyboard
void query_keys()
This function should be pretty easy to figure out. I know that this function could be implemented a little differently, but I am going for ease of understanding right now and not necessary all-out speed. You might wonder where I got the 176 & 96. Easy!
240 (screen width) - 64 (sprite width) = 176
160 (screen height) - 64 (sprite height) = 96

// Redraws the plane
void redraw_plane()
Another function that is almost identical to Day 7.

Well, that's it for Day 8, isn't that great, can't wait ... for Day 9.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Graphics: background.pal.c, object.pal.c, red_plane_anim.bmp, red_plane_anim.raw.c, water.bmp, water.map.c, water.raw.c
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day8_Sprites2_Animation.zip

HAM Version 2.30 And Lower
Necessary for all HAM programs: makefile
Graphics: background.pal.c, object.pal.c, red_plane_anim.bmp, red_plane_anim.raw.c, water.bmp, water.map.c, water.raw.c
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day8_Sprites2_Animation.zip

View Demo Now

Discuss Day 8

< < HOME > >

HAM Tutorial :: Day 9 :: Maps

Some of you may wonder after looking at this tutorial what the purpose is or how it is different from Day 6. Honestly, there isn't a lot that's different except that today the map size will be 512x512 and you can see how to make a map that can be scrolled around.

We are going to do things a little differently for today's graphics. The reason is that I don't want an optimized palette. If you've been paying attention, you should have noticed that the *gfx2gba* optimizes the palette by removing colors not used in your images. This is usually fine. Today, however, I want to reference a specific color when I use *ham_SetTextCol()* and it's much easier to figure out the index number of the color in Photoshop ahead of time.

So, to convert the graphics, type:
gfx2gba -fsrc -m -t8 -x simple_world.bmp

If you look at the gfx2gba readme, you'll see that *-x* prevents optimizing your palette. You'll also notice that I left out *-p<palette_name>* because *gfx2gba* ignores it when you use -x.

Your **gfx** directory should now have the following files:
simple_world.pal.c
simple_world.map.c
simple_world.raw.c

Now let's get to the code:

```
// The Main HAM Library
#include "mygba.h"

// Graphics Includes
#include "gfx/simple_world.pal.c"
#include "gfx/simple_world.raw.c"
#include "gfx/simple_world.map.c"

// Global Variables
u8 newframe; // Signifies a new frame
u16 map_x=0; // X position of the map
u16 map_y=0; // Y position of the map

// Function Prototypes
void vblFunc(); // VBL function

// Main function
int main()
{
  // Variables
  map_fragment_info_ptr bg_simple_world; // Pointer to our background
```

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

```
// Initialize HAMlib
ham_Init();

// Initialize the text display system
// on the background of your choice
ham_InitText(0);

// Setup the background mode
ham_SetBgMode(0);

// Initialize the palettes
ham_LoadBGPal(&simple_world_Palette,256); // Background palette

// Set the text color
ham_SetTextCol(195, 40);

// Setup the tileset for our image
ham_bg[1].ti = ham_InitTileSet(&simple_world_Tiles,
SIZEOF_16BIT(simple_world_Tiles),1,1);

// Setup the map for our image
ham_bg[1].mi = ham_InitMapEmptySet(3,0);
bg_simple_world = ham_InitMapFragment(&simple_world_Map,
                                      64,64,0,0,64,64,0);

// Copy (in this case the whole) map to BG1 at x=0, y=0
ham_InsertMapFragment(bg_simple_world,1,0,0);

// Display the background
ham_InitBg(1,1,2,0);

// Start the VBL interrupt handler
ham_StartIntHandler(INT_TYPE_VBL,&vblFunc);

while(1)
{
 if(newframe)
 {
  // Write some stuff to the screen ...
  ham_DrawText(1,1,"HAM 512 by 512 Map example");
  ham_DrawText(1,2," use pad to scroll around");

  ham_DrawText(1,17,"scroll-x: %5d",map_x);
  ham_DrawText(1,18,"scroll-y: %5d",map_y);

  // ... and let people move the map around
  if(F_CTRLINPUT_DOWN_PRESSED)
  {
   if(map_y<(512-160))
    M_BG1SCRLY_SET(map_y++)
```

```
    }
    if(F_CTRLINPUT_UP_PRESSED)
    {
     if(map_y>0)
       M_BG1SCRLY_SET(map_y--)
    }
    if(F_CTRLINPUT_LEFT_PRESSED)
    {
     if(map_x>0)
       M_BG1SCRLX_SET(map_x--)
    }
    if(F_CTRLINPUT_RIGHT_PRESSED)
    {
     if(map_x<(512-240))
       M_BG1SCRLX_SET(map_x++)
    }
    newframe=0;
  } // End of if(newframe)
 } // End of while(1)

 return 0;
} // End of main()

// VBL function
void vblFunc()
{
 newframe=1; // Starting a new frame

 return;
} // End of vblFunc()
```

Code Explanation

// Global Variables
newframe is used in *main()* in the *while* loop to determine if it's a new frame or not. 'When does a new frame occur?' you may be thinking. Well, every vertical blank (VBL), of course!
map_x and *map_y* should hopefully be clear to you. They keep track of the top-left corner of the part of the map that will be displayed on the screen.

Now on to *main()*

// Set the text color
ham_SetTextCol(195, 40);
This should be a pretty simple function to figure out. Basically you pass it the index number from the background palette of the color you want for the foreground and background font color. 'How do I figure out the index number?' you ask. Well, take a look at the Graphics FAQ. If you look at the palette used with *simple_world.bmp*, you'll notice that 195 is red and 40 is a purplish color.

// Setup the map for our image
ham_bg[1].mi = ham_InitMapEmptySet(3,0);
mymap = ham_InitMapFragment(&simple_world_Map,64,64,0,0,64,64,0);
You hopefully noticed that I am passing 64,64 instead of 30,20. Why am I doing this? Well, I am setting up a map which is 512x512 pixels instead of a 240x160 background.

Now let's examine *while(1)*

if(newframe)...
This will be set to 1, or true, every VBL.

if(F_CTRLINPUT_DOWN_PRESSED)... if(map_y<(512-160))
This will allow the map to move down if it's not already at the bottom. Not sure why we use 512 - 160? Well, there the screen height is 160 pixels. The image height is 512 pixels. The top line of the image will be at 512 - 160 pixels when the map is all the way at the bottom.
Still not sure? Click here.

M_BG1SCRLY_SET(map_y++)
This is just a macro which is used to scroll a background on the Y-axis (up and down axis). *map_y* is incremented 1 because down was pressed. I think the rest of these checks and the other background scrolling macros should be understandable.

There is one more thing you need to know about here - tiles. There is a limit to the number of tiles you can load at once - 651 (I believe). Because my image is 512x512 pixels and because I am loading the whole image at once, this becomes something I have to consider. When you run *gfx2gba* you'll notice that before optimization there are 4096 tiles and after optimization there are only 431. If my image were only a bit more complicated, then this would be a problem and I would have to load the image differently. There are other ways to do this kind of scrolling, but this is one of the easiest ways to understand for now.

That's pretty much it for Day 9. I hope this isn't too complicated to understand.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Graphics: simple_world.map.c, simple_world.pal.c, simple_world.raw.c, simple_world.bmp
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day9_Maps.zip

HAM Version 2.30 and Lower
Necessary for all HAM programs: makefile
Graphics: simple_world.map.c, simple_world.pal.c, simple_world.raw.c, simple_world.bmp
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day9_Maps.zip

View Demo Now

Discuss Day 9

<< HOME >>

HAM Tutorial :: Day 10 :: Sprites #3 - Animation #2

Today's tutorial is very similar to Day 8, except that we will animate the plane image so that it looks like the propellor is spinning. I started out with the graphics from Day 8 and then manipulated them. You can see what I mean here. For the sake of saving time I decided that the plane will only move up, down, left or right and not in any diagonal direction.

One of the purposes of this tutorial is to give you a basic understanding of timers. Make sure that you understand the *frames* variable and how it is used to animate the propellor.

To convert the graphic, use the following:
gfx2gba -fsrc -m -pbackground.pal -t8 water.bmp

Again, the sprite will not be in a tile/map format, so type:
gfx2gba -D -fsrc -pobject.pal -t8 red_plane_anim.bmp

Your **gfx** directory should have the following files:
background.pal.c
object.pal.c
red_plane_anim.raw.c
water.map.c
water.raw.c

On to the code:

```
// The Main HAM Library
#include "mygba.h"

// Graphics Includes
#include "gfx/background.pal.c"
#include "gfx/object.pal.c"
#include "gfx/water.raw.c" // gfx2gba -fsrc -m -pbackground.pal -t8
                                water.bmp
#include "gfx/water.map.c"
#include "gfx/red_plane_anim.raw.c" // gfx2gba -D -fsrc -pobject.pal -t8
                                        red_plane_anim.bmp


// Defines
#define ANIM_UP         0
#define ANIM_RIGHT      1
#define ANIM_DOWN       2
#define ANIM_LEFT       3


// Global Variables
u32 dir_plane = 1; // Direction the plane is facing, 0-3
u32 plane[1]; // Sprite index for the plane
```

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

```
u32 plane_x = 110; // X position of the plane
u32 plane_y = 50; // Y position of the plane
u32 animcnt = 0; // Current frame of animation, 0-3
u32 frames = 0; // Global frame counter
u32 array_spot = 0; // Stores the location of the sprite's current image

// Function Prototypes
void vblFunc(); // VBL function
void query_keys(); // Query the keyboard
void redraw_plane(); // Redraws the falling block

// Main function
int main()
{
 // Variables
 map_fragment_info_ptr bg_water; // Pointer to our background

 // Initialize HAMlib
 ham_Init();

 // Setup the background mode
 ham_SetBgMode(1);

 // Initialize the palettes
 ham_LoadBGPal(&background_Palette,256); // Background palette
 ham_LoadObjPal(&object_Palette,256); // Sprite palette

 // Setup the tileset for our image
 ham_bg[0].ti = ham_InitTileSet(&water_Tiles,
                                SIZEOF_16BIT(water_Tiles),1,1);

 // Setup the map for our image
 ham_bg[0].mi = ham_InitMapEmptySet(3,0);
 bg_water = ham_InitMapFragment(&water_Map,30,20,0,0,30,20,0);
 ham_InsertMapFragment(bg_water,0,0,0);

 // Display the background
 ham_InitBg(0,1,0,0);

 // Setup the array spot
 array_spot = (16384 * dir_plane) + (4096 * animcnt);

 // Setup the plane
 plane[0] = ham_CreateObj(
                (void *)&red_plane_anim_Bitmap[array_spot],
                0,3,OBJ_MODE_NORMAL,1,0,0,0,0,0,0,0,plane_x,plane_y);

 // Start the VBL interrupt handler
 ham_StartIntHandler(INT_TYPE_VBL,&vblFunc);

 while(1)
 {
  // Infinite loop to keep the program running
 }

 return EXIT_SUCCESS;
```

```
} // End of main()


// VBL function
void vblFunc()
{
 ham_CopyObjToOAM(); // Copy plane sprite to hardware
 query_keys(); // Check for movement
 redraw_plane(); // Redraw the plane
 frames++; // Increment the frame counter

 return;
} // End of vblFunc()


// Query the keyboard
void query_keys()
{
 // UP only
 if(F_CTRLINPUT_UP_PRESSED && !F_CTRLINPUT_RIGHT_PRESSED
   && !F_CTRLINPUT_LEFT_PRESSED)
 {
  if (plane_y > 0) plane_y--;
  dir_plane = ANIM_UP;
 }

 // RIGHT only
 if(F_CTRLINPUT_RIGHT_PRESSED && !F_CTRLINPUT_UP_PRESSED
   && !F_CTRLINPUT_DOWN_PRESSED)
 {
  if (plane_x < 176) plane_x++;
  dir_plane = ANIM_RIGHT;
 }

 // DOWN only
 if(F_CTRLINPUT_DOWN_PRESSED && !F_CTRLINPUT_RIGHT_PRESSED
   && !F_CTRLINPUT_LEFT_PRESSED)
 {
  if (plane_y < 96) plane_y++;
  dir_plane = ANIM_DOWN;
 }

 // LEFT only
 if(F_CTRLINPUT_LEFT_PRESSED && !F_CTRLINPUT_UP_PRESSED
   && !F_CTRLINPUT_DOWN_PRESSED)
 {
  if (plane_x > 0) plane_x--;
  dir_plane = ANIM_LEFT;
 }

 return;
} // End of query_keys()


// Redraws the plane
void redraw_plane()
```

```
{
 // We'll only update the animation every 5th frame
 if (frames > 5) {
  frames = 0; // Reset the frame counter
  array_spot = (16384 * dir_plane) + (4096 * animcnt); // Figure out
                                                  where to load the image from
  ham_UpdateObjGfx(plane[0],
              (void*)&red_plane_anim_Bitmap[array_spot]); // Update it
  // Increment the animation counter
  if (animcnt == 3) {
    animcnt = 0;
  } else {
    animcnt++;
  }
 }

 // Make sure to set the (new) plane co-ordinates
 ham_SetObjX(plane[0],plane_x);
 ham_SetObjY(plane[0],plane_y);

 return;
} // End of redraw_plane()
```

Code Explanation

// Defines ...
This is similar to Day 8. There are fewer directions because it would have taken a while to create the extra graphics.

// Global Variables
All of these are basically the same except for *animcnt*. This is used to keep track of the current image (out of 4 possible) for the current plane direction.

// Main function
int main()
The main thing here the following code:

// Setup the array spot
*array_spot = (16384 * dir_plane) + (4096 * animcnt);*
Let me explain the numbers. Remember the plane is a 64x64 pixel graphic and that 64 * 64 = 4096. Therefore one plane image is an array of 4096 pixels. There are 4 frames per direction (to make the propellor appear to be moving). 4 * 4096 = 16384. Huh? Take a look at the following chart.

	Index	Index	Index	Index
Direction	**Frame 1**	**Frame 2**	**Frame 3**	**Frame 4**
UP	0	4096	8192	12288
RIGHT	16384	20480	24576	28672

DOWN	32768	36864	40960	45056
LEFT	49152	53428	57344	61440

// VBL function
void vblFunc()
This is the same as in previous days. Notice that *frames* is incremented every VBL.

// Query the keyboard
void query_keys()
This should be pretty easy to understand as well.

// Redraws the plane
void redraw_plane()
This is where the animation happens. Every five frames the plane sprite gets updated with the next frame of animation.

Hmm. I guess that's it for Day 10. Make sure to tinker with the code to see that you really understand it. You may want to try updating the graphic every 10 or 20 frames to see what happens.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Graphics: background.pal.c, object.pal.c, red_plane_anim.bmp, red_plane_anim.raw.c, water.bmp, water.map.c, water.raw.c
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day10_Sprites3_Animation2.zip

View Demo Now

Discuss Day 10

<< HOME >>

HAM Tutorial :: Day 11 :: Backgrounds - Rotation

For Day 11 I'll explain how to rotate backgrounds. HAM makes this very simple with the RotBgEx() function. This function is very powerful because not only can you rotate backgrounds but also zoom in or out and scroll backgrounds.
NOTE: There is also a RotBg() function, but I prefer the newer RotBgEx() because you have more control over the rotation.

Oh, and don't forget. You can only rotate in Mode 1 and 2. In Mode 1, background 2 can rotate/scale. In Mode 2, backgrounds 2 & 3 can rotate/scale. In case you forgot, take a look at Day 1 again.

To convert the graphic, use the following:
gfx2gba -fsrc -m -protate.pal -rs -t8 rotate.bmp

Your **gfx** directory should have the following files:
rotate.map.c
rotate.pal.c
rotate.raw.c

On to the code:

```
// The Main HAM Library
#include "mygba.h"

// Graphics Includes
// gfx2gba -fsrc -m -protate.pal -rs -t8 rotate.bmp
#include "gfx/rotate.map.c"
#include "gfx/rotate.pal.c"
#include "gfx/rotate.raw.c"

// Global Variables
u32 frames = 0;
u32 zoomx = 256; // X co-ordinate of zoom center
u32 zoomy = 256; // Y co-ordinate of zoom center
u32 scrollx = 64; // X co-ordinate of scrolling center
u32 scrolly = 64; // Y co-ordinate of scrolling center
bool zoomin = 1;

// Function Prototypes
void vblFunc(); // VBL function
void query_keys(); // Query the keyboard
void redraw(); // Redraw the screen


// Main function
int main()
{
  // Initialize HAMlib
  ham_Init();

  // Setup the background mode
```

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

```
  ham_SetBgMode(1);

  // Initialize the palettes
  ham_LoadBGPal(&rotate_Palette,256); // Background palette

  // Initialize the rotating background
  ham_bg[2].ti = ham_InitTileSet(&rotate_Tiles,
                                 SIZEOF_16BIT(rotate_Tiles),1,1);
  ham_bg[2].mi = ham_InitMapSet(&rotate_Map,256,0,1);

  // Display the background
  ham_InitBg(2,1,1,0);

  // Move the background to the center
  ham_RotBgEx(2,0,120,80,scrollx,scrolly,zoomx,zoomy);

  // Start the VBL interrupt handler
  ham_StartIntHandler(INT_TYPE_VBL,&vblFunc);

  while(1)
  {
    // Infinite loop to keep the program running
  }

  return EXIT_SUCCESS;
} // End of main()


// VBL function
void vblFunc()
{
  ham_CopyObjToOAM(); // Copy plane sprite to hardware
  query_keys(); // Check for movement
  redraw(); // Redraw the plane

  // Rotate the background
  ham_RotBgEx(2,frames%360,120,80,scrollx,scrolly,zoomx,zoomy);

  // Increment the frames
  ++frames;

  return;
} // End of vblFunc()


// Query the keyboard
void query_keys()
{
  // RIGHT only
  if (F_CTRLINPUT_RIGHT_PRESSED)
  {
    if (scrollx < 127) ++scrollx;
  }

  // LEFT only
  if (F_CTRLINPUT_LEFT_PRESSED)
  {
```

```
    if (scrollx > 0) --scrollx;
  }

  // START only
  if (F_CTRLINPUT_START_PRESSED) {
    scrollx = 64;
    scrolly = 64;
  }

  return;
} // End of query_keys()


// Redraw the screen
void redraw()
{
  if (zoomin) { // Zoom in
   if (zoomx != 396) {
    zoomx+=2;
    zoomy+=2;
   } else {
    zoomin = FALSE;
   }
  } else { // Zoom out
   if (zoomx != 116) {
    zoomx-=2;
    zoomy-=2;
   } else {
    zoomin = TRUE;
   }
  }

  return;
} // End of redraw()
```

Code Explanation

// Global Variables

u32 zoomx=256;
u32 zoomy=256;
These are used for the amount of zooming in the X and Y direction. In the HAM documentation, you will see that the author uses 0x100 for no zoom. This is a hexadecimal number which equals 256 in decimal. You can use either system when you call this function. Increasing the number increases the zoom, decreasing decreases.

u32 scrollx=64;
u32 scrolly=64;
These are used to scroll the point in the image that corresponds to the point of rotation in the X and Y directions. In this case I set them to the center of the image (it's a 128x128 pixel image). Be careful that you don't try to set the scroll point outside of the image's boundaries. This is explained in more detail later.

```
// Main()
```

You'll notice that the way I load the background here is different from how it was done in previous days. This is a shorter way to do it. Either method works fine, though.

```
// Initialize the rotating background
ham_bg[2].ti = ham_InitTileSet(&rotate_Tiles,
                               SIZEOF_16BIT(rotate_Tiles),1,1);
```

This call is exactly the same. The next part is different.

```
ham_bg[2].mi = ham_InitMapSet(&rotate_Map,256,0,1);
```

The first thing passed is the array that contains the map data. Next is "the size of the tiles to be copied into BG RAM, in number of 16bit chunks." The next number depends on the final argument (whether it is a rotation map or not). In this case it is, so the corresponding value for a 128x128 pixel image for a rotation background is 0. Finally, it is passed a 1, of course, because it is a rotation background.

NOTE: You could do the same thing with the following code:

```
// Variables
map_fragment_info_ptr bg_rotate; // Pointer to the background
// Setup the tileset for our image
ham_bg[2].ti = ham_InitTileSet(&rotate_Tiles,
                               SIZEOF_16BIT(rotate_Tiles),1,1);
// Setup the map for our image
ham_bg[2].mi = ham_InitMapEmptySet(0,1);
bg_rotate = ham_InitMapFragment(&rotate_Map,16,16,0,0,16,16,1);
ham_InsertMapFragment(bg_rotate,2,0,0);
// Display the background
ham_InitBg(2,1,0,0);
```

```
// Move the background to the center
ham_RotBgEx(2,0,120,80,scrollx,scrolly,zoomx,zoomy);
```

This is a sneak peak at RotBgEx(). As I said it is a powerful function. I am using it here not to rotate the background image, but to center it on the screen. I will rotate later in the VBL function and explain the arguments then.

```
// VBL function
```

```
// Rotate the background
ham_RotBgEx(2,frames%360,120,80,scrollx,scrolly,zoomx,zoomy);
```

Okay, now this is where the rotation (and zooming) is actually happening. The first argument passed is the background number. (It can only be 2 or 3 as you can't rotate BGs 0 and 1.) The next argument is the angle of rotation. Using *frames%360* the value will be 0 - 359. After that comes the center of rotation on the X and Y plane. I am using the center of the screen which is at 120, 80 (well, almost). Stop for a moment and make sure you understand that. This is the point around which the image will rotate. Next you pass the X and Y co-ordinates for the scroll offset. This is the point of the image that will correspond to the point of rotation. I chose 64,64 - the center of the image (again, almost). Take another moment to picture that

properly. The center of the image will rotate around the center of the screen. Finally you pass the amount you want to zoom in the X and Y direction. It starts out at 256 (or 0x100 in hex). This means no zoom. If you look at the redraw function, you will see how *zoomx* and *zoomy* are modified.

// query_keys()

if (scrollx < 127) ++scrollx;
if (scrollx > 0) --scrollx;
While the demo is running, the user can press left or right to move the point of the image that corresponds to the center of rotation. For example, if you move all the way to the left, the image will rotate with the left-hand side around the middle. Notice the 'R' in *Rotate* is near the center. If you press to the right, then the right-hand side will be near the middle and the 'e' in *Rotate* will be near the center. Oh, and if the user presses Start, it resets the scroll offset to the middle.

// redraw()

The code here is not difficult. I set the max for zooming in at 396 and the max for zooming out at 116. When a max is met, I swap *zoomin* so that it will swap between zooming in and then zooming out.

Ahh, that's it. Another Day, another wonderful lesson in the fine art of GBA programming with HAM. You should definitely tinker with the code to see what all you can do with RotBgEx() before moving on to Day 12.

Download Code

NOTE: You may need to Right-click and choose Save As.

HAM Version 2.40 And Higher
Necessary for all HAM programs: makefile
Graphics: rotate.map.c, rotate.pal.c, rotate.raw.c
Main source file: main.c
Compiled binary: hello.gba

Download All Files In One Zip: Day11_BG_Rotation.zip

View Demo Now

Discuss Day 11

<< HOME >>

HAM Tutorial :: Version History

Version 1.7.7 :: 01/24/2003
- Downloads: Updated PDF files

Version 1.7.6 :: 01/17/2003
- HAM Tutorial Hearts : Version 3.4

01/17/03 - GBAx Competition
- Make sure you make a cool game and enter to win here

Version 1.7.5 :: 01/15/2003- Wow, has it been almost 2 months?
- Removed Advertisements from the website. I am not bitter, they just aren't doing anything for me. I finally got a job anyway!
- Day 2 :: Fixed makefile. Don't know how/why the previous one was uploaded.

Version 1.7.4 :: 11/20/2002
- Lend a helping HAM day! I am going to mirror the official HAM downloads to help Emanuel save bandwidth. Go here!

Version 1.7.3 :: 11/05/2002
- HAM Tutorial Hearts : Version 3.3

Version 1.7.2 :: 11/04/2002
- HAM Tutorial Hearts : Version 3.2

Version 1.7.1 :: 11/03/2002
- HAM Tutorial Hearts : Version 3.1
  Added some screenshots of the game, too.

Version 1.7.0 :: 11/02/2002
- HAM Tutorial Hearts : Version 3.0
  (Now released under the GPL)
- Games: I added this section to show you some of the games I have been working on. Right now there are two, TicTacToe and Hearts. Both of them are basically complete. They were written with C++.

Version 1.6.9 :: 10/31/2002 - HAPPY HALLOWEEN!
- HAM Tutorial Hearts : Version 2.8

Version 1.6.8 :: 10/27/2002
- HAM Tutorial Hearts : Version 2.7

Version 1.6.7 :: 10/26/2002
- HAM Tutorial Hearts : Version 2.6

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

Version 1.6.6 :: 10/25/2002
- HAM Tutorial Hearts : Version 2.5

Version 1.6.5 :: 10/24/2002
- HAM Tutorial Hearts : Version 2.4

Version 1.6.4 :: 10/23/2002
- Day 7, 9 : Cleaned up references to *AgbMain()* instead of *main()*.
- Day 9 : Fixed a comment that refered to the wrong background number.
- HAM Tutorial Hearts : I just found out that I took 3rd place in the GbaDev.org Hearts competition. Pretty sweet! I wish I had more time, 'cause the the version I submitted could have been better. Oh well. I have since updated it and it is now very playable! Check out the page here.

Version 1.6.3 :: 10/15/2002
- Removed link to Flash Linker on LikSang.com and added one to EasyBuy2000.com.

Version 1.6.2 :: 10/13/2002
- Links : Added more links.
- Advertisements: Added more advertisements. :- |

Version 1.6.1 :: 10/12/2002
- Links : Added some cool development links.

10/11/02 - Happy 1/2 Birthday!
- The HAM Tutorial website is now 6 months old.
  In the first 6 months it got over 22,000 hits!!!

Version 1.6.0 :: 10/10/2002
- Day 11 : Backgrounds - Rotation added.
- Advertisements. Yup, I have decided to add them. I hope this doesn't make you all too mad, but it would be nice to get something in return for what I've done in my spare time. On the bright side, there will be no popups - I promise.

Version 1.5.1 :: 10/07/2002
- Day 5 : Typo fixed.
- Day 6 : Typo fixed. Broken link removed. Code cleanup.

Version 1.5.0 :: 10/06/2002
- Day 10 : Done. Check it out.
- Poll : Finally updated.

Version 1.4.5 :: 10/05/2002
- Forgot to mention this a while back - gfx2gba 0.13. It comes with HAM 2.50. Get it here and take a look at the *Readme.txt*.

Version 1.4.4 :: 10/04/2002

- Graphics FAQ : Added info

Version 1.4.3 :: 10/03/2002
- General site cleanup
- Day 7 : Typo fixed
- Links : Added more game development links & reordered current ones

Version 1.4.2 :: 10/02/2002
- Downloads : PDF and source files have all been updated to the current version.

Version 1.4.1 :: 09/12/2002
- Demos are fixed! Well, not really. I just recompiled them with HAM 2.40. The problem is that sound is not supported in Boycott Advance Online.

Version 1.4.0 :: 09/11/2002
- HAM 2.50 Released
- All tutorials have been tested (and work) with HAM 2.50
- Demos are not working! I am trying to figure this out.

Version 1.3.11 :: 09/06/2002
- Links : Added a few more game development links

Version 1.3.10 :: 08/22/2002
- Project 3 (Slide Show) : Updated for HAM 2.40.
- Day 9 : Small typos fixed. Updated *Readme.txt* file.

Version 1.3.9 :: 08/21/02 - Back In The USA!!!
- Day 8 : Updated for HAM 2.40.
- Day 9 : Updated for HAM 2.40.

Version 1.3.8 :: 07/28/2002 - HAM 2.40 Updates Again
- Day 5 : Typo fixed.
- Day 6 : Updated for HAM 2.40.
  Fixed the zip file for HAM 2.30. It was really messed up. Oops!
- Day 7 : Updated for HAM 2.40.
- History : Typo fixed.

Version 1.3.7 :: 07/11/2002 - More HAM 2.40 Updating
- Day 3 : Updated for HAM 2.40.
- Day 4 : Updated for HAM 2.40.
- Day 5 : Updated for HAM 2.40.

Version 1.3.6 :: 07/03/2002 - Updating Time
- Well, I am finally getting settled in here in Japan. I don't know how often I will be able to find the time, but I am trying very hard to get the software udated for HAM 2.40.
- Day 2 : Updated for HAM 2.40.
- HAM 2.40 : For those of you who have been living under a rock, HAM is up to version 2.40. There were lots of fixes and additions in this version, some of which broke old code.

Go here to get it.
- gfx2gba 0.11 & 0.12 : Click here to download the file.
Click here for the readme file.

06/16/2002 - HAM Tutorial Status Update
- You may have noticed the lack of updates and news lately. Don't panic, I haven't discontinued work on the site. Instead, real life has caused me to put my work on the HAM Tutorials on hold for a couple of months. I just packed up my stuff and moved from Ohio to San Francisco this week with my girlfriend. I will then be in Japan for an internship from June 20th through August 16th. When I return I intend to get back to working on the tutorials again as much as I can.
- Thank you all for your positive feedback. These past couple of months I have received so many emails from you expressing your thanks for the tutorials. I have precious little spare time to devote to the tutorials and this makes it all feel worthwhile. Okay, enough of that...
- Oh yeah, HAM is now up to 2.4! I have not had the time to test my tutorials with the new version so 'use at your own risk.' There are a few neat additions, as well as the usual bug fixes, so it may be worth it. If you want to play it safe, just rename your current ham directory to something like *ham_2.30* and then install the new one. You can then use either version to compile your code.

Version 1.3.5 :: 05/19/2002
- Day 9 : Fixed a lot of typos. (Thanks **marke_gba**)
- Games: I've been working on my first full game for the GBA. I'll let you all know about it when it's ready.
- gfx2gba : There's a new version 0.10 out. I haven't had a chance to check it out yet, though. Download, Readme
- Project 3 : Added an intro describing the project. Small typos fixed.

Version 1.3.4 :: 05/14/2002
- Projects : Added a new section called Projects. Currently only my latest (and greatest?) project is ready for you. I've got to redo the code on the other two a bit before I release them to everyone.
- Downloads : Some cleanups here and there.

Version 1.3.3 :: 05/08/2002
- Day 5 : Added a link to clarify what I meant by *the chart*. Sorry about the confusion.

Version 1.3.2 :: 05/06/2002
- Day 9 : Seems to be okay, so it's official.
- Polls : New poll added today. Looking forward to HAM 2.40.
- PDFs : Updated
- Added a link on the contents panel to the translations

Version 1.3.1 :: 05/05/2002
- Downloadable PDFs are now available! Check the Downloads page!
- Day 9 : Currently in beta. It is being checked for accuracy and should be officially released tomorrow.

- Credits, Links, GFX2GBA Readme : Minor cleanups.

Version 1.3.0 :: 05/01/2002 - Correspondence Day!
- Polls : I decided to add polls to the web site. What better way than this to get feedback from you?! Check it out here.
- Discussion Boards : Each day now has a discussion board. Click the link at the bottom of each page or click here to see them all.
- Fixed some small typos on this page
- Added links to the MVB2 FAQ to the homepage and Links page

Version 1.2.3 :: 04/29/2002
- Links : Added a link to VisualHAM
- Removed comment on the homepage about the source being in C++ This may get added again later if I actually use C++ code.

Version 1.2.2 :: 04/25/2002
- Translations: Added more links. I hope this helps everyone.
- Day 2: Removed comment about copying *crt0.s* to your source directory.
- Added downloadable *am.bat* for those who want it.
- Added info about the MBV2 cable and a link to where it can be bought.

Version 1.2.1 :: 04/24/2002 - Translation Day!
- Translations: Added a page with links to sites that will translate my tutorial automatically. There will be some weirdness, but I hope y'all can figure it out.

Version 1.2.0 :: 04/23/2002
- Update to HAM 2.30. Everything seems to be okay again. There is one thing to note. You no longer need the *crt0.s* file in your source directory. The *makefile* is different now as well. Take a look at the *changelog* to see what's new.
- Day 3: Updating to HAM 2.30 seems to have fixed the problem.

Version 1.1.4 :: 04/23/2002
- Links: Adding more links to great stuff on the Links page

Version 1.1.3 :: 04/20/2002
- Day 6: Typo fixed. (thanks **kiwibonga**)
  Added info about map_fragment_info_ptr

Version 1.1.2 :: 04/16/2002
- Day 8: Sprites #2 - Animation added
- Small updates to Links page
- Fixed link to old domain on the Graphics FAQ page

Version 1.1.1 :: 04/15/2002 - Tax Day Edition #2
- Added a single zip file for every day so that you can download all the source easily.
- Updated Credits page. I think it is very important to give credit where credit is due. I know I had to read a lot of documentation on the GBA

before I could get started.

Version 1.1.0 :: 04/15/2002 - Tax Day Edition
- Update to HAM 2.20. Everything works fine except for Day 3. I'll add more info when a fix is found. UPDATE: This runs fine in Boycott Advance but not in VBA Windows or VBA SDL.
- Small code cleanup in Demos section

Version 1.0.3 :: 04/14/2002
- Links: Added a link to HAM's official documentation page

Version 1.0.2 :: 04/13/2002
- Day 1: Fixed explanation about sprite palettes. Remember, there can be one 256 color palette for all of your sprites, or there can be up to 16 different 16 bit palettes.
- Day 2: Added to some words to add intonation (?)
- Day 7: Fixed typo in ham_StartIntHandler() explanation
- Added a link on the home page to HAM's website (doh!)

Version 1.0.1 :: 04/12/2002
- Day 4: Alphabetized parameters for gfx2gba
- Day 5: Minor update to ham_CreateObj() explanation. Alphabetized parameters for gfx2gba
- Day 6: Alphabetized parameters for gfx2gba
- Day 7: Fixed instructions for graphic creation. Fixed typo in link for blocks_L.bmp (thanks **thorston**). Alphabetized parameters for gfx2gba
- Fixed Initial Public release date (oops)
- Somewhat major bug fix, thanks to **thorston** for the heads up Apparently there was some extra junk at the bottom of the makefiles which came with HAM. It could cause compilation to fail. I removed the junk and updated all the makefiles for the tutorial. He also mentioned that when the makefile target has the same name as the directory the files are in, it also could break the build.

Version 1.0.0 :: 04/11/2002
- Initial Public Release

< < HOME > >

HAM Tutorial :: Graphics FAQ

General Questions

Question: I get the following when compiling my program. Why?
warning: `background_Palette' initialized and declared `extern'

Answer: The version of *gfx2gba* that came with HAM 2.50 (version 0.13) added *extern* to the beggining of the array name. This will be changed in version 0.14. You can ignore the error or remove *extern* from the begging of each of your .c graphic files.

NOTE: A common problem people have is creating a bitmap with an 8-bit palette. When I first started GBA programming, I had no idea how to do it. Honestly, I am not a graphics expert, as you can tell by my backgrounds and sprites I've created for my tutorials, but I will try to give you an idea of where to start with what I've found to be common problems.

Photoshop 6.0 - Windows

Currently I do almost everything in Photoshop, so let's start there.

HOWTO: Setup Your Image For *gfx2gba*

Start by creating a new image. You can begin with the Mode set to RGB. Draw the image and then save it as a standard PSD file. The next part is important, so pay attention.

Click **Image** -> **Mode** -> **Indexed Color**
Under **Palette**, choose either **Exact** or **Local (Perceptive)**
  If you pick Local (Perspective you need to do the following as well
  - Set the **Colors** to **256**
  - Set **Forced** to **None**
Click **OK**
Click **File** -> **Save As**
Next to **Format**, choose **BMP**
Under **File Format**, **Windows** should be selected

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

Under **Depth**, **8 bit** should be selected
Click **OK**

That's it. It's pretty simple to get the basic palette setup.

HOWTO: Work With You Palette / Color Index

This is how you determine the index number of a color in your palette or add new ones to it for use with *ham_SetTextCol()*, etc.

Your image should be set to *Indexed Color*. If not, take a look at the the HOWTO for setting up image and do that first.

Click **Window** -> **Show Info**
NOTE: If you only see *Hide Info*, then the info window is already there
Click **Image** -> **Mode** -> **Color Table**

Now, as you move your cursor over the colors, you'll see the index of the color show up in the **Info** window next to **Idx:**. Easy, huh?!

One more thing... If you want to add colors to your index, just click on the first empty spot in the palette. Empty colors should be black and near the end of the chart.

Gimp 1.2.3 - Windows

What the heck is Gimp? Gimp is a Unix program similar to Photoshop which has been ported to Windows. It is free, works very well and there are tons of free plugins, too. If you don't already have Photoshop, I recommend you give it a try. The actual homepage is here, but you can find the Windows port here.

HOWTO: Setup Your Image For *gfx2gba*

Click **File** -> **New** (Set **width** & **height**, everything else default)
Click **OK**
Right-click on the image & choose **Image** -> **Mode** -> **Indexed...**
Generate Optimal Palette should be selected.
# of Colors should be **256** (or 16 if you are doing a 16 bit palette)
Click **OK**

That's all there is to it. Another easy one!

HOWTO: Work With You Palette / Color Index

Click **File** -> **Dialogs** -> **Indexed Palette**
The **Indexed Color Palette** should pop up and you'll see the all of the colors in your palette. Click on the color, who's index you want to know, and it will appear in the **Index** window.

< < HOME > >

HAM Tutorial :: GFX2GBA Readme

The latest version of the readme can be found here.
You can look for the latest file here.

HAM 2.50 Version: 0.13
HAM 2.40 Version: 0.10
HAM 2.30 Version: 0.08

==========
gfx2gba v0.13 README
==========

this tool converts

PCX - 8 bit (256 color)
TGA - 8 bit (256 color)
SPR - 8 bit (256 color)
BMP - 8 bit (256 color) or 4 bit (16 color)
TIM - 8 bit (256 color) or 4 bit (16 color)

graphic files to GBA useable data (raw, tiles, map). The special thing about this tool is that it is able to combine the palettes of several gfx files into one single "master palette". it removes/remaps double colors. of course if you try to combine 2 files with 256 unique colors each it will fail, heh ... :p

Usage: gfx2gba [options] bmp/pcx/tga/spr/tim files ...

Options are:

-pPalettename -m generate map (optimized)
-oOutputdir -M generate map (not optimized)
-fOutput format -mm generate metatiled map (optimized)
-sSection -MM generate metatiled map (not optimized)
-SpSuffix for palette -bBank for map
-SmSuffix for map -mc save map in column order *new*
-StSuffix for tiledata -F turn off check for flipped tiles
-TMetatilesize -rs output a rotate/scale BG screen map
-tTilesize -P don't save palette data
-tc save tiles as columns *new* -G don't save tile/image data
-cColordepth -D don't save map data
-CColor-Offset -X don't save anything
-A force color offset add -Z compress everything
-aTransparent color -zt compress tile/bitmap data
-vVRAM-Offset -zp compress palette data
-x don't merge palettes -zm compress map data
-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 *new*

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

-align adds alignment info to arrays (needed sometimes for GCC)

-mc saves map's in column order instead of rows

-tc saves the tiles in column order instead of rows

-B only optimize/remove BLANK tiles and don't touch others

-A forces color offset adding to ALL colors (it doesn't add the offset to index 0 by default)

-ap use aPLib as compressor instead of lz77

-aps use aPLib (safe mode) as compressor instead of lz77

NOTE: for detailed info about aPLib check
http://home19.inet.tele.dk/jibz/apack/
thanks to Jørgen Ibsen for this excellent piece of software

-q quiet mode reduces output to a minimum

-p specifies the name of the master palette (default: master.pal)

-o specifies the output directory (default is current dir)

-f specifies the output format (raw/src/asm/arm) (default is raw)

-s will allow you to select the ROM section used in the assembler outputfiles (default is .rodata)

-Sp specifices a custom extension for palette files (ie replace ".pal" by something else)

-Sm like above but to customize/replace ".map"

-St like above but to customize/replace ".raw"

NOTE: if C or ASM output is selected, .c or .s will ALWAYS be added to the filename, regardless of what custom extension you selected!

-T specifies the metatilesize (1/8/16/32/64) (default is 1, which means NO metatiling at all)

-t specifies the tilesize (1/8/16/32/64) (default is 1, which means NO tiling at all, giving you plain raw data for mode4/5 bg's)

-c specifies input/output color depth (16/256/32k) (default is 256)

NOTE: in 256 color mode you can also READ 16 color images, but they'll be saved as 256 color images!
NOTE2: 32k setting only works for OUTPUT of raw 15+1bit gfx data for use in MODE3/5, you can NOT READ other formats than 16/256 color BMP or 256 color PCX. this tool was not designed for this mode anyway, so expect some bugs there (especially if you mix it with other options)

-C will add an offset (0-255) to the bitmap color indexes (default 0) it will NOT add the offset to color index #0 (transparent) !!!

-a changes the transparent color (0-255). (default 0)

-v specifies the offset to be added when saving tilemaps (default is 0)

-b specifies the color bank to use for 16 color tile maps (0-15) (default is 0)

NOTE: DO NOT USE SPACES between -p/-o/-s/-Sp/-Sm/-St-t/-c/-v/-C/-b and it's parameter!!! use -t16 or something like that, but -t 16 won't work!!!

-m will generate an optimized tile map from the inputfile. optimized means it removes double tiles and also checks for horizontal, vertical and h+v flipped doubles. of course it sets the correct flags in the tile map data. the output .raw file will ONLY contain the used tiles (of course).

-M will generate a NON optimized tilemap

-mm will tile a generated map into 32x32 tile sub-maps for easy DMA'ing them around

-MM same as -mm just for non optimized maps

-F will turn off check for flipped tiles in map. only "real" doubles will be removed.

NOTE: activating map generation will change default tile size from 1 to a more GBA friendly size of 8. you can still set your own size using the -t option.
fixed a stupid bug that produced an endless loop in v0.6 if your input gfx bitmap was >= 4mbyte.

-rs will force the map to be saved in 1 byte per tile format making it suitable for rotate/scale backgrounds. you lose color bank info and h/v flip info obviously and the max number of tiles is 256. but hey, you knew that already, don't ya?
NOTE: this one is fixed now, had a silly bug that was not catched by -Wall ... :/

-P will disable palette output at all

-G will disable graphics (tiles/images) output at all

-D will disable map output at all

-X will disable all data output

-x will prevent gfx2gba from merging/optimizing your palettes and save a separate .pal file for each input gfx file.

-zt will lz77 compress tile/gfx data

-zp will lz77 compress palette data

-zm will lz77 compress map data

-Z will lz77 compress everything

NOTE: all compression stuff is compatible to the GBA internal Lz77 depacker.
if it fails to compress it will show you a warning and save the normal data.

==========
Frequently Asked Questions
==========

Q: so far so good, but what if i want to convert all my 36 .bmp files and 12 .pcx files ?!
A: easy: gfx2gba *.pcx *.bmp
could it be any simpler?! :)

Q: what -t value do i need for correct sprites and background tiles?
A: use -t8 to get what you need ...

Q: but i am using Linux for GBA dev?!
A: just use the included Linux binary (you may have to set exec flag)

Q: wtf is -v good for? why would i want to add an offset?!
A: HINT: to maximize the number of useable tiles.

Q: is it possible that your tool is the only free available one that also supports 16 color / 4 bit gfx?!
A: erm... well... seems so... looks like other tool coders ignore the presence of the 16 color modes on GBA ...

Q: v0.4 features are the same as in v0.3?! what's new or changed?!

A: *cough* this release has 2 major bugs fixed. bitmap export to 4bit (16 color) was broken (pixel swapped) (export to tiles was working, that's why i found this nasty bug so late) and my color optimizing stuff made ALL black (0x0000) colors transparent (thanks to Michael Jagger for yelling at me about this)

Q: soo, what's new in v0.5 ? i still can't see any new options?!
A: *cough*cough* the black color bug was nastier than i thought but at the end they all DIE ... or short: the last bug in color optimizing is removed now ... fixed bitmap remapping (last pixel wasn't remapped) and finally it compiles now without even a warning! :-)

Q: hey, i can mix all kind of options and gfx2gba doesn't complain about stuff that doesn't make sense?!
A: it is a tool to help you developing GBA stuff, not a brain replacement and it assumes you know at least a little bit what you are doing.

Q: I have a problem with 32k mode...
A: I don't care ... 32k mode is just a little add-on and not something high-priority for me ... :)

Q: hey, who's responsible for the color bank & rot/scale option? i don't think you're smart enough to think about 'em yourself!?
A: sad but true ... thanks to Alex D. for this obviously useful ideas. cheers mate, keep the ideas coming (even if they arrive in my inbox at 00:30 in the morning)!

Q: are there any other people who are giving you their valuable ideas free of charge?
A: yeah! many thanks to Dennis R. for the idea to change some of the default values and kicking my butt to implement C/C++ array output at least. and then we have Bengt Johannesson who found out that the symbol name creation was f*cked when you used files not placed in the current directory (it had the complete path in the label name instead of just the filename) ...

Q: who wanted the l33t flipped tile check to be removable!?
A: Nikita Mikros needed it so i added it ... :)

Q: metatiling? wow! cool name for a feature, BUT WHAT THE HELL IS IT?!
A: it more or less means that your input gfx will get tiled twice. for example my game "engine" is building large objects from 16x16 sprites. but 16x16 sprites have to made from 8x8 data internally anyway. so with a normal tool you have to group your objects in 16xY gfx to get a good result. now here comes metatiling into play: it first tiles your gfx into X*X and then in Y*Y. so -T16 -t8 will split up the gfx into 16x16 blocks first and then this blocks into 8x8 blocks ... i hope you get the idea ... :-)

Q: who's responsible for the compressing stuff?
A: thanks to Tony McB. for permisson to use his packer source!

Q: what else is changed/added/fixed in v0.9?
A: added check for incorrect BMP headers (some tools don't set the "used colors" entry) and use the depth as default if header info is missing.
also fixed rot/scale BG generation (thanks to Alex D. for reporting)
added some headers in asm/c outputfiles (that's for you Alex! :))
added size info for linker to asm outputfiles

Q: who brought up the idea to add options to disable data output?
A: Erik G. asked for it and now he got it ... :p

Q: not many *NEW*'s in v0.10, what has changed?
A: added function to convert artist-filenames to coder-filenams
(ie replacing illegal characters in label names by _'s)
fixed filename generation and handling if sourcefiles are in
a sub-directory
message if saving was successful or not
(thanks to Dennis R. for all this ideas/reports)
Andreas T. told me that -p together with -x was not working so
i fixed it :)
also for Andreas T. are the -mm and -MM options
thanks to Peter H. (PED) you can now also enjoy the -q option
made -C work with -x (thanks to Thomas H. for this one)
added -a option to set the transparent color (also for Thomas H.)

Q: any updates in v0.11 ?
A: added -B (only blank tile optimize) for light1
added -A (for Thomas H.)
added -ap and -aps (aPlib compression) (for Dennis R.)
added -farm
added TGA 256 color (packed/unpacked) loader
fixed color index 0 color in master.pal (for Tyler Gregg)
fixed GCC ASM .align 4 to .align 2
fixed "***ERROR: need 256 free colors, only 255 available!" problem
included a static linked linux version (for Joeri)

Q: no new features in v0.12???
A: no, just 2 important bug fixes:
- fixed -mm and -MM for metatiled maps (thanks to Phantasm for this one)
- fixed RAW palette output

Q: what's really new in v0.13?
A: added ProMotion .spr loader
added Playstation(tm) .tim loader
added more precise error texts to image loader
added -mc option to convert row based maps to column based maps (Luc B.)
added -tc option to convert tiles/bitmaps to columns (Luc B.)
added possibility to have a ' ' (space) between argument and parameter (Krystian W.)
added -align option to add __attribute__ ((aligned (4))) to C/C++ arrays (Thomas H.)
added extern to C/C++ arrays (Thomas H.)

added try{} catch(){} around new to prevent termination without error message

=======
Contact
=======

any questions, reports, suggestions are welcome ...
contact me at: markus@console-dev.de

==========
gfx2gba v0.10 README
==========

this tool converts 8 bit (256 color) or 4 bit (16 color) BMP or 8 bit (256 color) PCX graphic files to GBA useable data (raw, tiles, map). the special thing about this tool is that it is able to combine the palettes of several gfx files into one single "master palette". it removes/remaps double colors. of course if you try to combine 2 files with 256 unique colors each it will fail, heh ... :p

Usage: gfx2gba [options] bmp/pcx files ...

Options are:

-pPalettename -m generate map (optimized)
-oOutputdir -M generate map (not optimized)
-fOutput format -mm generate metatiled map (optimized) *new*
-sSection -MM generate metatiled map (not optimized) *new*
-SpSuffix for palette *new* -bBank for map
-SmSuffix for map *new* -F turn of check for flipped tiles
-StSuffix for tiledata *new* -rs output a rotate/scale BG screen map
-TMetatilesize -P don't save palette data
-tTilesize -G don't save tile/image data
-cColordepth -D don't save map data
-CColor-Offset -X don't save anything
-aTransparent color *new* -Z compress everything
-vVRAM-Offset -zt compress tile data
-x don't merge palettes -zp compress palette data
-q quiet mode *new* -zm compress map data

-q quiet mode reduces output to a minimum

-p specifies the name of the master palette (default: master.pal)

-o specifies the output directory (default is current dir)

-f specifies the output format (raw/src/asm) (default is raw)

-s will allow you to select the ROM section used in the assembler outputfiles (default is .rodata)

-Sp specifices a custom extension for palette files (ie replace ".pal" by something else)

-Sm like above but to customize/replace ".map"

-St like above but to customize/replace ".raw"

NOTE: if C or ASM output is selected, .c or .s will ALWAYS be added to the filename, regardless of what custom extension you selected!

-T specifies the metatilesize (1/8/16/32/64) (default is 1, which means NO metatiling at all)

-t specifies the tilesize (1/8/16/32/64) (default is 1, which means NO tiling at all, giving you plain raw data for mode4/5 bg's)

-c specifies input/output color depth (16/256/32k) (default is 256)
NOTE: in 256 color mode you can also READ 16 color images, but they'll be saved as 256 color images!
NOTE2: 32k setting only works for OUTPUT of raw 15+1bit gfx data for use in MODE3/5, you can NOT READ other formats than 16/256 color BMP or 256 color PCX. this tool was not designed for this mode anyway, so expect some bugs there (especially if you mix it with other options)

-C will add an offset (0-255) to the bitmap color indexes (default 0) it will NOT add the offset to color index #0 (transparent) !!!

-a changes the transparent color (0-255). (default 0)

-v specifies the offset to be added when saving tilemaps (default is 0)

-b specifies the color bank to use for 16 color tile maps (0-15) (default is 0)

NOTE: DO NOT USE SPACES between -p/-o/-s/-Sp/-Sm/-St-t/-c/-v/-C/-b and it's parameter!!! use -t16 or something like that, but -t 16 won't work!!!

-m will generate an optimized tile map from the inputfile. optimized means it removes double tiles and also checks for horizontal, vertical and h+v flipped doubles. of course it sets the correct flags in the tile map data. the output .raw file will ONLY contain the used tiles (of course).

-M will generate a NON optimized tilemap

-mm will tile a generated map into 32x32 tile sub-maps for easy DMA'ing them around

-MM same as -mm just for non optimized maps

-F will turn off check for flipped tiles in map. only "real" doubles will be removed.

NOTE: activating map generation will change default tile size from 1 to a more GBA friendly size of 8. you can still set your own size using the -t option.
fixed a stupid bug that produced an endless loop in v0.6 if your input gfx bitmap was >= 4mbyte.

-rs will force the map to be saved in 1 byte per tile format making it suitable for rotate/scale backgrounds. you lose color bank info and h/v flip info obviously and the max number of tiles is 256. but hey, you knew that already, don't ya?
NOTE: this one is fixed now, had a silly bug that was not catched by -Wall ... :/

-P will disable palette output at all

-G will disable graphics (tiles/images) output at all

-D will disable map output at all

-X will disable all data output

-x will prevent gfx2gba from merging/optimizing your palettes and save a separate .pal file for each input gfx file.

-zt will lz77 compress tile/gfx data

-zp will lz77 compress palette data

-zm will lz77 compress map data

-Z will lz77 compress everything

NOTE: all compression stuff is compatible to the GBA internal Lz77 depacker.
if it fails to compress it will show you a warning and save the normal data.

==========
Frequently Asked Questions

==========

Q: so far so good, but what if i want to convert all my 36 .bmp files and 12 .pcx files ?!
A: easy: gfx2gba *.pcx *.bmp
could it be any simpler?! :)

Q: what -t value do i need for correct sprites and background tiles?
A: use -t8 to get what you need ...

Q: but i am using Linux for GBA dev?!
A: just use the included Linux binary (you may have to set exec flag)

Q: wtf is -v good for? why would i want to add an offset?!
A: HINT: to maximize the number of useable tiles.

Q: is it possible that your tool is the only free available one that also supports 16 color / 4 bit gfx?!
A: erm... well... seems so... looks like other tool coders ignore the presence of the 16 color modes on GBA ...

Q: v0.4 features are the same as in v0.3?! what's new or changed?!
A: *cough* this release has 2 major bugs fixed. bitmap export to 4bit (16 color) was broken (pixel swapped) (export to tiles was working, that's why i found this nasty bug so late) and my color optimizing stuff made ALL black (0x0000) colors transparent (thanks to Michael Jagger for yelling at me about this)

Q: soo, what's new in v0.5 ? i still can't see any new options?!
A: *cough*cough* the black color bug was nastier than i thought but at the end they all DIE ... or short: the last bug in color optimizing is removed now ... fixed bitmap remapping (last pixel wasn't remapped) and finally it compiles now without even a warning! :-)

Q: hey, i can mix all kind of options and gfx2gba doesn't complain about stuff that doesn't make sense?!
A: it is a tool to help you developing GBA stuff, not a brain replacement and it assumes you know at least a little bit what you are doing.

Q: I have a problem with 32k mode...
A: I don't care ... 32k mode is just a little add-on and not something high-priority for me ... :)

Q: hey, who's responsible for the color bank & rot/scale option? i don't think you're smart enough to think about 'em yourself!?
A: sad but true ... thanks to Alex D. for this obviously useful ideas. cheers mate, keep the ideas coming (even if they arrive in my inbox at 00:30 in the morning)!

Q: are there any other people who are giving you their valuable ideas free of charge?

A: yeah! many thanks to Dennis R. for the idea to change some of the default values and kicking my butt to implement C/C++ array output at least. and then we have Bengt Johannesson who found out that the symbol name creation was f*cked when you used files not placed in the current directory (it had the complete path in the label name instead of just the filename) ...

Q: who wanted the l33t flipped tile check to be removable!?
A: Nikita Mikros needed it so i added it ... :)

Q: metatiling? wow! cool name for a feature, BUT WHAT THE HELL IS IT?!
A: it more or less means that your input gfx will get tiled twice. for example my game "engine" is building large objects from 16x16 sprites. but 16x16 sprites have to made from 8x8 data internally anyway. so with a normal tool you have to group your objects in 16xY gfx to get a good result. now here comes metatiling into play: it first tiles your gfx into X*X and then in Y*Y. so -T16 -t8 will split up the gfx into 16x16 blocks first and then this blocks into 8x8 blocks ...
i hope you get the idea ... :-)

Q: who's responsible for the compressing stuff?
A: thanks to Tony McB. for permisson to use his packer source!

Q: what else is changed/added/fixed in v0.9?
A: added check for incorrect BMP headers (some tools don't set the "used colors" entry) and use the depth as default if header info is missing.
also fixed rot/scale BG generation (thanks to Alex D. for reporting)
added some headers in asm/c outputfiles (that's for you Alex! :))
added size info for linker to asm outputfiles

Q: who brought up the idea to add options to disable data output?
A: Erik G. asked for it and now he got it ... :p

Q: not many *NEW*'s in v0.10, what has changed?
A: added function to convert artist-filenames to coder-filenams (ie replacing illegal characters in label names by _'s)
fixed filename generation and handling if sourcefiles are in a sub-directory
message if saving was successful or not
(thanks to Dennis R. for all this ideas/reports)
Andreas T. told me that -p together with -x was not working so i fixed it :)
also for Andreas T. are the -mm and -MM options
thanks to Peter H. (PED) you can now also enjoy the -q option
made -C work with -x (thanks to Thomas H. for this one)
added -a option to set the transparent color (also for Thomas H.)

=======
Contact
=======

any questions, reports, suggestions are welcome ...
contact me at: markus@console-dev.de

==========
gfx2gba v0.8 README
==========

this tool converts 8 bit (256 color) or 4 bit (16 color) BMP or 8 bit (256 color) PCX graphic files to GBA useable data (raw, tiles, map). the special thing about this tool is that it is able to combine the palettes of several gfx files into one single "master palette". it removes/remaps double colors. of course if you try to combine 2 files with 256 unique colors each it will fail, heh ... :p

Usage: gfx2gba [options] bmp/pcx files ...

Options are: -pPalettename
-oOutputdir
-fOutput format
-TMetatilesize
-tTilesize
-cColordepth
-CColor-Offset
-vVRAM-Offset
-bBank for map
-m generate map (optimized)
-M generate map (not optimized)
-F turn of check for flipped tiles
-rs output a rotate/scale BG screen map
-P don't save palette data
-x don't merge palettes
-zt compress tile data *NEW*
-zp compress palette data *NEW*
-zm compress map data *NEW*
-Z compress everything *NEW*

-p specifies the name of the master palette (default: master.pal)

-o specifies the output directory (default is current dir)

-f specifies the output format (raw/src/asm) (default is raw)

-T specifies the metatilesize (1/8/16/32/64) (default is 1, which means NO metatiling at all)

-t specifies the tilesize (1/8/16/32/64) (default is 1, which means NO tiling at all, giving you plain raw data for mode4/5 bg's)

-c specifies input/output color depth (16/256/32k) (default is 256)
NOTE: in 256 color mode you can also READ 16 color images, but they'll be saved as 256 color images!
NOTE2: 32k setting only works for OUTPUT of raw 15+1bit gfx data for use in MODE3/5, you can NOT READ other formats than 16/256 color BMP or 256 color PCX. this tool was not designed for this mode anyway, so expect some bugs there (especially if you mix it with other options)

-C will add an offset (0-255) to the bitmap color indexes (default 0) it will NOT add the offset to color index #0 (transparent) !!!

-v specifies the offset to be added when saving tilemaps (default is 0)

-b specifies the color bank to use for 16 color tile maps (0-15) (default is 0)

NOTE: DO NOT USE SPACES between -p/-o/-t/-c/-v/-C/-b and it's parameter!!!
use -t16 or something like that, but -t 16 won't work!!!

-m will generate an optimized tile map from the inputfile. optimized means it removes double tiles and also checks for horizontal, vertical and h+v flipped doubles. of course it sets the correct flags in the tile map data. the output .raw file will ONLY contain the used tiles (of course).

-M will generate a NON optimized tilemap

-F will turn off check for flipped tiles in map. only "real" doubles will be removed.

NOTE: activating map generation will change default tile size from 1 to a more GBA friendly size of 8. you can still set your own size using the -t option.
fixed a stupid bug that produced an endless loop in v0.6 if your input gfx bitmap was >= 4mbyte.

-rs will force the map to be saved in 1 byte per tile format making it suitable for rotate/scale backgrounds. you lose color bank info and h/v flip info obviously and the max number of tiles is 256. but hey, you knew that already, don't ya?

-P will disable palette output at all

-x will prevent gfx2gba from merging/optimizing your palettes and save a separate .pal file for each input gfx file.

-zt will lz77 compress tile/gfx data

-zp will lz77 compress palette data

-zm will lz77 compress map data

-Z will lz77 compress everything

NOTE: all compression stuff is compatible to the GBA internal Lz77 depacker.
if it fails to compress it will show you a warning and save the normal data.

==========
Frequently Asked Questions
==========

Q: so far so good, but what if i want to convert all my 36 .bmp files and 12 .pcx files ?!
A: easy: gfx2gba *.pcx *.bmp
could it be any simpler?! :)

Q: what -t value do i need for correct sprites and background tiles?
A: use -t8 to get what you need ...

Q: but i am using Linux for GBA dev?!
A: just use the included Linux binary (you may have to set exec flag)

Q: wtf is -v good for? why would i want to add an offset?!
A: HINT: to maximize the number of useable tiles.

Q: is it possible that your tool is the only free available one that also supports 16 color / 4 bit gfx?!
A: erm... well... seems so... looks like other tool coders ignore the presence of the 16 color modes on GBA ...

Q: v0.4 features are the same as in v0.3?! what's new or changed?!
A: *cough* this release has 2 major bugs fixed. bitmap export to 4bit (16 color) was broken (pixel swapped) (export to tiles was working, that's why i found this nasty bug so late) and my color optimizing stuff made ALL black (0x0000) colors transparent (thanks to Michael Jagger for yelling at me about this)

Q: soo, what's new in v0.5 ? i still can't see any new options?!
A: *cough*cough* the black color bug was nastier than i thought but at the end they all DIE ... or short: the last bug in color optimizing is removed now ... fixed bitmap remapping (last pixel wasn't remapped) and finally it compiles now without even a warning! :-)

Q: hey, i can mix all kind of options and gfx2gba doesn't complain about stuff that doesn't make sense?!
A: it is a tool to help you developing GBA stuff, not a brain replacement

and it assumes you know at least a little bit what you are doing.

Q: I have a problem with 32k mode...
A: I don't care ... 32k mode is just a little add-on and not something high-priority for me ... :)

Q: hey, who's responsible for the color bank & rot/scale option? i don't think you're smart enough to think about 'em yourself!?
A: sad but true ... thanks to Alex D. for this obviously useful ideas. cheers mate, keep the ideas coming (even if they arrive in my inbox at 00:30 in the morning)!

Q: are there any other people who are giving you their valuable ideas free of charge?
A: yeah! many thanks to Dennis R. for the idea to change some of the default values and kicking my butt to implement C/C++ array output at least. and then we have Bengt Johannesson who found out that the symbol name creation was f*cked when you used files not placed in the current directory (it had the complete path in the label name instead of just the filename) ...

Q: who wanted the l33t flipped tile check to be removable!?
A: Nikita Mikros needed it so i added it ... :)

Q: metatiling? wow! cool name for a feature, BUT WHAT THE HELL IS IT?!
A: it more or less means that your input gfx will get tiled twice. for example my game "engine" is building large objects from 16x16 sprites. but 16x16 sprites have to made from 8x8 data internally anyway. so with a normal tool you have to group your objects in 16xY gfx to get a good result. now here comes metatiling into play: it first tiles your gfx into X*X and then in Y*Y. so -T16 -t8 will split up the gfx into 16x16 blocks first and then this blocks into 8x8 blocks ...
i hope you get the idea ... :-)

Q: who's responsible for the compressing stuff?
A: thanks to Tony McB. for permisson to use his packer source!

=======
Contact
=======

any questions, reports, suggestions are welcome ...
contact me at: markus@console-dev.de

enjoy another cool tool from your friends at MUPS!!!

<< HOME >>

HAM Tutorial :: Translations

Hopefully you can find a good translation on this page.

NOTE: Any code examples will not work with translated words.
You will have to use the English code.

Chinese: AltaVista

French: AltaVista :: Google

German: AltaVista :: Google

Italian: AltaVista :: Google

Japanese: AltaVista

Korean: AltaVista

Portuguese: AltaVista :: Google

Spanish: AltaVista :: Google

<< HOME >>

HAM Tutorial :: Downloads

HAM Files

Latest Version

* **HAM 2.60 update from 2.52** *(23rd Dec 2002)* (1 MB) Register Now! (Windows Version with installer, works for Windows 98/ME/2000/XP)

* **HAM v2.52 update from 2.50** *(18th Nov 2002)* (8 MB) Register Now! (Windows Version with installer, works for Windows 98/ME/2000/XP)

* **HAM v2.50 freeware full installer** *(09th Sep 2002)* (40 MB) Register Now! (Windows Version with installer, works for Windows 98/ME/2000/XP)

* **HAM v2.50 freeware compiler builder edition** *(09th Sep 2002)* (2 MB) Register Now! (Cygwin Version, works for Windows 98/ME/2000/XP system running CygWin, will download ~50MB compiler sources)

* **HAM v2.50 freeware compiler builder edition** *(09th Sep 2002)* (2 MB) Register Now! (Linux Version , tested on SuSE Linux 7.3 / 8.0, will download ~50MB compiler sources)

Previous Versions

* **HAM v2.40 freeware full installer** *(12th Jun 2002)* (25 MB) Register Now!

Documentation

HAM Tutorial 1.4.2 :: Entire Tutorial Docs:: .pdf (1484 KB), .zip (458 KB)
HAM Tutorial 1.6.0 :: Entire Tutorial Source :: .zip (755 KB)

HAM Tutorial :: Day 1 Docs 1.7.7 :: .pdf (21 KB), .zip (09 KB)
HAM Tutorial :: Day 2 Docs 1.7.7 :: .pdf (25 KB), .zip (10 KB)
HAM Tutorial :: Day 3 Docs 1.7.7 :: .pdf (25 KB), .zip (11 KB)
HAM Tutorial :: Day 4 Docs 1.7.7 :: .pdf (26 KB), .zip (11 KB)
HAM Tutorial :: Day 5 Docs 1.7.7 :: .pdf (27 KB), .zip (12 KB)
HAM Tutorial :: Day 6 Docs 1.7.7 :: .pdf (30 KB), .zip (13 KB)
HAM Tutorial :: Day 7 Docs 1.7.7 :: .pdf (34 KB), .zip (16 KB)
HAM Tutorial :: Day 8 Docs 1.7.7 :: .pdf (36 KB), .zip (16 KB)
HAM Tutorial :: Day 9 Docs 1.7.7 :: .pdf (31 KB), .zip (14 KB)
HAM Tutorial :: Day 10 Docs 1.7.7 :: .pdf (32 KB), .zip (16 KB)

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

HAM Tutorial :: Day 11 Docs 1.7.7 :: .pdf (30 KB), .zip (14 KB)

HAM Tutorial :: Proj 1 Docs 1.4.2 :: .pdf (KB), .zip (KB)
HAM Tutorial :: Proj 2 Docs 1.4.2 :: .pdf (KB), .zip (KB)
HAM Tutorial :: Proj 3 Docs 1.4.2 :: .pdf (KB), .zip (KB)

< < HOME > >

HAM Tutorial :: Credits

The only reason for the order of sites listed here is our alphabet.

Boycott Advance Online - I used the Java version of Boycott Advance for my Online Demonstrations.

Cowbite Virtual Hardware Specifications - I don't know how many times I've checked this page and referred others to it regarding GBA hardware.

Devrs.com/gba, GBADev.org & GBAEmu.com - A few of my daily stops on the information superhighway.

Ngine.de - Duh! The home of HAM, of course.

The Pern Project - They were one of the first sites I went to when I started learning about GBA development. I decided to start the HAM Tutorial because of their site.

Sympoll - I am using my own modified of this for my polls. (Coming Soon!)

VisualBoy Advance - My emulator of choice. I use it more than any other emu for testing my demos, games, etc.

Yahoo! Groups : hamdev - The place to go for discussing HAM development.

<< HOME >>

HAM Tutorial :: Links

Development & Tutorials

AI

Implementing Artificial Intelligence (by Jonathan Nix)
Using Genetic Algorithms for Game AI (by Greg James)
Beginners Guide to Pathfinding Algorithms

ARM, Thumb, Assembly

Re-eject References

C/C++

Advanced String Techniques in C++ Part I :: Part II
Bjarne Stroustrup's C++ Glossary
C++ Coding Style (Albert Sandberg)
C++ Exception Handling (by Denton Woods)
C++ Exception Handling (by Illco of Mayday Software Productions)
C++ Tutorial (by Forest J. Hanford)
CPP-Home.com
Introduction to C++: Lesson 1 - Intro to C++ and Programming Overview
Introduction to C++: Lesson 2 - Dealing With Data
Introduction to C++: Lesson 3 - Doing Things!
Introduction to C++: Lesson 4 - Functions
Introduction to C++: Lesson 5 - Handling More Complex Data
Introduction to C++: Lesson 6 - Pointers
Introduction to C++: Lesson 7 - Grouping Data
Introduction to C++: Lesson 8 - Classes
Introduction to C++: Lesson 9 - Classes and Operator Overloading
Introduction to C++: Lesson 10 - Inheritance
Introduction to C++: Lesson 11 - Virtual Functions, Runtime Polymorphism
Introduction to C++: Lesson 12 - Advanced I/O
Reducing Dependencies In C++
Using Inheritance And Virtual Functions

Design / Development

21st Century Game Design Primer (by Joe Hitchens)
Character-Based Game Design (by François Dominic Laramée)
Design Document Skeleton (by William Anderson)
Designing for the Mainstream (by François Dominic Laramée)
Game Design: The Essence of Computer Games
Grid Based Game Design (by William Anderson)
How to Get Your Design Ideas into Production (by William Anderson)
Making A Game - The Idea (Part One)
Making A Game - The Design (Part Two)
Making A Game - The Document (Part Three)
Making A Game - The Development (Part Four)

Discussions

Graphics FAQ

GFX2GBA Readme

Translations

Downloads

Games

Projects

Credits

Links

Support

The Art of Computer Game Design (by Chris Crawford)
The Journal of Computer Game Design Volume 1 (1987 - 1988)
The Journal of Computer Game Design Volume 2 (1988 - 1989)
The Journal of Computer Game Design Volume 3 (1989 - 1990)
The Journal of Computer Game Design Volume 4 (1990 - 1991)
The Journal of Computer Game Design Volume 5 (1991 - 1992)
The Journal of Computer Game Design Volume 6 (1992 - 1993)
Interactive Entertainment Design Volume 7 (1993 - 1994)
Interactive Entertainment Design Volume 8 (1994 - 1995)
Interactive Entertainment Design Volume 9 (1995 - 1996)
Lilan (1996 - 1997)
Pedersen's Principles on Game Design and Production
The Developer's Life Archive (by François Dominic Laramée)
The Game Designer's Tool Kit (by François Dominic Laramée)
The Making of a Hero (by William Anderson)
The Making of a Monster (by William Anderson)

Development Environments
Catapult
Devkit Advance
HAM

Fonts
Bitmap Font Builder (convert TTFs to bitmaps for your game)

GBA
GBA Development From The Ground Up, Volume 1 :: Volume 2 :: Volume 3
GBAJunkie.co.uk
Palletized JPEG for Game Boy Advance
Resource Management
The Pern Project
Yahoo! Groups : hamdev

Graphics
Background Tiles
Converting Art to Sprites
Graphic Design for Non-Designers Lesson 1 - Who, Me? An Artist?
Graphic Design for Non-Designers Lesson 2 - Taking the Next Step
Graphic Design for Non-Designers Lesson 3 - Show It Again, Sam
Graphic Design for Non-Designers Lesson 4 - The Ink Is Black, the Page...
Graphic Design for Non-Designers Lesson 5 - Simple Elegance
Graphic Design for Non-Designers Lesson 6 - Hard Copy: The Final ...

Integrated Development Environments (IDEs)
VisualHAM

Math / Physics
3D Geometry Primer (by Bram de Grav)

3D Matrix Math Demystified

Optimization
Isometric Game Engine Optimization

Other
10 Minutes To Document Your Code
Being A Better Programmer
Collision Detection
Collision Detection, Part 1 :: Part 2 (by Greg Magarshak)
Generating Names Phonetically
MVB2 FAQ
The Standalone Programmer: A Question Of Quality (Matt Gullett)
The Elements of Game Programming

Photoshop
Creating A Texture
Creating A Tileable Texture

Documentation

HAM Documentation

Emulators

VisualBoy Advance
Boycott Advance
Boycott Advance Online (Java version)

Hardware

Cowbite Virtual Hardware Specifications

News/Information Sites

Dev'rs GBA Dev FAQs
GameTutorials.com
Mappy VM SDK 0.1
The Audio Advance

< < HOME > >

HAM Tutorial :: Support

Some of you may be wondering how you can help out the site. Well, you can help out financially or by donating source code.

Source Code

If you want to work on a tutorial covering a topic that I haven't gotten to yet, then follow these steps.
- write code that is easy to understand
- try to make your style match what I have done
- make sure your code is well documented (& include a Readme.txt)
- test your code on various emulators
- if possible, test your code on hardware
- zip up all files necessary to compile the code and email it to me

I will go through the code and, if necessary, modify it to fit in with the rest of the tutorials. When I upload it to the website, I will give you credit.

Financial Support

I may open a PayPal account someday so that you can donate money directly to me. Until I do that, feel free to click on banners and if you live in England, order something from Gamer.co.uk.

Either way, thank you very much for any and all support!

<< HOME >>