NAME

rgbgfx — Game Boy graphics converter

SYNOPSIS

rgbgfx [−CmOuVZ] [−v [−v ...]] [−a *attrmap* | −A] [−b *base_ids*] [−c *color_spec*]
[−d *depth*] [−L *slice*] [−N *nb_tiles*] [−n *nb_pals*] [−o *out_file*]
[−p *pal_file* | −P] [−q *pal_map* | −Q] [−r *stride*] [−s *nb_colors*]
[−t *tilemap* | −T] [−x *quantity*] *file*

DESCRIPTION

The **rgbgfx** program converts PNG images into data suitable for display on the Game Boy and Game Boy Color, or vice-versa.

The main function of **rgbgfx** is to divide the input PNG into 8×8 pixel *squares*, convert each of those squares into 1bpp or 2bpp tile data, and save all of the tile data in a file. It also has options to generate a tile map, attribute map, and/or palette set as well; more on that and how the conversion process can be tweaked below.

ARGUMENTS

Note that options can be abbreviated as long as the abbreviation is unambiguous: `--verb` is `--verbose`, but `--ver` is invalid because it could also be `--version`.

rgbgfx accepts decimal, binary, and hexadecimal numbers in option arguments. Decimal numbers are written as usual; binary numbers must be prefixed with either '`%`' or '`0b`', and hexadecimal numbers must be prefixed with either '`$`' (which will likely need escaping or quoting to avoid being interpreted by the shell), or '`0x`'. Leading zeros (after the base prefix, if any) are accepted, and letters are not case-sensitive. All of these are equivalent: '`42`', `042`, `0b00101010`, `0B101010`, `0x2A`, `0X2A`, `0x2a`.

Unless otherwise noted, passing '`-`' (a single dash) as a file name makes **rgbgfx** use standard input (for input files) or standard output (for output files). To suppress this behavior, and open a file in the current directory actually called '`-`', pass `./-` instead. Using standard input or output more than once in a single command will likely produce unexpected results.

The following options are accepted:

`-a` *attrmap*, `--attr-map` *attrmap*
: Generate an attribute map, which is a file containing tile "attributes". For each square of the input image, its corresponding attribute map byte contains the mirroring bits (if `-m` was specified), the bank bit (see `-N`), and the palette index. See *Pan Docs*: https://gbdev.io/pandocs/Tile_Maps#bg-map-attributes-cgb-mode-only for the individual bytes' format. The output is written just like the tile map (see `-t`), follows the same order (`-Z`), and has the same size.

`-A`, `--auto-attr-map`
: Same as `-a` *base_path*.attrmap (see "Automatic output paths").

`-b` *base_ids*, `--base-tiles` *base_ids*
: Set the base IDs for tile map output. *base_ids* should be one or two numbers between 0 and 255, separated by a comma; they are for bank 0 and bank 1 respectively. Both default to 0.

`-C`, `--color-curve`
: When generating palettes, use a color curve mimicking the Game Boy Color's screen. The resulting colors may look closer to the input image's **on hardware and accurate emulators**.

`-c` *color_spec*, `--colors` *color_spec*
: Use the specified color palettes instead of having **rgbgfx** automatically determine some. *color_spec* can be one of the following:

  inline palette spec
  : If *color_spec* begins with a hash character '`#`', it is treated as an inline palette specification. It should contain a comma-separated list of hexadecimal colors, each beginning with a hash. Colors in are accepted either as `#rgb` or `#rrggbb` format. Palettes must be separated by a colon or semicolon (the latter may require quoting to avoid special

handling by the shell), and spaces are allowed around colons, semicolons and commas; trailing commas and semicolons are allowed. See "EXAMPLES" for an example of an inline palette specification.

embedded palette spec
: If *`color_spec`* is the case-insensitive word `embedded`, then the first four colors of the input PNG's embedded palette are used. It is an error if the PNG is not indexed, or if colors other than these 4 are used. (This is different from the default behavior of indexed PNGs, as then unused entries in the embedded palette are ignored, whereas they are not with `-c embedded`).

external palette spec
: Otherwise, *`color_spec`* is assumed to be an external palette specification. The expected format is `format:path`, where *`path`* is a path to a file (`-` is not treated specially), which will be processed according to the *`format`*. See "PALETTE SPECIFICATION FORMATS" for a list of formats and their descriptions.

`-d` *`depth`*, `--depth` *`depth`*
: Set the bit depth of the output tile data, in bits per pixel (bpp), either 1 or 2 (the default). This changes how tile data is output, and the maximum number of colors per palette (2 and 4 respectively).

`-L` *`slice`*, `--slice` *`slice`*
: Only process a given rectangle of the image. This is useful for example if the input image is a sheet of some sort, and you want to convert each cel individually. The default is to process the whole image as-is.

  `slice` must be two number pairs, separated by a colon. The numbers must be separated by commas; space is allowed around all punctuation. The first number pair specifies the X and Y coordinates of the top-left pixel that will be processed (anything above it or to its left will be ignored). The second number pair specifies how many tiles to process horizontally and vertically, respectively.

  `-L` **is ignored in reverse mode**, no padding is inserted.

`-m`, `--mirror-tiles`
: Deduplicate tiles that are symmetrical mirror images of each other. Only one of each unique tile will be saved in the tile data file, with mirror images counting as duplicates. Tiles are checked for horizontal, vertical, and horizontal-vertical mirroring. Useful with a tile map and attribute map together (see `-a` and `-t`) to keep track of the duplicated tiles and the dimension(s) mirrored. Implies `-u`.

`-N` *`nb_tiles`*, `--nb-tiles` *`nb_tiles`*
: Set a maximum number of tiles that can be placed in each VRAM bank. *`nb_tiles`* should be one or two numbers between 0 and 256, separated by a comma; if the latter is omitted, it defaults to 0. Setting either number to 0 prevents any tiles from being output in that bank.

  If more tiles are generated than can fit in the two banks combined, **`rgbgfx`** will abort. If `-N` is not specified, no limit will be set on the amount of tiles placed in bank 0, and tiles will not be placed in bank 1.

`-n` *`nb_pals`*, `--nb-palettes` *`nb_pals`*
: Abort if more than *`nb_pals`* palettes are generated. This may not be more than 256.

  Note that attribute map output only has 3 bits for the palette ID, so a limit higher than 8 may yield incomplete data unless relying on a palette map (see `-q`).

`-O`, `--group-outputs`
: Sets the 'base path' to be the output tile data path from `-o` instead of the input image path (see "Automatic output paths").

`-o` *`out_file`*, `--output` *`out_file`*
: Output the tile data in native 2bpp format or in 1bpp (depending on `-d`) to this file.

`-p` *`pal_file`*, `--palette` *`pal_file`*
: Output the image's palette set to this file.

`-P`, `--auto-palette`
: Same as `-p` *`base_path`*.pal (see "Automatic output paths").

`-q` *`pal_file`*, `--palette-map` *`pal_file`*
: Output the image's palette map to this file. This is useful if the input image contains more than 8 palettes, as the attribute map only contains the lower 3 bits of the palette indices.

`-Q`, `--auto-palette-map`
: Same as `-q` *`base_path`*.palmap (see "Automatic output paths").

`-r` *`width`*, `--reverse` *`width`*
: Switches **`rgbgfx`** into "**reverse**" mode. In this mode, instead of converting a PNG image into Game Boy data, **`rgbgfx`** will attempt to reverse the process, and render Game Boy data into an image. See "REVERSE MODE" below for details.

  `width` is the width of the image to generate, in tiles.

`-s` *`nb_colors`*, `--palette-size` *`nb_colors`*
: Specify how many colors each palette contains, including the transparent one if any. *`nb_colors`* cannot be more than `1 <<` *`depth`* (see `-d`).

`-t` *`tilemap`*, `--tilemap` *`tilemap`*
: Generate a file of tile indices. For each square of the input image, its corresponding tile map byte contains the index of the associated tile in the tile data file. The IDs wrap around from 255 back to 0, and do not include the bank bit; use `-a` for that. Useful in combination with `-u` and/or `-m` to keep track of duplicate tiles.

`-T`, `--auto-tilemap`
: Same as `-t` *`base_path`*.tilemap (see "Automatic output paths").

`-u`, `--unique-tiles`
: Deduplicate identical tiles. Only one of each unique tile will be saved in the tile data file. Useful with a tile map (see `-t`) to keep track of the duplicated tiles.

  Note that if this option is enabled, no guarantee is made on the order in which tiles are output; while it *should* be consistent across identical runs of a given **`rgbgfx`** release, the same is not true for different releases.

`-V`, `--version`
: Print the version of the program and exit.

`-v`, `--verbose`
: Be verbose. The verbosity level is increased by one each time the flag is specified, with each level including the previous:
  1. **`rgbgfx`** prints out its configuration before doing anything.
  2. A generic message is printed before doing most actions.
  3. Some of the actions' intermediate results are printed.
  4. Some internal debug printing is enabled.

  The verbosity level does not go past 6.

  Note that verbose output is only intended to be consumed by humans, and may change without notice between RGBDS releases; relying on those for scripts is not advised.

`-x` *`quantity`*, `--trim-end` *`quantity`*
: Do not output the last *`quantity`* tiles to the tile data file; no other output is affected. This is useful for trimming "filler" / blank squares at the end of an image. If fewer than *`quantity`* tiles

would have been emitted, the file will be empty.

Note that this is done *after* deduplication if `-u` was enabled, so you probably don't want to use this option in combination with `-u`. Note also that the tiles that don't get output will not count towards `-N`'s limit.

`-Z`, `--columns`
: Read squares from the PNG in column-major order (column by column), instead of the default row-major order (line by line). This primarily affects tile map and attribute map output, although it may also change generated tile data and palettes.

At-files

In a given project, many images are to be converted with different flags. The traditional way of solving this problem has been to specify the different flags for each image in the Makefile / build script; this can be inconvenient, as it centralizes all those flags away from the images they concern.

To avoid these drawbacks, **`rgbgfx`** supports "at-files": any command-line argument that begins with an at sign ('`@`') is interpreted as one. The rest of the argument (without the @, that is) is interpreted as the path to a file, whose contents are interpreted as if given on the command line. At-files can be stored right next to the corresponding image, for example:

```
$ rgbgfx -o image.2bpp -t image.tilemap @image.flags image.png
```

This will read additional flags from file `image.flags`, which could contains for example `-b 128` to specify a base offset for the image's tiles. The above command could be generated from the following *make*(1) rule, for example:

```
%.2bpp %.tilemap: %.flags %.png
	rgbgfx -o $*.2bpp -t $*.tilemap @$*.flags $*.png
```

Since the contents of at-files are interpreted by **`rgbgfx`**, **no shell processing is performed**; for example, shell variables are not expanded (`$PWD`, `%WINDIR%`, etc.). In at-files, lines that are empty or contain only whitespace are ignored; lines that begin with a hash sign ('`#`'), optionally preceded by whitespace, are considered comments and also ignored. Each line can contain any number of arguments, which are separated by whitespace. (No quoting feature to prevent this is provided.)

Note that a leading '`@`' has no special meaning on option arguments, and that the standard '`--`' to stop option processing also disables at-file processing. For example, the following command line reads command-line options from `tilesets/town.flags` then `tilesets.flags`, but processes `@tilesets/town.png` as the input image and outputs tile data to `@tilesets/town.2bpp`:

```
$ rgbgfx -o @tilesets/town.2bpp @tilesets/town.flags
@tilesets.flags -- @tilesets/town.png
```

At-files can also specify the input image directly, and call for more at-files, both using the regular syntax. Note that while '`--`' can be used in an at-file (with identical semantics), it is only effective inside of it—normal option processing continues in the parent scope.

PALETTE SPECIFICATION FORMATS

The following formats are supported:

`act` *Adobe Photoshop color table*: https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/#50577411_pgfId-1070626.

`aco` *Adobe Photoshop color swatch*: https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/#50577411_pgfId-1055819.

`gbc` A GBC palette memory dump, as emitted by **`rgbgfx`** `-p`. Useful to force several images to share the same palette.

`gpl` *GIMP palette*: https://docs.gimp.org/2.10/en/gimp-concepts-palettes.html.

`hex` Plaintext lines of hexadecimal colors in `rrggbb` format.

`psp` *Paint Shop Pro palette*: https://www.selapa.net/swatches/colors/fileformats.php#psp_pal.

If you wish for another format to be supported, please open an issue (see "BUGS" below) or contact us, and supply a few sample files.

PALETTE GENERATION

`rgbgfx` must generate palettes from the colors in the input image, unless `-c` was used; in that case, the provided palettes will be used. **If the order of colors in the palettes is important to y ou**, for example because you want to use palette swaps, please use `-c` to specify the palette e xplicitly.

First, if the image contains *any* transparent pixel, color #0 of *all* palettes will be allocated to it. This is done **even if palettes were explicitly specified using** `-c`; then the specification only co vers color #1 onwards. (If you do not want this, ask your image editor to remove the alpha channel.)

After generating palettes, **`rgbgfx`** sorts colors within those palettes using the following rules: delim $$

- If the PNG file internally contains a palette (often dubbed an "indexed" PNG), then colors in each output palette will be sorted according to their order in the PNG's palette. Any unused entries will be ignored, and only the first entry is considered if there are any duplicates. (If you want a given color to appear more than once, or an unused color to appear at all, you should specify the palettes explicitly instead using `-c`; `-c embedded` may be appropriate.)
- Otherwise, if the PNG only contains shades of gray, they will be categorized into as many "bins" as there are colors per palette, and the palette is set to these bins. The darkest gray will end up in bin #0, and so on; note that this is the opposite of the RGB method below. If two distinct grays end up in the same bin, the RGB method is used instead.

  Be careful that **`rgbgfx`** is picky about what it considers "grays": the red, green, and blue components of each color must *all* be *exactly* the same.
- If none of the above apply, colors are sorted from lightest (first) to darkest (last). The definition of luminance that **`rgbgfx`** uses is "$2126 times red + 7152 times green + 722 times blue$".

delim off

Note that the "indexed" behavior depends on an internal detail of how the PNG is saved, specifically its `PLTE` chunk. Since few image editors (such as GIMP) expose that detail, this behavior is only kept for compatibility and should be considered deprecated.

OUTPUT FILES

All files output by **`rgbgfx`** are binary files, and designed to follow the Game Boy and Game Boy Color's native formats. What follows is succinct descriptions of those formats, including **`rgbgfx`**-specific details. For more complete, beginner-friendly descriptions of the native formats with illustrations, please check out *Pan Docs*: https://gbdev.io/pandocs/Graphics.

Tile data

Tile data is output like a binary dump of VRAM, with no padding between tiles. Each tile is 16 bytes, 2 per row of 8 pixels; the bits of color IDs are split into each byte (or "bitplane"). The leftmost pixel's color ID is stored in the two bytes' most significant bits, and the rightmost pixel's color ID in their least significant bits.

When the bit depth (`-d`) is set to 1, the most significant bitplane (second byte) of each ro w, being all zeros, is simply not output.

Palette data

Palette data is output like a dump of palette memory. Each color is written as GBC-native little-endian RGB555, with the unused bit 15 set to 0. There is no padding between colors, nor between palettes; however, empty colors in the palettes are output as 0xFFFF. delim $$ For example, if 5 palettes are generated with `-s 4`, the palette data file will be $2 times 4 times 5 = 40$ bytes long, even if some palettes contain less than 3 colors. delim off Note that `-n` only caps ho w many palettes are generated (and thus this file's size), but fewer may be generated still.

Tile map data

A tile map is an array of tile IDs, with one byte per tile ID. The first byte always corresponds to the ID of the tile in top-left corner of the input image; the second byte is either the ID of the tile to its right (by default), or below it (with `-Z`); and so on, continuing in the same direction. Rows / columns (respectively) are stored consecutively, with no padding.

Attribute map data

Attribute maps mirror the format of tile maps, like on the GBC, especially the order in which bytes are output. The contents of individual bytes follows the GBC's native format:

Bit 7	BG-to-OAM Priority	Set to 0
Bit 6	Vertical Flip	0=Normal, 1=Mirror vertically
Bit 5	Horizontal Flip	0=Normal, 1=Mirror horizontally
Bit 4	Not used	Set to 0
Bit 3	Tile VRAM Bank number	0=Bank 0, 1=Bank 1
Bit 2–0	Background Palette number	BGP0-7

Note that if more than 8 palettes are used, only the lowest 3 bits of the palette ID are output.

Automatic output paths

For convenience, **rgbgfx** provides shortcuts to generate all files in the same directory. This is done by using the uppercase version of a flag (for example, `-A` instead of `-a`). The *base_path* is the input image path (or the output tile data path from `-o`, if `-O` was given) with its extension, if any, removed.

For example, these two commands are equivalent:

```
$ rgbgfx img/player.png -o build/player.2bpp -P
$ rgbgfx img/player.png -o build/player.2bpp -p img/player.pal
```

And so are these two:

```
$ rgbgfx img/player.png -o build/player.2bpp -O -P
$ rgbgfx img/player.png -o build/player.2bpp -p build/player.pal
```

REVERSE MODE

rgbgfx can produce a PNG image from valid data. This may be useful for ripping graphics, recovering lost source images, etc. An important caveat on that last one, though: the conversion process is **lossy** both ways, so the "reversed" image won't be perfectly identical to the original—but it should be close to a Game Boy's output. (Keep in mind that many of consoles output different colors, so there is no true reference rendering.)

When using reverse mode, make sure to pass the same flags that were given when generating the data, especially `-C`, `-d`, `-N`, `-s`, `-x`, and `-Z`. ""At-files" may help with this". **rgbgfx** will warn about any inconsistencies it detects.

Files that are normally outputs (`-a`, `-p`, `-t`) become inputs, and *file* will be written to instead of read from, and thus needs not exist beforehand. Any of these inputs not passed is assumed to be some default:

palettes	Unspecified palette data makes **rgbgfx** assume DMG (monochrome Game Boy) mode: a single palette of 4 grays. It is possible to pass palettes using `-c` instead of `-p`.
tile data	Tile data must be provided, as there is no reasonable assumption to fall back on.
tile map	A missing tile map makes **rgbgfx** assume that tiles were not deduplicated, and should be laid out in the order they are stored.
attribute map	Without an attribute map, **rgbgfx** assumes that no tiles were mirrored.

NOTES

Some flags have had their functionality removed. `-D`, `-f`, and `-F` are now ignored, and `-h` is an alias for the new (and less confusingly named) `-Z`. These will be removed and/or repurposed in future versions of **rgbgfx**, so relying on them is not recommended. The same applies to the corresponding long options.

If you are curious, you may find out that palette generation is an NP-complete problem, so **rgbgfx** does not attempt to find the optimal solution, but instead to find a good one in a reasonable amount of time. It is

possible to compute the optimal solution externally (using a solver, for example), and then provide it to **rgbgfx** via `-c`.

EXAMPLES

The following will only validate the `tileset.png` image (check its size, that all tiles have a suitable amount of colors, etc.), but output nothing:

```
$ rgbgfx src/res/maps/overworld/tileset.png
```

The following will convert the `tileset.png` image using the two given palettes (and only those), and store the generated 2bpp tile data in `tileset.2bpp`, and the attribute map in `tileset.attrmap`.

```
$ rgbgfx -c '#ffffff,#8d05de, #dc7905,#000000; #fff,#8d05de, #7e0000 , #000' -A -o tileset.2bpp tileset.png
```

The following will deduplicate the tiles in the `title_screen.png` image, keeping only one of each unique tile, and store the generated 2bpp tile data in `title_screen.2bpp`, and the tile map in `title_screen.tilemap`.

```
$ rgbgfx -u title_screen.png -o title_screen.2bpp -t title_screen.tilemap
```

The following will convert the given inline palette specification to a palette set, and store the palette set in `colors.pal`, without needing an input image.

```
$ rgbgfx -c '#fff,#ff0,#f80,#000' -p colors.pal
```

TODO: more examples.

BUGS

Please report bugs and mistakes in this man page on *GitHub*: https://github.com/gbdev/rgbds/issues. Bug reports and feature requests about RGBDS are also welcome!

SEE ALSO

rgbasm(1), *rgblink*(1), *rgbfix*(1), *rgbds*(7)

The Game Boy hardware reference *Pan Docs*: https://gbdev.io/pandocs/Graphics, particularly the section about graphics.

HISTORY

rgbgfx was originally written by stag019 as a program to be packaged in RGBDS. It was later rewritten by ISSOtm, and is now maintained by a number of contributors at https://github.com/gbdev/rgbds.