York Graphics Reference Manual

Oliver Dixon March 7, 2022

WARNING: do not use this library. It is poorly implemented, contains severe bugs, does not perform comprehensive error-checking, and interfaces terribly with any client programs. Use Allegro instead; complete manuals, documentation, and examples can be found within the Allegro 5 GitHub repository: `https://github.com/liballeg/allegro5`. ShockSoc are offering support sessions for Allegro at 2pm on Wednesdays in P/T/401; we are not able to help with *York Graphics*.

The in-house *York Graphics* library purports to reduce the burden on Allegro programmers by maintaining an internal set of global variables pertaining to the current instance of Allegro and its various displays. When a "`GFX_*`" function is invoked, those global state variables are automatically passed through to an Allegro "`al_*`" function. Notwithstanding the few spared keypresses, *York Graphics* suffers from many issues, to the point of becoming a greater hindrance than help. However, for students adamant to continue using its interface, this document attempts to remedy the lack of quality documentation associated with its implementation. For each function defined in `graphics_lib.h`, there corresponding function signature and short description. This is a reference document, and has been written to be easily searchable by humans.

— `GFX_InitWindow`

```
void GFX_InitWindow ( int x, int y );
```

Initialise a window with dimensions (x, y). The initial pen colour is set to white, and the background colour is set to black. This function is a wrapper for `al_create_display`, but also implicitly initialises Allegro and various add-ons with the `al_init` and `al_init_{image, primitives}_addon` functions. If any of these constructors fail, an error message is printed to `stdout` and the program is killed with a nonzero status code.

— `GFX_CloseWindow`

```
void GFX_CloseWindow ( void );
```

Deinitialise the window and Allegro instance created by `GFX_InitWindow`. This function is a wrapper for `al_destroy_display`.

— `GFX_InitFont`

```
void GFX_InitFont ( void );
```

Initialises a fixed Allegro font from `./data/fixed_font.tga`, which may or may not be included with your *York Graphics* distribution. This function is a wrapper for the Allegro `al_init_font_addon` and `al_load_font`. If the latter function fails, or the aforementioned TGA font file does not exist, multiple error lines to `stdout` will be printed, and the process will be killed with a zero (success) status code. There is no corresponding destructor routine for this function; it will always cause a memory leak.

— `GFX_PauseFor`

```
void GFX_PauseFor ( int time_in_milliseconds );
```

This function is a one-to-one wrapper for the Allegro `al_rest` routine. Passing a negative number will likely cause undefined behaviour, henceforth denoted by "UB".

— `GFX_RandNumber`
`int GFX_RandNumber ( int lower_range, int upper_range );`

This function supposedly uses the standard `rand(3)` function to generate a random number within the range [`lower_range`, `upper_range`]. However, it does not seed the random number generator, so it must be patched in order to work correctly:

1. Append "`#include <time.h>`" to the list of preprocessor directives in `graphics_lib.c`.
2. Add "`srand ( time ( NULL ) );`" to the beginning of `GFX_RandNumber`.

Once this is completed, the function should return a random number within the desired range. **This is not cryptographically secure.**

— `GFX_MakeRGB`
`COLOUR GFX_MakeRGB ( unsigned colour );`

Pack a value from the `colour` enumerable—implemented as a list of `#define` statements—into a `COLOUR` structure. See below for a comprehensive list of available colours:

BLACK	RED	DARKGRAY	LIGHTRED
BLUE	MAGENTA	LIGHTBLUE	LIGHTMAGENTA
GREEN	BROWN	LIGHTGREEN	YELLOW
CYAN	LIGHTGRAY	LIGHTCYAN	WHITE

The return value is a packed `COLOUR` struct, consisting of red, green, and blue values. If a `colour` not within the recognised list is passed, the content of the return value is undefined. Attempting to use the RGB fields thereof will always invoke UB. Similar colour-mapping behaviour can be achieved through the `al_map_rgba` function, which also provides an optional field for setting the alpha (opacity) channel.

— `GFX_SetColour`
`void GFX_SetColour ( unsigned x );`

Sets the global colour state variable, `RGB`, to a value indicated by a human-readable constant, as above. No value is returned, but all future primitives will be drawn with the given colour.

— `GFX_UpdateDisplay`
`void GFX_UpdateDisplay ( void );`

This function is a one-to-one wrapper to the Allegro-provided `al_flip_display`. Writing directly to the screen hardware buffer is extremely expensive, so a buffer is maintained in RAM and all drawing operations blit to the latter software buffer. To copy the software buffer to the screen, this routine can be used. Due to its significant cost, invocations of this function should be kept to an absolute minimum.

— `GFX_DrawFilledCircle`
`void GFX_DrawFilledCircle ( int x, int y, int radius, unsigned fillcolour );`

This function draws a circle with centre (x, y) and radius "`radius`" to the buffer. It is filled with the colour indicated with `fillcolour`. This function is a wrapper for `al_draw_filled_circle`.

— `GFX_DrawCircle`
`void GFX_DrawCircle ( int x, int y, int radius, int thickness );`

This function behaves the same as `GFX_DrawFilledCircle`, but does not fill the circle with a specified colour. The thickness of the border is denoted by "`thickness`". This function is a wrapper for `al_draw_circle`.

— GFX_DrawFilledRectangle

```
void GFX_DrawFilledRectangle ( int upper_left_x, int upper_left_y,
        int lower_right_x, int lower_right_y, unsigned fillcolour );
```

Draw a filled rectangle to the buffer with corners at the specified positions. The rectangle is then filled with a specified colour, denoted by "`fillcolour`". This function is a wrapper for `al_draw_filled_rectangle`.

— GFX_DrawRectangle

```
void GFX_DrawRectangle(int upper_left_x, int upper_left_y, int lower_right_x,
        int lower_right_y, int thickness);
```

This function behaves the same as `GFX_DrawFilledRectangle`, but does not fill the rectangle with a specified colour. The thickness of the border is denoted by "`thickness`". This function is a wrapper for `al_draw_rectangle`.

— GFX_DrawFilledTriangle

```
void GFX_DrawFilledTriangle ( int x1, int y1, int x2, int y2, int x3, int y3,
       unsigned colour );
```

This function draws a triangle to the buffer with points at (x_1, y_1), (x_2, y_2), and (x_3, y_3). The resultant triangle is then filled with the colour denoted by "`colour`". Although this function is a wrapper for `al_draw_filled_triangle`, there is no analogue for a non-filled triangle. If desired, the native Allegro `al_draw_triangle` can provide this functionality.

— GFX_DrawFilledEllipse

```
void GFX_DrawFilledEllipse ( int centre_x, int centre_y, int radius_x,
        int radius_y, unsigned fillcolour );
```

This function draws an ellipse to the screen, with centre (c_x, c_y) and radii r_x and r_y at the loci. It is filled with the colour denoted by "`fillcolour`". This function is a wrapper for the Allegro `al_draw_filled_ellipse` routine.

— GFX_DrawEllipse

```
void GFX_DrawEllipse ( int centre_x, int centre_y, int radius_x,
       int radius_y, int thickness );
```

This function behaves the same `GFX_DrawFilledEllipse`, but does not fill the ellipse with a specified colour. The thickness of the border is denoted by "`thickness`". This function is a wrapper for `al_draw_ellipse`.

— GFX_DrawArc

```
void GFX_DrawArc ( int centre_x, int centre_y, int radius, double angle_start,
        double angle_end, int thickness );
```

Draw an arc, with a border width denoted by "`thickness`", about (c_x, c_y), with point angles "`angle_start`" and "`angle_end`". Angles are specified in radians and converted to degrees internally. This function is a wrapper for the `al_draw_arc` Allegro function.

— GFX_DrawLine

```
void GFX_DrawLine ( int xstart, int ystart, int xend, int yend,
       int thickness );
```

This function draws a line, of width "`thickness`", between the points $(x_{\text{start}}, y_{\text{start}})$ and $(x_{\text{end}}, y_{\text{end}})$. This function is a wrapper for the Allegro `al_draw_line` routine.

— GFX_DrawLineTo
```
void GFX_DrawLineTo ( int x, int y, int thickness );
```

This function draws a line from the current global cursor position, updated on every primitive draw call, or `GFX_MoveTo`, to a given point (x, y). This function is an indirect wrapper for the `al_draw_line` routine.

— GFX_SetBackgroundColour
```
void GFX_SetBackgroundColour ( unsigned colour );
```

This function sets the internal global background colour state with a single colour, denoted by "`colour`", that is ultimately denoted mapped to a `COLOUR` struct with `GFX_MapRGB`. Future `GFX_ClearWindow` invocations will respect any state-changes applied through this interface.

— GFX_ClearWindow
```
void GFX_ClearWindow ( void );
```

This function is a one-to-one wrapper for the Allegro `al_clear_to_colour` function, clearing the current global display with the colour stored in the "`BACKGROUND_RGB`" state; this variable can be set through an invocation of `GFX_SetBackgroundColour`.

— GFX_MoveTo
```
void GFX_MoveTo ( int x, int y );
```

This function alters the internal global cursor state to the given point (x, y).

— GFX_CreateEventQueue
```
void GFX_CreateEventQueue ( void );
```

This function is a direct wrapper for the Allegro `al_create_event_queue` routine, and assigns the return value to the internal "`event_queue`" global. This must be invoked before *any* events are attempted to be captured, as only the resultant queue may be polled. As this function does not contain any error-handling, *York Graphics* should be rewritten to export the resultant queue global, thus allowing client programs to ensure its validity before attempting to use further event-handling routines. However, since anyone using the library will be inherently indifferent to such risks, this is not an important concern.

— GFX_RegisterDisplayEvents
```
void GFX_RegisterDisplayEvents ( void );
```

This is a wrapper for the `al_register_event_source` Allegro routine. If the wrapped function fails, an error is written to `stdout` and the process is killed with a status code indicating successful execution. On success, the event queue will become stackable with display events, pertaining to the state of any interactions performed *on* the window; this function must be successfully invoked before attempting to detect such events.

— GFX_RegisterMouseEvents
```
void GFX_RegisterMouseEvents ( void );
```

This is a wrapper for the `al_register_event_source` Allegro routine. If the wrapped function fails, an error is written to `stdout` and the process is killed with a status code indicating successful execution. On success, the event queue will become stackable with mouse events, recording any events reported via the standard OpenGL/DirectX mouse interface; this function must be successfully invoked before attempting to detect such events.

— GFX_RegisterKeyboardEvents
```
void GFX_RegisterKeyboardEvents ( void );
```

This is a wrapper for the `al_register_event_source` Allegro routine. If the wrapped function fails, an error is written to `stdout` and the process is killed with a status code indicating successful execution. On success, the event queue will become stackable with keyboard events, recording any events reported via the standard OpenGL/DirectX keyboard interface; this function must be successfully invoked before attempting to detect such events.

— GFX_DrawText
```
void GFX_DrawText ( int x, int y, const char * text );
```

This function is a wrapper for the Allegro `al_draw_text` routine. The string given by "`text`" is rendered to the left of the point specified by (x, y), using the font colour specified by the `GFX_SetColour` function.

— GFX_HideCursor
```
void GFX_HideCursor ( void );
```

This function is a one-to-one wrapper for the `al_hide_mouse_cursor` native Allegro routine. There is no function within *York Graphics* to perform the opposite operation, so the native `al_show_mouse_cursor` must be called, passing the internal global variable pointing to the active Allegro display as the only argument.

— GFX_IsEventWaiting
```
void GFX_IsEventWaiting ( void );
```

This function enables non-blocking access to the event queue. If events are pending on the internal queue, one is returned; otherwise zero. This is a thin wrapper for the native Allegro `al_is_event_queue_empty` routine.

— GFX_WaitForEvent
```
void GFX_WaitForEvent ( void );
```

This is the blocking variant of `GFX_IsEventWaiting`, such that the function will hang the calling thread until an event arrives and can be inspected. This routine is a one-to-one wrapper on the Allegro `al_wait_for_event` native function.

— GFX_IsEventCloseDisplay
```
int GFX_IsEventCloseDisplay ( void );
```

This function checks the latest loaded event; if the display should be closed, one is returned. Otherwise, zero is returned. (`ALLEGRO_EVENT_DISPLAY_CLOSE`)

— GFX_IsEventIsMouseMoved
```
int GFX_IsEventIsMouseMoved ( void );
```

This function checks the latest loaded event; if the mouse is reporting a new position, one is returned. Otherwise, zero is returned. Use the `GFX_GetMouseCoordinates` routine to retrieve the exact axes values. (`ALLEGRO_EVENT_MOUSE_AXES`)

— GFX_IsEventMouseButton
```
int GFX_IsEventMouseButton ( void );
```

This function checks the latest loaded event; if a mouse button has been pressed, one is returned. Otherwise, zero is returned. (`ALLEGRO_EVENT_MOUSE_BUTTON_DOWN`)

— GFX_IsEventKeyDown
```
int GFX_IsEventKeyDown ( void );
```

This function checks the latest loaded event; if a key was pressed and not released, one is returned. Otherwise, zero is returned. (ALLEGRO_EVENT_KEY_DOWN)

— GFX_IsEventKeyUp
```
int GFX_IsEventKeyUp ( void );
```

This function checks the latest loaded event; if a key was released, one is returned. Otherwise, zero is returned. (ALLEGRO_EVENT_KEY_UP)

— GFX_GetMouseCoordinates
```
int GFX_GetMouseCoordinates ( int * x_ptr, int * y_ptr );
```

If the mouse is reporting new coordinates, this function returns one and stores the horizontal and vertical positions in the integers referenced by "x_ptr" and "y_ptr" respectively. If no new coordinates have been reported, zero is returned and the referenced values are unchanged.

— GFX_GetMouseButton
```
int GFX_GetMouseButton ( int * button_ptr );
```

If the latest event indicates a button has been pressed, this function returns one and stores the Allegro code of the corresponding button in the value referenced by "button_ptr". If the latest event did not indicate a button press, zero is returned.

— GFX_GetKeyPress
```
int GFX_GetKeyPress ( int * keypress_ptr );
```

If the latest event indicates a keyboard button has been pressed, and potentially released, this function returns one and stores the corresponding Allegro keycode in the value referenced by "keypress_ptr". If there is no appropriate event on the queue, zero is returned.

— GFX_InitMouse
```
void GFX_InitMouse ( void );
```

This function attempts to initialise the Allegro mouse interface; this routine must be invoked before any mouse activity is expected. On failure, an error is printed to stdout, and the process is killed with a status code indicating success. This function is a thin wrapper for the al_install_mouse native Allegro routine, and should always be destroyed with GFX_CloseMouse.

— GFX_InitKeyboard
```
void GFX_InitKeyboard ( void );
```

This function attempts to initialise the Allegro keyboard interface; this routine must be invoked before any keyboard/keycode activity is expected. On failure, an error is printed to stdout, and the process is killed with a status code indicating success. This function is a thin wrapper for the al_install_keyboard native Allegro routine, and should always be destroyed with GFX_CloseKeyboard.

— GFX_CloseMouse
```
void GFX_CloseMouse ( void );
```

This function destructs the Allegro mouse interface; it is a one-to-one wrapper to the native al_uninstall_mouse routine, and should be used to clean up GFX_InitMouse invocations.

— GFX_CloseKeyboard
```
void GFX_CloseKeyboard ( void );
```

This function destructs the Allegro keyboard/keycode interface; it is a one-to-one wrapper to the native `al_uninstall_keyboard` routine, and should be used to clean up `GFX_InitKeyboard` invocations.

— GFX_InitBitmap
```
BITMAP GFX_InitBitmap ( void );
```

This function initialises and returns a `BITMAP` type for later use, which happens to be an inconvenient typedef from `ALLEGRO_BITMAP *`. (The only thing worse than typedeffing is a struct is typedeffing a pointer type to a non-pointer type!) The client programmer must maintain the return value of this routine himself; it is not stored as an internal global.

— GFX_LoadBitmap
```
BITMAP GFX_LoadBitmap ( char * name );
```

This function is a one-to-one wrapper to the native `al_load_bitmap` routine. A bitmap image from the path specified by "`name`" is loaded into an `ALLEGRO_BITMAP` structure, and its pointer is returned. On failure, `NULL` is returned.

— GFX_MakeImageBGTransparent
```
void GFX_MakeImageBGTransparent ( BITMAP image, int red, int green,
        int blue );
```

This function is a one-to-one wrapper to the native `al_convert_mask_to_alpha` Allegro routine. Given an "`image`" and colour (r, g, b), this function replaces all matching pixels with transparency. This can be occasionally useful for removing backgrounds at runtime, however it is rather expensive, and in most cases could be easily obviated by linking a pre-processed partially transparent image at compile-time. This function returns no value; any changes are written back to "`image`".

— GFX_DrawBitmap
```
void GFX_DrawBitmap ( BITMAP image, int x, int y );
```

This routine is an indirect wrapper for the native Allegro `al_draw_bitmap` function, among others. Given an "`image`" and point (x, y), this function writes the bitmap pixel data to the display buffer, with the image centred at the given point.

— GFX_FreeBitmap
```
void GFX_FreeBitmap ( BITMAP image );
```

This function destructs a bitmap; it is a one-to-one wrapper for the `al_destroy_bitmap` native Allegro routine, and should be used for any instantiated `BITMAP` objects before the program ends.

NEVER USE THIS LIBRARY.