LittlevGL Documentation

Release 6.0

Gabor Kiss-Vamosi

Jul 19, 2019

CONTENTS

English (en) - (zh-CN) Magyar (hu) - Türk (tr)

PDF version: `LittlevGL.pdf`

LittlevGL is a free and open-source graphics library providing everything you need to create embedded GUI with easy-to-use graphical elements, beautiful visual effects and low memory footprint.

Website · **GitHub** · **Forum** · **Live demo** · ***Simulator*** · **Blog**

CHAPTER

ONE

KEY FEATURES

- Powerful building blocks buttons, charts, lists, sliders, images etc
- Advanced graphics with animations, anti-aliasing, opacity, smooth scrolling
- Various input devices touchpad, mouse, keyboard, encoder etc
- Multi-language support with UTF-8 encoding
- Multi-display support, i.e. use more TFT, monochrome displays simultaneously
- Fully customizable graphical elements
- Hardware independent to use with any microcontroller or display
- Scalable to operate with little memory (64 kB Flash, 16 kB RAM)
- OS, External memory and GPU supported but not required
- Single frame buffer operation even with advanced graphical effects
- Written in C for maximal compatibility (C++ compatible)
- Simulator to start embedded GUI design on PC without embedded hardware
- Tutorials, examples, themes for rapid GUI design
- Documentation online and offline
- Free and open-source under MIT license

CHAPTER

TWO

REQUIREMENTS

- 16, 32 or 64 bit microcontroller or processor
- > 16 MHz clock speed is recommended
- Flash/ROM: > 64 kB for the very essential components (> 180 kB is recommended)
- RAM:
  - Static RAM usage: ~8..16 kB depending on the used features and objects types
  - Stack: > 2kB (> 4 kB is recommended)
  - Dynamic data (heap): > 4 KB (> 16 kB is recommended if using several objects). Set by `LV_MEM_SIZE` in *lv_conf.h*.
  - Display buffer: > *"Horizontal resolution"* pixels (> 10 × *"Horizontal resolution"* is recommended)
- C99 or newer compiler
- Basic C (or C++) knowledge: pointers, structs, callbacks.

Note that the memory usage might vary depending on the architecture, compiler and build options.

CHAPTER

THREE

FAQ

3.1 Where to get started?

- For a general overview of LittlevGL visit littlevgl.com
- To make some experiments with LittlevGL in a simulator on your PC or in even in your browser see the *Get started* guide.
- To see how you can port LittlevGL to your device go to the *Porting* section.
- To learn how LittlevGL works start to read the *Overview*.
- To read tutorials or share your own experiences go to the Blog
- To see the source doe of the library go to GitHub: https://github.com/littlevgl/lvgl/.

3.2 Where can I ask questions?

To ask questions in the Forum: https://forum.littlevgl.com/.

We use GitHub issues for development related discussion. So you should use them only if your question or issue is tightly related to the development of the library.

3.3 Is my MCU/hardware supported?

Every MCU which is capable of driving a display via Parallel port, SPI, RGB interface or anything else and fulfills the *Requirements* is supported by LittlevGL. It includes

- "Common" MCUs like STM32F, STM32H, NXP Kinetis, LPC, iMX, dsPIC33, PIC32 etc.
- Bluetooth, GSM, WiFi modules like Nordic NRF and Espressif ESP32
- Linux frame buffer like /dev/fb0 which includes Single board computers too like Raspberry
- and anything else with a strong enough MCU and a periphery to drive a display

3.4 Is my display supported?

LittlevGL needs just one simple driver to copy an array of pixels to a given area of the display. If you can do this your display then you use that display with LittlevGL. It includes

- TFTs with 16 or 24 bit color depth
- Monitors with HDMI port
- Small monochrome displays
- Gray-scale displays
- LED matrices
- or any other display where you can control the color/state of the pixels

See the *Porting* section to learn more.

3.5 Is LittlevGL free? How can I use it in a commercial product?

LittlevGL comes with MIT license which means you can download and use it for any purpose you want without any obligations.

3.6 Nothing happens, my display driver is not called. What have I missed?

Be sure you are calling `lv_tick_inc(x)` in an interrupt and `lv_task_handler()` in your main `while(1)`.

Learn more in the *Tick* and *Task handler* section.

3.7 Why the display driver is called only one? Only the upper part of the display is refreshed.

Be sure you are calling `lv_disp_flush_ready(drv)` at the end of you *display flush callback*.

3.8 Why I see only garbage on the screen?

Probably there a bug in your display driver. Try the following code without using LittlevGL:

```
#define BUF_W 20
#define BUF_H 10
lv_color_t buf[BUF_W * BUF_H];
lv_color_t * buf_p = buf;
uint16_t x, y;
for(y = 0; y < BUF_H; y++) {
    lv_color_t c = lv_color_mix(LV_COLOR_BLUE, LV_COLOR_RED, (y * 255) / BUF_H);
    for(x = 0; x < BUF_W; x++){
        (*buf_p) =  c;
        buf_p++;
    }
}

lv_area_t a;
```

(continues on next page)

(continued from previous page)

```
a.x1 = 10;
a.y1 = 40;
a.x2 = a.x1 + BUF_W - 1;
a.y2 = a.y1 + BUF_H - 1;
my_flush_cb(NULL, &a, buf);
```

3.9 Why I see non-sense colors on the screen?

Probably LittlevGL's the color format is not compatible with your displays color format. Check `LV_COLOR_DEPTH` in *lv_conf.h*.

If you are using 16 bit colors with SPI (or other byte-oriented) interface probably you need to set `LV_COLOR_16_SWAP 1` in *lv_conf.h*. It swaps the upper and lower bytes of the pixels.

3.10 How to speed up my UI?

- Turn on compiler optimization
- Increase the size of the display buffer
- Use 2 display buffers and flush the buffer with DMA (or similar periphery) in the background
- Increase the clock speed of the SPI or Parallel port if you use them to drive the display
- If you display has SPI port consider changing to a model with parallel because it has much higher throughput
- Keep the display buffer in the internal RAM (not external SRAM) because LittlevGL uses it a lot and it should have a small access time

3.11 How to reduce flash/ROM usage?

You can disable all the unused feature (like animations, file system, GPU etc) and object types in *lv_conf.h*.

IF you are using GCC you can add

- `-fdata-sections -ffunction-sections` compiler flags
- `--gc-sections` linker flag

to remove unused functions and variables. ‘

3.12 How to reduce the RAM usage

- Lower the size of the *Display buffer*
- Reduce `LV_MEM_SIZE` in *lv_conf.h*. This memory used when you create objects like buttons, labels, etc
- To work with lower `LV_MEM_SIZE` you can create the objects only when required and deleted them when they are not required anymore.

3.13 How to work with an operating system?

To work with an operating system where tasks can interrupt each other you should protect LittlevGL related function calls with a mutex. See the *Operation system* section to learn more.

3.14 How to contribute to LittlevGL?

There are several ways to contribute to LittlevGL:

- write a few lines about your project to inspire others
- answer other's questions
- report and/or fix bugs
- suggest and/or implement new features
- improve and/or translate the documentation
- write a blog post about your experiences

To learn more see Contributing guide

3.15 Where can I find the documentation of the previous version (v5.3)?

You can download it here and open offline:

`Docs-v5-3.zip`

3.15.1 Get started

Live demos

You can see how LittlevGL looks like without installing and downloading anything. There some ready made user interfaces which you can easily try in your browser.

Go to the Live demo page and choose a demo you are interested in.

Simulator on PC

You can try out the LittlevGL **using only your PC** without any development boards. Write a code, run it on the PC and see the result on the monitor. It is cross-platform: Windows, Linux and OSX are supported.The written code is portable, you can simply copy it when using an embedded hardware.

The simulator is also very useful to report bugs because it means common platform for every user. So it's a good idea to reproduce a bug in simulator and use the code snippet in the Forum.

Select an IDE

The simulator is ported to various IDEs. Choose your favorite IDE, read its README on GitHub, download the project, and load it to the IDE.

In followings the set-up guide of Eclipse CDT is described in more details.

Set-up Eclipse CDT

Install Eclipse CDT

Eclipse CDT is C/C++ IDE. You can use other IDEs as well but in this tutorial the configuration for Eclipse CDT is shown.

Eclipse is a Java based software therefore be sure **Java Runtime Environment** is installed on your system.

On Debian-based distros (e.g. Ubuntu): `sudo apt-get install default-jre`

You can download Eclipse's CDT from: https://eclipse.org/cdt/. Start the installer and choose *Eclipse CDT* from the list.

Install SDL 2

The PC simulator uses the SDL 2 cross platform library to simulate a TFT display and a touch pad.

Linux

On **Linux** you can easily install SDL2 using a terminal:

1. Find the current version of SDL2: `apt-cache search libsdl2 (e.g. libsdl2-2.0-0)`
2. Install SDL2: `sudo apt-get install libsdl2-2.0-0` (replace with the found version)
3. Install SDL2 development package: `sudo apt-get install libsdl2-dev`
4. If build essentials are not installed yet: `sudo apt-get install build-essential`

Windows

If you are using **Windows** firstly you need to install MinGW (64 bit version). After it do the following steps to add SDL2:

1. Download the development libraries of SDL.Go to https://www.libsdl.org/download-2.0.php and download *Development Libraries: SDL2-devel-2.0.5-mingw.tar.gz*
2. Decompress the file and go to *x86_64-w64-mingw32* directory (for 64 bit MinGW) or to *i686-w64-mingw32* (for 32 bit MinGW)
3. Copy *_…mingw32/include/SDL2* folder to *C:/MinGW/…/x86_64-w64-mingw32/include*
4. Copy *_…mingw32/lib/* content to *C:/MinGW/…/x86_64-w64-mingw32/lib*
5. Copy *_…mingw32/bin/SDL2.dll* to *{eclipse_worksapce}/pc_simulator/Debug/*. Do it later when Eclipse is installed.

Note: If you will use **Microsoft Visual Studio** instead of Eclipse then you don't have to install MinGW.

OSX

On **OSX** you can easily install SDL2 with brew: `brew install sdl2`

If something is not working I suggest this tutorial to get started with SDL.

Pre-configured project

A pre-configured graphics library project (based on the latest release) is always available. You can find it on GitHub or on the Download page. (The project is configured for Eclipse CDT.)

Add the pre-configured project to Eclipse CDT

Run Eclipse CDT. It will show a dialogue about the **workspace path**. Before accepting it check that path and copy (and unzip) the downloaded pre-configured project there. Now you can accept the workspace path. Of course you can modify this path but in that case copy the project to that location.

Close the start up window and go to **File->Import** and choose **General->Existing project into Workspace**. **Browse the root directory** of the project and click **Finish**

On **Windows** you have to do two additional things:

- Copy the **SDL2.dll** into the project's Debug folder
- Right click on the project -> Project properties -> C/C++ Build -> Settings -> Libraries -> Add … and add *mingw32* above SDLmain and SDL. (The order is important: mingw32, SDLmain, SDL)

Compile and Run

Now you are ready to run the LittlevGL Graphics Library on your PC. Click on the Hammer Icon on the top menu bar to Build the project. If you have done everything right you will not get any errors. Note that on some systems additional steps might be required to "see" SDL 2 from Eclipse but in most of cases the configurations in the downloaded project is enough.

After a success build click on the Play button on the top menu bar to run the project. Now a window should appear in the middle of your screen.

Now everything is ready to use the LittlevGL Graphics Library in the practice or begin the development on your PC.

Quick overview

Here you can learn the most important things about LittlevGL. You should read it first to get a general impression and read the detailed *Porting* and *Overview* sections after that.

Add LittlevGL into your project

The steps below show how to setup LittlevGL on an embedded system with a display and a touchpad. You can use the *Simulators* to get ready to use projects which can be run on your PC.

- Download or Clone the library
- Copy the `lvgl` folder into your project

- Copy `lvgl/lv_conf_templ.h` as `lv_conf.h` next to the `lvgl` folder and set at least `LV_HOR_RES_MAX`, `LV_VER_RES_MAX` and `LV_COLOR_DEPTH`.
- Include `lvgl/lvgl.h` where you need to use LittlevGL related functions.
- Call `lv_tick_inc(x)` every `x` milliseconds **in a Timer or Task** (`x` should be between 1 and 10). It is required for the internal timing of LittlevGL.
- Call `lv_init()`
- Create a display buffer for LittlevGL

```
static lv_disp_buf_t disp_buf;
static lv_color_t buf[LV_HOR_RES_MAX * 10];                    /*Declare a buffer␣
↪for 10 lines*/
lv_disp_buf_init(&disp_buf, buf, NULL, LV_HOR_RES_MAX * 10);   /*Initialize the␣
↪display buffer*/
```

- Implement and register a function which can **copy a pixel array** to an area of your display:

```
lv_disp_drv_t disp_drv;               /*Descriptor of a display driver*/
lv_disp_drv_init(&disp_drv);          /*Basic initialization*/
disp_drv.flush_cb = my_disp_flush;    /*Set your driver function*/
disp_drv.buffer = &disp_buf;          /*Assign the buffer to the display*/
lv_disp_drv_register(&disp_drv);      /*Finally register the driver*/

void my_disp_flush(lv_disp_t * disp, const lv_area_t * area, lv_color_t * color_p)
{
    int32_t x, y;
    for(y = area->y1; y <= area->y2; y++) {
        for(x = area->x1; x <= area->x2; x++) {
            set_pixel(x, y, *color_p);  /* Put a pixel to the display.*/
            color_p++;
        }
    }

    lv_disp_flush_ready(disp);         /* Indicate you are ready with the flushing*/
}
```

- Implement and register a function which can **read an input device**. E.g. for a touch pad:

```
lv_indev_drv_init(&indev_drv);              /*Descriptor of a input device driver*/
indev_drv.type = LV_INDEV_TYPE_POINTER;     /*Touch pad is a pointer-like device*/
indev_drv.read_cb = my_touchpad_read;       /*Set your driver function*/
lv_indev_drv_register(&indev_drv);          /*Finally register the driver*/

bool my_touchpad_read(lv_indev_t * indev, lv_indev_data_t * data)
{
    static lv_coord_t last_x = 0;
    static lv_coord_t last_y = 0;

    /*Save the state and save the pressed coordinate*/
    data->state = touchpad_is_pressed() ? LV_INDEV_STATE_PR : LV_INDEV_STATE_REL;
    if(data->state == LV_INDEV_STATE_PR) touchpad_get_xy(&last_x, &last_y);

    /*Set the coordinates (if released use the last pressed coordinates)*/
    data->point.x = last_x;
    data->point.y = last_y;
```

(continues on next page)

(continued from previous page)

```
    return false; /*Return `false` because we are not buffering and no more data to␣
↪read*/
}
```

- Call `lv_task_handler()` periodically every few milliseconds in the main `while(1)` loop, in Timer interrupt or in an Operation system task. It will redraw the screen if required, handle input devices etc.

Learn the basics

Objects (Widgets)

The graphical elements like Buttons, Labels, Sliders, Charts etc are called objects in LittelvGL. Go to *Object types* to see the full list of available types.

Every object has a parent object. The child object moves with the parent and if you delete the parent the children will be deleted too. Children can be visible only on their parent.

The *screen* are the "root" parents. To get the current screen call `lv_scr_act()`.

You can create a new object with `lv_<type>_create(parent, obj_to_copy)`. It will return an `lv_obj_t *` variable which should be used as a reference to the object to set its parameters. The first parameter is the desired *parent*, te second parameters can be an object to copy (`NULL` is unused). For example:

```
lv_obj_t * slider1 = lv_slider_create(lv_scr_act(), NULL);
```

To set some basic attribute `lv_obj_set_<paramters_name>(obj, <value>)` function can be used. For example:

```
lv_obj_set_x(btn1, 30);
lv_obj_set_y(btn1, 10);
lv_obj_set_size(btn1, 200, 50);
```

The objects has type specific parameters too which can be set by `lv_<type>_set_<paramters_name>(obj, <value>)` functions. For example:

```
lv_slider_set_value(slider1, 70, LV_ANIM_ON);
```

To see the full API visit the documentation of the object types or the related header file (e.g. `lvgl/src/lv_objx/lv_slider.h`).

Styles

Styles can be assigned to the objects to changed their appearance. A style describes the appearance of rectangle-like objects (like a button or slider), texts, images and lines at once.

You can create a new style like this:

```
static lv_style_t style1;                  /*Declare a new style. Should be `static`*/
lv_style_copy(&style1, &lv_style_plain);  /*Copy a buil-in style*/
style1.body.main_color = LV_COLOR_RED;          /*Main color*/
```

(continues on next page)

(continued from previous page)

```
style1.body.grad_color = lv_color_hex(0xffd83c) /*Gradient color (orange)*/
style1.body.radius = 3;
style1.text.color = lv_color_hex3(0x0F0)        /*Label color (green)*/
style1.text.font = &lv_font_dejavu_22;          /*Change font*/
...
```

To set a new style for an object use the `lv_<type>set_style(obj, LV_<TYPE>_STYLE_<NAME>, &my_style)` functions. For example:

```
lv_slider_set_style(slider1, LV_SLIDER_STYLE_BG, &slider_bg_style);
lv_slider_set_style(slider1, LV_SLIDER_STYLE_INDIC, &slider_indic_style);
lv_slider_set_style(slider1, LV_SLIDER_STYLE_KNOB, &slider_knob_style);
```

If an object's style is `NULL` then it will inherit its parent's style. For example, the labels' style are `NULL` by default. If you place them on a button then they will use the `style.text` properties from the button's style.

Learn more in *Style overview* section.

Events

Events are used to inform the user if something has happened with an object. You can assign a callback to an object which will be called if the object is clicked, released, dragged, being deleted etc. It should look like this:

```
lv_obj_set_event_cb(btn, btn_event_cb);                 /*Assign a callback to the␣
↪button*/

...

void btn_event_cb(lv_obj_t * btn, lv_event_t event)
{
    if(event == LV_EVENT_CLICKED) {
        printf("Clicked\n");
    }
}
```

Learn more about the events in the *Event overview* section.

Examples

Button with label

```
lv_obj_t * btn = lv_btn_create(lv_scr_act(), NULL);     /*Add a button the current␣
↪screen*/
lv_obj_set_pos(btn, 10, 10);                            /*Set its position*/
lv_obj_set_size(btn, 100, 50);                          /*Set its size*/
lv_obj_set_event_cb(btn, btn_event_cb);                 /*Assign a callback to the␣
↪button*/

lv_obj_t * label = lv_label_create(btn, NULL);          /*Add a label to the button*/
lv_label_set_text(label, "Button");                     /*Set the labels text*/
```

(continues on next page)