LVGL: Construire une application graphique

Table of Contents

Cet article analyse le code proposé par https://github.com/AIWintermuteAI/Seeed_reTerminal_LVGL_UI_Demo afin de:

1. présenter un exemple d'application **LVGL**
2. montrer les principes de base de l'utilisation des images dans **LVGL**

Application Terminal_LVGL_UI_Demo

Prérequis

La bibliothèque **SDL** de pilotes de bas niveau pour utiliser des graphiques, gérer la souris, le clavier, etc, doit être installlée.

```
sudo apt-get update && sudo apt-get install -y build-essential libsdl2-dev
```

Usage

Cloner le projet et les sous-modules associés :

```
git clone --recursive
https://github.com/AIWintermuteAI/Seeed_reTerminal_LVGL_UI_Demo.git
```

Les étapes suivantes peuvent être utilisées avec CMake sur un système de type Unix.

- **CMake** doit être installé, c'est-à-dire la commande **cmake** fonctionne sur le terminal.
- créer un nouveau répertoire. Le nom n'a pas d'importance (par exemple build).
- passer dans le répertoire build: `cd build`.
- Taper `cmake ...` CMake générera les fichiers de construction appropriés.
- Taper `make -j4` ou (plus portable) `cmake --build. --parallel`.

l'opérateur `parallel` est pris en charge à partir de CMake v3.12. Si on utilise une ancienne version de CMake, supprimer `--parallel` de la commande ou utiliser l'option make.

Le binaire sera dans ../bin/main, et peut être exécuté en tapant cette commande.

Description

Dans cette section on va décrire le contenu du fichier `lv_reterminal_demos/lv_demo_reterminal_UI/demo_reterminal_UI.c` qui est le principal code de l'application.

Tous les composants nécessaires sont importés et initialisés dans `main.c`, après quoi la fonction principale de l'interface utilisateur est appelée. La description de l'interface utilisateur, les appels et les fonctions d'assistance se trouvent à l'intérieur de `lv_demo_reterminal_UI/lv_demo_reterminal_UI.c`.

```
    tv = lv_tabview_create(lv_scr_act(), LV_DIR_TOP, tab_h);

    lv_obj_set_style_text_font(lv_scr_act(), font_normal, 0);

    lv_obj_t * tab_btns = lv_tabview_get_tab_btns(tv);
    lv_obj_set_style_pad_left(tab_btns, 0, 0);

    lv_obj_t * t1 = lv_tabview_add_tab(tv, "Assistant");
    lv_obj_t * t2 = lv_tabview_add_tab(tv, "Debug");
    lv_obj_t * t3 = lv_tabview_add_tab(tv, "Stats");
```

On créer un widget **Tabview** sur l'écran actif et on le remplit avec trois onglets : **Assistant**, **Debug** et **Stats**.

Le contenu de chaque onglet est initialisé séparément dans une fonction correspondante :

```
    assistant_create(t1) ;
    debug_create(t2) ;
    stats_create(t3) ;

    color_changer_create(tv);

    evdev_lis3dh_init();
```

De plus, des éléments de changement de couleur sont créés sur le widget **Tabview** et l'accéléromètre intégré est initialisé. Après cela, on créer trois rappels de minuterie avec des données d'entrée factices :

```
    static uint32_t user_data = 10;
    lv_timer_t * time_timer = lv_timer_create(time_timer_cb, 1,
&user_data);
    lv_timer_t * system_timer = lv_timer_create(system_timer_cb, 500,
&user_data);
    lv_timer_t * accelerometer_timer =
lv_timer_create(accelerometer_timer_cb, 50, &user_data);
```

Ceux-ci sont chargés d'obtenir respectivement l'heure du système, l'état du système (CPU, Mem, Espace disque, vitesse actuelle Ethernet, etc.) et les lectures de l'accéléromètre. on peut trouver ces trois fonctions de rappel au bas du fichier `lv_demo_reterminal_UI.c`.

```

void time_timer_cb(lv_timer_t * timer)
{

    time_t t = time(NULL);
    struct tm *local = localtime(&t);

    sprintf(timeString, "%02d:%02d:%02d", local->tm_hour, local->tm_min,
local->tm_sec);
    sprintf(dateString, "%s\n%s %02d %04d", DAY[local->tm_wday],
MONTH[local->tm_mon], local->tm_mday, local->tm_year + 1900);

    lv_label_set_text(clock_label, timeString);
    lv_label_set_text(date_label, dateString);

}

void system_timer_cb(lv_timer_t * timer)
{

    lv_meter_indicator_t *indic1 = timer->user_data;
    cpu_pct = 100 - lv_timer_get_idle();

    lv_mem_monitor_t mon;
    lv_mem_monitor(&mon);

    uint32_t used_size = mon.total_size - mon.free_size;;
    uint32_t used_kb = used_size / 1024;
    uint32_t used_kb_tenth = (used_size - (used_kb * 1024)) / 102;
    mem_pct = mon.used_pct;

    dsk_pct = get_available_space();
    eth0_num = get_current_network_speed();
    //light_num = get_light_sensor();

}

void accelerometer_timer_cb(lv_timer_t * timer)
{
```

```
    evdev_lis3dh_read(&data);

    lv_chart_set_next_value(chart1, x_ser, data.x_val);
    lv_chart_set_next_value(chart1, y_ser, data.y_val);
    lv_chart_set_next_value(chart1, z_ser, data.z_val);

}
```

Pour une application particulière, il peut être plus approprié d'utiliser d'autres widgets que Tabview. on peut consulter la description complète des widgets LVGL 8.0 ici pour une utilisation et des exemples.

Exemple d'utilisation des images

Seeed_reTerminal_LVGL_UI_Demo fournit un bureau uni, c'est un peu triste, l'objectif de cette section est d'adapter le code pour ajouter une image en arrière plan.

La source de l'image peut être :

- une variable dans le code (un tableau C avec les pixels).
- un fichier stocké en externe (comme sur une carte SD).
- un texte avec des symboles.

Image convertie en tableau de pixel

Pour définir la source d'une image, il faut utiliser `lv_img_set_src(img, src)`. Pour générer un tableau de pixels à partir d'une image PNG, JPG ou BMP, on peut utiliser l'outil de conversion d'image en ligne et définir l'image convertie avec son pointeur : `lv_img_set_src(img1, &convert_img_var);` Pour rendre la variable visible dans le fichier C, il faut la déclarer avec `LV_IMG_DECLARE (converted_img_var)`.

Pour ajouter une image d'arrière-plan on on va donc modifier la section suivante dans `lv_reterminal_demos/lv_demo_reterminal_UI/demo_reterminal_UI.c`:

```
lv_obj_t * panel1 = lv_obj_create(parent);
lv_obj_set_height(panel1, lv_pct(100));
```

```
LV_IMG_DECLARE(img_bubble_pattern);
lv_obj_t * panel1 = lv_obj_create(parent);
lv_obj_set_height(panel1, lv_pct(100));
lv_obj_set_pos(panel1, 0, 0);
lv_img_set_src(panel1, &img_bubble_pattern);
```

Utilisation d'une image externe

Pour utiliser des fichiers externes, il faut encoder l'image et utiliser le module de système de fichiers de LVGL pour utiliser un lecteur avec certaines fonctions de base des fichiers.

https://github.com/lvgl/lv_lib_png propose un décodeur PNG pour LVGL prêt à l'emploi

- Télécharger ou cloner ce dépôt
  - Télécharger depuis GitHub
  - Cloner : clone git https://github.com/littlevgl/lv_lib_png.git
- Inclure la bibliothèque : #include “lv*lib*png/lv*png.h*” * *Initialiser le decodeur avec lv*png_init()

L'implémentation de l'utilisation d'images PNG en LVGL utilise la bibliothèque **lodepng**.

Par défaut, **lodepng** utilise l'API IO de fichier C (par exemple, fopen) et les images peuvent être ouvertes comme ceci :
`lv_img_set_src(img, "./lv_lib_png/png_decoder_test.png");`

Si on veut que **lodepng** utilise l'API du système de fichiers de LVGL, il faut ajouter `#define LV_PNG_USE_LV_FILESYSTEM 1` à la fin de `lv_conf.h`. Après cela, l'image peut être ouverte comme ceci :
`lv_img_set_src(img, "P:lv_lib_lodepng/png_decoder_test.png");`