

Manuel de la librairie **uvsqgraphics**

Franck.Quessette@uvsq.fr

Octobre 2018



Table des matières

1	Introduction	2
2	Manuel de l'utilisateur	3
2.1	Types, Variables, Constantes	3
2.1.1	Types	3
2.1.2	Variables	3
2.1.3	Constantes	3
2.2	Affichage	4
2.2.1	Initialisation de la fenêtre graphique	4
2.2.2	Affichage automatique ou manuel	4
2.2.3	Création de couleur	4
2.3	Gestion du clavier et de la souris	4
2.3.1	Clavier	4
2.3.2	Déplacements la souris	4
2.3.3	Clics de la souris	5
2.3.4	Fin de programme	5
2.4	Dessin d'objets	5
2.5	Écriture de texte	6
2.6	Lecture d'entier	7
2.7	Gestion du temps	7
2.7.1	Chronomètre	7
2.7.2	Attendre	7
2.7.3	L'heure	7
2.8	Valeur aléatoires	7
2.9	Divers	8
3	Installation, compilation et Makefile	9
3.1	Installation	9
3.2	Utilisation	9
3.3	Compilation à la main dans le shell	9
3.4	Compilation avec Makefile	9
4	Un exemple complet	12
A	Annexes	14
A.1	Liste des caractères accentués	14
A.2	Liste des codes de couleur en français	14
A.3	Liste des noms des couleurs disponibles en français et en anglais	15

Chapitre 1

Introduction

Cette librairie en langage C est basée sur la version 1.2 de librairie SDL (www.libsdl.org). Elle permet de s'initier au langage C sans se soucier des entrées sorties, qui sont essentiellement graphiques.

Une documentation de la librairie SDL 1.2 est disponible sur le site www.libsdl.org/release/SDL-1.2.15/docs/.

Dans ce document le chapitre 2 décrit l'ensemble des fonctions disponibles ; le chapitre 3 explique comment installer la librairie sous Linux et le chapitre 4 donne un exemple complet d'utilisation.

Chapitre 2

Manuel de l'utilisateur

2.1 Types, Variables, Constantes

2.1.1 Types

Le type `POINT` permet de stocker les coordonnées d'un point.

```
typedef struct point {int x,y;} POINT;
typedef Uint32 COULEUR;
typedef int BOOL;
```

2.1.2 Variables

La largeur et la hauteur de la fenêtre graphique.

```
int WIDTH;
int HEIGHT;
#define HAUTEUR.FENETRE HEIGHT
#define LARGEUR.FENETRE WIDTH
```

Les variables `WIDTH` et `HEIGHT` sont initialisées lors de l'appel à `init_graphics()`.

2.1.3 Constantes

Les 16 couleurs en français et 140 en anglais (voir la liste en annexe). Quelques couleurs de base en français :

```
#define noir 0x000000
#define gris 0x777777
#define blanc 0xffffffff
#define rouge 0xff0000
#define vert 0x00ff00
#define bleu 0x0000ff
#define jaune 0x00ffff
#define cyan 0xffff00
#define magenta 0xff00ff
```

Les constantes booléennes :

```
#define TRUE 1
#define True 1
#define true 1
#define FALSE 0
#define False 0
```

```
#define false 0
```

2.2 Affichage

2.2.1 Initialisation de la fenêtre graphique

```
void init_graphics(int W, int H);
```

2.2.2 Affichage automatique ou manuel

Il y a deux modes d’affichage, automatique ou non. Quand l’affichage automatique est activé, chaque dessin d’objet s’affiche automatiquement. Quand il n’est pas automatique c’est l’appel à la fonction :

```
void affiche_all();
```

qui affiche les objets.

Pour basculer de l’affichage automatique au non automatique, il y a deux fonctions :

```
void affiche_auto_on();  
void affiche_auto_off();
```

Sur les ordinateurs lents, il vaut mieux ne pas afficher les objets un par un, mais les afficher que quand c’est nécessaire. C’est aussi utile pour avoir un affichage plus fluide quand on fait des animations.

Par défaut on est en mode automatique.

2.2.3 Création de couleur

La fonction

```
COULEUR couleur_RGB(int r, int g, int b);
```

prend en argument les 3 composantes rouge (**r**), vert (**g**) et bleue (**b**) qui doivent être comprises dans l’intervalle $[0..256[$ et renvoie la couleur.

2.3 Gestion du clavier et de la souris

2.3.1 Clavier

La fonction (écrite par Yann Strozecki)

```
int get_key();
```

renvoie le code ascii du caractère de la dernière touche du clavier qui a été pressée. Si aucune touche n’a été pressée, renvoie -1. Cette fonction est non bloquante.

Si depuis le dernier appel à la fonction :

```
POINT get_arrow();
```

il y a eu n_g appuis sur la flèche gauche, n_d appuis sur la flèche droite n_h appuis sur la flèche haut et n_b appuis sur la flèche bas. Le point renvoyé vaudra en $x = n_d - n_g$ et en $y = n_h - n_b$.

Cette fonction est non bloquante, c’est à dire que si aucune flèche n’a été appuyée les champs **x** et **y** vaudront 0.

2.3.2 Déplacements la souris

La fonction :

```
POINT get_mouse();
```

renvoie le déplacement de souris avec la même sémantique que `POINT get_arrow()`. Cette fonction est non bloquante : si la souris n'a pas bougé les champs `x` et `y` vaudront 0.

2.3.3 Clics de la souris

La fonction :

```
POINT wait_clic();
```

attend que l'utilisateur clique avec le bouton gauche de la souris et renvoie les coordonnées du point cliqué. Cette fonction est bloquante.

La fonction :

```
POINT wait_clic_GMD(char *button);
```

attend que l'utilisateur clique et renvoie dans `*button` le bouton cliqué : le caractère `G` pour gauche, le `M` pour milieu et le `D` pour droit. Cette fonction est bloquante.

La fonction (écrite par Yann Strozecki) :

```
POINT get_clic();
```

renvoie la position du dernier clic de souris. Si le bouton gauche n'a pas été pressée, renvoie $(-1, -1)$. Cette fonction non bloquante.

La fonction (écrite par Pierre Coucheney et Franck Quessette)

```
int wait_key_arrow_clic (char *touche, int *fleche, POINT *P);
```

renvoie une des trois constantes `EST_FLECHE`, `EST_TOUCHE` ou `EST_CLIC` selon l'action effectuée. Selon l'action, un des trois argument est modifié, les autres sont inchangés.

- Si une flèche est appuyée, la valeur mise dans l'argument `*fleche` est une des quatre constantes `FLECHE_GAUCHE`, `FLECHE_DROITE`, `FLECHE_HAUT` ou `FLECHE_BAS`.
- Si une touche est appuyée, la valeur mise dans l'argument `*touche` est la lettre majuscule correspondant à la touche.
- Si un clic est effectuée, les valeurs mises dans l'argument `*P` sont les coordonnées du point cliqué.

Cette fonction est bloquante tant qu'une des trois actions n'a pas été effectuée.

2.3.4 Fin de programme

La fonction :

```
POINT wait_escape();
```

attend que l'on tape la touche `Echap` ou `esc` et termine le programme.

2.4 Dessin d'objets

Cette fonction remplit tout l'écran de la couleur en argument :

```
void fill_screen(COULEUR coul);
```

Les fonctions ci-dessous dessinent l'objet dont elle porte le nom. Quand le nom de la fonction contient `fill`, l'intérieur est également colorié.

```

void pixel(POINT p, COULEUR coul);
void draw_line(POINT p1, POINT p2, COULEUR coul);

// Les rectangles ont leurs côtés parallèles aux bords
void draw_rectangle(POINT p1, POINT p2, COULEUR coul);
void draw_fill_rectangle(POINT p1, POINT p2, COULEUR coul);

void draw_circle(POINT centre, int rayon, COULEUR coul);
void draw_fill_circle(POINT centre, int rayon, COULEUR coul);

void draw_triangle(POINT p1, POINT p2, POINT p3, COULEUR coul);
void draw_fill_triangle(POINT p1, POINT p2, POINT p3, COULEUR coul);

void draw_ellipse(POINT f1, POINT f2, int r, COULEUR coul);
void draw_fill_ellipse(POINT f1, POINT f2, int r, COULEUR coul);

```

2.5 Écriture de texte

Pour l’affichage de texte ; il n’y a qu’une seule police : verdana. La police peut-être changée dans le fichier `graphics.h`, mais nécessite ensuite une recompilation de la librairie et une réinstallation.

Pour les lettres accentuées, il faut respecter l’encodage ttf. Pour ce faire on peut utiliser les constantes définies (voir annexe). Par exemple :

```
char *s = "Cha"icirc"ne de caract"egrave"re";
```

`icirc` vaut la chaîne `"\xee"` qui est `î` et on utilise la concaténation de constantes de type chaînes de caractère.

L’écriture se fait avec la fonction :

```
void aff_pol(char *a_ecrire, int taille, POINT hg, COULEUR coul);
```

- `a_ecrire` est la chaîne de caractère contenant le texte à écrire ;
- `taille` est la taille de la police ;
- `hg` fourni les coordonnées en haut à gauche où sera positionnée le texte ;
- `coul` est la couleur d’écriture.

La même chose mais en donnant le point central du texte et non plus le point en haut à gauche :

```
void aff_pol.centre(char *a_ecrire, int taille, POINT centre, COULEUR coul);
```

Pour positionner comme on le souhaite un texte, les deux fonctions :

```
int largeur_texte(char *a_ecrire, int taille);
int hauteur_texte(char *a_ecrire, int taille);
```

renvoie la largeur et la hauteur, en pixels, du texte à écrire.

Le style peut être changé grâce à la fonction

```
void pol_style(int style);
```

Les styles possibles sont : `NORMAL`, `GRAS`, `ITALIQUE` et `SOULIGNE`. Le style est appliqué jusqu’au prochain appel de la fonction. Au départ le style est `NORMAL`.

La fonction ci-dessous affiche directement un entier sans avoir à le mettre dans une chaîne de caractères :

```
void aff_int(int n, int taille, POINT p, COULEUR C);
```

Les fonctions suivantes affichent dans la fenêtre graphique comme dans une fenêtre shell. Commence en haut et se termine en bas mais il n’y a pas de scroll. La police est de taille 20 et de couleur blanc.

Écriture d’une chaîne de caractères, d’un entier, d’un flottant ou d’un booléen :

```
void write_text(char *a_ecrire);
void write_int(int n);
void write_float(float x);
void write_bool(BOOL b);
```

La fonction :

```
void writeln();
```

effectue un retour à la ligne.

2.6 Lecture d'entier

Renvoie d'un entier tapé au clavier.

```
int lire_entier_clavier();
```

Cette fonction est bloquante

2.7 Gestion du temps

2.7.1 Chronomètre

Ce chronomètre est précis à la microseconde. Il y a deux fonctions, l'une démarre le chrono ou le remet à zéro s'il a déjà été démarré :

```
void chrono_start();
```

L'autre renvoie la valeur courante du chrono en secondes sans l'arrêter :

```
float chrono_lap();
```

Elle peut donc être appelée autant de fois que nécessaire.

2.7.2 Attendre

Attend le nombre de millisecondes passé en argument :

```
void attendre(int millisecondes);
```

2.7.3 L'heure

Renvoie la valeur de l'heure courante ([0..23]) :

```
int heure();
```

Renvoie la valeur de la minute courante ([0..59]) :

```
int minute();
```

Renvoie les secondes courantes à la microseconde près ([0;60[) :

```
float seconde();
```

2.8 Valeur aléatoires

Renvoie d'un float uniformément distribué dans l'intervalle [0;1[:


```
float alea_float();
```

Renvoie d'un `int` uniformément distribué dans l'intervalle $[0..N[$ soit N valeurs différentes de 0 à $N - 1$:

```
int alea_int(int N);
```

2.9 Divers

Calcule de la distance entre deux points :

```
float distance(POINT P1, POINT P2);
```

qui est : $\sqrt{(P1.x - P2.x)^2 + (P1.y - P2.y)^2}$.

Chapitre 3

Installation, compilation et Makefile

3.1 Installation

L'installation se fait dans une fenêtre shell sur une machine linux ubuntu.

1. récupérer le fichier `UVSQ_graphics.zip`

http://e-campus2.uvsq.fr/Members/franques/Fichiers/element_view?idElement=Fichier-franques-20181017215540

2. puis décompresser et aller dans le dossier `UVSQ_graphics` :



```
$> unzip UVSQ_graphics.zip
$> cd UVSQ_graphics
```

3. enfin, installer :

```
$> make install
```

À la fin de l'installation une fenêtre de l'éditeur geany se lance avec le code du programme `demo1.c` qui est également en Annexe de ce document.

3.2 Utilisation

Une fois que l'on a un programme dans l'éditeur geany, le bouton  ou bien `<>` permet de compiler puis le bouton à sa droite  ou bien une coche `✓`) permet d'exécuter.

3.3 Compilation à la main dans le shell

Si tout le programme est dans un seul fichier, la commande de compilation `makeuvsq` est disponible (après avoir fait l'installation). Par exemple pour compiler et exécuter le programme `demo1.c`, taper :

```
$> makeuvsq demo1
$> ./demo1
```

3.4 Compilation avec Makefile

L'exemple ci-dessous montre un projet avec trois fichiers `affiche.h`, `affiche.c`, `principal.c` et le `Makefile` qui va avec. Ces quatre fichiers sont dans le sous dossier `Compilation_separee_avec_uvsqgraphics` du dossier `UVSQ_graphics` décrit dans la section installation ci-dessus.

Le fichier `affiche.h` :

```
void initialiser_affichage();
void ecrire_texte_dans_cercle(char *texte);
```

Le fichier `affiche.c` qui contiennent des fonctions d’affichage qui utilisent la librairie `uvsqgraphics` :

```
#include <uvsqgraphics.h>
#define TAILLE_POLICE 30

void initialiser_affichage() {
    init_graphics(600,400);
}

void ecrire_texte_dans_cercle(char *texte) {
    int rayon = 2 + largeur_texte(texte,TAILLE_POLICE)/2;
    POINT centre; centre.x = LARGEUR_FENETRE/2; centre.y = HAUTEUR_FENETRE/2;
    POINT bas; bas.x = centre.x; bas.y = centre.y - 50;
    fill_screen(blanc);
    int i;
    char lenombre[4];
    for (i=0 ; i<=50 ; i++) {
        draw_fill_circle(centre,rayon,gold);
        if (i%2) aff_pol_centre(texte, TAILLE_POLICE, centre, red);
        sprintf(lenombre,"%d",50-i);
        aff_pol_centre(lenombre,TAILLE_POLICE+3*i,bas,red);
        attendre(100);
    }
}
```

Le fichier `principal.c` qui contient la fonction `main` et qui appelle des fonctions de `affiche.h` :

```
#include "affiche.h"
#include <stdlib.h>

int main() {
    initialiser_affichage();
    ecrire_texte_dans_cercle("Bonjour le monde !");
    exit(0);
}
```

Le fichier `Makefile` :

```
all: principal
    ./principal

# Exemple donnant les options nécessaires pour faire l’édition de lien avec uvsqgraphics
# Attention, ce sont des back-apostrophes et non pas des apostrophes
principal: principal.o affiche.o
    gcc -o principal principal.o affiche.o -luvsqgraphics ‘sdl-config --libs’ -lm -lSDL_ttf

principal.o: principal.c affiche.h
    gcc -c principal.c

# Exemple donnant les options nécessaires pour compiler avec uvsqgraphics
# Attention, ce sont des back-apostrophes et non pas des apostrophes
affiche.o: affiche.c affiche.h
```

```
gcc -c 'sdl-config --cflags' affiche.c

clean:
    rm -f affiche.o
    rm -f principal.o
    rm -f principal
```

Commentaires :

— La ligne

```
#include <uvsqgraphics.h>
```

au début d'`affiche.c` permet d'utiliser la librairie `uvsqgraphics` dans ce fichier.

— La commande de compilation de `affiche.c` est :

```
gcc -c 'sdl-config --cflags' affiche.c
```

qui produit `affiche.o`.

— La commande de compilation de `principal.c` est :

```
gcc -c 'sdl-config --cflags' principal.c
```

qui produit `principal.o`.

— La command d'édition de lien pour produire l'exécutable final qui s'appelle `principal` est :

```
gcc -o principal affiche.o principal.o -luvsqgraphics 'sdl-config --libs' -lm -lSDL_ttf
```

Vous pouvez ajouter des options à ces commandes comme `-g` ou `-Wall`.

Attention : '`sdl-config --cflags`' pour la compilation et '`sdl-config --libs`' pour l'édition de liens sont entourés de back quotes (apostrophes à l'envers). Dans les deux cas, c'est la commande `sdl-config` qui est exécutée, seule l'option change. Ces commandes permettent d'inclure la librairie SDL. Sur une machine lubuntu 64 bits ces commandes donnent :

```
$> sdl-config --cflags
-I/usr/include/SDL -D_GNU_SOURCE=1 -D_REENTRANT
$> sdl-config --libs
-L/usr/lib/x86_64-linux-gnu -lSDL
```

On pourrait donc mettre

```
-I/usr/include/SDL -D_GNU_SOURCE=1 -D_REENTRANT
```

au lieu de

```
'sdl-config --cflags'
```

dans la commande de compilation. Ce serait néanmoins moins bien du point de vue de la portabilité.

La commande d'édition de lien inclus en plus la librairie `uvsqgraphics` avec l'option `-luvsqgraphics`, la librairie mathématique avec l'option `-lm` et la librairie SDL spécialisé dans l'écriture de texte avec l'option `-lSDL_ttf`. L'ordre dans lequel ces options sont mises est important.

Chapitre 4

Un exemple complet

Le fichier `demo1.c` :

```
#include <uvsqgraphics.h>

// #####
// Démo 1 de l'API uvsqgraphics
// Dans geany, cliquer sur <.> pour compiler
// et sur la coche à droite de <.> pour exécuter
// #####

int main () {
    POINT P1, P2;
    init_graphics (800,600);
    fill_screen (antiquewhite);

    // Affichage d'un disque et d'un cercle
    P1.x = 200; P1.y = 300;
    draw_fill_circle(P1,50,blueviolet);
    draw_circle(P1,40,white);

    // Affichage d'un rectangle avec un bord d'une couleur différente
    P2.x = 100; P2.y = 100;
    draw_fill_rectangle(P1,P2,violetlight);
    draw_rectangle(P1,P2,violet);

    // Affichage de texte de taille 20
    char *a_ecrire = "Cliquer dans la fen"ecirc"tre";
    P1.x = LARGEUR_FENETRE/2 - largeur_texte(a_ecrire,20)/2;
    P1.y = 2*hauteur_texte(a_ecrire,20);
    aff_pol(a_ecrire,20,P1,gris);

    // Attente d'un clic
    P2 = wait_clic();
    // Remplissage de l'écrans
    fill_screen(lavender);
    // Affichage d'un cercle centré sur le point cliqué
    P1.x = LARGEUR_FENETRE/2 - largeur_texte(a_ecrire,20)/2;
    aff_pol(a_ecrire,20,P1,gris);
```

```
P2 = wait_clc();
draw_fill_circle(P2,80,crimson);

// Affichage d'un cercles centré sur le point cliqué
P2 = wait_clc();
draw_fill_circle(P2,80,rouge);

// Attente de l'appui sur la touche Echap pour terminer
wait_escape();
exit(0);
}
```

Annexe A

Annexes

A.1 Liste des caractères accentués

La dénomination est identique à celle utilisée en html.

à	#define	agrave	"\xe0"
á	#define	aacute	"\xe1"
â	#define	acirc	"\xe2"
ä	#define	auml	"\xe4"
ç	#define	ccedil	"\xe7"
è	#define	egrave	"\xe8"
é	#define	eacute	"\xe9"
ê	#define	ecirc	"\xea"
ë	#define	euml	"\xeb"
î	#define	icirc	"\xee"
ï	#define	iuml	"\xef"
ô	#define	ocirc	"\xf4"
ù	#define	ugrave	"\xf9"
û	#define	ucirc	"\xfb"
ü	#define	uuml	"\xfc"
Ç	#define	Ccedil	"\xc7"
È	#define	Egrave	"\xc8"
É	#define	Eacute	"\xc9"
Ê	#define	Ecirc	"\xca"

A.2 Liste des codes de couleur en français

```
#define argent      0xC0C0C0
#define blanc      0xFFFFFFFF
#define bleu       0x0000FF
#define bleumarine 0x000080
#define citronvert 0x00FF00
#define cyan       0x00FFFF
#define magenta    0xFF00FF
#define gris       0x808080
#define jaune      0xFFFF00
#define marron     0x800000
#define noir       0x000000
```

```
#define rouge      0xFF0000
#define sarcelle   0x008080
#define vert       0x00FF00
#define vertclair  0x008000
#define vertolive  0x808000
#define violet     0x800080
```

A.3 Liste des noms des couleurs disponibles en français et en anglais

La section 2.2.3 explique comment créer de nouvelles couleurs.

Français			
argent	blanc	bleu	bleumarine
citronvert	cyan	magenta	gris
jaune	marron	noir	rouge
sarcelle	vert	vertolive	violet
Anglais			
aliceblue	antiquewhite	aqua	aquamarine
azure	beige	bisque	black
blanchedalmond	blue	blueviolet	brown
burlywood	cadetblue	chartreuse	chocolate
coral	cornflowerblue	cornsilk	crimson
cyan	darkblue	darkcyan	darkgoldenrod
darkgray	darkgreen	darkkhaki	darkmagenta
darkolivegreen	darkorange	darkorchid	darkred
darksalmon	darkseagreen	darkslateblue	darkslategray
darkturquoise	deeppink	deeppink	deepskyblue
dimgray	dodgerblue	firebrick	floralwhite
forestgreen	fuchsia	gainsboro	ghostwhite
gold	goldenrod	gray	green
greenyellow	honeydew	hotpink	indianred
indigo	ivory	khaki	lavender
lavenderblush	lawngreen	lemonchiffon	lightblue
lightcoral	lightcyan	lightgoldenrodyellow	lightgreen
lightgrey	lightpink	lightsalmon	lightseagreen
lightskyblue	lightslategray	lightsteelblue	lightyellow
lime	limegreen	linen	magenta
maroon	mediumaquamarine	mediumblue	mediumorchid
mediumpurple	mediumseagreen	mediumslateblue	mediumspringgreen
mediumturquoise	mediumslateblue	midnightblue	mintcream
mistyrose	moccasin	navajowhite	navy
oldlace	olive	olivedrab	orange
orangered	orchid	palegoldenrod	palegreen
paleturquoise	palevioletred	papayawhip	peachpuff
peru	pink	plum	powderblue
purple	red	rosybrown	royalblue
saddlebrown	salmon	sandybrown	seagreen
seashell	sienna	silver	skyblue
slateblue	slategray	snow	springgreen
steelblue	tan	teal	thistle
tomato	turquoise	violetlight	wheat
white	whitesmoke	yellow	yellowgreen

http://fr.wikipedia.org/wiki/Liste_de_couleurs
http://fr.wikipedia.org/wiki/Liste_de_couleurs