Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Documentation `libg2x`(version 6e-23)

`libg2x` *est une bibliothèque graphique, écrite en* `C`*, basée sur* `OpenGl` *et* `freeglut` [a].

Plus précisément, elle est destinée au prototypage rapide de programmes graphiques simples en 2D.

Les fonctionalités `OpenGl` *sont totalement masquées et souvent contraintes (mais restent accessibles) pour fournir à l'utilisateur un environnement graphique bridé mais très simple.*

[a] (alternative à `GLU (GL-Utility)` et `GLUT (GL-Utility-Toolkit)`)

Principes généraux et structure des codes

L'architecture de fonctionnement repose sur un découpage imposé des tâches semblable à une architecture "Modèle-Vue-Contrôleur".

De nombreuses fonctionalités, habituellement à la charge du programmeur, sont prédéfinies ici, en particulier pour tout ce qui touche à la gestion des fenêtres et des options de contrôles.

Un programme utilisant la `libg2x` sera structuré sur la base de quelques fonctions essentielles présentées dans l'exemple `work/src/g2x_00_squelette.c` et détaillées ci-dessous.

```
-------------------------------------------------------------------------------
01| #include <g2x.h>
02|
03| static int WWIDTH=512, WHEIGHT=512;
04| static double xmin=-10.,ymin=-10.,xmax=+10.,ymax=+10.;
05|
06| static void init(void) { /* la fonction d'initialisation          */ }
07| static void ctrl(void) { /* la fonction de contrôle               */ }
08| static void evts(void) { /* la fonction de gestion des événements */ }
09| static void draw(void) { /* la fonction de dessin                  */ }
10| static void anim(void) { /* la fonction d'animation                */ }
11| static void quit(void) { /* la fonction de sortie                  */ }
12|
13| int main(int argc, char *argv)
14| {
15|   g2x_InitWindow(*argv,WWIDTH,WHEIGHT);
16|   g2x_SetWindowCoord(xmin,ymin,xmax,ymax);
17|
18|   g2x_SetInitFunction(init); /* fonction d'initialisation */
19|   g2x_SetCtrlFunction(ctrl); /* fonction de contrôle      */
20|   g2x_SetEvtsFunction(evts); /* fonction d'événements     */
21|   g2x_SetDrawFunction(draw); /* fonction de dessin        */
22|   g2x_SetAnimFunction(anim); /* fonction d'animation      */
23|   g2x_SetExitFunction(quit); /* fonction de sortie        */
24|
25|   return g2x_MainStart();
26| }
-------------------------------------------------------------------------------
```

Convention de nommage

- Les **modules** : de la forme `g2x_nom.[h,c]` - à l'exception du *header* principal `<g2x.h>`
- Les **types** : de la forme `G2Xnom` - à l'exception des (re-)définitions du module `<g2x_types.h>` qui crée/renomme quelques types standards (`bool` , `uchar` , `uint` ....)
- Les **macros** : de la forme `G2XNOM` - à l'exception de quelques macros classiques, définies dans `<g2x_types.h>` (`PI, ZERO, SQR(), MIN(), MAX()` ....)
- Les **fonctions** : de la forme `g2x_Nom()` - pas d'exception à cette règle

Les dépendances

`01| #include <g2x.h>` : seule dépendance essentielle.

Cet appel contient déjà la plupart des `include` standards (`stdio.h` , `stdlib.h` , `math.h` ...)

La fenêtre graphique

`03| int WWIDTH, WHEIGHT;` → `15| g2x_InitWindow(*argv,WWIDTH,WHEIGHT)`

Création de la fenêtre, avec ses dimensions initiales (en pixels), positionnée au coin supérieur gauche de l'écran (coord. `(0,0)`).

Dimensions et position de la fenêtre sont ajustables en cours d'exécution et un mode 'plein-écran' est disponible (`<Ctrl+'f'>` ou `<F11>`). Les dimensions courantes sont accessibles à tout moment via des fonctions `get` : `g2x_GetPixWidth(); g2x_GetPixHeight();`

→ C'est la 1ère fonction à appeler et elle est **indispensable** pour initialiser le système graphique.

La zone de travail réelle

`04| double xmin,ymin,xmax,ymax;` → `16| g2x_SetWindowCoord(xmin,ymin,xmax,ymax)`

Définition de la zone de travail (réelle) associée à la fenêtre graphique.

La fenêtre graphique en tant que 'matrice de pixels' n'est pas directement accessible à l'utilisateur (sauf manip. particulière, hors sujet ici)

La zone de travail associée est en réalité un espace réel infini dans lequel on peut naviguer. Il faut néanmoins initialiser le positionnement et les dimensions de la fenêtre graphique sur cet espace. Cela se fait en associant des coordonnées réelles aux coins inf. gauche (x_{min}, y_{min}) et sup. droit (x_{max}, y_{max}).

attention : ces valeurs initiales doivent être compatibles avec les tailles `WWIDTH` et `WHEIGHT` pour assurer un rapport d'aspect cohérent (un cercle apparaît bien comme un cercle, et non une ellipse) : `(xmax-xmin)/(ymax-ymin) = WWIDTH/WHEIGHT` .

Ces coordonnées s'adaptent automatiquement lors d'un redimensionnement de la fenêtre graphique ou lors d'un zoom, d'un *panscan* mais le rapport d'apsect initial est conservé.

Si la fonction `g2x_SetWindowCoord` n'est pas appelée, le système utilisera la zone par défaut définie par les tailles de la fenêtre graphique : `(0.,0.,WWIDTH,WHEIGHT);`

Les dimensions courantes (pixels et réelles) sont accessibles à tout moment via des fonctions d'accès dédiées `g2x_GetPix{Width|Height}` et `g2x_Get{X|Y}{Min|Max}();` .

☞ Cela permet de contrôler/ajuster les comportements des programmes en cours d'éxécution.

note : on peut aussi connaître la taille 'réelle' d'un pixel écran : `g2x_Get{X|Y}PixSize();`

☞ Ca peut permettre de limiter la précision des calculs/dessins en coordonnées réelles (inutile d'aller au dela de la précision d'un pixel écran) ou d'ajuster certains tracés en fonction du rapport d'échelle (pixel/réel).

La couleur de fond (`BackGround`) de l'écran 2D est limitée à une valeur de gris ($\in [0,1]$) réglable et accessible via les fonctions `g2x_SetBkGdCol` et `g2x_GetBkGdCol` . La commande prédéfinie `<Ctrl+'w'>` inverse ce niveau de gris.

Les commandes de *navigation* dans la fenêtre réelle sont :

- le `zoom` avant/arrière : associé à la molette (*wheel*) de la souris et aux touches `'+'/'-'` du clavier.
- le `panscan` (déplacements x/y) associé au clic-milieu de la souris
- la touche `'='` remet tout ça aux conditions initiales définies au lancement.
- Les touches `<Ctrl+'f'>` ou `<F11>` basculent en mode plein écran

☞ Les touches `'?'` ou `<F12>` affichent une aide qui résume les principales commandes prédéfinies. Les options de contrôle crées par l'utilisateur viendront s'ajouter automatiquement (ou presque) à cette liste (cf. `<g2x_REFERENCE.c>`)

La structuration du reste du code, c'est-à-dire, l'interface utilisateur `libg2x`, passe par la définition de 6 fonctions spécifiques, de format imposé, chacune avec un rôle très précis.

A chacune de ces fonctions correspond un *handler* spécifique `g2x_Set***Function(void (*f)(void));`.

Cette structure, parfois un peu contraignante, à l'avantage de rendre les codes très facile à maintenir et à faire évoluer. Cela permet également de passer très simplement de la `libg2x` à une autre lib. graphique (telle que la `libMLV`).

L'architecture qui en découle peut être assimilée à ce que l'on peut trouver dans un jeu vidéo, avec ses différents "moteurs" :

① Chargement

② Boucle de jeu

ⓐ Capture des événements (clavier, souris, joystick....)

ⓑ Interprétation de événements ("moteur de jeu")

ⓒ Calculs divers ("moteurs physiques" ....)

ⓓ Affichage ("moteur graphique")

③ Sortie

les 6 fonctions spécifiques et leurs *handlers*

Le protoype imposé `void f(void)` à ces fonctions, sans paramètre ni valeur de retour, implique que les différentes fonctions devront communiquer via des variables globales. Bien que ça ne fasse pas partie du *"manuel des bonnes pratiques en C"*, il n'y pas vraiment d'alternative.

1. procédure d'initialisation : `06| void init(void) {...}` → `18| g2x_SetInitFunction(init);`

C'est l'étape chargement des données. Appelée une seule fois, avant le lancement de la boucle principale, cette fonction crée et initialise les données globales. C'est ici que doivent se faire les éventuelles allocations mémoire, chargement de fichier...

Cette fonction ne doit contenir aucun appel à des routines d'affichage (ils ne seraient pas pris en compte).

Elle peut ne pas être définie → `18| g2x_SetInitFunction(NULL);`

2. procédure de contrôle : `07| void ctrl(void) {...}` → `19| g2x_SetCtrlFunction(ctrl);`

Définit les outils de contrôle et d'interaction (création des `bouttons`, `scrollbars` ...) . Appelée une seule fois, juste après la fonction d'initialisation et avant le lancement de la boucle principale.

Tout ce qu'il y a ici pourrait être directement écrit dans la fonction `init()`, mais c'est plus 'propre' et plus pratique de séparer.

Cette fonction ne doit contenir aucun appel à des routines d'affichage (ils ne seraient pas pris en compte)

Si elle n'est pas définie → `19| g2x_SetCtrlFunction(NULL);`

3. procédure d'événements : `08| void evts(void) {...}` → `20| g2x_SetEvtsFunction(evts);`

Réception et traitement des interruptions clavier, souris.... Cette fonction est appelée, à chaque cycle, juste avant la fonction d'affichage `draw()`. Tout ce qu'il y a ici pourrait d'ailleurs être directement intégré à la fonction d'affichage mais c'est plus 'propre' et plus pratique de séparer.

Si elle n'est pas définie → `20| g2x_SetEvtsFunction(NULL);`

4. procédure d'affichage principal : `09| void draw(void) {...}` → `21| g2x_SetDrawFunction(draw);`

C'est la seule fonction (parmi les 6) qui soit **indispensable**. Sans elle, rien ne s'affichera. Elle est appelée par la boucle principale qui gère également le rafraîchissement, les `zoom/panscan` et la synchro avec les fonctions de contrôle (`ctrl`), capture d'événement (`evts`) et d'animation (`anim`).

Etant appelée en boucle infinie, elle ne doit pas contenir d'appel de gestion de mémoire et ne doit pas modifier les données globales. Si celles-ci doivent être modifiées dynamiquement, c'est par l'une des 2 autres fonctions `evts` (clavier/souris) ou `anim` (changement d'état).

5. procédure d'animation : `10| void anim(void) {...}` → `22| g2x_SetAnimFunction(anim);`

Réalise les calculs de changement d'état des données. Comme son nom l'indique, son usage est réservé aux applications produisant une animation, c'est à dire une séquence d'images (produites par la fonction d'affichage draw) entre lesquelles les "données" (points, vecteurs, couleurs...) changent.

C'est cette fonction `anim` qui pilote cette modification des données.

Elle ne doit contenir aucun appel graphique ni aucun appel de gestion de mémoire. Elle est appellée en boucle, en synchronisation, avec la fonction d'affichage.

Si elle n'est pas définie → `22| g2x_SetAnimFunction(NULL);`

6. procédure de sortie : `11| void quit(void) {...}` → `23| g2x_SetExitFunction(quit);`

A priori lorsque l'appication se termine (sortie de la boucle infinie) elle ne revient pas dans le programme source (c'est `X11` ou `freeglut` qui envoie le signal d'arrêt `exit()`).

Les tâches à effectuer en fin de programme (affichage terminal, libération de mémoire, sauvegarde sur fichier....) sont à placer dans cette fonction `quit()` qui sera passée à `atexit()` à l'arrêt du programme.

Si il n'y a rien à faire en sortie → `23| g2x_SetExitFunction(NULL);`

la boucle d'éxécution : `25| return g2x_MainStart();`

L'appel à cette fonction démarre tout le processus : branchements, exécution et synchronisation des fonctions spécialisées et surtout lancement de la boucle infinie (`<ESC>` ou `<Ctrl+'q'>` pour quitter proprement.

Cette boucle met en séquence les 3 étapes que sont la récupération des événements (`events`) les calculs d'évolution (`anim`) et l'affichage (`draw`).

Au final, la fonction principale `int main(int argc, char* argv[ ]);` ne changera jamais, mis à part pour mettre à `NULL` un des *handlers* (ou supprimer la ligne d'appel) lorsqu'une des fonctions d'interface est inutile (jamais `draw`, rarement `init`, souvent `anim`).

Rapide survol des modules

La documentation complète et précise de la `libg2x` est encore en cours. A ce stade, elle se résume aux commentaires présents dans les fichiers d'en-tête `<g2x_*.h>`

Les basiques

- `<g2x_types.h>` : contient des (re-)définition (raccourcis) de types standards (par exemple le type `uchar` pour `unsigned char`, le type énuméré `bool` ...) et la définition de quelques constantes (`PI, ZERO...`) et macro (`SQR, MIN, MAX ...`) classiques .

  On trouve également ici la constante `ZERO` et les macros `IS_ZERO` et `G2Xiszero` (identiques), qui servent à remplacer les tests d'égalité sur les réels. Par exemple, avec 2 réels simples `double a=1., b=0.1;`, le test `if (10.*b==a)` donne pour résultat `false` à cause de l'imprécision des réels au format IEEE.

  On utilisera plutôt le test `if (IS_ZERO(10*b-a))` ....'

  De manière générale, les tests d'égalité sur les réels sont à proscrire.

- `<g2x_geom.h>` : contient les définitions des types `G2Xpoint` et `G2Xvector`, les deux entités élémentaires de la géométrie, ainsi que de nombreuses fonctions associées à ces deux types : constructeurs, opérateurs scalaires et vectoriels, fonctionalités de normes et distances ...

  Les 2 types sont identiques, mais géométriquement les deux objets sont bien différents. Il est donc utile de pouvoir les distinguer.

  Ce même type possède un 3° nom (`G2Xcoord`) qui peut être utilisé lorsque l'objet peut prendre les deux formes (point et/ou vecteur).

  ```
  typedef struct { double x,y; } G2Xcoord, G2Xpoint, G2Xvector;
  ```

- `<g2x_tools.h>` : contient quelques outils périphériques comme des générateurs aléatoires (dans un intervalle réel, ou centré sur une valeur) et quelques outils de msesure du temps (`G2Xclock`).

- `<g2x_colors.h>`: contient les types, constantes et fonctions associées à la gestion des couleurs. Les couleurs sont définies par défaut en mode `RGBA` (structure de 4 `float`, dans l'intervalle `[0,1]`, pour les composantes `Rouge`, `Vert`, `Bleu` et le canal de 'transparence' `Alpha`)

  ```
  typedef struct { float r,g,b,a; } G2Xcolor;
  ```

  attention : contrairement à de nombreux systèmes (`OpenGl, SDL` ...), le champ `a` (composante `alpha`) représente bien la **transparence** et non l'opacité : la couleur est pleinement visible pour `a=0.` et disparaît pour `a=1.`

  Ce module propose également quelques fonctions de conversion / représentation des couleurs en mode `HSVA (Hue, Saturation, Value, Alpha)` ainsi que des fonctions de création de cartes de couleurs de type "arc-en-ciel" (dégradé de teintes)

  Enfin il propose de nombreuses couleurs prédéfinies sous forme de macros, telles que `G2Xr`, `G2Xo`, `G2Xy`, `G2Xg` .... pour les couleurs rouge, orange, jaune, vert....

- `<g2x_control.h>`: contient l'ensemble des objets et fonctionalités de contrôle associées à l'interface (`button`, `switch`, `popup`, `scrollbar`), les fonctionalités de gestion des interruptions (clavier, souris) ainsi que la gestion des points de contrôle (`G2Xctrlpt` : point "cliquables", déplaçable en "drag & drop").

- `<g2x_window.h>` : contient les fonctions de gestion de la fenêtre `g2x_InitWindow(), g2x_SetWindowCoord()`, les 6 *handlers* de fonctions spécifiques `g2x_Set***Function(void (*f)(void))` et la fonction de démarrage `g2x_MainStart()`

  On y trouve également les fonctions `g2x_Set*` et `g2x_Get*` d'attribution/récupération de nombreuses variables d'environnement, des fonctions d'affichage formaté de chaînes de caratères, des fonctions de tracé d'axes et grilles de graduation, ainsi que diverses fonctions utilitaires, plus ou moins marginales

Les "haut-niveau"

- `<g2x_draw.h>` : contient les fonctions de dessin de quelques primitives simples telles que points, droites, cercle & ellipses, triangles, rectangles, quadrilatères quelconques...
- `<g2x_transfo.h>` : contient les fonctions de transformations géométriques en coordonnées homogènes 2D (translation, homothétie, rotation), le type `G2Xhmat` (matrice 3x3) associé et les opérateurs matriciels simples (produits `Matrice`×`Point`, `Matrice`×`Vecteur` et `Matrice`×`Matrice`)
- `<g2x_geoalgo.h>` : ce module, beaucoup plus évolué, contient des fonctions réalisant des algorithmes simples mais d'usage assez courant sur les primitives géométriques (intersction de segments, de cercles)
- `<g2x_polygon.h>` : un module de gestion de polygône quelconque à base de listes doublement chaînées circulaires.

Les périphériques

- `<g2x_pixmap.h>` : un module permettant de manipuler des images au format brut `PNM`[1].

  L'utilisation de ce module (lecture/écriture d'images) utilise la boîte à outils `Netpbm` (spécifique `linux`).

  Supporte les formats `pnm` | `bmp` | `gif` | `jpeg` | `png` | `tiff`.
- `<g2x_capture.h>` : module pour les captures d'écran, les vidéo. Ces éléments sont tous activés automatiquement, il n'y a donc aucune raison de l'utiliser.

  Ce module utilise la suite Netpbm pour l'image et `mencoder` pour la vidéo (formats `mp4` | `mkv` | `flv`)

[1] PNM: Portable aNy Map