Manuel de la bibliothèque GfxLib

Présentation liminaire

La bibliothèque *GfxLib* a été conçue dans le but de mettre à disposition rapidement et simplement une architecture graphique portable permettant de faire du dessin 2D, tout en restant ouverte au dessin 3D si le système supporte OpenGL.

Ainsi, *GfxLib* est basée sur *OpenGL/GLUT*. Le choix de cette architecture vient de raisons multiples :

- *OpenGL est disponible, sous forme logicielle ou matérielle, sur la plupart des plateformes informatiques existantes ;*
- *OpenGL utilise un monde de représentation basé sur un repère orienté de manière classique (l'écran est le premier quadrant, dont l'origine se trouve en bas à gauche), et pour lequel les coordonnées sont des nombres en virgule flottante ;*
- *OpenGL/GLUT permet de créer des programmes simples et portables : GfxLib a ainsi été testée sur plusieurs distributions Linux, sous Windows 2000/XP/Vista/7/8/10, ainsi que sous macOS ;*
- *l'API OpenGL permet de tracer simplement des primitives 2D, mais en plus permet l'insertion aisée de primitives 3D : la transition du 2D vers la 3D devrait être simple si jamais il est besoin d'avoir des représentations tridimensionnelles complexes.*

GfxLib gère non seulement les graphiques vectoriels classiques, mais :

- *gère aussi l'interaction avec le clavier ;*
- *gère aussi l'interaction avec la souris ;*
- *permet le dessin bitmap avec ou sans gestion de la transparence ;*
- *permet de garantir le nombre d'images par seconde d'une animation de manière relativement simple ;*
- *gère le redimensionnement de la fenêtre de travail et propose de plus un mode plein écran ;*
- *permet d'optimiser en performance les affichages graphiques à l'aide de listes d'affichage ;*
- *facilite le dessin en 3D.*

Le document ci-présent explique rapidement la philosophie de fonctionnement d'une grande partie de *GfxLib*. Le lecteur est ensuite invité à parcourir le code source de la bibliothèque pour en apprendre plus sur ses possibilités, ainsi que de tester les exemples de code fournis avec elle.

Fonctionnement interne

GLUT fonctionne principalement à l'aide du système de *callbacks* (appel différé de fonction). Bien que *GfxLib* tire parti de cette caractéristique et permette de l'exploiter au mieux, *GfxLib* ajoute cependant une couche supplémentaire simplifiant énormément la gestion des événements. Elle utilise ainsi une fonction unique de gestion des événements, qui reçoit une variable de type *EvenementGfx*, décrivant l'événement à traiter (et qui doit avoir le prototype suivant : *void gestionEvenement()*).

Ainsi, à l'aide d'un simple *switch-case*, il est possible de réagir aux événements suivants :

- *Inactivite : ce message est envoyé lorsqu'aucun autre message n'est disponible ; comme il remplit rapidement la file des messages d'une application, il est désactivé par défaut et pour le recevoir, il faut appeler la fonction activeGestionInactivite() ;*
- *Temporisation : ce message peut être envoyé régulièrement au programme, afin de permettre la réalisation répétitive de tâches qui nécessitent un intervalle de temps relativement précis entre elles (réglable à la micro-seconde, mais avec une précision qui peut varier selon le système) ;*
- *Affichage : une demande implicite ou explicite de réaffichage de l'écran a été demandée, et il faut redessiner le contenu de la fenêtre ;*
- *Clavier : une touche clavier a été enfoncée, le caractère correspondant à cette touche est récupéré à l'aide de la fonction caractereClavier(), et l'état des modificateurs Shift, Ctrl et Alt peuvent être scannés à l'aide de toucheShiftAppuyee(), toucheCtrlAppuyee() et toucheAltAppuyee() ;*
- *ClavierSpecial : similaire à l'événement précédent, hormis que cette fois-ci ce sont uniquement les touches F1 à F12 qui sont prises en compte, ainsi que les touches fléchées du clavier ; la fonction toucheClavier() renvoie un entier de type TouchesSpeciales qui contient l'identifiant de la touche appuyée ;*
- *Souris : la souris a été déplacée, on récupère ses coordonnées à l'aide des fonctions abscisseSouris() et ordonneeSouris() ; notez que par défaut, les déplacements de la souris ne sont suivis que lorsqu'un bouton est appuyé ; pour suivre tout le temps la souris, il faut activer cette possibilité (qui remplit rapidement la file des messages) en appelant la fonction activeGestionDeplacementPassifSouris() ;*
- *BoutonSouris : un bouton de la souris a été actionné, on identifie lequel à l'aide de etatBoutonSouris() qui peut renvoyer GaucheAppuye, GaucheRelache, DroiteAppuye ou DroiteRelache ; abscisseSouris() et ordonneeSouris() sont aussi bien évidemment utilisables pour répondre à ce message ;*
- *Initialisation : ce message est envoyé avant l'arrivée de tout autre message, il permet d'initialiser certaines données avant de commencer quelque action que ce soit (note : il est conseillé d'utiliser les variables static, afin d'avoir tous les avantages des variables globales sans en avoir les inconvénients) ;*
- *Redimensionnement : ce message est envoyé lorsque la fenêtre est redimensionnée par l'utilisateur, ou suite à l'appel de redimensionneFenetre() ou de*

modePleinEcran() ; la nouvelle taille de fenêtre peut alors être connue grâce à largeurFenetre() et hauteurFenetre().

On initialise tous ces services en appelant *prepareFenetreGraphique()* qui définit le nom et la taille de la fenêtre, puis on lance la boucle infinie de gestion des événements à l'aide de *lanceBoucleEvenements()*. Ces deux actions doivent impérativement se trouver après la fonction *initialiseGfx()*, et idéalement toutes trois devraient être placées dans la fonction *main* du programme.

Petite entorse au formalisme *OpenGL* (qui gère les couleurs par leurs niveaux de rouge, de vert et de bleu de 0.f à1.f), ce sont des niveaux entiers de 0 à 255 qui sont utilisés par la *GfxLib* pour décrire une couleur, pour des raisons de cohérence avec le format de stockage des couleurs dans les fichiers *BMP*, etc. Bien entendu, les fonctions classiques *OpenGL* conservent leur fonctionnement original, même lorsqu'elle sont utilisées en conjonction avec GfxLib.

Ainsi, *effaceFenetre()* et *couleurCourante()* respectivement efface la fenêtre et fixe la couleur du pinceau courant avec les trois valeurs entières qui doivent leur être passées en paramètre (rouge, puis vert puis bleu).

Le dessin vectoriel sous GfxLib

epaisseurDeTrait() fixe la taille (en pixels de la fenêtre) du pinceau qui va dessiner tout ce qui est filaire. Elle aura de l'influence sur *point()* et *ligne()*, mais aussi sur *afficheChaine()*.

triangle() et *rectangle()* complètent la liste des primitives graphiques de base.

Mentionnons aussi *tailleChaine()* qui renvoie un flottant donnant la longueur « graphique » de la chaîne de caractères une fois affichée à l'écran.

rafraichisFenetre() envoie enfin une demande explicite de re-dessin de la fenêtre, ce qui se traduit par l'envoi du message *Affichage* à la fonction *gestionEvenement()*. C'est utile lorsque, en réponse à un message quelconque, on a modifié l'état du dessin et qu'on veut qu'il soit mis effectivement à jour dans la fenêtre.

Le dessin bitmap sous GfxLib

GfxLib permet d'afficher directement dans la fenêtre une image de type BVR (bleu-vert-rouge) comme celles fournies par BMPLib. Cela se fait à l'aide de la fonction *ecrisImage()*.

Son homologue *lisImage()* permet au contraire de lire une portion de la fenêtre et de la sauvegarder dans une image de type BVR. Des pixels isolés de la fenêtre peuvent aussi être lus grâce à *lisPixel()*, et les composantes rouge, vert et bleu peuvent être extraites de l'entier qu'elle renvoie par *rougeDuPixel()*, *vertDuPixel()* et *bleuDuPixel()*.

La fonction *ecrisImageARVB()* attend quant à elle des données des données au format Alpha-Rouge-Vert-Bleu. Le « canal Alpha » permet de fixer l'opacité de la couleur rouge-vert-bleu spécifiée pour chaque pixel, et ainsi de pouvoir afficher des images avec des effets de transparence.

L’animation/l'automatisation sous GfxLib

L’animation/l'automatisation peut se faire de deux manières différentes sous *GfxLib*, chacune aboutissant à un résultat original.

demandeRedessinDans_ms() demande au système d’exploitation d’envoyer un message *Affichage* après que ce soient écoulés au moins le nombre de millisecondes spécifié. Il faut répéter l’appel de cette fonction si nécessaire.

demandeTemporisation() demande au système d’exploitation un envoi régulier de messages *Temporisation* (en spécifiant le nombre de millisecondes minimum entre chaque message). Ceci permet d'effectuer des calculs à intervalles réguliers, et éventuellement, de lancer un rafraîchissement régulier de l'affichage si on utilise *rafraichisFenetre()*.