Manuel de la bibliothèque isentlib 2.0

Présentation

La bibliothèque isentlib est une bibliothèque dont le but est de promouvoir et de faciliter l'utilisation de diverses technologies informatiques, tout en offrant un accès simplifié à ces technologies au maximum d'étudiants possible. Isentlib est basée sur les technologies suivantes :

- bibliothèque C standard ;
- OpenGL pour le graphisme ;
- pthread et threads Win32 pour le parallélisme d'exécution ;
- SSE pour le calcul vectoriel ;
- socket et WinSocket pour la programmation réseau.

La bibliothèque « encapsule » ces technologies complexes afin qu'elles puissent être utilisées avec le minimum de connaissances informatiques possible. Isentlib peut ainsi être utilisée de deux manières :

- soit en tant que telle, en profitant des facilités qu'elle offre pour construire rapidement un programme basé sur une interface graphique, sur des threads, de la communication réseau, etc. ;
- soit de manière didactique, car le code source est livré : il est ainsi possible à tout un chacun de comprendre comment la bibliothèque fonctionne, afin de s'en inspirer ou bien de personnaliser le comportement de l'application en ajoutant ses propres codes OpenGL, en mélangeant des fonctions de la bibliothèque isentlib avec d'autres fonctions, etc.

Contenu de la bibliothèque

La bibliothèque isentlib est en fait constituée d'une multitude de petites bibliothèques, le plus indépendantes possible les unes des autres. Il est ainsi possible de « piocher » dans isentlib afin d'en retirer juste ce qui est nécessaire, et rien de plus.

isentlib comprend donc :

- BmpLib, déclarée dans BmpLib.h et utilisant OutilsLib ; cette bibliothèque permet de lire et écrire facilement des images bitmap au format BMP 24 bits non-compressée ;
- ESLib, déclarée dans ESLib.h et utilisant ErreurLib ; cette bibliothèque permet de lire au clavier et écrire sur l'écran des données texte classiques, sans avoir à comprendre les commandes C classiques mais complexes que sont printf et scanf ; ESLib est tout particulièrement ciblée pour les programmes des classes préparatoires ; ESLib propose aussi quelques fonctions supplémentaires liées au tirage de valeurs aléatoires et au temps ; elle est décrite plus en détails ci-après ;

- GfxLib, déclarée dans GfxLib.h ; elle facilite la création d'une interface graphique, propose la gestion du clavier, de la souris, de l'animation, etc. ; cette bibliothèque imposante est décrite plus en détails dans le « Manuel de la bibliothèque GfxLib » ci-après ;

- SocketLib, déclarée dans SocketLib.h ; cette bibliothèque permet d'établir facilement la communication par l'intermédiaire du réseau Internet entre deux machines distantes, ou même entre deux processus sur une même machine ;

- ThreadLib, déclarée dans ThreadLib.h ; cette bibliothèque permet de lancer des threads et propose aussi des mécanismes de synchronisation pour faciliter la programmation parallèle ;

- VectorLib, déclarée dans VectorLib.h ; cette bibliothèque offre quelques fonctions de calcul simples mais utilisant l'unité vectorielle du CPU, afin d'accélérer la vitesse de traitement des informations.

Exploitation des codes sources

isentlib est en général fournie sous la forme d'une bibliothèque statique pour Linux et Windows, mais les codes sources sont aussi mis à disposition.

Ces codes sources sont un support pédagogique commenté permettant au lecteur de comprendre et d'apprendre le fonctionnement de diverses technologies standard (comme OpenGL, les socket, etc.). Ils sont aussi le moyen le plus simple pour permettre à tout un chacun de personnaliser son application, voire de se faire sa propre bibliothèque, mieux taillée pour ses besoins particuliers.

Conclusion

Ci-après dans ce document suivent des descriptions plus détaillées des diverses bibliothèques composant isentlib. À tout un chacun d'en faire bon usage.

Manuel de la bibliothèque ESLib

Présentation liminaire

La bibliothèque *ESLib* a été conçue dans le but de mettre à disposition des étudiants, dès la première année du cycle ISEN, des moyens permettant de gérer les actions courantes d'interaction avec l'utilisateur, et ce sans entrer dans la complexité de mise en œuvre du langage C.

Ainsi *ESLib*, bien qu'étant basée sur les entrées/sorties standard, permet à des débutants en langage C de demander à l'utilisateur des entrées de données, et permet d'afficher en mode texte nombre de données, avec un formalisme très simple.

Fonctionnement interne

ESLib s'attache à respecter la plus grande symétrie possible entre les entrées et les sorties de données. Ainsi il est possible de lire au clavier les données suivantes :

• un entier par *int lisEntier()* ;

• un nombre en virgule flottante par *float lisFlottant()* ;

• un caractère par *char lisCaractere()*.

Les opérations d'écriture correspondantes sont :

• *ecrisEntier(int valeur)* ;

• *ecrisFlottant(float valeur)* (le nombre de chiffres à afficher après la virgule peut être modifié à l'aide de *fixePrecision(int n)*) ;

• *ecrisCaractere(char valeur)*.

*ecrisChaine(char *chaine)* peut aussi être utilisée pour afficher une chaîne de caractères complète. *tabulation()* et *sautDeLigne()* insèrent respectivement une tabulation et un saut de ligne dans le flot courant des données.

Ces opérations sont basiques et incomplètes, car elles ne doivent être utilisées que par commodité durant les premiers temps de la formation au langage C.

Pour aller plus loin

effaceEcran(), pour des raisons de compatibilité entre systèmes du code source de la bibliothèque, affiche une série de sauts de ligne, faisant croire à un effacement de la fenêtre texte.

double tempsCPU() renvoie le nombre de seconde écoulé relativement au programme qui a été lancé.

float valeurAleatoire() renvoie une valeur aléatoire comprise entre 0 inclus et 1 exclu. Le générateur pseudo-aléatoire peut être initialisé à l'aide de *initialiseValeurAleatoire(float valeur)*.

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.

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, ainsi que sous MacOSX ;*
- *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*. Bien que *GfxLib* tire parti de cette caractéristique et permette de l'exploiter au mieux, elle 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() ;*
- *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 impérati-

vement se trouver après la fonction *initialiseGLUT()*, 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 vectorisé 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.

rafraichitFenetre() 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.

effectueAffichage() force l'envoi des commandes graphiques au driver *OpenGL*.

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 sous GfxLib

L’animation peut se faire de deux manières différentes sous *GfxLib*, chacune aboutissant au même résultat : un message *Affichage* sera envoyé à la fonction *gestionEvenement()* à un horaire bien précis.

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.

demandeAnimation_ips() demande au système d’exploitation un envoi régulier de messages *Affichage* afin de maintenir le nombre d’images par seconde (ips) spécifié. Notez que pour maintenir cette cadence en moyenne, et si le programme a subi un ralentissement (système surchargé par exemple), *GfxLib* peut être amené à afficher plus d’images par seconde pendant un instant limité, afin de « rattraper » les images perdues.

Pour aller plus loin

Plutôt que de perpétuellement envoyer des ordres à la carte graphique, il existe sous *OpenGL* la notion de liste graphique. Les ordres de dessin, si ceux-ci ne changent pas, sont compilés et stockés dans la mémoire de la carte graphique, qui peut ainsi, lorsqu’on lui demande d’exécuter une liste, ressortir toutes ces informations de manière bien plus performante. *GfxLib* permet de faire des listes de manière très simple, en appelant *demarreListeGraphique()* qui renvoie le numéro de la liste démarrée. *termineListeGraphique()* termine la liste démarrée, qui peut ensuite être exécutée d’un coup par *afficheListeGraphique()*.

Nous avons dit que *Glut* utilisait des *callbacks*, et que *GfxLib* était venu se greffer sur ce système. *GfxLib* utilise en fait les *callback* de *Glut*, structure l’information qui en découle et appelle *gestionEvenement()*. Cependant, il est possible d’utiliser les fonctions *callback Glut*. *GfxLib* simplifie ce procédé à l’aide de *modifieFonctionAffichage(), modifieFonctionInactivite(), modifieFonctionClavier(), modifieFonctionClavierSpecial(), modifieFonctionSouris() et modifieFonctionDeplacementSouris()*. Il faut passer à ces fonctions la fonction *callback* de prototype convenable, et :

- soit s’assurer que la gestion du message n’est plus faite dans la fonction *gestionEvenement()* ;
- soit utiliser la *callback* pour effectuer un travail supplémentaire, mais ajouter l’appel explicite de la fonction *gestionEvenement()* avec l’événement correspondant comme paramètre.

La 3D sous GfxLib

Il est possible de mêler du code *OpenGL* pur au code écrit pour *GfxLib*. Cependant, *GfxLib* propose quand même quelques primitives simplifiant l’utilisation de la 3D. Le lecteur est invité à lire les exemples en 3D fournis avec la bibliothèque, ainsi que *GfxLib.h* pour en savoir plus sur ce support.