TP OpenGL 1
Introduction à OpenGL et GLUT

Licence Informatique 3ème année

Année 2016-2017

1 Introduction

OpenGL est une librairie graphique, comportant environ 120 fonctions distinctes et permettant la création et la manipulation d'objets tri-dimensionnels en vue de leur affichage au sein d'applications interactives. L'un des objectifs des concepteurs de cette librairie a été de la définir indépendamment de toute plateforme. Ceci implique que rien n'est prévu dans cette librairie pour gérer les aspects haut niveau liés à l'interaction avec l'utilisateur (multi-fenêtrage, gestion des événements, ...). De même, par souci de simplicité et « d'universalité », OpenGL ne fournit que des primitives graphiques de très bas niveau pour construire les objets à afficher et à manipuler. À charge à l'utilisateur ou à d'autres bibliothèques spécialisées de construire des objets plus complexes à l'aide de ces primitives.

OpenGL doit être vu comme une « machine à état » : la librairie possède un certain nombre de variables d'état, réunies au sein d'un *contexte graphique*. Chaque opération graphique s'effectuera en tenant compte des valeurs présentes dans ce contexte. À titre d'exemple, la couleur ou l'épaisseur d'un trait représentent chacune une variable d'état ; tous les affichages se feront dans la couleur et l'épaisseur actuelles, jusqu'à ce que le programmme change explicitement ces valeurs.

OpenGL étant indépendant de tout système d'interface homme-machine, le programmeur a à sa charge la gestion du multi-fenêtrage et des événements. Sur les systèmes Unix, cela implique qu'il a à développer l'interface de son application sous XWindow [1] qui représente la couche logicielle « standard » gérant le multi-fenêtrage, les événements, etc ... Mais si XWindow présente une richesse énorme, il en résulte une complexité très importante de programmation qui focalise davantage le programmeur sur les aspects interface plutôt que sur le centre du problème, c'est-à-dire le développement des aspects graphiques. Pour pallier ce problème, l'API [2] GLUT (OpenGL Graphics Utility) a été développée en vue de fournir, avec un très petit nombre de fonctions, les moyens de gérer facilement les aspects « fenêtrage » , gestion d'événements, gestion de menus dynamiques etc ...

Ce TP a donc pour buts de présenter en parallèle les fonctionnalités de GLUT pour la gestion des événements et une introduction à OpenGL pour les aspects purement graphiques, sachant que nous nous limiterons au 2D dans un premier temps.

2 Un premier exemple

Tapez l'exemple qui suit dans un fichier nommé `tp1.c` et lisez attentivement les explications qui le suivent, sans chercher à compiler cette application pour le moment.

```
01 #include <GL/gl.h>
02 #include <GL/glu.h>
03 #include <glut.h>
04
05 void dessiner(void)
06 {
07   return;
08 }
09
10 int main (int argc, char *argv[])
11 {
```

1. à ne surtout pas confondre avec Windows ...
2. application programming interface

```
12   /* initialiser glut */
13   glutInit (&argc, argv);
14
15   /* creer la fenetre */
16   glutCreateWindow(argv[0]);
17
18   /* choix de la fonction de rafraichissement */
19   glutDisplayFunc(dessiner);
20
21  /* demarrer la boucle evenementielle */
22   glutMainLoop();
23 }
```

2.1 Quelques explications

Les appels de fonctions qui apparaissent dans la fonction `main` représentent le minimum nécessaire à l'utilisation de fenêtres par l'intermédiaire de `glut`. Ces 4 fonctions doivent donc toujours être présentes et apparaître dans cet ordre. Leur effet respectif est détaillé ci-après.

2.1.1 glutInit

Cette fonction, appelée en ligne 13, est chargée d'initialiser la librairie `glut` et, en particulier, d'effectuer une connexion avec le système de fenêtrage. Sa syntaxe est la suivante :

```
void glutInit(int *argc, char *argv[]);
```

Comme leurs noms l'indiquent, les paramètres sont les mêmes que ceux de la fonction `main`. Notez cependant que la fonction attend **l'adresse** de la variable mémorisant le nombre d'arguments figurant sur la ligne de commande. Cette fonction doit toujours être la première fonction `glut` appelée.

2.1.2 glutCreateWindow

Cette fonction, appelée en ligne 16, permet de créer « physiquement » la fenêtre. Sa syntaxe est la suivante :

```
int glutCreateWindow(char *nom);
```

Le paramètre `nom` permet de spécifier le texte qui apparaîtra dans la barre de titre de la fenêtre. La valeur de retour est un entier qui permet d'identifier la fenêtre créée, en cas de création de plusieurs fenêtres. Notez que la numérotation des fenêtres commence à 1 et que chaque fenêtre créée possède son propre contexte graphique OpenGL.

2.1.3 glutDisplayFunc

Cette fonction permet de spécifier à `glut` le nom de la fonction à appeler lorsqu'un affichage est jugé nécessaire (par exemple lorsque la fenêtre est ouverte, lorsqu'elle redevient visible, ...). Sa syntaxe est la suivante :

```
void glutDisplayFunc(void (*func)(void));
```

Son paramètre `func` est donc un pointeur vers la fonction à appeler. Notez que cette fonction ne doit retourner aucun résultat et ne prendre aucun paramètre. La ligne 19 montre un exemple d'utilisation de cette fonction, la fonction appelée (`dessiner`) n'ayant pour le moment aucun effet.

2.1.4 glutMainLoop

Dernière fonction à être appelée, `glutMainLoop` permet à votre programme d'entrer en phase événementielle, cette fonction tournant sous forme de boucle infinie et récupérant les différents événements qui peuvent se produire. En fonction du type d'un événement, cette fonction appelera une fonction apte à le traiter (sous réserve qu'elle ait été définie auparavant ...). Sa syntaxe est la suivante :

```
void glutMainLoop(void);
```

Notez que le code qui vous écrivez **après** cette fonction ne sera jamais exécuté ...

2.2 Compilation

Avant de pouvoir compiler ce programme, il est nécessaire de préciser un certain nombre d'options de compilation supplémentaires :

- l'option `-I` : elle permet de préciser au compilateur des répertoires supplémentaires pour la recherche des fichiers bibliothèque (fichiers `.h` inclus dans le source C). En effet le compilateur recherche ce type de fichier, par défaut dans le répertoire `/usr/include`. Mais les bibliothèques non standards pour le compilateur (OpenGL par exemple) peuvent avoir été installées dans un autre répertoire, a priori inconnu du compilateur. Il faut alors préciser au compilateur l'emplacement supplémentaire où il est susceptible de trouver des bibliothèques additionnelles avec l'option `-I`, suivie du chemin d'accès à celles-ci ;
- l'option `-L` : elle permet de préciser au compilateur des répertoires supplémentaires pour la recherche des librairies [3] dans lesquelles trouver certaines fonctions non standards utilisées par le programme. Le répertoire de recherche par défaut est `/usr/lib`. L'option `-L` suivie du chemin d'accès aux librairies non standards à utiliser permet ainsi d'élargir les capacités de recherche du compilateur.
- l'option `-l` : elle permet de préciser le nom des librairies qui doivent effectivement être utilisées. Un nom de librairie commencant toujours par `lib`, ces 3 caractères sont omis et le `-l` est uniquement suivi par les caractères figurant après `lib`. Par exemple, la librairie contenant les fonctions de `glut` porte le nom `libglut`. On utilisera alors l'option `-l` sous la forme `-lglut`.

Ainsi sur les PC Linux de l'Université, la commande de compilation complète pour le fichier `tp1.c` est la suivante :

```
gcc -o tp1 tp1.c -I /usr/include/GL  -lglut -lGLU -lGL  -lm
```

Exécutez cette commande et lancez le programme `tp1`. Notez les points suivants :

- le contenu de la fenêtre est initialisé avec ce qui se trouve « dessous » au moment où la fenêtre est ouverte ;
- les boutons « classiques » liés aux fenêtres sont disponibles et utilisables (icônifier, plein-écran, fermeture) ;
- le nom de la fenêtre est bien celui de la commande.

2.3 Paramètrage de la fenêtre

`Glut` offre d'autres fonctions permettant de configurer la fenêtre avant sa création. Deux de ces fonctions sont décrites ci-après. Notez bien qu'elles doivent être appelées **avant** la création de la fenêtre.

- `glutInitWindowSize` : cette fonction permet de spécifier la taille (en pixels) de la fenêtre lorsqu'elle est ouverte. Sa syntaxe est la suivante :

  ```
  void glutInitWindowSize(int largeur, int hauteur);
  ```

- `glutInitWindowPosition` : cette fonction permet de spécifier les coordonnées écran à partir desquelles la fenêtre doit être ouverte. Sa syntaxe est la suivante :

  ```
  void glutInitWindowPosition(int x, int y);
  ```

  Les paramètres `x` et `y` représentent les coordonnées écran (en termes de pixels) du coin supérieur gauche de la fenêtre.

Application : Modifier le source de `tp1.c` de manière à ce que la fenêtre créée soit de taille 256×256 et s'ouvre à la position $(100, 100)$.

---

3. On appelle librairie un fichier dans lequel se trouvent des fonctions **déjà compilées**, utilisables par d'autres programmes. On distingue les librairies statiques (le code des fonctions à utiliser est incorporé dans le code du programme qui les utilise, au moment de la compilation) des librairies dynamiques (le code d'une fonction n'est pas incorporé dans l'exécutable ; il est chargé en mémoire depuis la librairie uniquement au moment de son exécution, ce qui permet de réduire la taille des fichiers exécutables sur disque).

2.4 Premiers affichages OpenGL

2.4.1 glClearColor

Cette fonction permet de spécifier la couleur qui doit être utilisée pour effacer la fenêtre. Sa syntaxe est la suivante :

```
void glClearColor(GLclampf rouge, GLclampf vert,
                  GLclampf bleu, GLclampf alpha);
```

Les quatres paramètres représentent respectivement les niveaux de rouge, vert, bleu et « alpha » utilisés pour représenter la couleur. Leurs valeurs (flottantes [4]) doivent être comprises entre 0 et 1. Notez bien que cette commande **n'efface pas la fenêtre** ; elle change simplement la valeur de la couleur « de fond » dans le contexte graphique d'OpenGL.

2.4.2 glClear

Cette fonction permet de déclencher l'effacement des différents « *buffers* » utilisés par OpenGL. Parmi ceux-ci, le « *Frame buffer* » , que l'on peut traduire par « mémoire image » est le buffer dans lequel toutes les fonctions d'affichage écrivent. La fonction `glClear` permet d'effacer simultanément un ou plusieurs des buffers d'OpenGL, en utilisant la valeur de réinitialisation récupérée dans le contexte graphique d'OpenGL. Sa syntaxe est la suivante :

```
void glClear(GLbitfield mask);
```

Le paramètre `mask` permet d'indiquer les buffers à réinitialiser : chaque bit de `mask` correspond à un buffer particulier ; lorsque ce bit est à 1, cela signifie que le buffer correspondant doit être réinitialisé, sinon il n'est pas modifié.

OpenGL définit des constantes associées à chaque buffer, la valeur de chaque constante correspondant à un bit particulier. La valeur de `mask` peut alors être obtenue par un *ou-logique* entre les constantes associées à chacun des buffers que l'on souhaite réinitialiser. Pour le moment, seule la constante associée au « *Frame buffer* » nous intéresse ; elle est notée `GL_COLOR_BUFFER_BIT` (soit « *le bit associé à la mémoire d'images d'OpenGL* »). Un appel de la forme :

```
glClear(GL_COLOR_BUFFER_BIT);
```

aura alors pour effet de réinitialiser l'image avec la couleur d'effacement définie par le précédent appel de la fonction `glClearColor`.

2.4.3 glFlush

Pour diverses raisons assez complexes à évoquer pour le moment, l'exécution d'ordres d'affichage peut être différée. L'effet en est que l'image semble ne pas être complète (il manque des figures par exemple) sans que le programme lui-même ne soit en cause [5]. La fonction `glFlush` permet de forcer l'exécution des instructions graphiques éventuellement en attente. Sa syntaxe est la suivante :

```
void glFlush(void);
```

Il est conseillé d'effectuer un appel à cette fonction dès que l'ensemble des fonctions utilisées pour générer une nouvelle image a été appelé. Eviter cependant (sauf cas particuliers) de l'appeler après chaque instruction graphique, car elle aurait pour effet de ralentir certains programmes...

4. Pour des raisons de portabilité, OpenGL redéfinit ses propres types de données. Le type `GLclampf` est ici redéfini sous la forme d'un `float` (voir le fichier `gl.h` à ce sujet).

5. Par analogie, on peut comparer cela au problème de la « bufferisation » des sorties texte sous Unix : lors de l'exécution d'un `printf`, l'affichage peut être différé, ce qui donne l'impression que le programme n'effectue pas cette instruction. Pour forcer l'affichage, on vous conseille alors soit d'insérer un `'\n'` à la fin de votre format d'affichage, soit de forcer l'affichage avec la fonction `fflush()`.

Application : Modifiez la fonction `dessiner` pour qu'elle effectue, lors de son appel, un effacement de la fenêtre avec la couleur de votre choix.

3 Paramétrisation d'un Makefile

Avant de poursuivre plus avant cette présentation d'OpenGL et de Glut, nous allons nous intéresser à la façon d'écrire un fichier `Makefile` plus ergonomique et plus simple à gérer.

3.1 Création du Makefile

Dans un premier temps, effectuez les opérations suivantes :
— mettre `dessiner` dans un fichier nommé `graphique.c`
— écrire le fichier `graphique.h`
— mettre à jour le fichier `tp1.c`
Editez ensuite le fichier `Makefile` suivant, en respectant les règles de syntaxe données ci-après :
— le symbole ' :' doit être suivi d'une tabulation ;
— chaque commande de compilation doit être précédée d'au moins une tabulation ;
— une ligne trop longue peu être écrite sur deux lignes la première étant terminée par le caractère '\' .

```
tp1:        tp1.o graphique.o
        gcc -o tp1 tp1.o graphique.o -lglut -lGLU -lGL -lm

tp1.o:      tp1.c
        gcc -c tp1.c -I /usr/include/GL

graphique.o:    graphique.c
        gcc -c graphique.c -I /usr/include/GL
```

Vous remarquerez que les options de compilation supplémentaires décrites ci-avant n'apparaissent pas systématiquement pour chaque commande de compilation. En effet, les librairies à utiliser (options `-L` et `-lnom`) ne doivent être précisées qu'au moment de l'édition de liens (réunion des fichiers `.o` en un fichier exécutable). De la même manière, l'emplacement des fichiers additionnels à inclure ne doit être précisé qu'au moment de la première phase de compilation (génération des fichiers `.o`).

Tester ce nouveau `Makefile` en recompilant votre application.

3.2 Utilisation des variables du Makefile

L'utilitaire `make` admet la définition de variables simples dans le fichier `Makefile`. La syntaxe de cette définition est la suivante :

```
NOM= chaine de caractères quelconques
```

Lorsque l'on souhaite utiliser la valeur de cette variable, la syntaxe est la suivante :

```
$(NOM)
```

D'un point de vue pratique, lorsque l'utilitaire `make` rencontre `$(NOM)`, il le remplace par la valeur de la variable `NOM`. On peut alors réécrire le fichier `Makefile` qui nous intéresse sous la forme suivante :

```
# définition des dossiers d'inclusion des bibliothèques .h
INCLUDE= -I /usr/include/GL

# définition des librairies à utiliser lors de l'étape d'édition de liens
LIB=  -lglut -lGLU -lGL  -lm

# définition du compilateur à utiliser
CC=gcc
```

```
# définition des options de compilation à utiliser
# -g : en vue d'autoriser l'utilisation du débugger
# -Wall : afficher le maximum de messages d'erreurs
OPTION= -g -Wall

tp1:        tp1.o graphique.o
        $(CC) -o tp1 tp1.o graphique.o   $(LIB)

tp1.o:      tp1.c
        $(CC) -c tp1.c  $(INCLUDE) $(OPTION)

graphique.o:    graphique.c
        $(CC) -c graphique.c  $(INCLUDE) $(OPTION)

# cible d'effacement des fichiers objets et de l'exécutable

clean:
        rm -r *.o  tp1
```

Le premier avantage de cette formulation est d'offrir une plus grande facilité de maintenance du fichier `Makefile` :

- une modification du chemin d'accès aux fichiers à inclure ou des librairies à utiliser n'est effectuée qu'en un seul endroit ;
- le rajout d'un fichier dans le cadre du projet est simplifié.

Enfin elle offre une plus grande lisibilité du contenu du fichier.

Modifier votre fichier `Makefile` comme indiqué ci-dessus, effacer les fichier `.o` **uniquement** (!!!), puis recompiler votre application.

Remarque : Une *cible* particulière a été ajoutée à votre fichier `Makefile`. Il s'agit de la cible `clean`. Son objectif est de permettre l'effacement de tous les fichiers objet générés lors de la compilation. Elle est déclenchée par l'appel de la commande suivante :

```
make clean
```

4 Tracés 2D

4.1 Repère écran

Avant de pouvoir tracer quoi que ce soit, il est nécessaire de connaître les limites de l'écran associées à votre fenêtre. Par défaut, OpenGL considère que le centre du repère image se trouve au centre de la fenêtre et que les coordonnées x et y des points qui s'y trouvent appartiennent à $[-1, +1] \times [-1, +1]$. Vous manipulerez donc des coordonnées **réelles**. Notez qu'en cas de dépassement de ces intervalles de définition, OpenGL réalise un « clipping » automatique des valeurs extérieures.

4.2 Spécification d'une primitive géométrique

Avant de pouvoir tracer un « objet » , il est nécessaire de définir ce que l'on souhaite afficher. OpenGL permet la définition de primitives géometriques simples, en 2 ou 3 dimensions. La définition d'une primitive doit toujours se faire **entre** 2 fonctions particulières, `glBegin` et `glEnd`. Elle consiste alors à énumérer une liste de points qui vont être considérés comme les sommets de la primitive à construire. La syntaxe des 2 fonctions `glBegin` et `glEnd` est la suivante :

```
void glBegin(GLenum mode);
void glEnd(void);
```

Le paramètre `mode` spécifie le type de primitive qui doit être construite. Parmi les valeurs possibles de ce paramètre, en voici quelques unes : `GL_POINTS`, `GL_LINES`, `GL_POLYGON`, `GL_TRIANGLES`. La primitive sera tracée à l'issue de l'exécution de la fonction `glEnd`.

La définition d'une primitive géométrique s'écrit alors de la manière suivante :

```
glBegin(TYPE_DE_LA_PRIMITIVE);
  /* definition sommet 0
  /* definition sommet 1
  /* ...................
```

```
  /* definition sommet N
glEnd();
```

OpenGL utilisera alors les différents sommets dans l'ordre de leur apparition pour définir la ou les primitives souhaitées. Ci-dessous est résumée la façon dont sont utilisés les sommets selon le type de la primitive choisie :

- `GL_POINTS` : définition d'un ensemble de points $P_0, P_1, P_2, ...P_N$
- `GL_LINES` : définition d'un ensemble de lignes ; les P_i sont interprétés comme étant une suite de **segments disjoints** (P_0, P_1), (P_2, P_3), ...
- `GL_POLYGON` : définition d'un polygone ; les P_i sont interprétés comme étant les sommets successifs d'un polygone, le dernier sommet (P_N) étant connecté au premier sommet (P_0) ;
- `GL_TRIANGLES` : définition d'un ensemble de triangles ; les P_i sont interprétés 3 par 3 comme étant les sommets d'un triangle, le dernier sommet de chaque triplet étant connecté au premier.

Il reste à présent à préciser la façon de définir un sommet. Cela se fait par l'intermédiaire des fonctions `glVertex`. Il existe plusieurs fonctions `glVertex`, dont le nom et le prototype varient en fonction de la façon dont le sommet doit être défini : nombre de dimensions du point (2,3 ou 4), type du point (`int`, `float` ou `double`, etc ...). Voici quelques exemples de fonctions existantes :

- `glVertex2f(float x, float y)`
- `glVertex3d(double x, double y, double z)`
- `glVertex2i(int x, int y)`
- `glVertex4f(float x, float y, float z, float w)`

Exemple : l'exemple ci-dessous permettra de définir deux segments d'extrémités respectives $(0, 0) - (0.5, 0.5)$ et $(-0.6, -0.6) - (-0.6, 0.6)$.

```
glBegin(GL_LINES);
  glVertex2f(0.0, 0.0);
  glVertex2f(0.5, 0.5);
  glVertex2f(-0.6, -0.6);
  glVertex2f(-0.6, 0.6);
glEnd();
```

Applications :

- Modifiez la fonction `dessiner` de telle manière qu'elle trace un carré de côté $\frac{1}{2}$ et de centre O, origine du repère ;
- Modifiez la fonction `dessiner` de telle manière qu'elle trace un cercle de centre O et de rayon 0.9. Le cercle sera approximé par un polygone.

4.3 Attributs de tracé

Différents attributs sont disponibles pour faire varier la façon dont le tracé sera effectué, tels que la couleur, l'épaisseur des points et des lignes, etc...

La couleur La fonction `glColor3f` permet de définir la couleur par défaut qu'utilisera OpenGL pour dessiner. Son prototype est le suivant :

```
void glColor3f(float r, float v, float b);
```

Ses paramètres représentent les valeurs de couleur en mode (R,V,B), chaque valeur pouvant prendre une valeur comprise entre 0 et 1. Notez bien que l'exécution de cette fonction a pour effet de fixer la couleur d'affichage dans le contexte graphique, ce qui implique que toute fonction d'affichage utilisera cette couleur jusqu'à ce qu'elle soit à nouveau modifiée par un autre appel à `glColor3f`.

Applications : Modifiez la fonction `dessiner` de telle manière qu'elle trace le carré précédent en **rouge** et le cercle en **bleu**.

Taille des points De la même manière, il est possible de modifier la taille des points affichés à l'aide de la fonction suivante :

```
void glPointSize(GLfloat taille);
```

La taille doit être un nombre ≥ 0, la taille par défaut étant 1.0. De la même manière que précédemment, cette fonction modifie la taille des points dans le contexte graphique d'OpenGL.

Notez que, pour être pris en compte, l'appel à cette fonction doit être fait avant l'appel à la fonction `glBegin()`.

Epaisseur des lignes L'épaisseur des lignes peut être modifiée en utilisant la fonction suivante :

```
void glLineWidth(GLfloat epaisseur);
```

L'épaisseur doit être un nombre ≥ 0, l'épaisseur par défaut étant 1.0. De la même manière que précédemment, cette fonction modifie l'épaisseur des lignes dans le contexte graphique d'OpenGL.

Notez que, pour être pris en compte, l'appel à cette fonction doit être fait avant l'appel à la fonction `glBegin()`.

Recommandations Dans toute la suite du Tp et dans tous les Tps suivants :

- les variables globales utilisées seront définies dans le fichier `tpN.c` (avec N = 1, 2, 3, ...) et déclarées `extern` dans un fichier nommé `globales.h`.
  Tout module utilisant une variable globale devra donc inclure le fichier `globales.h`
- pour permettre la compilation séparée, chaque fois que vous créerez un fichier d'extension `.c` vous écrirez le fichier correspondant d'extension `.h` contenant les prototypes des fonctions définies dans le fichier d'extension `.c`.

Applications :

1. Définissez une variable globale nommée `taillePoint`. Vous initialiserez cette variable à 1.0.
2. Utilisez cette variable dans la fonction `dessiner` pour fixer la taille des points et l'épaisseur des lignes.
3. Pour tester la validité de votre code donnez différentes valeurs à la variable `taillePoint` et notez la variation des tracés.
4. Revenez à la valeur par défaut qui est 1.0.

---

5 Gestion des événements

La librairie Glut permet de gérer à la fois la partie multi-fenêtrage et la partie événementielle. Nous allons à présent décrire succintement la façon d'utiliser les fonctionnalités de Glut pour gérer les événements issus de la souris ou du clavier.

5.1 Gestion des événements clavier

La gestion des événements liés au clavier s'effectue par l'intermédiaire de la fonction `glutKeyboardFunc`, dont le prototype est le suivant :

```
void glutKeyboardFunc(void (*func)(unsigned char touche,
                                   int x, y));
```

Son fonctionnement est similaire à celui de la fonction `glutDisplayFunc` : elle définit la fonction qui doit être appelée lorsqu'un événement souris est détecté.

Les paramètres qui seront passés à cette fonction lors de son appel par le gestionnaire d'événements de Glut sont les suivants :

- `touche` : le code ASCII de la touche pressée ;
- `(x,y)` : la position du curseur de la souris au moment de l'appel.

Application :

- Écrivez dans un fichier nommé `clavier.c` une fonction nommée `gestionClavier` permettant de faire varier la taille des points et l'épaisseur des lignes. Un appui sur la touche `+` incrémente la variable globale `taillePoint` de 1.0, un appui sur la touche `-` la décrémente de 1.0. Pensez à vérifier que la taille des points reste toujours supérieure à 1.0 ;
- Terminez le code de la fonction `gestionClavier` par un appel à la fonction `glutPostRedisplay()` qui permet de forcer glut à appeler la fonction de dessin. Ceci permettra d'éviter des problèmes de rafraîchissement de l'affichage ;
- Modifiez la méthode `main` pour que la gestion du clavier soit faite par la fonction `gestionClavier` ;
- Modifiez le fichier `Makefile` puis testez les fonctionnalités de votre programme modifié.

5.2 Gestion des événements souris

De la même manière, il est possible de gérer les événements liés à la souris par l'intermédiaire de la fonction :

```
void glutMouseFunc(void (*func)(int bouton, int etat,
                                int x, int y));
```

Comme précédemment, elle définit la fonction qui doit être appelée lorsqu'un événement souris est détecté. Les paramètres qui seront passés à cette fonction lors de son appel par le gestionnaire d'événements de Glut sont les suivants :

- `bouton` : ce paramètre précise le bouton qui a été la cause de l'événement ; les valeurs possibles sont `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON`[6] ou `GLUT_RIGHT_BUTTON` ;
- `etat` : précise l'état du bouton qui a généré l'événement avec les 2 valeurs possibles `GLUT_DOWN` (bouton enfoncé) et `GLUT_UP` (bouton relevé) ;
- `x, y` : les coordonnées du pixel dans lequel l'événement a eu lieu.

Application :

- définir, dans un fichier `geometrie.h`, le type et la constante suivants :

```
typedef struct { float x, y;} point2D;

#define UNDEFINED  -1E10
```

- définir, dans le module `tp1.c`, une variable globale de type `point2D`. Cette variable sera utilisée pour saisir et communiquer les coordonnées du point de l'écran à allumer. Les coordonnées de cette variable globale seront initialisées avec la valeur `UNDEFINED` ;
- écrire, dans un fichier nommé `souris.c`, une fonction permettant de saisir les coordonnées du point sur lequel se trouve la souris lorsque le bouton gauche est **relaché**. Vous terminerez cette fonction par un appel à la fonction `glutPostRedisplay()` qui permet de forcer `glut` à appeler la fonction de dessin. Ceci permettra d'éviter des problèmes de rafraîchissement de l'affichage.
  - Notez bien que le repère écran est défini par l'intervalle $[-1, 1] \times [-1, 1]$, alors que les valeurs des paramètres `x` et `y` de la fonction sont les coordonnées pixels (origine en haut à gauche de la fenêtre). Il sera donc nécessaire d'effectuer le calcul de conversion dans la fonction qui saisit les coordonnées de la souris ...
  - Pour récupérer les dimensions de la fenêtre d'affichage, vous pourrez utiliser la fonction `int glutGet(int param)` avec comme valeur de paramètre `GLUT_WINDOW_WIDTH` (largeur de la fenêtre) ou `GLUT_WINDOW_HEIGHT` (hauteur de la fenêtre).
- modifier la fonction `dessiner` de manière à ce qu'à chaque appel qui lui est fait, elle efface l'écran puis affiche le point qui vient d'être saisi, avec la taille courante, sous réserve que ses coordonnées soient bien définies ;
- pensez enfin à modifier la fonction `main` pour pouvoir prendre en compte les événements souris ...

---

6. Pour les souris à 2 boutons, cette valeur ne pourra pas être générée.

5.3 Gestion des événements clavier spéciaux

L'utilisation de la fonction `glutKeyboardFunc` ne permet de gérer que les touches « classiques » d'un clavier. Pour gérer les touches spéciales, il est nécessaire de passer par la fonction suivante :

```
void glutSpecialUpFunc(void (*func)(unsigned char touche,
                                    int x, y));
```

De manière similaire à ce qui a été vu précédemment, la fonction passée en paramètre récupérera la touche utilisée dans le paramètre `touche`. La différence réside dans le fait que cette touche sera nécessairement un touche spéciale, qui devra être testée via des constantes prédéfinies. Par exemple, les touches de déplacement du curseur dont définies par :

- `GLUT_KEY_DOWN` : déplacement vers le bas
- `GLUT_KEY_LEFT` : déplacement vers la gauche
- `GLUT_KEY_RIGHT` : déplacement vers la droite
- `GLUT_KEY_UP` : déplacement vers le haut

Application :

- Ajoutez dans le fichier `clavier.c` une fonction permettant de saisir dans une variable globale de type `point2D` la translation qui doit être appliquée au point affiché (s'il existe). Cette translation prendra les valeurs suivantes :
  - `(0.1,0.0)` si la touche `droite` est enfoncée ;
  - `(-0.1,0.0)` si la touche `gauche` est enfoncée ;
  - `(0.0,0.1)` si la touche `haut` est enfoncée ;
  - `(0.0,-0.1)` si la touche `bas` est enfoncée ;
- modifier la fonction `dessiner` de manière à ce qu'elle prenne en compte cette translation en ajoutant sa valeur aux coordonnées du point à afficher et en la remettant à 0 après utilisation ;
- tester les fonctionnalités de votre programme modifié ;

Partie optionnelle

6 utilisation de menus

Glut vous permet de créer, de manière très simple, des petits menus, utilisables à la souris. Nous allons voir dans ce qui suit comment opérer.

6.1 Création d'un menu

La fonction permettant la création d'un menu est la suivante :

```
int glutCreateMenu(void (*func)(int valeur));
```

La valeur retournée est le numéro du menu créé, numéro qui peut éventuellement être utilisé par la suite lors d'une gestion **dynamique** des menus. Il ne sera pas utilisé dans le cadre de ce TP.

Le paramètre passé à `glutCreateMenu` correspond à la fonction qui devra gérer les choix de l'utilisateur dans le menu. Le paramètre de cette fonction correspondra au choix qui a été fait dans le menu.

6.2 Initialisation du menu

Chaque menu possède des choix ; après avoir créé votre menu, il est nécessaire de lui définir ces choix, à l'aide de la fonction suivante :

```
void  glutAddMenuEntry(char *label, int cle);
```

Le paramètre `label` correspond à la chaîne de caractères qui apparaîtra dans le menu et qui correspondra à l'intitulé du choix. Le paramètre `clé` quant à lui sera la valeur passée à la fonction chargée de gérer le menu si l'option correspondante est choisie.

6.3 Exemple

Soit à gérer un menu proposant 2 choix :
— effacer l'écran ;
— quitter.

À chacun de ces choix on va affecter une `cle` numérique qui permettra de déterminer le choix sélectionné ; au premier choix, on va associer la valeur 1 et au second la valeur 2. On pourra alors écrire les 2 lignes suivantes :

```
glutAddMenuEntry("effacer", 1);
glutAddMenuEntry("quitter", 2);
```

Il faut ensuite écrire la fonction permettant de traiter chacun des choix de l'utilisateur :

```
void gerer_menu(int cle)
{
  switch(cle){
  case 1 : /* traiter l'effacement */
  case 2 : /* traiter l'arret */
  }
}
```

Il reste ensuite à écrire, dans l'ordre, toutes les instructions nécessaires à l'initialisation de notre menu :

```
....
void gerer_menu(int cle)
{
  switch(cle){
  case 1 : /* traiter l'effacement */
  case 2 : /* traiter l'arret */
  }
}
...
glutCreateMenu(gerer_menu);
glutAddMenuEntry("effacer", 1);
glutAddMenuEntry("quitter", 2);
```

Enfin il est nécessaire d'attacher ce menu à l'un des boutons de la souris afin que l'on puisse l'appeler ... Pour cela, on peut utiliser la fonction :

```
void glutAttachMenu(int bouton);
```

le paramètre `bouton` étant l'une des 3 valeurs prédéfinies :
— `GLUT_RIGHT_BUTTON`
— `GLUT_MIDDLE_BUTTON`
— `GLUT_LEFT_BUTTON`

Attention : lorsqu'un menu est attaché à un bouton, celui-ci ne peut plus être utilisé pour autre chose que l'appel du menu ...

Remarque : à aucun moment n'apparaît de référence au numéro de menu auquel font référence les différentes fonctions. Cela est dû au fait que Glut gére un *menu courant*. Après création d'un menu par `glutCreateMenu`, le menu obtenu est considéré comme le menu courant et toutes les instructions qui suivent et qui se rapportent à un menu lui seront appliquées.

6.4 Exercice

Modifier votre application *tp1* pour qu'elle fasse apparaître un menu permettant de choisir la couleur de tracé des points. Les couleurs proposées par ce menu seront le rouge, le vert et le bleu. Pour réaliser ceci, il vous est conseillé :
— d'écrire les fonctions de gestion du menu dans un fichier nommé `menu.c` ;
— d'écrire, dans ce fichier, une fonction nommée `init_menu`, dont le rôle est de créer et d'initialiser le menu et qui sera appelée par la fonction `main` de `tp1.c`.

6.5 Création de sous-menus

Un menu simple est souvent insuffisant lorsque le nombre d'options augmente. Glut permet donc d'associer un sous-menu aux différentes entrées d'un menu. Cela se fait par l'intermédiaire de la fonction :

```
void glutAddSubMenu(char *label, int menu);
```

avec :
- `label` : le label qui apparaîtra dans le menu principal (donc le « nom » du sous-menu ;
- `menu` : le numéro du menu qui doit être associé au label dans le menu principal.

Le sous-menu doit donc être créé à part, avec la fonction `glutCreateMenu`. Cette création implique aussi la création de ses différentes options et de la fonction chargée de les gérer.

Exemple :

```
....
void gerer_menu_principal(int cle)
{
  switch(cle){
  case 1 : /* traiter l'impression */
  case 2 : /* traiter l'arret */
  }
}
...
void gerer_sous_menu_fichier(int cle)
{
  switch(cle){
  case 1 : /* traiter le chargement */
  case 2 : /* traiter la sauvegarde */
  }
}
...
/* -- creer le sous-menu -- */
id_sous_menu = glutCreateMenu(gerer_sous_menu_fichier);
glutAddMenuEntry("charger", 1);
glutAddMenuEntry("sauver", 2);

/* -- creer le menu principal -- */
glutCreateMenu(gerer_menu_principal);
glutAddSubMenu("fichier", id_sous_menu); /* insertion sous menu */
glutAddMenuEntry("imprimer", 1);         /* dans le menu courant */
glutAddMenuEntry("quitter", 2);
```

6.6 Exercice

Modifier votre application de manière à ce que son menu offre le choix entre 3 options :
- choisir une couleur de tracé ; dans ce cas un sous-menu proposera les couleurs rouge, vert et bleu ;
- choisir une épaisseur de tracé ; dans ce cas un sous-menu proposera les valeurs 1.0, 2.0 ou 3.0 ;
- effacer l'écran.

6.7 Exercice supplémentaire

Rajouter une option à votre menu principal, permettant de choisir la primitive géométrique à tracer : un point, un segment ou un cercle. Dans le cas du point, un clic sur le bouton gauche de la souris indiquera l'emplacement auquel le tracer. Dans le cas du segment, le point de départ sera indiqué par un clic sur le bouton gauche de la souris et le point d'arrivée par un clic sur le bouton droit. Enfin, en cas de tracé d'un cercle, le centre de celui-ci sera indiqué par un clic sur le bouton gauche et un point de son contour par un clic sur le bouton droit.