Synthèse d'Image - TD01

Initiation à OpenGL et SDL

L'objectif de cette matière est de vous initier à **OpenGL** (en 2D, cette année). Ce premier TP va également vous permettre de prendre en main la **SDL** afin de créer une fenêtre dans laquelle un rendu OpenGL pourra être effectuée. Nous verrons également comment intégrer la gestion des évènements de type clavier, souris, etc...

EXERCICE 01 : Késako ?

Avant de commencer à travailler, il serait bon de savoir quels outils nous utilisons.

1. **A l'aide d'internet, définissez rapidement les termes suivants :**
   a. **API**
   b. **OpenGL**
   c. **SDL**
   d. **Open Source**

EXERCICE 02 : Votre première fenêtre

Vous savez maintenant ce que sont OpenGL et SDL. il est maintenant temps de s'en servir.

Récupérez les fichiers ***minimal.c*** et ***Makefile*** sur le site web du chargé de TD.

1. **Dans minimal.c, que signifient les premières instructions "#include" ?**
2. **Quelles sont les bibliothèques utilisées par OpenGL et SDL? (ie. quels sont les fichiers .so à utiliser ?)**
3. **Comment appeler/utiliser ces librairies à la compilation?**
4. **Compilez puis exécutez le programme minimal.c. Que constatez-vous ?**
5. **Quelle est l'utilité de la fonction** `SDL_SetVideoMode(...)` **?**

EXERCICE 03 : Domptage de la fenêtre

La fonction ***SDL_SetVideoMode*** permet d'ouvrir une fenêtre et de fixer ses paramètres. Elle a la signature suivante :

```
SDL_Surface* SDL_SetVideoMode(int width, int height, int bitsperpixel,
Uint32 flags)
```

Le pointeur qu'elle renvoit ne nous servira pas dans le cas d'une application OpenGL.

- Les paramètres ***width*** et ***height*** permettent de fixer les dimensions de la fenêtre.
- Le paramètre ***bitsperpixel*** spécifie le nombre de bits que la SDL doit utiliser pour encoder un pixel. Les valeurs qu'on utilise le plus souvent sont 8 pour une fenêtre affichant des images en niveau de gris, 24 pour un affichage rouge, vert, bleu et 32 pour un affichage rouge, vert, bleu + transparence.
- Le paramètre ***flags*** est un champs de bits permettant d'énumérer divers paramètres à l'aide de l'opérateur | (dit opérateur "pipe") de la manière suivante :

```
param1 | param2 | ... | param_n
```

Voici un exemple simple d'appel permettant d'ouvrir une fenêtre de dimensions 10x10 en niveaux de gris, pour OpenGL et en plein écran :

```
SDL_SetVideoMode(10, 10, 8, SDL_OPENGL | SDL_GL_DOUBLEBUFFER |
SDL_FULLSCREEN)
```

Attention ! Si vous exécutez ce code vous ne pourrez plus quitter l'application proprement car la croix permettant de fermer la fenêtre disparait.
Vous pourrez trouver sur cette page une liste des différents paramètres pouvant être placés dans flags http://sdl.beuc.net/sdl.wiki/SDL_SetVideoMode

La fonction **SDL_WM_SetCaption** permet quand à elle de modifier le titre de la fenêtre et son icône. La page suivante décrit son fonctionnement :
http://sdl.beuc.net/sdl.wiki/SDL_WM_SetCaption

A partir du fichier minimal.c et des fonctions ci-dessus :

1. **Créez une fenêtre redimensionnable (SDL_RESIZABLE) pour OpenGL de taille 400x400.**
2. **Changez le titre.**
3. **Finalement nous voulons effacer le contenu de la fenêtre (i.e. le remplir d'une couleur unique). Dans la fonction** `main()`**, insérez au bon endroit l'instruction** `glClear(GL_COLOR_BUFFER_BIT);`

EXERCICE 04 : Des événements ...

La SDL permet une gestion poussée et très libre des événements utilisateur. Lorsqu'elle détecte un événement, elle place dans une file interne une structure de type SDL_Event décrivant l'événement. Il est nécessaire de traiter tous les événements reçus à chaque tour de la boucle d'affichage. La fonction :

int SDL_PollEvent(SDL_Event *event)

permet de défiler (=retirer de la file) un événement de la file interne. Elle renvoie **0** s'il n'y a aucun événement à défiler. Sinon elle renvoie **1** et elle remplit ***event** avec les informations concernant l'événement. On peut donc défiler et traiter tous les événements de la file avec une boucle de la forme suivante :

```
SDL_Event e;
while(SDL_PollEvent(&e)) {
      /* Traiter evenement */
}
```

La structure **SDL_Event** est détaillée sur la page suivante :
http://sdl.beuc.net/sdl.wiki/SDL_Event.

Comme on peut le voir, ce n'est pas vraiment une structure mais une union.

1. **Rappeler la différence en C++ entre une structure et une union.**

Le champs type indique le type de l'événement reçu. Il indique quel champs doit être manipulé par la suite dans l'union. Par exemple si **e.type** vaut **SDL_KEYDOWN** ou **SDL_KEYUP**, on doit accéder au champs **e.key** pour connaître les propriétés de l'événement. Par contre si **e.type** vaut **SDL_MOUSEBUTTONUP** il faut accéder à **e.button**. La page : http://sdl.beuc.net/sdl.wiki/SDL_Event_Structures décrit chaque type de structure événement.

2. **Modifier le programme pour qu'il s'arrête lorsque l'utilisateur appuie sur la touche 'q'. Vous pourrez ainsi quitter le programme proprement même en plein écran.**
3. **Faire en sorte que lorsque l'utilisateur clique en position (x, y), la fenêtre soit redessinée avec une couleur proportionnelle aux coordonnées cliquées. En haut à gauche : tout noir, en haut à droite : tout rouge, en bas à gauche : tout vert et en bas à droite : tout jaune. Pour les autres pixels : un dégradé des 4 couleurs en fonction de la proximité du bord.**
4. **Faire de même mais lorsque la souris bouge (événement SDL_MOUSEMOTION).**

EXERCICE 05 : Dessinons !

Nous avons précédemment assigné une taille en pixels à notre fenêtre de rendu, grâce à la fonction `SDL_SetVideoMode(...)` (taille à l'écran). Or, pour pouvoir y dessiner une scène, nous devons également indiquer à OpenGL la taille virtuelle qu'aura notre fenêtre à l'intérieur de cette scène. Ainsi, à chaque fois qu'un point sera dessiné, les coordonnées de ce point seront fournies à OpenGL dans l'espace virtuel de la scène, et OpenGL se chargera de convertir celles-ci en coordonnées pixels dans le repère 2D de la fenêtre. Cette étape de conversion entre les coordonnées virtuelles et les coordonnées pixels (ou écran) s'appelle la **projection**.

1. **Créer une fonction, mettez-y les lignes suivantes, et faites le nécessaire pour qu'elle soit appelée à chaque redimensionnement de la fenêtre (événement SDL_VIDEORESIZE). N'oubliez pas de modifier les variables WINDOW_WIDTH et WINDOW_HEIGHT avant l'appel à la fonction en utilisant les paramètres de l'événement SDL.**

```
SDL_SetVideoMode(WINDOW_WIDTH, WINDOW_HEIGHT, BIT_PER_PIXEL, SDL_OPENGL |
SDL_RESIZABLE)
glViewport(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-1., 1., -1., 1.);
```

Le **viewport** défini la taille en pixel de la fenêtre réelle. Puis on passe en mode projection, et après avoir chargé la matrice identité, on crée une projection orthogonale qui, en fait, transforme les coordonnées virtuelles en coordonnées pixels.

Dessinons maintenant !

Pour spécifier une couleur, on utilise l'instruction

```
glColor3ub(r, v, b);
```

où **r**, **v**, **b** sont trois entiers compris entre 0 et 255 qui définissent les trois composantes rouge, verte et bleue de la couleur désirée. Une fois cette fonction appelée, tout se qui sera dessiné par la suite aura cette couleur.

2. **Dessiner un ou plusieurs points grâce aux instructions suivantes :**

```
glBegin(GL_POINTS);
  glVertex2f(-1 + 2. * x / WINDOW_WIDTH, -(-1 + 2. * y / WINDOW_HEIGHT));
glEnd();
```

ou **x** et **y** sont les coordonnées du pixel dans la fenêtre SDL.
La fonction `glBegin()` informe OpenGL qu'on va commencer à dessiner. Le paramètre à passer à cette fonction correspond au type de primitive qu'on veut dessiner (ici, **GL_POINTS** pour dessiner des points, mais il en existe d'autres comme **GL_LINES** ou **GL_TRIANGLES**. Le détail de la fonction est défini à cette page :
https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glBegin.xml

La fonction glEnd() informe OpenGL qu'on a terminé de dessiner.

3. **Pourquoi utilise t-on *-1 + 2. * x / WINDOW_WIDTH* et pas simplement x ? (idem pour y)**

4. **A l'aide de ces fonctions, faire une application de dessin :**

   Lorsqu'on clique dans la fenêtre, un point (de la couleur de votre choix) doit s'afficher à l'endroit cliqué. Utilisez les événements SDL appropriés pour récupérer l'information du clic souris.

   Attention, pour que votre dessin persiste, il faut :
   - désactiver le double buffering avant d'ouvrir la fenêtre avec SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 0)
   - nettoyer la fenêtre une seule fois avec glClear(GL_COLOR_BUFFER_BIT) **avant la boucle de rendu**.

EXERCICE 06 : De nouvelles fonctionnalités à l'application de dessin. (Cet exercice est à faire à la maison.)

Vous allez améliorer votre application de dessin en ajoutant les fonctionnalités suivantes :

Changement de primitives

Donnez la possibilité à l'utilisateur de dessiner plusieurs types de primitives :
- Des points (lorsqu'il appuie sur la touche P)
- Des lignes (lorsqu'il appuie sur la touche L)
- Des triangles (lorsqu'il appuie sur la touche T)

Pour cela, vous allez devoir jouer sur le mode de primitive que vous passez à la fonction glBegin(). Attention : Chaque primitive attend un certain nombre de point pour pouvoir être

dessinée (1 point pour GL_POINTS, 2 points pour GL_LINES, 3 points pour GL_TRIANGLES, etc.).

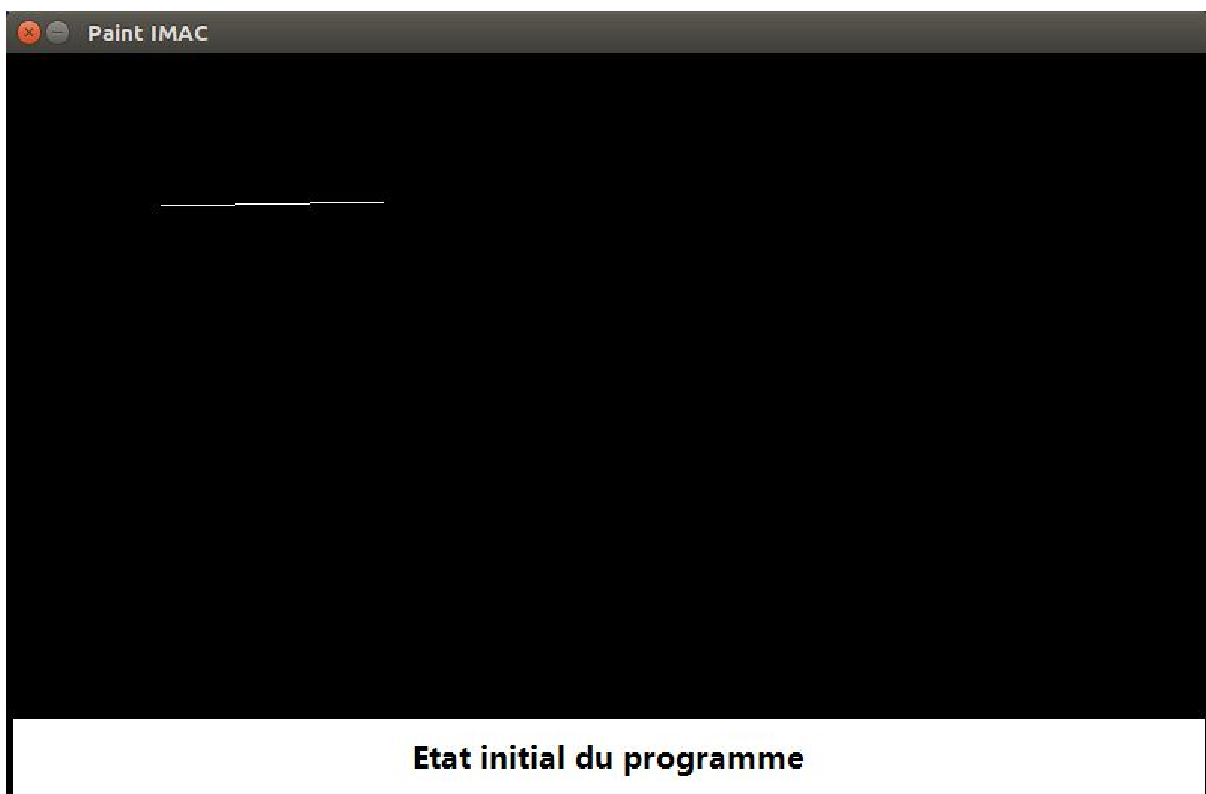
Choix de la couleur

Donnez la possibilité à l'utilisateur de pouvoir choisir une couleur de dessin sur une palette :

- Lorsqu'il appuie sur ESPACE, le viewport doit afficher plusieurs bandes de couleur correspondant à la palette.
- Lorsque l'utilisateur **clique** sur un de ces carrés, le programme enregistre la couleur sélectionnée.
- Lorsqu'il relâche ESPACE, le programme retourne en mode dessin et la couleur choisie est maintenant la couleur courante.

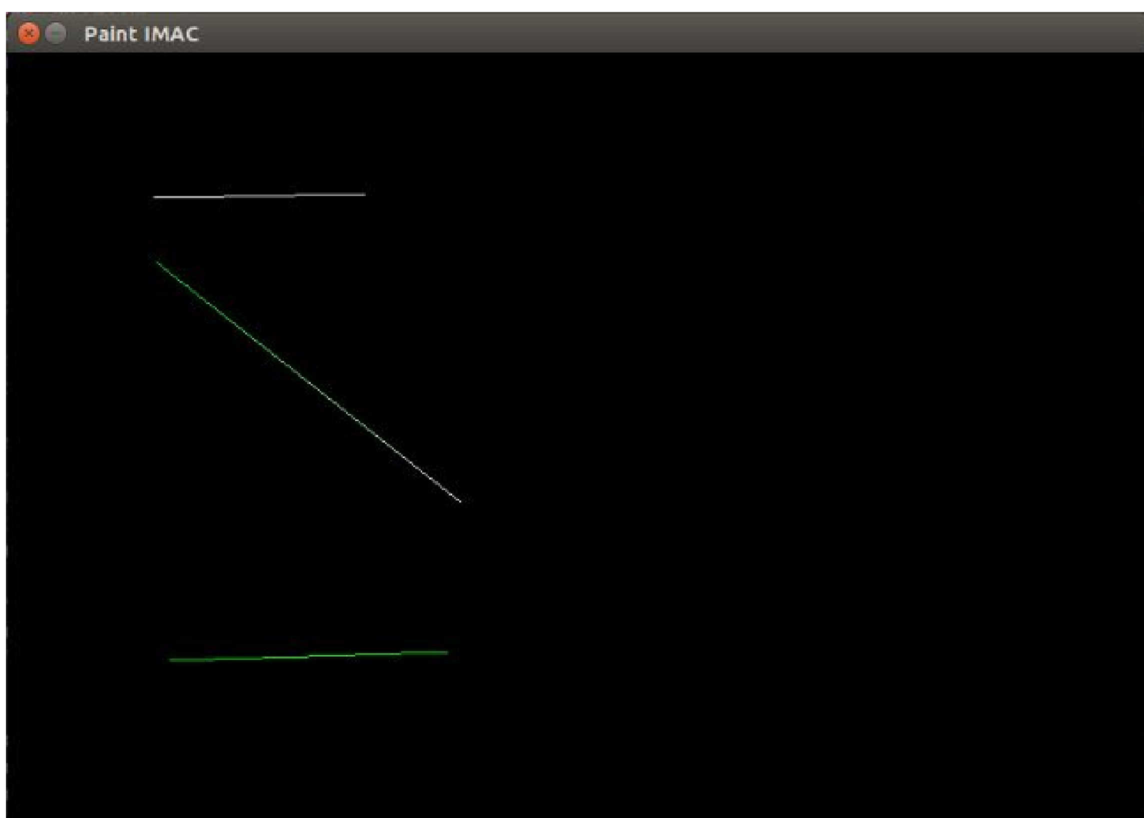
Vous allez devoir gérer deux modes de dessin : le mode normal et le mode "palette". Pour représenter le mode, un simple entier peut faire l'affaire. Vous devrez tester si cet entier vaut 0 (pour le mode normal) ou 1 (pour le mode palette) et dessiner en conséquence. Utilisez les événements SDL pour détecter que l'utilisateur appuie sur ESPACE, et changez la valeur du mode à ce moment là.

Proposez également un solution simple pour détecter sur quelle couleur de la palette l'utilisateur a cliqué (une seule des coordonnées du clic peut suffire).



Etat initial du programme

Passage en mode Palette lorsqu'on appuie sur ESPACE



Le programme enregistre la couleur pour les futurs dessins.

Structurer davantage le programme

Pour l'instant, vous ne gardez aucune trace des points que vous dessinez à l'écran. Vous allez mettre en place un système pour stocker les coordonnées et les couleurs des points dessinés, afin de les manipuler plus facilement.

1. **Ajoutez la structure suivante à votre programme :**

```
typedef struct Point{
	float x, y; // Position 2D du point
	unsigned char r, g, b; // Couleur du point
	struct Point* next; // Point suivant à dessiner
} Point, *PointList;
```

2. **Implémentez la fonction** allocPoint**, qui alloue l'espace mémoire nécessaire au stockage d'une structure Point, qui initialise le point, et qui renvoie le pointeur sur cet espace mémoire. La signature de la fonction doit être :**

```
Point* allocPoint(float x, float y, unsigned char r, unsigned char g,
unsigned char b);
```

3. **Implémentez la fonction** addPointToList **qui ajoute en fin de liste un point passé en paramètre.**

```
void addPointToList(Point* point, PointList* list);
```

4. **Implémentez la fonction** drawPoints **qui dessine tous les points de la liste passée en paramètre (avec les fonctions glColor3ub et glvertex2f).**

```
void drawPoints(PointList list);
```

5. **Implémentez la fonction** deletePoints **qui libère l'espace alloué à tous les points d'une liste.**

```
void deletePoints(PointList* list);
```

Vous avez également besoin de stocker des informations sur les primitives à utiliser. Une primitive encapsulera une liste de points à dessiner suivant un type de primitive spécifique. Le tout sera ordonné suivant une liste, pour enregistrer l'ordre dans lequel vous aurez dessiné les points.

6. **Ajoutez la structure suivante à votre programme :**

```
typedef struct Primitive{
    GLenum primitiveType;
    PointList points;
    struct Primitive* next;
} Primitive, *PrimitiveList;
```

7. **Implémentez la fonction** allocPrimitive **qui alloue la mémoire nécessaire à un objet Primitive. Le champs *points* doit rester à NULL.**

```
Primitive* allocPrimitive(GLenum primitiveType);
```

8. **Ajoutez la fonction** addPrimitive **qui ajoute une primitive à la liste de primitives.**

```
void addPrimitive(Primitive* primitive, PrimitiveList* list);
```

9. **Implémentez la fonction** drawPrimitives **qui dessine une liste de primitives passée en paramètre (réutilisez la fonction drawPoints pour chaque maillon de la chaîne de primitive).**

```
void drawPrimitives(PrimitiveList list);
```

10. **Implémentez la fonction** deletePrimitive **qui libère l'espace alloué à une primitive et à tous les points qu'elle contient.**

```
void deletePrimitive(PrimitiveList* list)
```

Désormais, vous pouvez utiliser vos nouvelles structures et fonctions pour structurer correctement votre programme :

11. **A l'initialisation, Créez une liste de primitives et créez une première primitive par défaut de type GL_POINTS.**
12. **Dans la boucle de rendu :**
    a. **Lorsque l'utilisateur appuie sur une touche pour changer de primitive (P, L, T), ajoutez une nouvelle primitive à la liste.**
    b. **Lorsque l'utilisateur clique dans le viewport, ajoutez un point de la couleur courante à la primitive courante.**
    c. **A chaque tour de boucle, dessinez l'ensemble des primitives.**
13. **En fin de programme, n'oubliez pas de libérer l'espace alloué aux primitives et aux points.**

Vous avez désormais toutes les cartes en main pour créer cette application de dessin qui concurrencera Photoshop à coup sûr !