

# Programmation de Jeux 2D: Un pong en SDL/OpenGL, Première partie

par [Jean Christophe Beyler](#)

Date de publication : 18/09/2006

Dernière mise à jour : 18/09/2006

Le jeu de pong est sûrement un des jeux les plus refait. Cette série de tutoriels va présenter une implémentation possible mais en rendant la tâche plus difficile. En effet, nous voulons avoir plusieurs balles qui peuvent être en collision et un jeu en multi-joueur. Voici l'introduction de ce tutoriel, il montre comment ouvrir une fenêtre SDL/OpenGL.

- 1 - Introduction
  - 1.1 - SDL
  - 1.2 - OpenGL
- 2 - Programme de base
- 3 - Conclusion
- 4 - Téléchargements
- 5 - Remerciements

## 1 - Introduction

Bienvenue à cette nouvelle série sur la programmation de jeux en 2D. Nous allons nous lancer dans un projet nettement plus intéressant que le [morpion](#). En effet, nous allons présenter le jeu *Pong*.

La version de base consiste à avoir deux raquettes et une balle. Il faut faire passer la balle derrière la raquette de l'adversaire et à ce moment là c'est gagné!

Encore un jeu suffisamment simple pour ne pas nous faire perdre nos cheveux mais nous allons tout de même aller un peu plus loin. Premièrement, nous allons ajouter le fait que le jeu va ajouter une balle de temps en temps, ce qui va bien sûr compliquer le jeu. Ensuite, nous allons ajouter une couche réseau pour pouvoir jouer à ce jeu en multijoueur.

### 1.1 - SDL

Je vais bien sûr reprendre beaucoup de choses de la série sur le [morpion](#), je vous conseille donc de le lire si vous n'êtes pas familier avec la bibliothèque SDL ou avec la programmation de jeu en général.

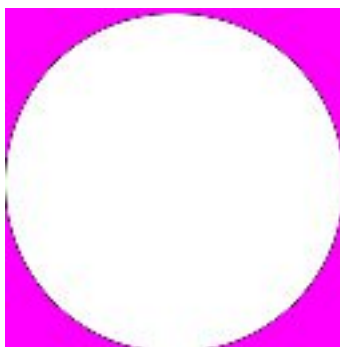
La bibliothèque SDL permet d'ouvrir la fenêtre et de gérer les entrées claviers et souris. Nous allons revoir rapidement dans la section 2 comment ouvrir la fenêtre. Mais, contrairement au [morpion](#), nous allons utiliser la bibliothèque [OpenGL](#) pour l'affichage.

### 1.2 - OpenGL

On pourra se demander pourquoi utiliser OpenGL pour faire de la 2D? Pourquoi ne pas utiliser les fonctions fournies par SDL?

Les réponses sont assez simples à comprendre : nous voulons afficher des balles de tailles différentes et de couleurs différentes.

La première solution serait d'utiliser une image comme celle-ci :



*Une image d'une balle*

Ensuite, lors du rendu on utilise un filtre de couleur pour dessiner la balle dans la couleur voulue, et on modifie la taille pour obtenir les dimensions souhaitées.

La bibliothèque SDL ne permet pas facilement (et rapidement) de faire ces modifications, par contre c'est très facile en OpenGL.

La deuxième solution serait d'avoir une image contenant toutes les tailles et couleurs des balles souhaitées. Inutile de vous dire que ce serait un vrai travail de titan et on limiterait le programme inutilement.

Nous avons donc choisi d'utiliser SDL pour la gestion de la fenêtre et OpenGL pour le rendu.

## 2 - Programme de base

Cette section va montrer comment ouvrir une fenêtre SDL/OpenGL correctement initialisée. Nous allons commencer par une fonction qui gère l'affichage OpenGL.

### Fonction d'affichage

```
void affiche()
{
    //Mettre a zero le tampon de couleurs
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();

    SDL_GL_SwapBuffers();
}
```

Comme tout programme OpenGL, pour l'affichage, on utilise la fonction **glClear** pour mettre le tampon de couleur à zéro. C'est comme cela qu'on efface toute l'image qui se trouve dans le tampon. Cela nous permet donc de dessiner de nouveau sur le tampon. Ensuite, on remet le repère à zéro utilisant **glLoadIdentity** et, lorsque le code du dessin est terminé, on met à jour les tampons avec **SDL\_GL\_SwapBuffers**.

Ce tutoriel n'est en rien un tutoriel sur OpenGL. On suppose que vous connaissez un minimum sur OpenGL pour que cette fonction **affiche** ne soit pas une surprise. La suite de ce tutoriel est une présentation rapide de comment ouvrir une fenêtre OpenGL avec la bibliothèque SDL, pour avoir plus de détail sur les fonctions d'initialisation SDL, regardez [le premier tutoriel SDL sur le morpion](#) ou les tutoriels de [loka](#).

La fonction **main** commence par des déclarations de variables locales.

### Début du main

```
int main(int argc, char **argv)
{
    SDL_Event event;
    SDL_Surface *screen;
    int done = 0;
    int fps, last, now;
```

Voici la signification de ces variables :

- **event** : pour gérer les événements;
- **screen** : pour gérer la surface de la fenêtre;
- **done** : pour savoir si le programme devait se terminer;
- **fps** : pour calculer le nombre d'affichage par seconde;
- **last, now** : pour gérer le temps (pour le calcul de **fps**).

Une fois les variables déclarées, nous allons commencer par initialiser SDL.

### Initialisation

```
//Initialisation
if(SDL_Init(SDL_INIT_VIDEO)!=0) {
    cerr << "Probleme pour initialiser SDL: " << SDL_GetError() << endl;
    return EXIT_FAILURE;
}

//Mettre un titre à la fenetre
SDL_WM_SetCaption("Pong Version 1.0", NULL);

//Double tampon
SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER,1);

//Ouvrir une fenetre
screen = SDL_SetVideoMode(WIDTH, HEIGHT, 32, SDL_OPENGL);
```

### Initialisation

```
if(screen==NULL)
    done = 1;
```

Comme vous le voyez, on initialise SDL, on met un titre à la fenêtre. Ensuite, la fonction **SDL\_GL\_SetAttribute** permet de dire qu'on utilise un double tampon pour l'affichage (sous SDL pur, nous utilisons le drapeau **SDL\_DOUBLEBUF**; pour OpenGL, le double tampon est par défaut dans l'implémentation courante de SDL mais je trouve important de rajouter cet appel). Enfin, on utilise la fonction **SDL\_SetVideoMode** pour dire qu'on veut utiliser **SDL\_OPENGL**. C'est ici qu'il y a un changement par rapport à un programme pur SDL.

Ensuite, nous avons le code d'initialisation OpenGL. Puisque nous sommes en train de faire un jeu 2D, la mise en place d'une vue orthogonale est logique. Voici comment on la met en place :

### Initialisation d'OpenGL

```
//Remettre a zero la matrice de projection
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

// Mettre les matrices en place pour une vue orthogonale
glOrtho(0,WIDTH,HEIGHT,0,-1,1);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Nous allons commencer par remettre à zéro la matrice de projection. Ensuite, nous appelons **glOrtho** pour avoir une vue orthogonale. Les deux premiers paramètres sont les bornes de la largeur de la fenêtre. Les deux paramètres qui suivent sont les bornes de la hauteur de la fenêtre. Enfin, les deux derniers paramètres définissent les bornes de profondeur. Ici, **WIDTH** et **HEIGHT** sont définies dans un .h comme ceci :

### Définition des variables

```
const int WIDTH=800;
const int HEIGHT=600;
```

Avant de commencer la boucle événementielle, nous initialisons **last** et **fps**. Prenons donc le temps d'expliquer l'utilité de ces deux variables. **fps** servira pour compter le nombre d'images par seconde. A chaque itération de la boucle événementielle, nous allons regarder l'heure actuelle en utilisant la fonction **SDL\_GetTicks** qui fournit l'heure en nombre de millisecondes (donc la division par 1000 permet d'avoir ce temps en secondes). Nous allons ensuite comparer l'heure actuelle avec la valeur de **last**. Si la valeur est différente, nous allons pouvoir calculer le nombre d'affichages par seconde.

### Initialisation des variables last et fps

```
last = SDL_GetTicks()/1000;
fps = 0;
```

### Boucle événementielle

```
//Boucle generale
while(!done)
{
    //Traiter les evenements
    while(SDL_PollEvent(&event))
    {
        switch(event.type)
        {
            case SDL_QUIT:
                done=1;
                break;
            case SDL_KEYUP:
                if(event.key.keysym.sym==SDLK_q)
                    done=1;
                break;
            default:
                break;
        }
    }
}
```

#### Boucle événementielle

```
}
}
```

Rien de spécial dans cette boucle. Nous parcourons les événements qui sont en attente. Si l'événement est de type **SDL\_QUIT**, on met la variable **done** à jour. Si l'événement est de type **SDL\_KEYUP** et que la touche relâchée est la lettre 'q' alors on met **done** à jour pour quitter le programme.

A la fin de la boucle événementielle, on incrémente la variable **fps**. Et on vérifie l'heure actuelle. Enfin, on appelle la fonction **affiche** pour appeler l'affichage de la scène. Je pense que le code qui suit est assez explicite :

#### Fin de la boucle événementielle

```
fps++;

now = SDL_GetTicks()/1000;
if(now!=last)
{
    cout << "FPS: " << fps/(now-last) << endl;
    last = now;
    fps = 0;
}

affiche();
}
```

La fin du programme se termine l'appel **SDL\_Quit**.

#### Fin du programme

```
SDL_Quit();
return EXIT_SUCCESS;
}
```

### 3 - Conclusion

Voilà pour cette première partie, nous avons fait une introduction de l'utilisation des bibliothèques SDL et OpenGL.

Comme vous le voyez, il est assez facile d'ouvrir une fenêtre et, pour ajouter du code d'affichage, il suffit de compléter la fonction **affiche**.

Cette première partie n'est qu'un début et on me dira que c'est encore un autre tutoriel d'ouverture d'une fenêtre SDL. Mais je préfère prendre le temps de le refaire encore une fois, ne serait-ce que parce que l'ajout de la gestion OpenGL change tout de même certaines choses. Les parties qui vont suivre vont gérer l'affichage des balles, leurs collisions et la gestion de la souris.

Jc

- [Sommaire du tutoriel](#);
- [Introduction](#);
- [Les bases du moteur](#);
- [Les collisions et un menu](#);
- [Améliorer les collisions](#);
- Le score, la souris et les joueurs;
- Le réseau;
- Conclusion.

## 4 - Téléchargements

Voici le code source pour ce premier tutoriel: [\(3 Ko\)](#).

Voici la version pdf de cet article: [\(35 Ko\)](#).

Si vous avez des suggestions, remarques, critiques, si vous avez remarqué une erreur, ou bien si vous souhaitez des informations complémentaires, n'hésitez pas à me contacter !

## 5 - Remerciements

J'aimerais remercier [loka](#) pour sa double relecture de cet article !