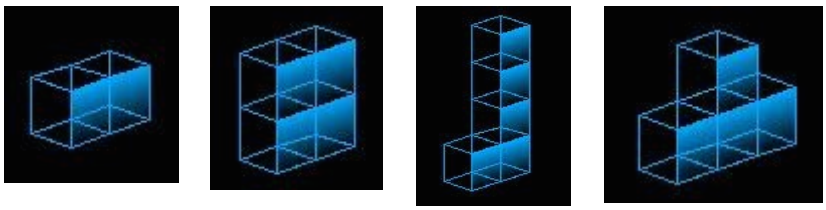
TETRIS OPENGL

Aurélien CEDEYN : aurelien.cedeyn@free.fr

Sommaire

Les objectifs du projet

Lors de ce projet nous avons créé un tétris en OpenGL tout d'abord en faisant descendre les pièces suivantes sur l'écran et sans accumulation :



L'espace de jeux devait avoir 8 unités en largeur et 16 en hauteur. Une fois ces pièces correctement implémentées, descendant sur notre espace de jeux de 16x8, il fallait les faire apparaître de façon aléatoire à l'écran, c'est à dire qu'il n'y ai pas d'ordre d'affichage précis des pièces et qu'elles ne devaient pas apparaître à un endroit constant en haut de l'écran lors du début de la descente.

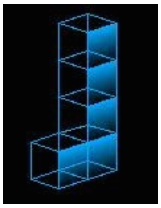
Une fois cette étape implémentée nous avons du gérer l'intervention de l'utilisateur sur les pièces ainsi que l'accumulation des pièces en bas de l'espace de jeux. Bien évidemment cette gestion implique des tests de collision sur les bord de l'espace de jeux.

Enfin ces deux étapes finies, nous devions gérer la rotation des pièces, la suppression des lignes effectuées par le joueur et la descente des pièces restantes.

Les structures de donnée

1.Les pièces

Chaque pièce est représentée par une matrice 4x4 composée de 0 et de 1.
Ex:
La pièce suivante



est représentée par la matrice :

0 0 1 0

0 0 1 0

0 0 1 0

0 1 1 0

Pour stocker toutes les pièces du tétris, j'ai choisi un tableau en trois dimension:
const char PIECES [NB-PIECES][HAUTEUR_PIECE][LARGEUR_PIECE]

Les pièces sont gérées par une structure pièce contenant les éléments suivants:

```
struct _piece{
  float x;
  float y;
  int tab[HAUTEUR_PIECE][LARGEUR_PIECE];
  int n_piece;
  int angle;
  char flag_stop_bas;
  char flag_stop_gauche;
  char flag_stop_droit;
  char flag_stop_rot;};
```

x: les coordonnées x du coin supérieur gauche de la pièce
y: les coordonnées y du coin supérieur gauche de la pièce

tab: l'ensemble des points de la pièce représentée par une matrice binaire 4x4

n_piece: le numéro de la pièce tiré au hasard lors de l'initialisation de la pièce

angle: l'angle de rotation de la pièce

flag_stop_*: ensemble de flag permettant de gérer les différentes collisions (gauche, bas, droit, rotation)

2.Le plateau de jeu

Le plateau de jeu est, comme les pièces représenté par une matrice. Cette matrice possède les dimensions HAUTEUR_PLATEAU+1 x LARGEUR_PLATEAU. Par défaut la hauteur du plateau est définit à 16 et la largeur à 8. La hauteur possède une ligne en plus afin de simplifier les collision des pièces.

Voici donc la structure utilisée :

```
struct _plateau{

  int largeur;

  int hauteur;

  int tab[HAUTEUR_PLATEAU][LARGEUR_PLATEAU];

};
```

largeur, hauteur: largeur du plateau, nous avons décidé de définir la largeur et la hauteur du plateau en tant que variable afin d'éventuellement permettre au joueur de définir la hauteur et la largeur qu'il souhaite. Dans la suite du programme nous avons implémenté chaque fonctions selon la largeur et la hauteur du plateau.

tab: la matrice contenant toutes les pièces déjà tombées

L'implémentation du jeu

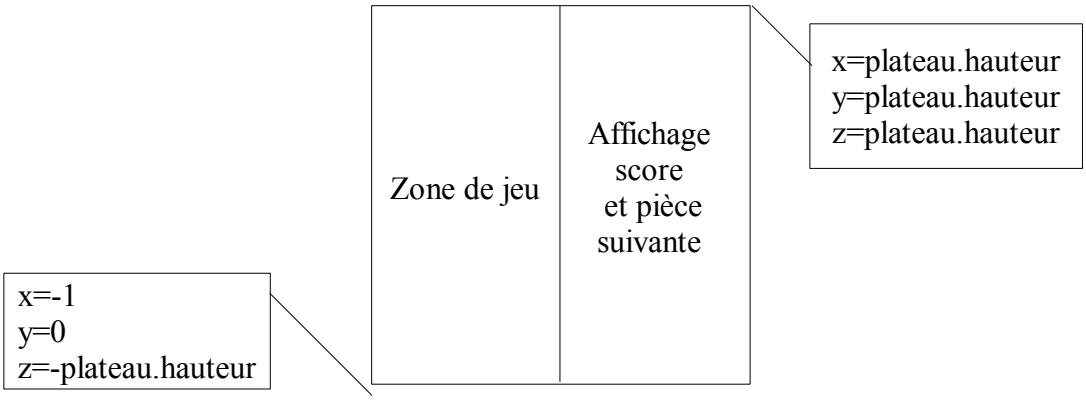
1.L'espace de jeux en 3d

Afin de simplifier la gestion des collisions et de la mise à jour de la matrice représentant le plateau, nous avons choisi de disposer d'un espace de jeu possédant les mêmes dimensions que cette matrice.

Cet espace est aisément configurable grâce à la fonction glOrtho().

Cette fonction à besoin de 6 paramètres définissant les coordonnées du coin inférieur gauche et du coin supérieur droit :

```
glOrtho(-1,plateau.hauteur,0,plateau.hauteur,-plateau.hauteur,plateau.hauteur);
```



L'espace d'affichage est donc d'une taille de plateau.hauteur +1 x plateau.hauteur.

Cet espace est donc séparé en deux parties:

- une partie dédié au tétris lui-même
- l'autre dédié à l'affichage du score et de la pièce suivante

Afin de donner un effet en 3d, nous avons dû déplacer la caméra virtuelle. Pour ce faire, nous avons utilisé la fonction gluLookAt() qui permet de placer la caméra virtuelle en prennant en compte les coordonnées par défaut d'OpenGL.

gluLookAt(-1 , 0.5, 1, 0, 0, 0, 0, 1, 0);

Nous avons choisi de placer l'observateur légèrement en avant et su la gauche du jeux.

Nous avons vu comment nous avons choisi la représentation de l'espace de jeu. Nous allons donc voir les différentes étapes du jeu.

2.La gestion de l'affichage des pièces

Pour gérer l'affichage de chaque éléments du tétris, nous avons choisi d'implémenter l'affichage dans une fonction différente pour chaque type d'élément ainsi nous avons les fonctions d'affichage suivantes:

void affiche_plateau_graphic(void): Procédure d'affichage du plateau

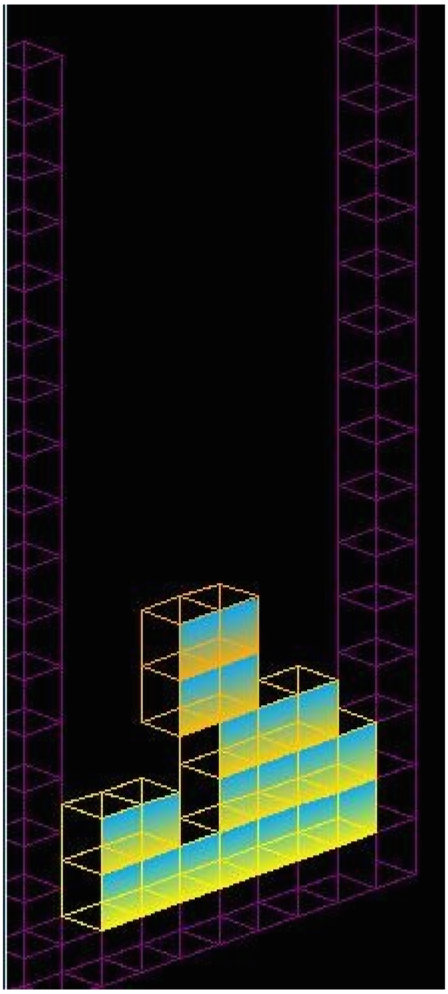
void affiche_piece_suiv(void): Procédure d'affichage de la pièce suivante

void affiche_piece(void): Procédure d'affichage de la pièce courante

Nous allons détailler ici la fonction d'affichage du plateau et celle de la pièce courante dans la mesure où la fonction d'affichage de la pièce suivante est identique.

- Affichage du plateau

Pour afficher le plateau, on effectue un simple parcours de la matrice du plateau et lorsque l'on rencontre un 1 dans la matrice on affiche un carré au coordonnées (n°colonne, n°ligne) du 1 dans la matrice puisque les coordonnées de la matrice sont identiques à celles de l'espace de jeux (cf <u>1. L'espace de jeux en 3d</u>).



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	1	1	1	0
0	1	1	0	1	1	1	1
0	1	1	1	1	1	1	1

- Affichage d'un cube dans la fonction d'affichage du plateau

Pour afficher un cube, nous dessinons d'abord un carré en dégradé (du bleu vers le jaune qui rougit en fonction de la hauteur de la pièce) puis nous affichons un cube en fil de fer grâce à la fonction glutWireCube().

Pour l'affichage du carré, comme pour le cube, nous dessinons ce dernier toujours au même endroit (c'est à dire au centre du repère d'affichage) et nous utilisons la fonction glTranslatef qui permet de déplacer ce repère aux coordonnée voulues.

Lors de l'utilisation de la fonction glTranslatef, nous utilisons toujours cette suite de fonctions:

```
glPushMatrix();
glLoadIdentity();
glTranslatef(j*Dimcarre, i*Dimcarre, 0);
glBegin(GL_POLYGON);
/*
Corps du dessin
*/
glEnd();
glPopMatrix();
```

glPushMatrix() permet de sauvegarder l'état des diverses matrices d'OpenGL

glLoadIdentity() réinitialise toutes les matrices d'OpenGL

glPopMatrix restaure les matrices d'OpenGL sauvegardées lors du dernier appel à glPushMatrix

Sans l'utilisation de ces fonctions, après le premier appel à glTranslatef, les coordonnées sont changées et le prochain appel à cette fonction calculera la position à adopter en fonction des dernières coordonnées.

- L'affichage de la pièce courante

La gestion de cet affichage est quasiment identique à l'affichage du plateau.

Nous nous plaçons d'abord aux coordonnées de la pièce (données par piece.x et piece.y) puis nous affichons, comme pour l'affichage du plateau, un carre puis un glutWireCube si dans le parcours de la matrice de la pièce nous rencontrons un 1.

3. Les différentes étapes

1. L'initialisation

L'étape indispensable lors de la programmation du tétris consiste à initialiser toutes les variables nécessaires. Les différents éléments à initialiser sont :

- Le plateau de jeu
- La pièce actuelle descendante
- La pièce suivante

L'initialisation du plateau se fait dans la fonction init(). Celle de la pièce courante et de la pièce suivante dans la fonction init_pièce().

La pièce courante et la pièce suivante sont toutes les deux du type pièce vu précédemment. Il nous faut donc initialiser chacun des champs de la structure pièce.

2. La descente d'une pièce

Lors du déroulement du jeu, la première chose dont il faut se préoccuper est la descente d'une pièce quelconque.

La descente s'effectue en fonction de la définition de FAC_VISTESSE. La valeur de FAC_VITESSE va définir la vitesse de descente de la pièce. Plus sa valeur sera élevée, plus la pièce descendra vite.

La descente proprement dite s'effectue dans la fonction Idle(). Nous utilisons cette fonction afin de gérer le comportement que doit utiliser glut lorsque l'utilisateur n'agit pas sur l'environnement du jeu. C'est dans cette fonction qu'est géré également la mise à jour de la matrice du plateau et la suppression des lignes.

Grâce à l'utilisation de notre structure pièce, il est très simple de faire descendre une piece: il suffit de soustraire à la coordonnée y de la pièce le facteur de vitesse FAC_VITESSE.

3. L'interaction de l'utilisateur avec la pièce

Afin de gérer l'interaction de l'utilisateur sur l'environnement du jeux, GLUT met à notre disposition diverses fonctions de gestion d'évènements

- **La fonction glutKeyboardFunc()**

Cette fonction permet d'établir le comportement des touches 'standard', c'est à dire toutes les touches hormis les touches directionnelles, SHIFT, CTRL, ALT, etc...

Cette fonction admet une fonction en paramètre qui sera charger de gérer le comportement des touches. Cette fonction possède le prototype suivant:

```
void fonction(unsigned char key, int x, int y)
```

Ainsi nous définissons la fonction Key possédant ce prototype. Cette fonction gère uniquement la touche ESC qui permet de quitter le programme.

- **La fonction glutSpecialFunc()**

La gestion de flèches directionnelles se fait sur le même principe que la fonction précédente. En effet elle prend également une fonction en paramètre qui gère ces touches. Pour pouvoir identifier ces touches, glut dispose des constantes suivantes:

GLUT_KEY_RIGHT: touche de direction droite (déplacement de la pièce à droite)

GLUT_KEY_LEFT: touche de direction gauche (déplacement de la pièce à gauche)

GLUT_KEY_UP: touche de direction haut (rotation de la pièce)

Lorsque l'utilisateur appuiera sur la touche de direction droit, grâce à la gestion de la pièce avec une structure, il nous suffit de tester si la pièce peut bouger (fonction collision_droit) et d'incrémenter les coordonnées x de la pièce d'une unité.

De la même façon quand l'utilisateur souhaite déplacer la pièce vers la gauche, on décrémente la valeur de l'abscisse x de la pièce après avoir appeler la fonction collision_gauche.

Voyons comment est implémentée la fonction collision_droit():

```
void collision_droit(void){

  int i,j;

  for (i=0;i<HAUTEUR_PIECE ; i++){
```

```
for (j=0 ; j<LARGEUR_PIECE ; j++){
  if (piece.tab[i][j]){
                if ( ((int)(piece.x + j + 1)>=plateau.largeur) || (plateau.tab[(int)(piece.y - i)][(int)(piece.x + j + 1)])){
                  piece.flag_stop_droit=1;
                  return;
                }
                piece.flag_stop_droit=0;
}}}
```

On peut voir très simplement que les collisions sont gérées par un simple parcours de la matrice de la pièce courante. En effet, on teste pour chaque petit cube de la pièce si ce dernier peut se déplacer à droite, si ce n'est pas le cas, on place le flag de collision à droite de la pièce à 1 empêchant ainsi l'utilisateur de se déplacer.

4. La mise à jour de la matrice de plateau

Cette mise à jour intervient lorsque la pièce se trouve en collision vers le bas, c'est à dire qu'elle ne peut plus descendre.

Comme pour les collisions gauche et droite, nous avons implémenté une fonction détectant la collision vers le bas. On procède de façon identique, nous testons chaque petit cube afin de savoir si un cube (dans la matrice du plateau) se trouve déjà en dessous de lui, si c'est le cas, on place le flag de collision bas de la pièce à 1.

Cette fonction est appelée dans la fonction Idle() ce qui nous permet de mettre ensuite, lorsqu'une pièce est en état de collision de l'enregistrer dans la matrice du plateau de jeu.

5. La suppression d'éventuelles lignes

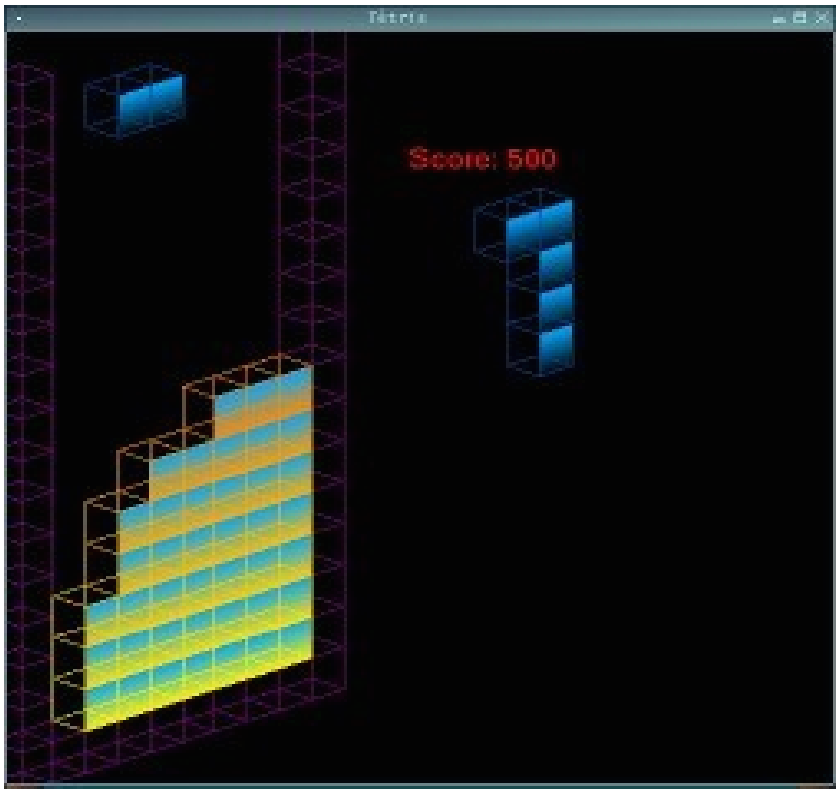
Pour supprimer les lignes de la matrice du plateau de jeu, nous procédons en deux étapes:

- Détection d'une ligne de 1
- Suppression de la ligne

Ces deux étapes sont représentées par les fonctions detect_line() et supp_lines().

La fonction detect_line renvoi l'indice de la première ligne de 1 dans la matrice du plateau et la fonction supp_lines décale l'ensemble de la matrice, pour chaque ligne détectée, afin de simuler la suppression de chaque ligne.

Dans cette fonction, nous utilisons la fonction glPrint() qui permet, en combinant les fonction OpenGL et GLUT d'écrire à une position donnée un texte à l'écran à la façon d'un printf(). Cette fonction va nous permettre l'affichage du score.


Tetris
Score: 500

L'évolution du projet

Le mini-projet qui nous a été demandé ne devait pas contenir énormément de fonctionnalités, mais on pourrait, dans le futur, s'intéresser plus à l'aspect graphique qu'OpenGL nous offre tel que l'ajout de texture sur les pièces, l'éclairage, etc...

Au niveau de la jouabilité, nous pourrions également ajouter diverses options, tel que des pièces bonus, l'accélération de la pièce lorsque le joueur atteint certaines tranche de score ou bien encore faire apparaître des pièces piège aléatoirement durant le jeux.

Concernant l'interface, il serait également possible de proposer à notre joueur de choisir la dimension du Tétris, le nombre de pièce, etc...

Conclusion

Ce projet nous a permis de découvrir un nouvel environnement OpenGL dont nous avons pu apprécier la puissance. Malgré un survole de ses capacités, nous avons implémenté un Tétris facilement et avec relativement peu de fonctions. L'éventail des capacité d'OpenGL est bien sûr bien plus vaste et ce projet mérite un approfondissement certain au niveau des capacités qui nous sont offertes.

Dans le futur nous allons nous diriger vers une amélioration graphique de ce jeu afin de le rendre plus agréable aux joueur, mais aussi leur offrir beaucoup plus de choix quand à la jouabilité d'un Tétris 'classique'.