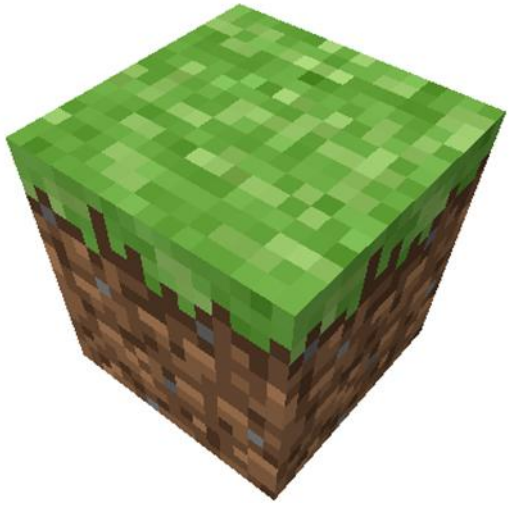
Rapport du Projet1

Réalité Virtuelle

Minecraft simplifié



Etudiants : Wei ZHOU & Ye ZHOU

Encadrant : Jean-Marie Normand

Le 27 janvier 2016

Index

I. Sommaire

Tout d'abord, notre sujet est < **Jeu vidéo OpenGL/OSG** >, ça va dire que nous devons écirie un jeu vidéo avec le langage OpenGL ou OSG, et il nous fait plaisir que nous pouvons définir quel jeu vidéo nous allons écrire.
Donc nous choisissions le jeu vidéo des cubes très connu – **Minecraft** comme notre thème du projet. Et pour écrire ce projet, nous avons beaucoup utilisé le bibliothèque **GLUT** (Open**GL U**tility **T**oolkit).

Pour le raison de temps, nous n'avons pas réalisé tous les fonctions dans le Minecraft, seulement une partie des fonctions, par exemple : se déplacer, enlever un cube, mettre un cube, créer des édifices avec des cubes différents.....
Nous allons les présenter en détail au texte suivant.

Nous ne pouvons pas nier qu'il existe quelque désavantages dans notre projet, mais nous nous avons fait du mieux et bien fait attention, et nous espérons que notre projet vous fera plaisir ☺

II. Modules

Nous avons écrit beaucoup de modules dans notre projet, nous croyons que c'est plus claire et plus facile pour lire et comprendre notre structure et les fonctions.

1. Eléments du monde : Cubes et plantes

Introduction : le monde virtuel est composé par les cubes et plants.

Classes: Cube, Plants

Location:

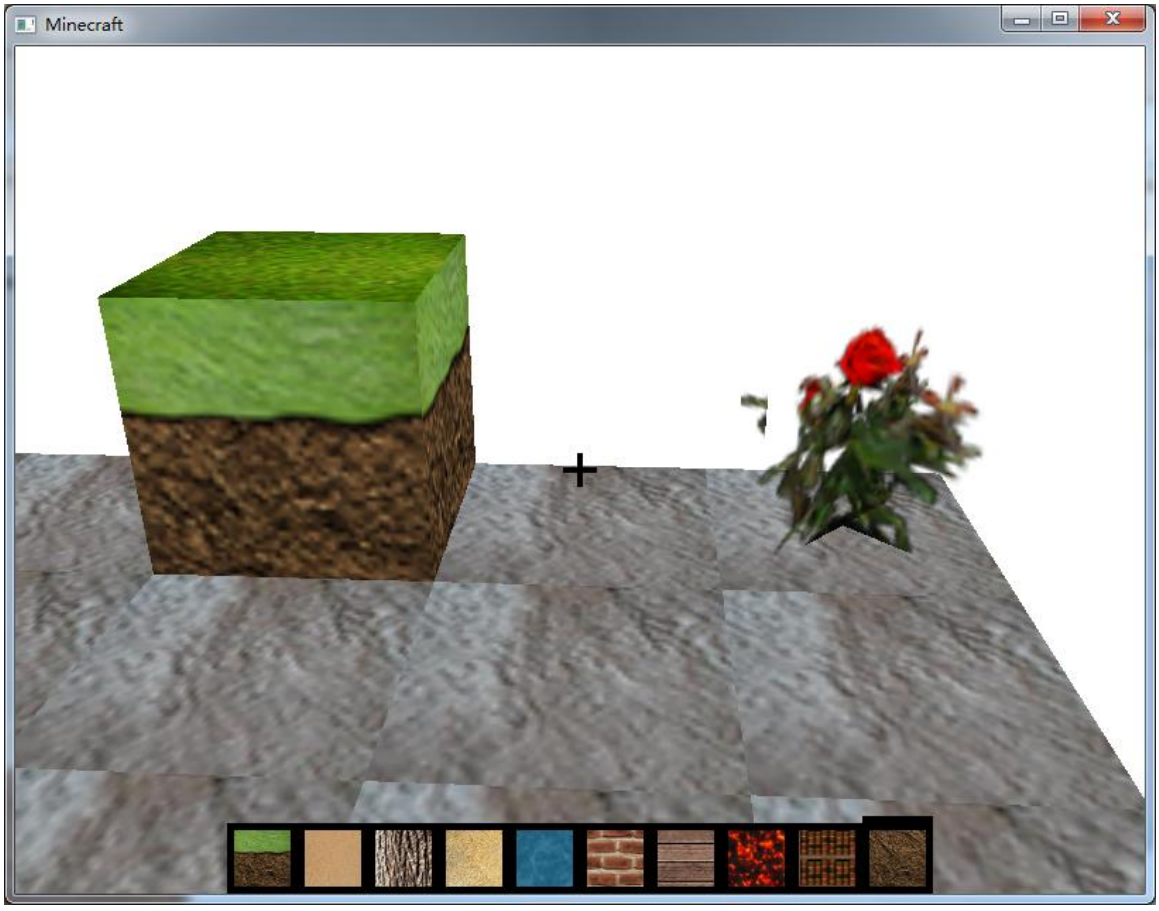
- cube.h cube.cpp
- plants.h plants.cpp

Membres principaux :

- Les coordonnées des cubes et plantes :
  float positionX, positionY, positionZ;
- Le type du cube et plante (bois, pierre, sol, eau...) :
  int type;
- Un identificateur pour juger si un objet est sélectionné ou pas :
  int chosen;

Méthodes principales:

- obtenir la texture à partir du type de ce cube :
  gettexture()
- affichage d'un cube/plante :
  afficheCube()
- obtenir les propriétés :
  getpositionX()
  getpositionY()
  getpositionZ()
  gettype()



Il est un cube et un plante

2. Le monde virtuel : world

Introduction : Comme Minecraft est un jeu bac à sable (Sandtable game). On a besoin tout d'abord un monde virtuel.

Classe: World

Location: world.h world.cpp

Membres principaux ::

- Enregistrer les cubes et plantes :

  map<int,Cube> cubes;

  map<int,Plants> plants;
- Un identificateur pour juger si un objet est sélectionné et ses coordonnées :

  int chosenx; int choseny; int chosenz;

  int chosen;
- Le cube ou plante avant le cube choix :

  int frontx; int fronty; int frontz;
- Le point de vu :

  View viewer;

Méthodes principales:

- Ajouter et supprimer les cubes and plantes :

  World ::addCube()

  World ::addplants()

  World ::deleteCube()

  World ::deleteplants()

- Affichage du monde virtuel :
  World ::generation()
  World ::afficheworld()
- La collision :
  World ::collision()

3. Le point de vue : viewer

Introduction : la caméra. En fait c'est un personne virtuel dans le monde.

Classe: View

Location:

- world.h world.cpp
- View.h view.cpp
- Mouse.h
- Keyboard.h

Membres principaux:

- Les coordonnées :
  float mypositionX, mypositionY, mypositionZ;
- La direction du point de vue :
  float object, object, objectZ;
- La vitesse :
  float speedx, speedy, speedz;
- L'accélération :
  float accelerx, accelery, accelerz;

Méthodes principales:

- Contrôle de la déplacement du observateur avec la souris :

  Mouse::mousemove()

- Déplacement : avancer, reculer, tourner à gauche, tourner à droité, saut à haut :

  Keyboard::keyboardmovement()

- Choisir un objet : On tracer une ligne droite à partir de la position de observateur vers la direction de cet objet :

  World:: viewerchoose()

- Clique évènement : cliquer gauche pour enlever un cube et cliquer droite pour mettre un cube :

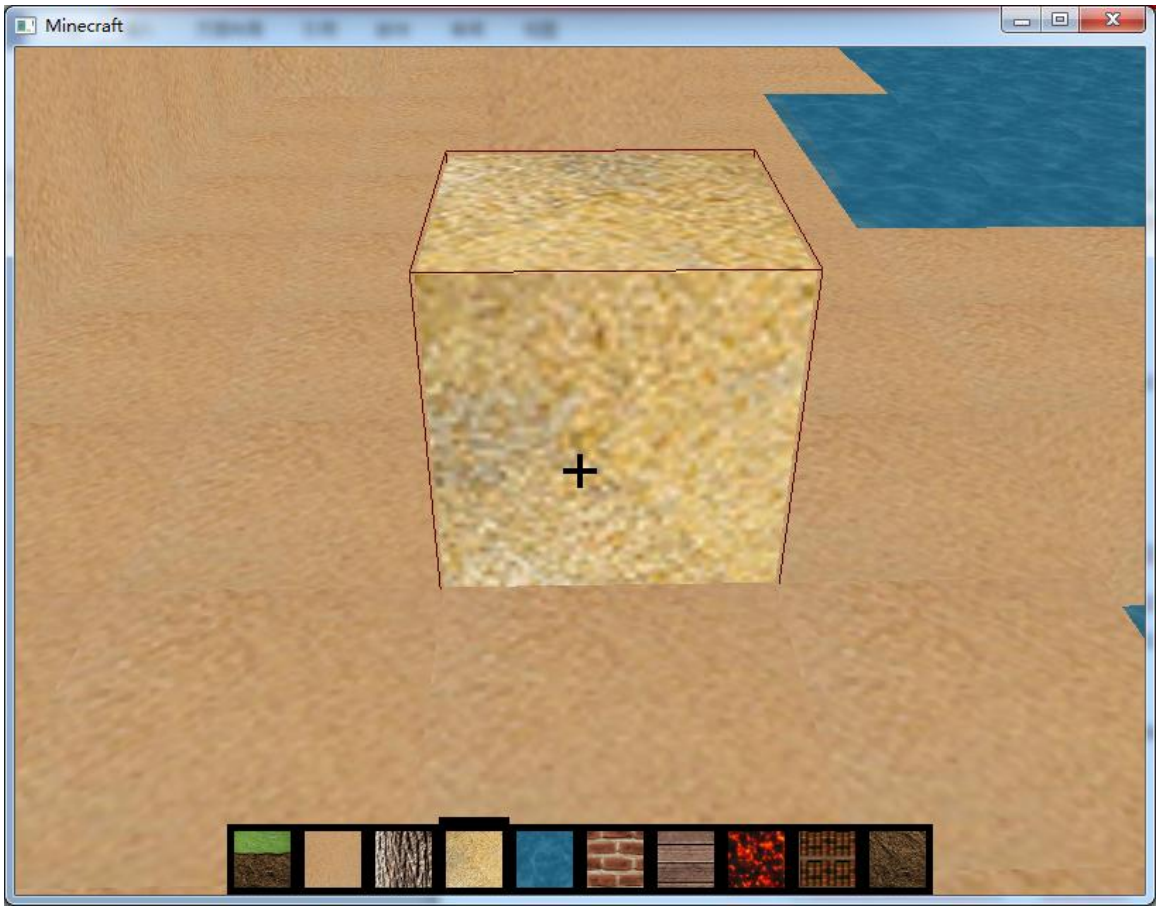
  Mouse:: leftclick()

  Mouse::rightclick()

  World:: vieweraddCube()

  World:: vieweraddplants()

  World:: viewerdelete()



Dans cette image, cette case est cochée. Il y a une bordure rouge autour

4. Les autres modules

1) SOIL

Introduction: Parce que nous avons besoin d'utiliser une texture transparent, donc l'utilisation image **png** est nécessaire. Mais le format **png** est difficile à lire, donc nous avons utilisé un tierce bibliothèque: soil

Location : main.cpp

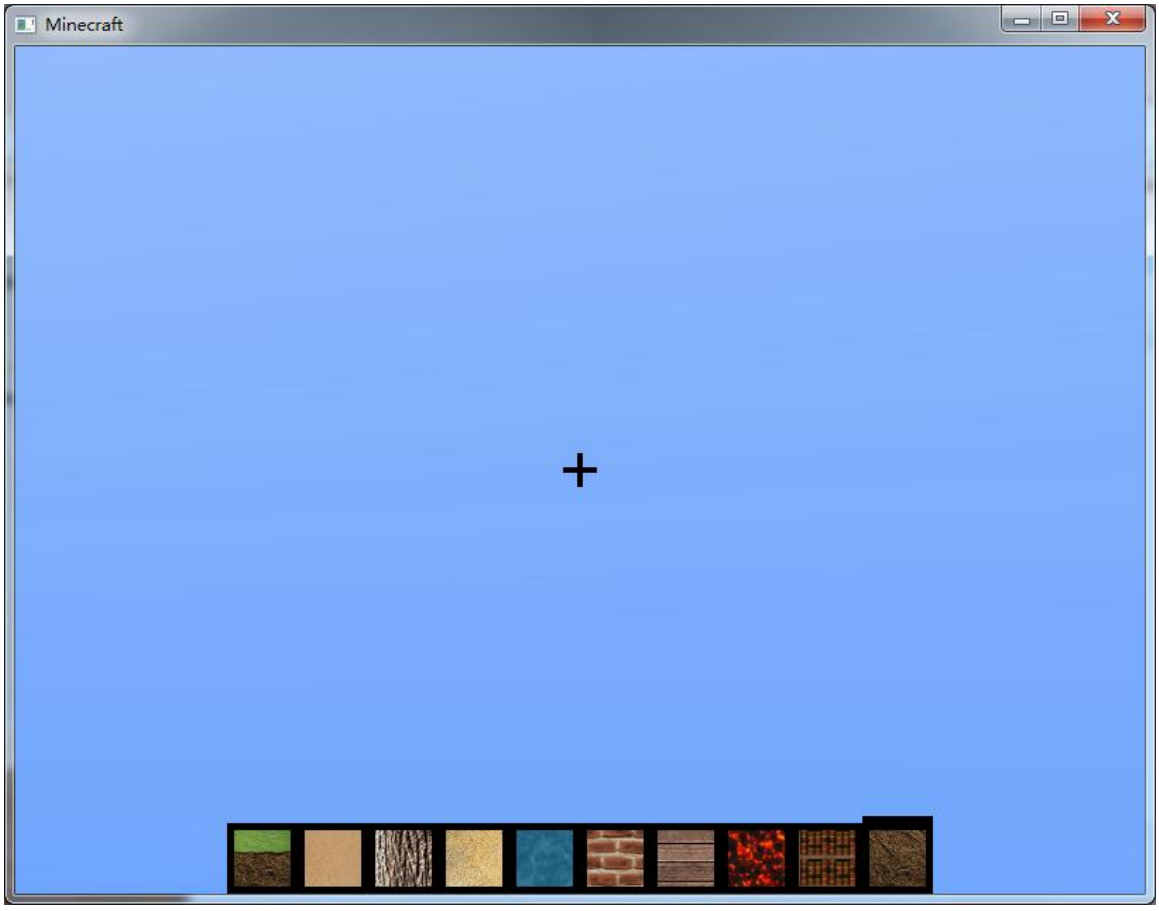
Méthodes principales: SOIL_load_image ()

2) 2D GUI

Introduction : Le 2D GUI est aussi important pour l'interaction avec l'utilisateur.

Location: GUI.h

Méthodes principales: drawGUI()



La petite croix dans au centre indique le curseur. La barre d'outils en bas montre les types des cubes. Le type du cube sélectionné sera marqué avec la bordure rouge.

3) Enregistrer et rapporter d'un monde

Introduction : ce module nous permette d'enregistrer et rapporter un monde à partir d'un fichier. Le fichier est save.txt.

Location: world.h world.cpp

Méthodes principales:

- void savetheworld();
- void readtheworld();

4) skybox

Introduction : Le skybox est un boîte avec texture qui presente le ciel.

Location : skybox.h

Methods Majors: skybox()

C'est le skybox et quelques nuages

5) Font

Introduction : Le skybox est un boîte avec texture qui presente le ciel.

Location: skybox.h

Méthodes principales:

- Font::BuildFont ()
- Font::glPrint()
- Font::KillFont()

Finalment on n'a pas utilisé le font.

III. Problèmes & Solutions

1. La combination entre 2D et 3D

Ce n'est pas facile pour combiner le 2d GUI et 3d monde virtuel ensemble avec glut. Nous avons fait beaucoup de recherche sur l'internet. Mais nous n'avons pas trouvé la méthode. Merci à monsieur Normand, il nous a conseillé de changer le mode de matrix avec :

```
glMatrixMode(GL_PROJECTION);
```

et

```
glMatrixMode(GL_MODELVIEW);
```

Nous l'avons trouvé très utile, cette méthode, mais il y a encore un problème : après avoir affiché le 2D GUI, le 3D monde est devenu une image 2D ! Ensuite nous avons essayé beaucoup de méthodes, et finalment, nous avons trouvé la solution :

```
glDisable(GL_DEPTH_TEST);
glDisable(GL_BLEND);
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
gluOrtho2D(0.0, 800, 0.0, 600);
```

```
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadIdentity();
..... // affichage de 2d interface.

glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
glPopMatrix();
glEnable(GL_DEPTH_TEST);
glEnable(GL_BLEND);// you enable blending function
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

L'étape clé est : interrompre le GL_DEPTH_TEST avant afficher le 2d et l'activer après.

2. L'affichage des cubes transparents

Dans la version archaique, nous avons utilisé une image BMP comme les textures. Quand nous avons écrit les plants, il faut utiliser transparence pour les afficher mieux. Mais il n'y a pas l'alpha dans BMP. Donc nous l'avons changé avec PNG.

Malheureusement, c'est plus compliqué d'apporter une image PNG. Ce format est difficile à lire.

Finalment nous avons utilisé un third-part-library : SOIL. Après avoir complié le code original et l'ajouté dans le projet, on peut rapporter un PNG avec SOIL_load_image() et glTexImage2D()

Mais après nous avons fait ça, le 2D GUI disparu !! Nous ne savons pas le raison et nous avons fait beaucoup de recherches sur l'internet. Finalement nous avons ajouté glDisable(GL_BLEND); avant l'affichage de 2d GUI et glEnable(GL_BLEND); Ca marche en fin!

Ceci est un matériau transparent de une plante, nous pouvons voir qu'il est partiellement transparent, pas occupé tout le cube.

3. Problèmes de déplacement

Parce que le point de vue du persenne changera la

définition de "front", donc pour lui c'est difficile de juger la direction de marcher. Ici nous aovns utilisé la méthode de vecteur. On utilise PositionX, PositionY, PositionZ : la position d'utilisateur dans le monde du jeu. ObjectX, ObjectY, ObjectZ : la direction de l'orientation. Dans le processus de déplacement de vue de point, nous obtenons la position et la direction de l'orientation d'utilisateur par calculer la déplacement et rotation d'utilisateur.

Par exemple , la rotation de vecteur:

```
float x = world.viewer.objectX;
float z = world.viewer.objectZ;
world.viewer.objectX = cos(ang) * x + sin(ang) * z;
world.viewer.objectZ = -sin(ang) * x + cos(ang) * z;
```

Aussi, nous utilisions les lois physique pour contrôler le mouvement des personnages à sa vitesse et accélération :

```
if(speedx != 0 || speedz != 0)
{
        accelerx = -0.5 * speedx;
        accelerz = -0.5 * speedz;
}
mypositionX += speedx;
mypositionY += speedy;
mypositionZ += speedz;
speedx += accelerx;
speedy += accelery;
speedz += accelerz;
```

Nous avons essayé à contrôler directement les appuis de déplacement. Mais OpenGL ne peut pas identifier qu'un seul événement. Plusieurs clés pressé dans la même temps sont invalide. Donc, nous avons créé une tableau d'état pour enregistrer les appuis qui sont en cours de presser. Cela peut gérer plusieurs touches pressées :

```
int keys[200];
void setkeydown(char key){keys[key] = 1;return;}
void setkeyup(char key){keys[key] = 0;return;}
```

4. Lentement si trop de cubes

Quand nous étions en train de fonder le monde virtuel, il était bloqué tout à coup. Nous avons trouvé que le problème est parce que nous avons mis trop de cubes et plantes dedans, et l'espace de dépôt n'est pas assez grande, donc la réaction de ce jeu est de plus en plus lent. Donc nous commençions à penser utiliser une structure des données différentes.

Finalement nous avons trouvé le recipient : Map dans STL de C++. C'est une structure à partir du mot clé et le contenu. Ensuite nous utilisions la méthode **hash**, alors nous pouvons trouvé un Cube ou une Plant plus vite avec la position XYZ. Le **hash** algorithme est :

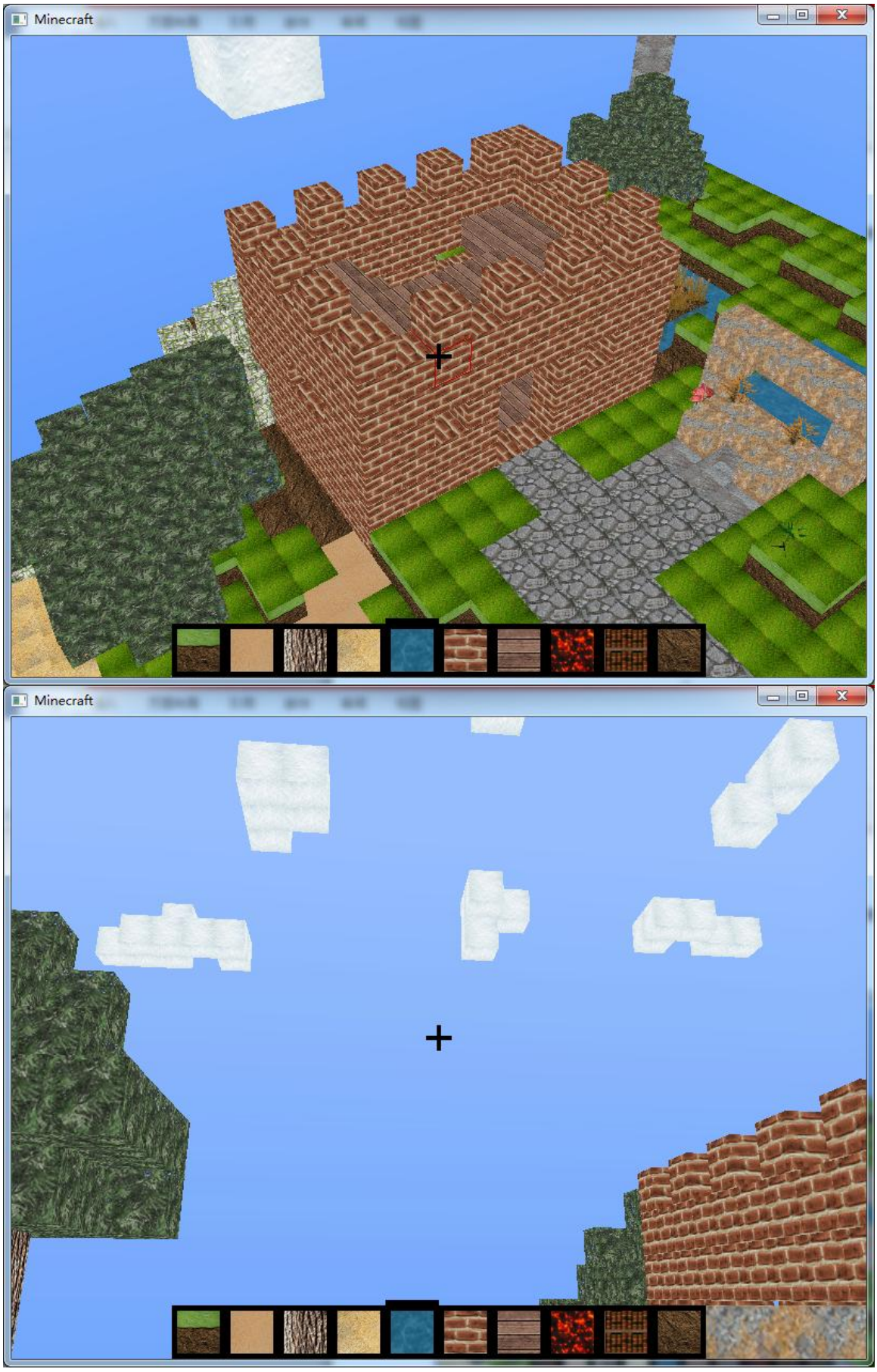
```
int num = 1000000*(MAX+xx) + 1000*(MAX+yy) + (MAX+zz);
plants.insert(map<int, Plants> :: value_type(num, p));
```

Par conséquent, le jeu peut fonctionner normalement.

IV. Résultat final

Enfin, nous avons construit notre beau monde dans le jeu!

Suivant est une vue aérienne :


Minecraft
Minecraft

Le manuel de la manipulation :

1. Appuyez sur **W, S, A, D**, pour aller de l'avant, arrière, gauche, aller à droite. Vous pouvez appuyer ensemble les deux cisailles.

2. Appuyez sur **espace** sur le clavier pour sauter, le pressez continûment pour voler.

3. Déplacez la souris pour changer l'angle de vue. Appuyez sur le bouton gauche de la souris pour enlever un cube, appuyez sur le bouton droite de la souris pour mettre un cube.

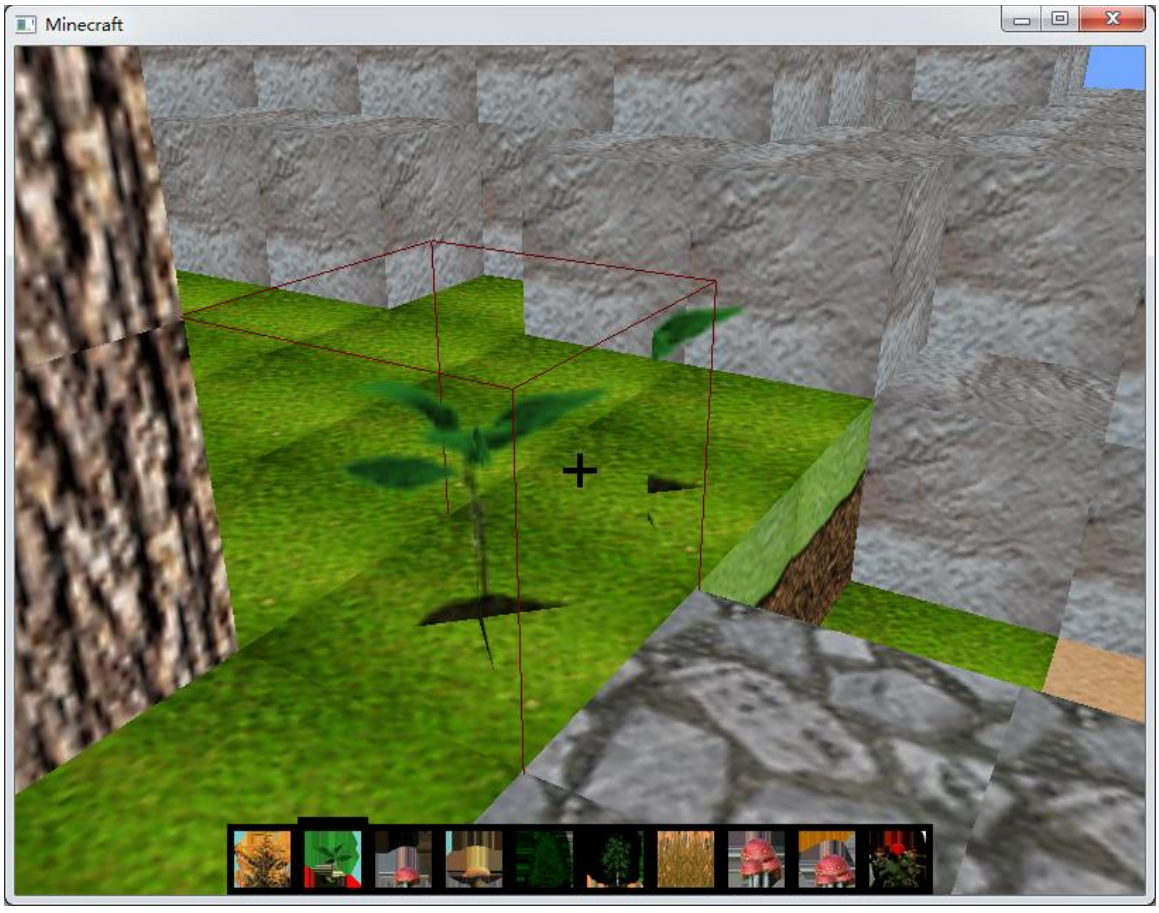
4. Appuyez sur les numéros **0-9** pour changer le type de cube sélectionné. Vous pouvez appuyer sur **+** et **-** pour retourner.

5. Appuyez sur Esc pour quitter le jeu, il va enregistrer vos modifications automatiquement, et la prochaine fois que vous ouvrez le jeu, ces changements seront conservés. Si vous ne voulez pas les enregistrer, vous pouvez quitter en appuyant **o**.

V. Désavantages et idées à réaliser

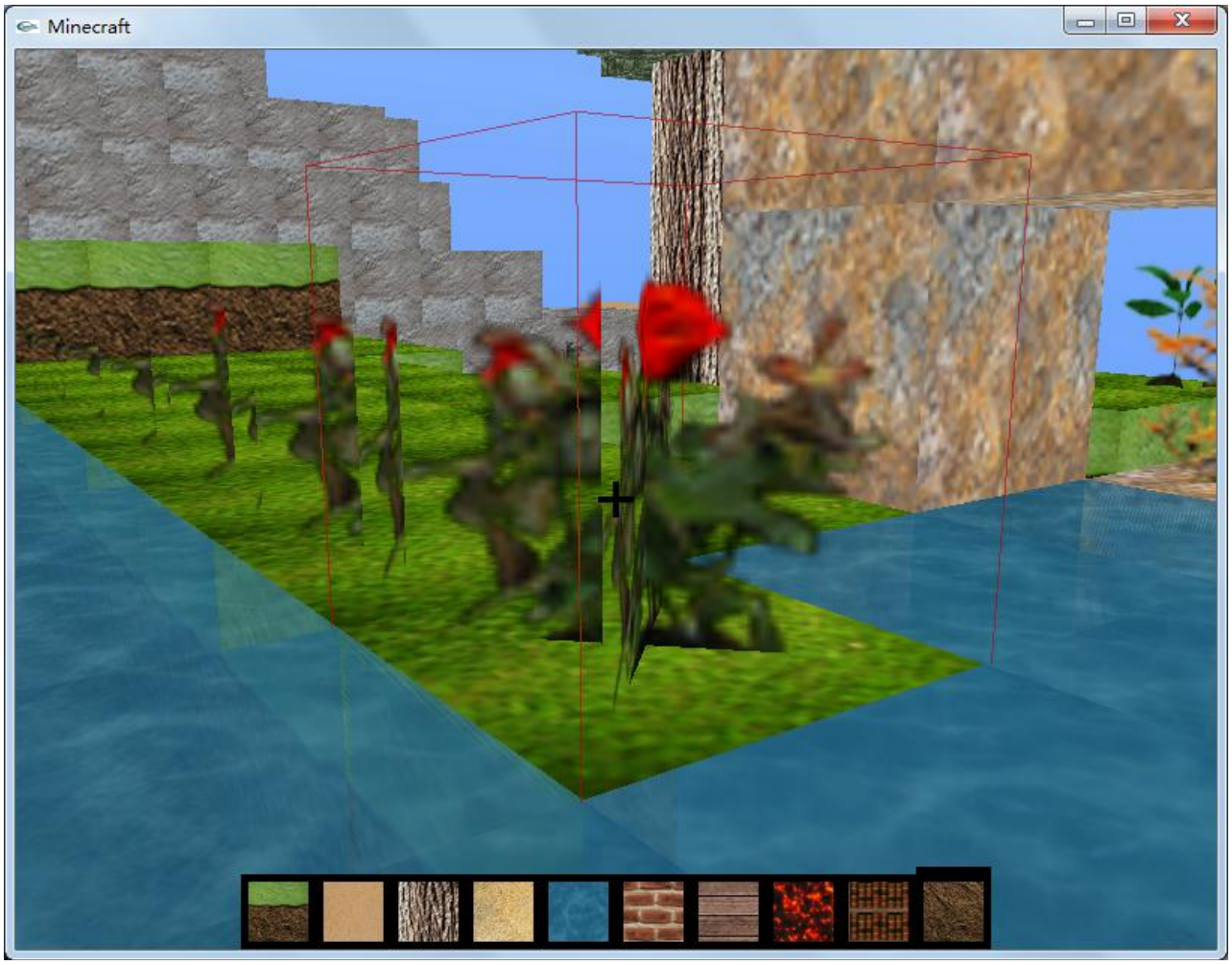
1. Cubes transparents :

Pour des cubes avec la diaphanéité, par exemple : des fleurs, des herbes ou des feuilles -- différents des cubes normal (sol, pierre, bois...), ils ne sont pas des cubes complets, il y a des epaces transparents, comme on peut voir de ces photos suivantes :



Mais dans notre projet, quand on change son angle visuel, de temps en temps, il existe un petit problème d'affichage :

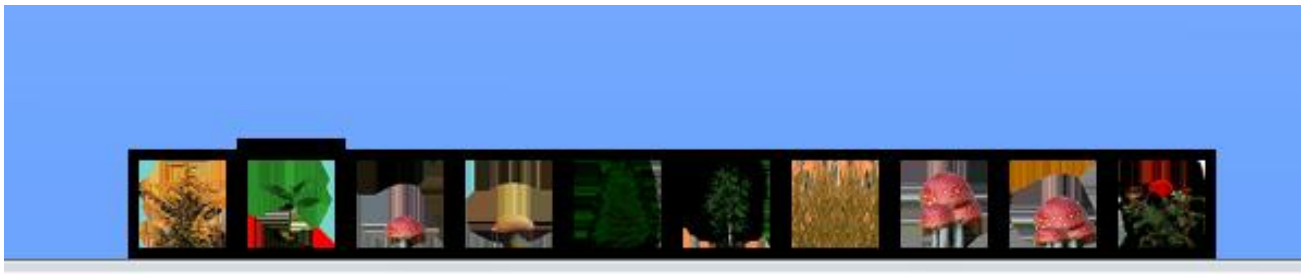
au centre d’objet, il y a une partie incomplet :



Nous ne savons pas le raison de ce problème, mais nous croyons que c’est le problème de notre moteur

2. Transparents dans le 2d GUI

Dans l'interface utilisateur, nous avons un problème étrange : quand nous imprimons l’image sur le 2D GUI, il y a des couleurs étranges dans la barre d’outil. Comme l’image dessous :



Ceci est un problème que nous ne pouvons pas résoudre

3. Trop seul..

Nous avons généré le monde qui est très beau, mais il y a encore un problème: le joueur est trop seul !! Nous pouvons également ajouter certains animaux et les gens dans le jeu, mais cette fois nous avons peu de temps. Si on peut continuer à faire notre projet, je vais d'abord ajouter les animaux et les humains.

4. Ne peut pas fonctionner sur d'autres OS

Nos jeu ne peut fonctionner que dans Windows. Bien que j'ai tenté d'exécuter en autres OS, ça marche pas. Je ne sais pas comment faire.

5. Manque musique

Nous avons essayé à ajouter la musique, mais quand on fait ça, le jeu s'arrêté et en attente de la musique, l'utilisateur ne peut jouer qu'après la musique est fini. Nous croyons que on a besoin de la technique parallèle.

VI. Conclusion

Après avoir fait ce projet, nous avons mieux compris et maîtrisé OpenGL. De plus, parce que le projet nous permettait beaucoup de liberté, donc nous somme lassé nous exprimer.

VII. Bibliothèque

1. [OpenGL] Combiner 2D et 3D : https://openclassrooms.com/forum/sujet/opengl-combiner-2d-et-3d-66523

2. Blending par Learn Opengl http://learnopengl.com/#!Advanced-OpenGL/Blending

3. OpenGL Texture Transparente http://www.developpez.net/forums/d415052/c-cpp/cpp/opengl-texture-transparente/

4. map - C++ Reference http://www.cplusplus.com/reference/map/map/

Enfin, merci à l'aide de professeur Jean-Marie Normand !