

Plateforme de Développement Continu

Comprendre la Théorie pour mieux Pratiquer
Sciences de l'Ingénieur
Cours - Tutoriels

OpenGL

Apprendre la programmation 3D en C++ avec OpenGL

J'aime, Je partage
Montez en Compétences

Auteur

Je suis **Gérard KESSE**,
Ingénieur en Développement Informatique C/C++/Qt,
Avec à la fois des compétences en Systèmes Embarqués et en Robotique.

Formé à Polytech'Montpellier, Je suis un professionnel de conception de projets logiciel applicatif ou embarqué dans les secteurs de l'Aéronautique, de la Robotique, des Drones et de la Vision par Ordinateur. Aussi, Je reste ouvert à d'autres types de secteurs tels que l'Énergie et les Finances.

Les Sciences de l'Ingénieur sont au cœur du métier d'ingénieur. Sur le site **ReadyDev**, la Plateforme de Développement Continu, dont j'en suis le concepteur, vous trouverez des cours et des tutoriels adaptés aux sciences de l'ingénieur.

J'aime, Je partage.

Gérard KESSE

[GitHub](#) | [LinkedIn](#) | [SiteWeb](#)



Sommaire

Auteur.....	2
Sommaire	3
Introduction	4
Installation sous Windows avec MinGW	4
Téléchargements	4
Installation de Notepad++	5
Installation de MinGW	5
Installation de FreeGLUT	5
Utiliser OpenGL avec FreeGLUT	7
Afficher une fenêtre	7
Ajouter une couleur de fond d'écran.....	9
Afficher un objet 3D	12
Ajouter une caméra à la scène 3D	16
Ajouter de la lumière à une scène 3D	19

Introduction

Le C++ est un langage de programmation orienté objet. OpenGL est une bibliothèque de création d'applications 3D. Le but de ce tutoriel est de vous apprendre la programmation 3D en C++ avec OpenGL.

Prérequis :

Apprendre à compiler un projet C++ avec QMake.

Installation sous Windows avec MinGW

OpenGL (Open Graphics Library) est un ensemble normalisé de fonctions de calcul d'images 2D ou 3D disponible sur de nombreuses plateformes où elle est utilisée pour des applications qui vont du jeu vidéo jusqu'à la CAO en passant par la modélisation.

OpenGL permet à un programme de déclarer la géométrie d'objets sous forme de points, de vecteurs, de polygones, de bitmaps et de textures. OpenGL effectue ensuite des calculs de projection en vue de déterminer l'image à l'écran, en tenant compte de la distance, de l'orientation, des ombres, de la transparence et du cadrage.

Dans ce tutoriel, nous utilisons la bibliothèque OpenGL avec l'utilitaire FreeGLUT, et l'IDE Qt Creator avec le compilateur C++ fourni par MinGW.

Téléchargements

Notepad++ :

<https://notepad-plus-plus.org/fr/>

MinGW :

<http://mingw.org/>

Qt Creator :

<https://download.qt.io/archive/qt/>

CMake :

<https://cmake.org/>

FreeGLUT :

<http://freeglut.sourceforge.net/>

Bullet :

<https://github.com/bulletphysics/bullet3>

Installation de Notepad++

Plugins Notepad++ :

TextFX

NppExport

Installation de MinGW

Packages MinGW :

mingw32-base

mingw32-gcc-g++

Installation de FreeGLUT

Dossier construction :

FreeGLUT/cmd_cmake.bat

FreeGLUT/cmd_build.bat

FreeGLUT/build/

Démarrer CMake :

FreeGLUT/cmd_cmake.bat

set PATH=C:\Program Files\CMake\bin;

set PATH=C:\Qt\Qt5.8.0\Tools\mingw530_32\bin;%PATH%

cmake-gui

pause

Configurer FreeGLUT avec CMake :

Where is the source code

C:/Users/gerar/Downloads/OpenGL/freeglut-3.0.0/freeglut-3.0.0

Where to build the binaries

C:/Users/gerar/Downloads/OpenGL/freeglut-3.0.0/freeglut-3.0.0/build

Configure

Specify the generator for this project

MinGW Makefiles

Specify native compilers

Next

Compilers

C

C:/Qt/Qt5.8.0/Tools/mingw530_32/bin/gcc.exe

C++

C:/Qt/Qt5.8.0/Tools/mingw530_32/bin/g++.exe

Finish

Configure

Configure

Generate

Construire FreeGLUT avec MinGW :

FreeGLUT/cmd_build.bat

set PATH=C:\Qt\Qt5.8.0\Tools\mingw530_32\bin

cd build

mingw32-make

pause

Dossier librairie FreeGLUT :

FreeGLUT/include/

FreeGLUT/build/lib/

FreeGLUT/build/bin/

Utiliser OpenGL avec FreeGLUT

Afficher une fenêtre

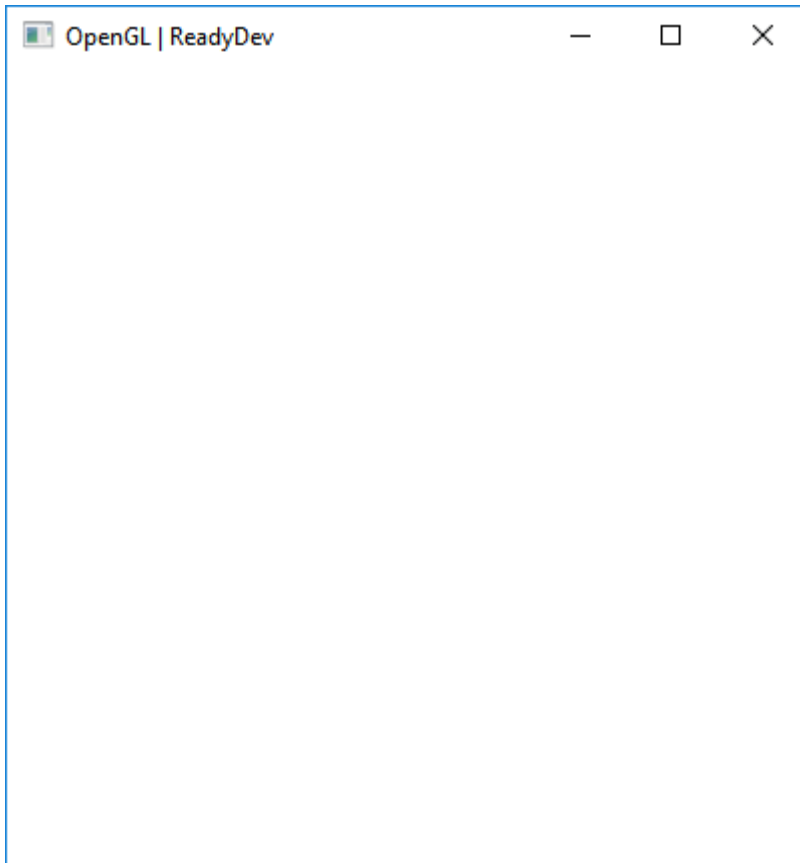
Objectif :

Afficher une fenêtre.

Implémentation :

Créer un gestionnaire de fenêtre (GWindow).

Afficher la fenêtre (show()).

Résultat :**Dossier projet :**

```
src/main.cpp  
win/GWindow.h  
win/GWindow.cpp  
win/bin/  
win/build/
```

Programme principale :

```
//=====
#include "GWindow.h"
//=====
int main(int argc, char** argv) {
    GWindow::Instance()->show(&argc, argv);
    return 0;
}
//=====
```

Affichage de la fenêtre :

```
//=====
void GWindow::show(int* argc, char** argv) {
    glutInit(argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA |
GLUT_DEPTH);
    glutInitWindowPosition(m_x, m_y);
    glutInitWindowSize(m_w, m_h);
    glutCreateWindow(m_title.toStdString().c_str());
    glutSetOption (GLUT_ACTION_ON_WINDOW_CLOSE,
GLUT_ACTION_GLUTMAINLOOP_RETURNS);
    glutDisplayFunc(DisplayFunc);
    glutMainLoop();
}
//=====
```

Initialisation des paramètres de la fenêtre :

```
//=====
GWindow::GWindow() {
    m_x = 0;
    m_y = 0;
    m_w = 400;
    m_h = 400;
    m_title = "OpenGL | ReadyDev";
}
//=====
```

Fonction de rappel de l'affichage de la fenêtre :

```
//=====
void GWindow::DisplayFunc() {

}
//=====
```


Ajouter une couleur de fond d'écran

Objectif :

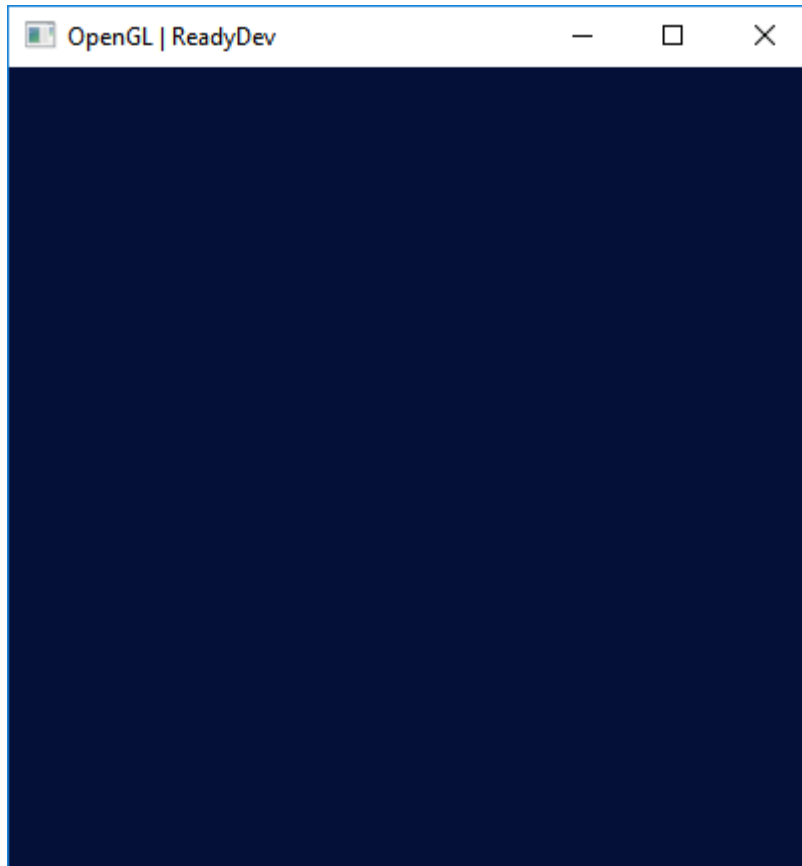
Ajouter une couleur de fond d'écran à une fenêtre.

Implémentation :

Créer un gestionnaire de fenêtre (GWindow).

Ajouter une couleur de fond d'écran (setBackground()).

Afficher la fenêtre (show()).

Résultat :**Dossier projet :**

src/main.cpp

src/GWindow.h

src/GWindow.cpp

win/bin/

win/build/

Programme principal :

```
//=====
#include "GWindow.h"
//=====
int main(int argc, char** argv) {
    GWindow::Instance()->show(&argc, argv);
    return 0;
}
//=====
```

Affichage de la fenêtre :

```
//=====
void GWindow::show(int* argc, char** argv) {
    glutInit(argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA |
GLUT_DEPTH);
    glutInitWindowPosition(m_x, m_y);
    glutInitWindowSize(m_w, m_h);
    glutCreateWindow(m_title.toStdString().c_str());
    glutSetOption (GLUT_ACTION_ON_WINDOW_CLOSE,
GLUT_ACTION_GLUTMAINLOOP_RETURNS);
    setBackground();
    glutIdleFunc (IdleFunc);
    glutDisplayFunc (DisplayFunc);
    glutMainLoop();
}
//=====
```

Ajout d'une couleur de fond d'écran :

```
//=====
void GWindow::setBackground() {
    float m_red = 5.0/255.0;
    float m_green = 16.0/255.0;
    float m_blue = 57.0/255.0;
    float m_alpha = 255.0/255.0;

    glClearColor(m_red, m_green, m_blue, m_alpha);
}
//=====
```

Fonction de rappel de la tâche de fond :

```
//=====
void GWindow::IdleFunc() {
    glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);
    glutSwapBuffers();
}
//=====
```

Afficher un objet 3D

Objectif :

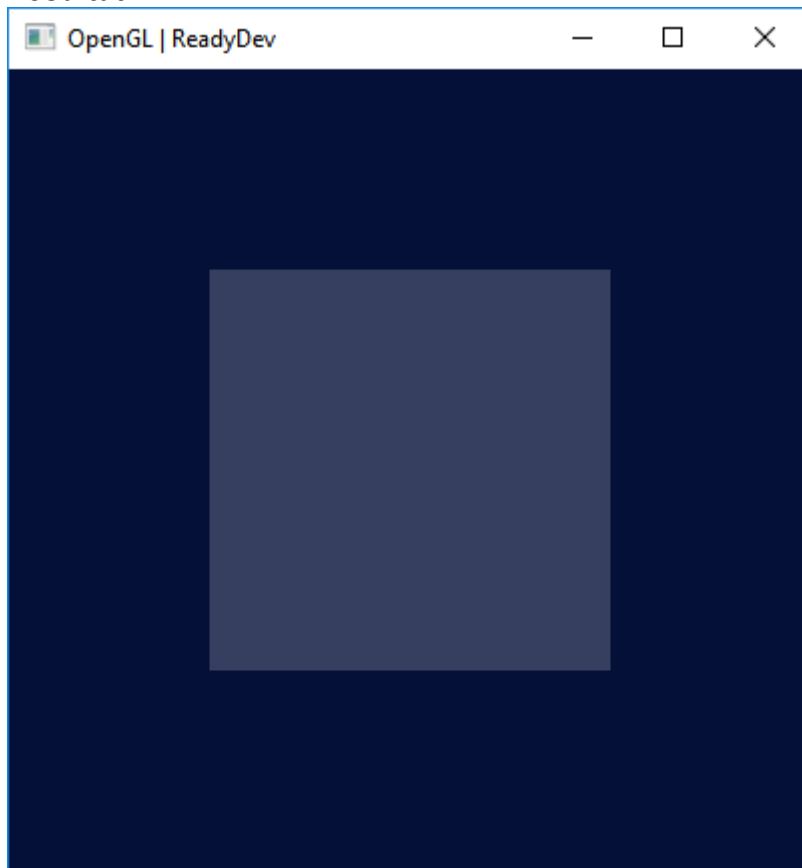
Afficher un objet 3D.

Implémentation :

Créer un gestionnaire de fenêtre (GWindow).

Créer un gestionnaire de dessin (GDraw).

Dessiner un objet 3D (draw()).

Résultat :**Dossier projet :**

```
src/main.cpp  
src/GWindow.h  
src/GWindow.cpp  
win/bin/  
win/build/
```

Programme principal :

```
//=====
#include "GWindow.h"
//=====
int main(int argc, char** argv) {
    GWindow::Instance()->show(&argc, argv);
    return 0;
}
//=====
```

Affichage de la fenêtre :

```
//=====
void GWindow::show(int* argc, char** argv) {
    glutInit(argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA |
GLUT_DEPTH);
    glutInitWindowPosition(m_x, m_y);
    glutInitWindowSize(m_w, m_h);
    glutCreateWindow(m_title.toStdString().c_str());
    glutSetOption (GLUT_ACTION_ON_WINDOW_CLOSE,
GLUT_ACTION_GLUTMAINLOOP_RETURNS);
    setBackground();
    glutIdleFunc (IdleFunc);
    glutDisplayFunc (DisplayFunc);
    glutMainLoop();
}
//=====
```

Fonction de rappel de la tâche de fond :

```
//=====
void GWindow::IdleFunc () {
    glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);
    GDraw::Instance()->draw();
    glutSwapBuffers();
}
//=====
```

Dessin de l'objet 3D :

```
//=====
```

```
void GDraw::draw() {
    float m_red = 55.0/255.0;
    float m_green = 63.0/255.0;
    float m_blue = 96.0/255.0;
    glColor3f(m_red, m_green, m_blue);

    float m_width = 0.5;
    float m_height = 0.5;
    float m_depth = 0.5;
    btVector3 m_vertices[8] = {
        btVector3(m_width, m_height, m_depth),
        btVector3(-m_width, m_height, m_depth),
        btVector3(m_width, -m_height, m_depth),
        btVector3(-m_width, -m_height, m_depth),
        btVector3(m_width, m_height, -m_depth),
        btVector3(-m_width, m_height, -m_depth),
        btVector3(m_width, -m_height, -m_depth),
        btVector3(-m_width, -m_height, -m_depth)
    };
    int m_indices[36] = {
        0,1,2,
        3,2,1,
        4,0,6,
        6,0,2,
        5,1,4,
        4,1,0,
        7,3,1,
        7,1,5,
        5,4,7,
        7,4,6,
        7,2,3,
        7,6,2
    };

    glBegin(GL_TRIANGLES);

    for(int i = 0; i < 36; i += 3) {
        const btVector3 &m_vert1 =
m_vertices[m_indices[i]];
        const btVector3 &m_vert2 =
m_vertices[m_indices[i+1]];
        const btVector3 &m_vert3 =
m_vertices[m_indices[i+2]];
```

```
        btVector3 m_normal = (m_vert3-
m_vert1).cross(m_vert2-m_vert1);
        m_normal.normalize ();

        glNormal3f(m_normal.getX(), m_normal.getY(),
m_normal.getZ());

        glVertex3f(m_vert1.x(), m_vert1.y(),
m_vert1.z());
        glVertex3f(m_vert2.x(), m_vert2.y(),
m_vert2.z());
        glVertex3f(m_vert3.x(), m_vert3.y(),
m_vert3.z());
    }
    glEnd();
}
//=====
```

Ajouter une caméra à la scène 3D

Objectif :

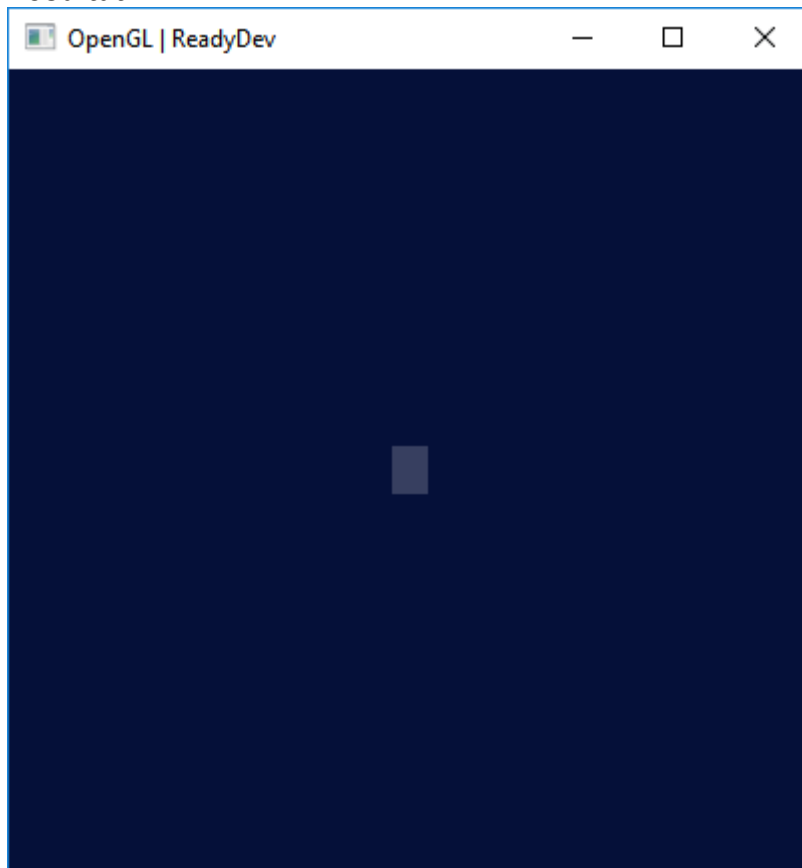
Ajouter une caméra à la scène 3D.

Implémentation :

Créer un gestionnaire de fenêtre (GWindow).

Créer un gestionnaire de camera (GCamera).

Afficher le champ de vision de la camera (update()).

Résultat :**Dossier projet :**

`src/main.cpp`

`src/GWindow.h`

`src/GWindow.cpp`

`win/bin/`

`win/build/`

Programme principal :

```
//=====
#include "GWindow.h"
//=====
int main(int argc, char** argv) {
    GWindow::Instance()->show(&argc, argv);
    return 0;
}
//=====
```

Affichage de la fenêtre :

```
//=====
void GWindow::show(int* argc, char** argv) {
    glutInit(argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA |
GLUT_DEPTH);
    glutInitWindowPosition(m_x, m_y);
    glutInitWindowSize(m_w, m_h);
    glutCreateWindow(m_title.toStdString().c_str());
    glutSetOption (GLUT_ACTION_ON_WINDOW_CLOSE,
GLUT_ACTION_GLUTMAINLOOP_RETURNS);
    setBackground();
    glutReshapeFunc(ReshapeFunc);
    glutIdleFunc(IdleFunc);
    glutDisplayFunc(DisplayFunc);
    glutMainLoop();
}
//=====
```

Fonction de rappel de la tâche de fond :

```
//=====
void GWindow::IdleFunc() {
    glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);
    GDraw::Instance()->draw();
    GCamera::Instance()->update();
    glutSwapBuffers();
}
//=====
```

Fonction de rappel du redimensionnement de la fenêtre :

```
//=====
void GWindow::ReshapeFunc(int w, int h) {
    glViewport(0, 0, w, h);
    GCamera::Instance()->setWindowSize(w, h);
    GCamera::Instance()->update();
}
//=====
```

Affichage du champ de vision de la camera :

```
//=====
void GCamera::update() {
    if(m_w == 0 && m_h == 0) return;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    float m_ratio = (float)m_w/m_h;
    float m_nearPlane = 1.0;
    float m_farPlane = 1000.0;
    float m_left = -m_ratio*m_nearPlane;
    float m_right = m_ratio*m_nearPlane;
    float m_top = -m_nearPlane;
    float m_bottom = m_nearPlane;
    float m_zNear = m_nearPlane;
    float m_zFar = m_farPlane;
    glFrustum(m_left, m_right, m_top, m_bottom,
m_zNear, m_zFar);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    float m_eyeX = 10.0;
    float m_eyeY = 5.0;
    float m_eyeZ = 0.0;
    float m_centerX = 0.0;
    float m_centerY = 0.0;
    float m_centerZ = 0.0;
    float m_upX = 0.0;
    float m_upY = 1.0;
    float m_upZ = 0.0;
    gluLookAt(m_eyeX, m_eyeY, m_eyeZ, m_centerX,
m_centerY, m_centerZ, m_upX, m_upY, m_upZ);
}
//=====
```

Ajouter de la lumière à une scène 3D

Objectif :

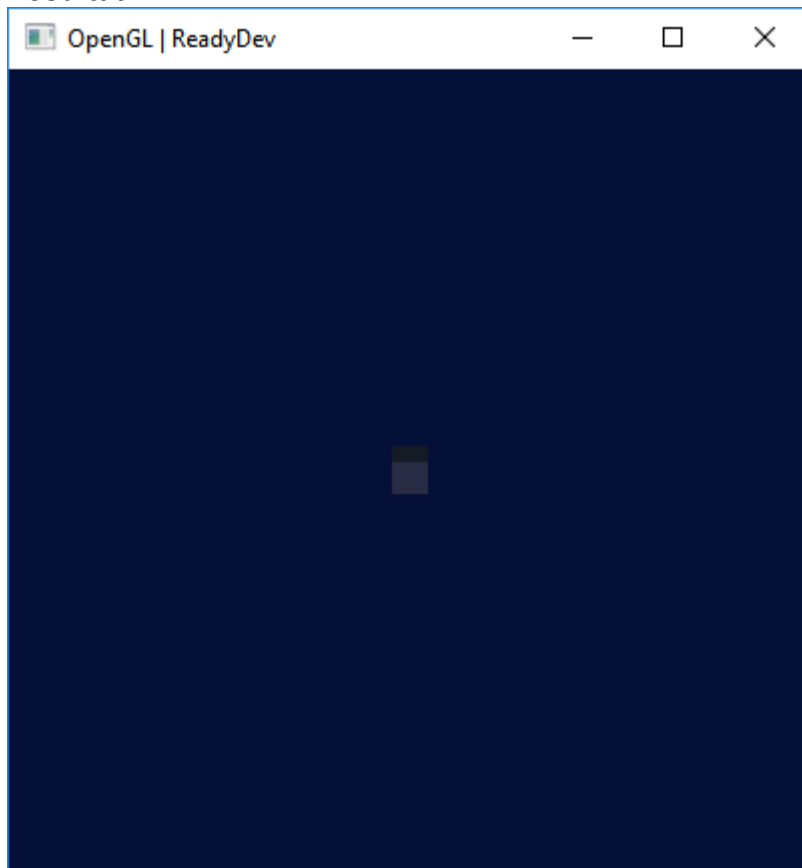
Ajouter de la lumière à une scène 3D.

Implémentation :

Créer un gestionnaire de fenêtre (GWindow).

Créer un gestionnaire de lumière (GLight).

Ajouter de la lumière à la scène (initLight()).

Résultat :**Dossier projet :**

src/main.cpp

src/GWindow.h

src/GWindow.cpp

win/bin/

win/build/

Programme principal :

```
//=====
#include "GWindow.h"
//=====
int main(int argc, char** argv) {
    GWindow::Instance()->show(&argc, argv);
    return 0;
}
//=====
```

Affichage de la fenêtre :

```
//=====
void GWindow::show(int* argc, char** argv) {
    glutInit(argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA |
GLUT_DEPTH);
    glutInitWindowPosition(m_x, m_y);
    glutInitWindowSize(m_w, m_h);
    glutCreateWindow(m_title.toStdString().c_str());
    glutSetOption (GLUT_ACTION_ON_WINDOW_CLOSE,
GLUT_ACTION_GLUTMAINLOOP_RETURNS);
    setLight();
    setBackground();
    glutReshapeFunc(ReshapeFunc);
    glutIdleFunc(IdleFunc);
    glutDisplayFunc(DisplayFunc);
    glutMainLoop();
}
//=====
```

Ajout de la lumière à la scène 3D :

```
//=====
void GWindow::setLight() {
    GLight::Instance()->initLight();
}
//=====
```

Création de la lumière ambiante, diffuse, spéculaire :

```
//=====
void GLight::initLight() {
    GLfloat m_ambient[] = {51.0/255.0, 51.0/255.0,
51.0/255.0, 255.0/255.0};
    GLfloat m_diffuse[] = {255.0/255.0, 255.0/255.0,
255.0/255.0, 255.0/255.0};
    GLfloat m_specular[] = {255.0/255.0, 255.0/255.0,
255.0/255.0, 255.0/255.0};
    GLfloat m_position[] = {5.0, 10.0, 1.0, 0.0};

    glLightfv(GL_LIGHT0, GL_AMBIENT, m_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, m_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, m_specular);
    glLightfv(GL_LIGHT0, GL_POSITION, m_position);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);

    glMaterialfv(GL_FRONT, GL_SPECULAR, m_specular);
    glMateriali(GL_FRONT, GL_SHININESS, 15);

    glShadeModel(GL_SMOOTH);

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);
}
//=====
```