READYDEV

[GitHub](https://github.com/gkesse) | [LinkedIn](https://www.linkedin.com/in/tia-gerard-kesse/) | [SiteWeb](http://readydev.freeboxos.fr/)

Par Gérard KESSE

Plateforme de Développement Continu

Comprendre la Théorie pour mieux Pratiquer

Sciences de l'Ingénieur

Cours - Tutoriels

READYDEV

[GitHub](https://github.com/gkesse) | [LinkedIn](https://www.linkedin.com/in/tia-gerard-kesse/) | [SiteWeb](http://readydev.freeboxos.fr/)

Par Gérard KESSE

J'aime, Je partage

Montez en Compétences

OpenGL

Maîtrisez la Programmation 3D

# Auteur

Je suis **Gerard 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'Energie 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](https://github.com/gkesse) | [LinkedIn](https://www.linkedin.com/in/tia-gerard-kesse/) | [SiteWeb](http://readydev.freeboxos.fr/)

[](http://readydev.freeboxos.fr/)

# Sommaire

[Auteur 1](#_Toc508615653)

[Sommaire 1](#_Toc508615654)

[Introduction 2](#_Toc508615655)

[Installation sous Windows 2](#_Toc508615656)

[Téléchargements 2](#_Toc508615657)

[Installation de MinGW 3](#_Toc508615658)

[Installation de GL3W 3](#_Toc508615659)

[Installation de GLFW 3](#_Toc508615660)

[Variables d’environnement 3](#_Toc508615661)

[Bibliothèque Manager 4](#_Toc508615662)

[Bibliothèque de classes communes 4](#_Toc508615663)

[Dessiner en dimension 2D 5](#_Toc508615664)

[Dessiner une ligne 5](#_Toc508615665)

[Dessiner un carré 9](#_Toc508615666)

# [Introduction](#_Sommaire_1)

OpenGL est une bibliothèque de création d’applications 3D. Le but de ce tutoriel est de vous apprendre à créer des applications 3D avec OpenGL en C++.

# [Installation sous Windows](#_Sommaire_1)

## [Téléchargements](#_Sommaire_1)

**Notepad++ :**

[https://notepad-plus-plus.org/](https://notepad-plus-plus.org/download/all-versions.html)

**MinGW :**

<http://www.mingw.org/>

**CMake :**

<https://cmake.org/download/>

**Python :**

<https://www.python.org/download/releases/2.7.8/>

**GL3W :**

<https://github.com/skaslev/gl3w>

**GLFW :**

<http://www.glfw.org/>

## [Installation de MinGW](#_Sommaire_1)

**Packages MinGW :**

mingw32-gcc-g++

mingw32-base

## [Installation de GL3W](#_Sommaire_1)

**Télécharger les sources :**

gl3w\_gen.py

## [Installation de GLFW](#_Sommaire_1)

**Construction CMake :**

Where is the source code

C:\Users\gerar\Downloads\OpenGL\glfw-3.2.1\glfw-3.2.1

Where to build the binaries

C:\Users\gerar\Downloads\OpenGL\glfw-3.2.1\glfw-3.2.1\build

Configure

Specify the generator for this project

MinGW Makefiles

Cocher

Use default native compilers

Finish

Configure

Configure

Generate

**Compilation MinGW :**

cd C:\Users\gerar\Downloads\OpenGL\glfw-3.2.1\glfw-3.2.1\build

mingw32-make

## [Variables d’environnement](#_Sommaire_1)

**Variables d’environnement :**

set PATH=C:\MinGW\bin;%PATH%

# [Bibliothèque Manager](#_Sommaire_1)

## [Bibliothèque de classes communes](#_Sommaire_1)

**Objectif :**

Présenter notre bibliothèque de classes communes aux projets.

**Dossier bibliothèque :**

manager/src/GFile.cpp

manager/src/GConfig.cpp

manager/src/GVertex.cpp

manager/src/GShader.cpp

manager/src/GWindow.cpp

manager/src/GDraw.cpp

**GFile :**

Singleton permettant de charger le contenu d'un fichier

**GConfig :**

Singleton permettant de gérer les données de configuration

**GVertex :**

Singleton permettant de charger les vertex correspondant à une application donnée

**GShader :**

Singleton de permettant de charger un shader et de renvoyer son id

**GWindow :**

Stratégie de permettant de créer la fenêtre principale

**GDraw :**

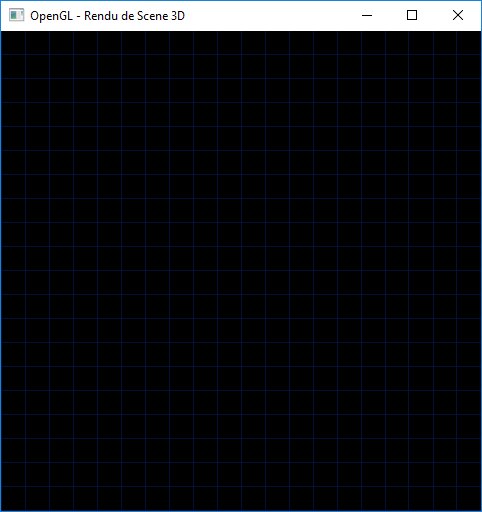
Stratégie de permettant d'afficher une figure

# [Dessiner en dimension 2D](#_Sommaire_1)

## [Dessiner une ligne](#_Sommaire_1)

**Objectif :**

Dessiner une grille en dimension 2D.



**Patrons de conception :**

Singleton

Stratégie

**Dossier projet :**

src/main.cpp

src/GWindowDraw.cpp

manager/src/GDrawLine.cpp

manager/src/GVertex.cpp

**src/main.cpp**

//================================================

#include "GWindowDraw.h"

//================================================

int main(int argc, char\*\* argv) {

cout << "### Start GProject\n";

GWindowDraw::Instance()->initGlfw();

GWindowDraw::Instance()->createWindow();

GWindowDraw::Instance()->createContext();

GWindowDraw::Instance()->initGl3w();

GWindowDraw::Instance()->run();

GWindowDraw::Instance()->release();

cout << "### End GProject\n";

return 0;

}

//================================================

**manager/src/GWindowDraw.cpp**

//================================================

void GWindowDraw::draw() {

GConfig::Instance()->setData("draw", "GL\_LINES");

GDraw::Instance()->draw();

}

//================================================

**manager/src/GDrawLine.cpp**

//================================================

void GDrawLine::drawShape() {

GLuint m\_vertexArray[1];

glGenVertexArrays(1, m\_vertexArray);

glBindVertexArray(m\_vertexArray[0]);

GLfloat m\_vertices[80][2];

GVertex::Instance()->loadVertexGrid(m\_vertices, 80);

GLuint m\_buffers[1];

glCreateBuffers(1, m\_buffers);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_buffers[0]);

glBufferStorage(GL\_ARRAY\_BUFFER, sizeof(m\_vertices), m\_vertices, 0);

GShaderInfo m\_shaders[] = {

{GL\_VERTEX\_SHADER, "res/shaders/shapes/shapes.vert"},

{GL\_FRAGMENT\_SHADER, "res/shaders/shapes/shapes.frag"},

{GL\_NONE}

};

GLuint m\_program = GShader::Instance()->loadShader(m\_shaders);

glUseProgram(m\_program);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 0, BUFFER\_OFFSET(0));

glEnableVertexAttribArray(1);

glBindVertexArray(m\_vertexArray[0]);

glDrawArrays(GL\_LINES, 0, 160);

//================================================

**manager/src/GVertex.cpp**

//================================================

void GVertex::loadVertexGrid(GLfloat vertex[][2], const int& row) {

int N = row/2;

for(int i = 0; i < N; i += 2) {

float dx = -1.0 + (i\*2.0)/N;

vertex[i][0] = dx;

vertex[i][1] = -1.0;

vertex[i + 1][0] = dx;

vertex[i + 1][1] = 1.0;

}

for(int i = 0; i < N; i += 2) {

float dy = -1.0 + (i\*2.0)/N;

vertex[N + i][0] = -1.0;

vertex[N + i][1] = dy;

vertex[N + i + 1][0] = 1.0;

vertex[N + i + 1][1] = dy;

}

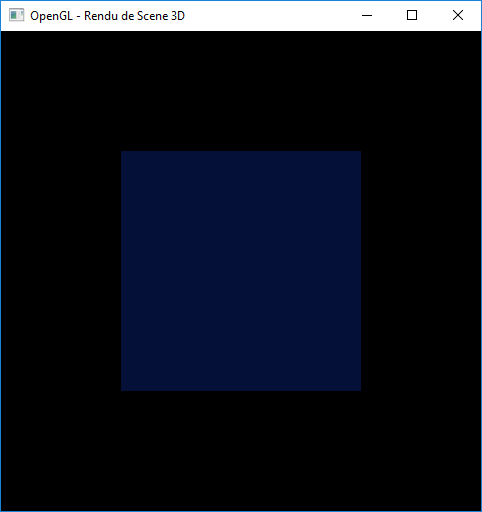
}

//================================================

## [Dessiner un carré](#_Sommaire_1)

**Objectif :**

Dessiner un carré en dimension 2D.



**Patrons de conception :**

Singleton

Stratégie

**Dossier projet :**

src/main.cpp

src/GWindowDraw.cpp

manager/src/GDrawQuad.cpp

**src/main.cpp**

//================================================

#include "GWindowDraw.h"

//================================================

int main(int argc, char\*\* argv) {

cout << "### Start GProject\n";

GWindowDraw::Instance()->initGlfw();

GWindowDraw::Instance()->createWindow();

GWindowDraw::Instance()->createContext();

GWindowDraw::Instance()->initGl3w();

GWindowDraw::Instance()->run();

GWindowDraw::Instance()->release();

cout << "### End GProject\n";

return 0;

}

//===================================================

**manager/src/GWindowDraw.cpp**

//================================================

void GWindowDraw::draw() {

GConfig::Instance()->setData("draw", "GL\_QUADS");

GDraw::Instance()->draw();

}

//================================================

**manager/src/GDrawQuad.cpp**

//================================================

void GDrawQuad::drawShape() {

GLuint m\_vertexArray[1];

glGenVertexArrays(1, m\_vertexArray);

glBindVertexArray(m\_vertexArray[0]);

GLfloat m\_vertices[4][2] = {

{-0.5f, 0.5f}, {-0.5f, -0.5f}, {0.5f, 0.5f}, {0.5f, -0.5f}

};

GLuint m\_buffers[1];

glCreateBuffers(1, m\_buffers);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_buffers[0]);

glBufferStorage(GL\_ARRAY\_BUFFER, sizeof(m\_vertices), m\_vertices, 0);

GShaderInfo m\_shaders[] = {

{GL\_VERTEX\_SHADER, "res/shaders/shapes/shapes.vert"},

{GL\_FRAGMENT\_SHADER, "res/shaders/shapes/shapes.frag"},

{GL\_NONE}

};

GLuint m\_program = GShader::Instance()->loadShader(m\_shaders);

glUseProgram(m\_program);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 0, BUFFER\_OFFSET(0));

glEnableVertexAttribArray(1);

glBindVertexArray(m\_vertexArray[0]);

glDrawArrays(GL\_TRIANGLE\_STRIP, 0, 4);

}

//================================================