# Sommaire

[Sommaire 1](#_Toc151500520)

[Introduction 2](#_Toc151500521)

[Création du programme principal 2](#_Toc151500522)

[Affichage de la fenêtre principale 2](#_Toc151500523)

[Création de la fenêtre principale 2](#_Toc151500524)

[Création de la fenêtre 3](#_Toc151500525)

[Création du timer de mise à jour 4](#_Toc151500526)

[Calcul du pas de progression 5](#_Toc151500527)

[Calcul du temps de progression 6](#_Toc151500528)

[Mise à jour de la position du robot 6](#_Toc151500529)

[Mise à jour de l’interface de la position du robot 7](#_Toc151500530)

[Mise à jour de la scène 3D 8](#_Toc151500531)

[Création du robot 9](#_Toc151500532)

[Chargement des modèles du robot 9](#_Toc151500533)

[Initialisation des coordonnées du robot 9](#_Toc151500534)

[Calcul de la position du robot 9](#_Toc151500535)

[Création du gestionnaire des modèles du robot 10](#_Toc151500536)

[Chargement des matériels, des vertex, des normales, des faces 10](#_Toc151500537)

[Chargement des matériels 12](#_Toc151500538)

[Remplacement des slashs par des espaces dans une chaine 13](#_Toc151500539)

[Création de la scène 3D avec OpenGL 13](#_Toc151500540)

[Initialisation des gestionnaires 13](#_Toc151500541)

[Initialisation de OpenGL 14](#_Toc151500542)

[Initialisation de la fenêtre d’affichage de la scène 14](#_Toc151500543)

[Initialisation des matrices de projection de la scène 14](#_Toc151500544)

[Initialisation de la caméra de la scène 15](#_Toc151500545)

[Initialisation de la lumière de la scène 15](#_Toc151500546)

[Initialisation de la grille du plan XY 16](#_Toc151500547)

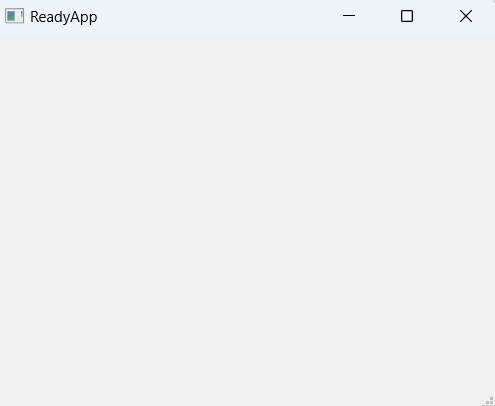
# Introduction

Le but de ce tutoriel est de vous apprendre à développer un simulateur de bras manipulateur robotisé de type SCARA en C++ - Qt – OpenGL – Modèle géométrique – Modèle cinématique – Cinématique inverse.

# Création du programme principal

## Affichage de la fenêtre principale

Résultat:



Programme C++:

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

// main.cpp

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

int **main**(int **\_argc**, char\*\* **\_argv**) {

QApplication **lApp**(*\_argc*, *\_argv*);

GMainWindow **lWindow**;

lWindow.show();

return lApp.exec();

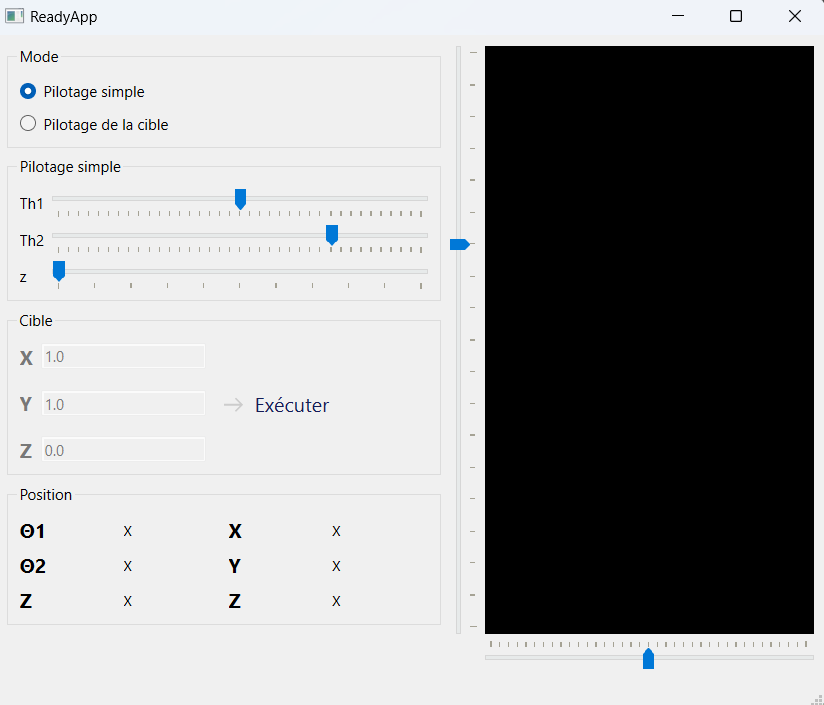
}

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

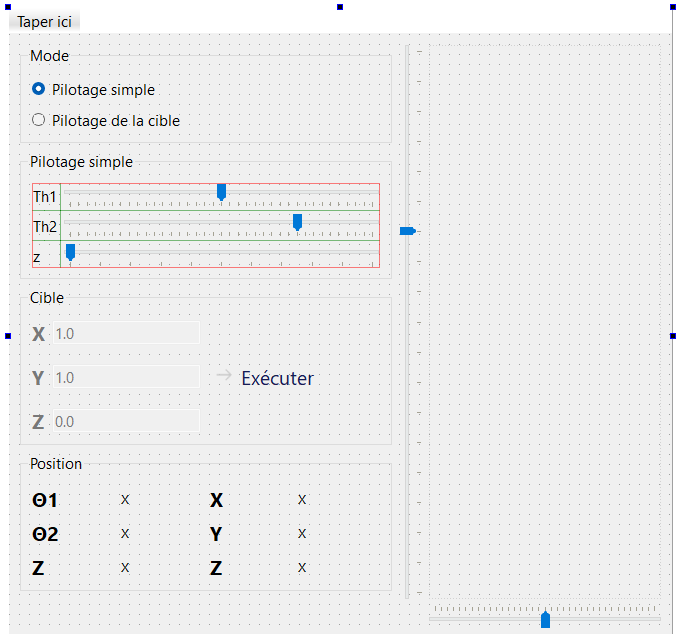
# Création de la fenêtre principale

## Création de la fenêtre

Résultat:



Résultat dans Qt Designer:



Programme C++:

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

// GMainWindow.cpp

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

GMainWindow::**GMainWindow**(QWidget\***\_parent**)

: QMainWindow(*\_parent*)

, ui(new Ui::GMainWindow) {

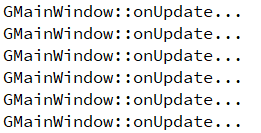
ui->setupUi(this);

}

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

## Création du timer de mise à jour

Résultat:



Programme C++:

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

// GMainWindow.cpp

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

GMainWindow::**GMainWindow**(QWidget\* **\_parent**)

: QMainWindow(*\_parent*)

, ui(new Ui::GMainWindow) {

ui->setupUi(this);

connect(&m\_timer, &QTimer::timeout, this, &GMainWindow::onUpdate);

m\_timer.setInterval(0);

m\_timer.start();

}

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

void GMainWindow::**onUpdate**() {

qDebug() << "GMainWindow::onUpdate...";

}

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

## Calcul du pas de progression

Résultat:



Programme C++:

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

// GMainWindow.cpp

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

GMainWindow::**GMainWindow**(QWidget\* **\_parent**)

: QMainWindow(*\_parent*)

, ui(new Ui::GMainWindow)

, t(0.0), dt(0.0) {

ui->setupUi(this);

connect(&m\_timer, &QTimer::timeout, this, &GMainWindow::onUpdate);

m\_timer.setInterval(0);

m\_timer.start();

m\_robot = new GScara(this);

m\_clock = clock();

}

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

void GMainWindow::**onUpdate**() {

dt = 1.0 \* (clock() - m\_clock) / CLOCKS\_PER\_SEC;

qDebug() << dt;

}

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

## Calcul du temps de progression

Résultat:



Programme C++:

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

// GMainWindow.cpp

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

GMainWindow::**GMainWindow**(QWidget\* **\_parent**)

: QMainWindow(*\_parent*)

, ui(new Ui::GMainWindow)

, t(0.0), dt(0.0) {

ui->setupUi(this);

connect(&m\_timer, &QTimer::timeout, this, &GMainWindow::onUpdate);

m\_timer.setInterval(0);

m\_timer.start();

m\_robot = new GScara(this);

m\_clock = clock();

}

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

void GMainWindow::**onUpdate**() {

dt = 1.0 \* (clock() - m\_clock) / CLOCKS\_PER\_SEC;

m\_clock = clock();

t += dt;

static double **t\_last** = t;

if(t\_last != t) {

t\_last = t;

qDebug() << t;

}

}

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

## Mise à jour de la position du robot

Programme C++:

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

// GMainWindow.cpp

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

void GMainWindow::**onUpdate**() {

dt = 1.0 \* (clock() - m\_clock) / CLOCKS\_PER\_SEC;

m\_clock = clock();

t += dt;

m\_robot->onUpdate(dt);

}

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

## Mise à jour de l’interface de la position du robot

Résultat :



Programme C++:

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

// GMainWindow.cpp

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

void GMainWindow::**onUpdate**() {

dt = 1.0 \* (clock() - m\_clock) / CLOCKS\_PER\_SEC;

m\_clock = clock();

t += dt;

m\_robot->onUpdate(dt);

bool **mode** = ui->radioButtonTarget->isChecked();

QString **text**;

text = QString::number(m\_robot->getTh1(), 'f', 3);

ui->labelTh1->setText(text);

text = QString::number(m\_robot->getTh2(), 'f', 3);

ui->labelTh2->setText(text);

text = QString::number(m\_robot->getZ(), 'f', 3);

ui->labelThZ->setText(text);

text = QString::number(m\_robot->getY(), 'f', 3);

ui->labelY->setText(text);

text = QString::number(m\_robot->getX(), 'f', 3);

ui->labelX->setText(text);

text = QString::number(m\_robot->getZ(), 'f', 3);

ui->labelZ->setText(text);

if(mode) {

ui->horizontalSliderTh1->setValue(m\_robot->getTh1());

ui->horizontalSliderTh2->setValue(m\_robot->getTh2());

ui->horizontalSliderZ->setValue(100.0 \* m\_robot->getZ());

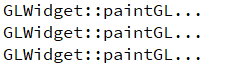
}

}

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

## Mise à jour de la scène 3D

Résultat :



Programme C++:

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

// GMainWindow.cpp

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

void GMainWindow::**onUpdate**() {

dt = 1.0 \* (clock() - m\_clock) / CLOCKS\_PER\_SEC;

m\_clock = clock();

t += dt;

m\_robot->onUpdate(dt);

bool **mode** = ui->radioButtonTarget->isChecked();

QString **text**;

text = QString::number(m\_robot->getTh1(), 'f', 3);

ui->labelTh1->setText(text);

text = QString::number(m\_robot->getTh2(), 'f', 3);

ui->labelTh2->setText(text);

text = QString::number(m\_robot->getZ(), 'f', 3);

ui->labelThZ->setText(text);

text = QString::number(m\_robot->getY(), 'f', 3);

ui->labelY->setText(text);

text = QString::number(m\_robot->getX(), 'f', 3);

ui->labelX->setText(text);

text = QString::number(m\_robot->getZ(), 'f', 3);

ui->labelZ->setText(text);

if(mode) {

ui->horizontalSliderTh1->setValue(m\_robot->getTh1());

ui->horizontalSliderTh2->setValue(m\_robot->getTh2());

ui->horizontalSliderZ->setValue(100.0 \* m\_robot->getZ());

}

ui->view->update();

}

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

# Création du robot

## Chargement des modèles du robot

Programme C++:

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

// GScara.cpp

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

GScara::**GScara**(QObject\* **\_parent**)

: QObject(*\_parent*) {

m\_baseModel.load("data/obj/base.obj");

m\_arm1Model.load("data/obj/arm1.obj");

m\_arm2Model.load("data/obj/arm2.obj");

m\_arm3Model.load("data/obj/arm3.obj");

}

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

## Initialisation des coordonnées du robot

Programme C++:

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

// GScara.cpp

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

GScara::**GScara**(QObject\* **\_parent**)

: QObject(*\_parent*)

, th1(0.0), th2(90.0), z(0.0)

, dth1(0.0), dth2(0.0), dz(0.0)

, posx(0.0), posy(0.0), posz(0.0)

, tx(1.0), ty(1.0), tz(0.0)

, r1(1.5), r2(1.5) {

m\_baseModel.load("data/obj/base.obj");

m\_arm1Model.load("data/obj/arm1.obj");

m\_arm2Model.load("data/obj/arm2.obj");

m\_arm3Model.load("data/obj/arm3.obj");

}

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

## Calcul de la position du robot

Résultat:



Programme C++:

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

GScara::**GScara**(QObject\* **\_parent**)

: QObject(*\_parent*)

, th1(0.0), th2(90.0), z(0.0)

, dth1(0.0), dth2(0.0), dz(0.0)

, posx(0.0), posy(0.0), posz(0.0)

, tx(1.0), ty(1.0), tz(0.0)

, r1(1.5), r2(1.5) {

m\_baseModel.load("data/obj/base.obj");

m\_arm1Model.load("data/obj/arm1.obj");

m\_arm2Model.load("data/obj/arm2.obj");

m\_arm3Model.load("data/obj/arm3.obj");

kinPr();

}

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

void GScara::**kinPr**() {

posx = r1 \* cos(M\_PI \* th1 / 180.0) + r2 \* cos(M\_PI \* (th1 + th2) / 180.0);

posy = r1 \* sin(M\_PI\*th1/180.0) + r2 \* sin(M\_PI\*(th1+th2)/180.0);

posz = z;

qDebug() << posx;

qDebug() << posy;

qDebug() << posz;

}

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

# Création du gestionnaire des modèles du robot

## Chargement des matériels, des vertex, des normales, des faces

Programme C++:

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

// GModel.cpp

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

void GModel::**load**(const char\* **\_filename**) {

std::ifstream **file**(\_filename);

if(!file) {

qDebug() << "Le fichier n'a pas été trouvé.";

exit(-1);

}

std::string **s**;

int **current\_material** = -1;

while(getline(*file*, *s*)) {

replace(*s*, '/', ' ');

std::stringstream **sstr**;

sstr << s;

std::string **cmd**;

sstr >> cmd;

if(cmd == "mtllib") {

std::string **matfile**;

sstr >> matfile;

loadMaterials(matfile);

}

if(cmd == "v") {

sGVertex **vtx**;

sstr >> vtx.x[0] >> vtx.x[1] >> vtx.x[2];

m\_vertex.push\_back(vtx);

}

if(cmd == "vn") {

sGVertex **nrm**;

sstr >> nrm.x[0] >> nrm.x[1] >> nrm.x[2];

for(int **i** = 0; i < 3; nrm.x[i++] \*= -1);

nrm.x[0] \*= -1;

nrm.x[1] \*= -1;

nrm.x[2] \*= -1;

m\_normals.push\_back(nrm);

}

if(cmd == "f") {

sGFace **face**;

for(int **i** = 0; i < 3; ++i) {

sstr >> face.vertex[i] >> face.normal;

}

face.material = current\_material;

m\_faces.push\_back(face);

}

if(cmd == "usemtl") {

std::string **matname**;

sstr >> matname;

int **i** = 0;

for(; i < (int)m\_materials.size(); ++i) {

if(matname == m\_materials[i].name) break;

}

current\_material = i;

}

file.close();

}

}

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

## Chargement des matériels

Programme C++:

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

// GModel.cpp

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

void GModel::**loadMaterials**(const std::string& **\_filename**) {

std::string **lFilename** = "data/obj/" + \_filename;

std::ifstream **file**(lFilename.c\_str());

if(!file) {

qDebug() << "Le fichier n'a pas été trouvé.";

exit(-1);

}

std::string **line**;

while(getline(*file*, *line*)) {

std::stringstream **sstr**;

sstr << line;

std::string **cmd**;

sstr >> cmd;

sGMaterial **nmat**;

if(cmd == "newmtl") {

sstr >> nmat.name;

}

if(cmd == "Ns") {

sstr >> nmat.shininess;

}

if(cmd == "Ka") {

sstr >> nmat.ambient[0] >> nmat.ambient[1] >> nmat.ambient[2];

}

if(cmd == "Kd") {

sstr >> nmat.diffuse[0] >> nmat.diffuse[1] >> nmat.diffuse[2];

}

if(cmd == "Ks") {

sstr >> nmat.specular[0] >> nmat.specular[1] >> nmat.specular[2];

}

if(cmd == "d") {

float **alpha**;

sstr >> alpha;

nmat.ambient[3] = alpha;

nmat.diffuse[3] = alpha;

nmat.specular[3] = alpha;

}

if(cmd == "illum") {

m\_materials.push\_back(nmat);

}

file.close();

}

}

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

## Remplacement des slashs par des espaces dans une chaine

Programme C++:

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

// GModel.cpp

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

void **replace**(std::string& **\_str**, const char **\_from**, const char **\_to**) {

std::string::iterator **i**;

for(i = \_str.begin(); i < \_str.end(); ++i)

if(\*i == \_from) \*i = \_to;

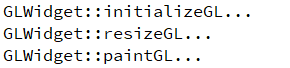
}

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

# Création de la scène 3D avec OpenGL

## Initialisation des gestionnaires

Résultat:



Programme C++:

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

// GLWidget.cpp

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

GLWidget::**GLWidget**(QWidget\* **\_parent**)

: QOpenGLWidget(*\_parent*) {

}

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

void GLWidget::***initializeGL***() {

qDebug() << "GLWidget::initializeGL...";

}

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

void GLWidget::***paintGL***() {

qDebug() << "GLWidget::paintGL...";

}

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

void GLWidget::***resizeGL***(int **width**, int **height**) {

qDebug() << "GLWidget::resizeGL...";

}

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

## Initialisation de OpenGL

Programme C++:

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

// GLWidget.cpp

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

void GLWidget::***initializeGL***() {

glEnable(GL\_DEPTH\_TEST);

glClearDepth(1.0);

glEnable(GL\_CULL\_FACE);

glEnable(GL\_POINT\_SMOOTH);

glPointSize(10.0);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glHint(GL\_POLYGON\_SMOOTH\_HINT, GL\_NICEST);

glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);

glClearColor(0.0, 0.0, 0.0, 0.0);

}

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

## Initialisation de la fenêtre d’affichage de la scène

Programme C++:

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

// GLWidget.cpp

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

void GLWidget::***resizeGL***(int **\_width**, int **\_height**) {

m\_width = \_width;

m\_height = \_height;

glViewport(0.0, 0.0, \_width, \_height);

setView();

}

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

## Initialisation des matrices de projection de la scène

Programme C++:

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

// GLWidget.cpp

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

void GLWidget::**setView**() {

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-30.0, 30.0, -30.0, 30.0, -30.0, 30.0);

GLdouble **aspect** = m\_width / (m\_height ? m\_height : 1);

const GLdouble **zNear** = -30.0, **zFar** = 30.0, **fov** = 30.0;

perspective(fov, aspect, zNear, zFar);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

## Initialisation de la caméra de la scène

Programme C++:

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

void GLWidget::**perspective**(GLdouble **\_fovY**, GLdouble **\_aspect**, GLdouble **\_zNear**, GLdouble **\_zFar**) {

GLdouble **xmin**, **xmax**, **ymin**, **ymax**;

ymax = \_zNear \* tan( \_fovY \* M\_PI / 360.0 );

ymin = -ymax;

xmin = ymin \* \_aspect;

xmax = ymax \* \_aspect;

glFrustum( xmin, xmax, ymin, ymax, \_zNear, \_zFar );

}

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

## Initialisation de la lumière de la scène

Programme C++:

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

// GLWidget.cpp

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

void GLWidget::**setLight**() {

GLfloat **lamb**[] = { 0.1, 0.1, 0.1, 1.0 };

GLfloat **ldif**[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat **lpos**[] = { -10.0, -10.0, 10.0, 1.0 };

glEnable(GL\_COLOR\_MATERIAL);

glDisable(GL\_LIGHTING);

glDisable(GL\_LIGHT0);

glColor4fv(ldif);

glBegin(GL\_POINTS);

glVertex4fv(lpos);

glEnd();

glDisable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_POSITION, lpos);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, lamb);

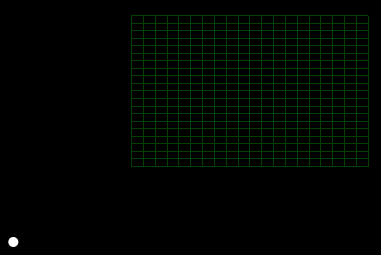
glLightfv(GL\_LIGHT0, GL\_DIFFUSE, ldif);

}

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

## Initialisation de la grille du plan XY

Résultat:



Programme C++:

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

// GLWidget.cpp

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

void GLWidget::***paintGL***() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

setView();

setLight();

glEnable(GL\_COLOR\_MATERIAL);

glDisable(GL\_LIGHTING);

glColor4f(0.0, 0.5, 0.0, 0.5);

for(int **i** = 0; i <= 20; ++i) {

glBegin(GL\_LINES);

glVertex3f(-5.0+0.5\*i, -5.0, 0.0); glVertex3f(-5.0+0.5\*i, 5.0, 0.0);

glVertex3f(-5.0, -5.0+0.5\*i, 0.0); glVertex3f(5.0, -5.0+0.5\*i, 0.0);

glEnd();

}

}

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