Εγχειρίδιο Εκμάθησης της Βιβλιοθήκης OpenGL

John Crabs

Αθήνα 2018

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Γενικά Στοιχεία για το Εγχειρίδιο

Το παρόν εγχειρίδιο προσπαθεί να αποτελέσει έναν εύκολα προσβάσιμο κι εύκολα καταννοητό οδηγό εκμάθησης της βιβλιοθήκης γραφικών OpenGL. Κύριο μέλημά του είναι η εξοικείωση του αναγνώστη, τόσο με τις συναρτήσεις/εντολές της βιβλιοθήκης, όσο και με τη φιλοσοφία που κρύβεται πίσω από αυτές (προγραμματιστική λογική, μαθηματικά μοντέλα...).

Προκειμένου να επιτευχθεί ο παραπάνω στόχος, ο οδηγός χωρίζεται σε ενότητες με παραδείγματα. Κάθε παράδειγμα στηρίζεται σε κάποια βασική θεωρεία, στην οποία θα εξηγούνται τόσο οι συναρτήσεις που χρησιμοποιούνται στο παράδειγμα, όσο και η λογική σειρά με την οποία αυτές πρέπει να εκτελεστούν. Τέλος για την καλύτερη κατανόηση, αλλά και για τη δημιουργία χρήσιμων παραδειγμάτων κώδικα, κάθε παράδειγμα θα εκφράζεται με δύο διαφορετικούς τρόπους.

Ο πρώτος τρόπος έκφρασης είναι η χρήση της βιβλιοθήκης GLUT, για τη δημιουργία παραθύρου. Αυτός ο τρόπος αποτελεί μία εύκολη μέθοδο για την εκμάθηση και καλύτερη κατανόηση της βιβλιοθήκης OpenGL. Είναι ο τρόπος υπόδειξης των παραδειγμάτων του οδηγού OpenGL Programming Guide 2nd Edition, από τον οποίο έχουν παρθεί και τροποποιηθεί αρκετά από τα παραδείγματα, ώστε να ικανοποιούν τις ανάγκες του παρόντος εγχειρίδιου.

Ο δεύτερος τρόπος έκφρασης των παραδειγμάτων, είναι με τη χρήση του περιβάλλοντος QtCreator για τη διαχείριση παραθύρων. Αν και ο κώδικας είναι ελαφρώς πιο σύνθετος, συγκριτικά με τον πρώτο τρόπο, τα αρχεία κώδικα που δημιουργούνται αποτελούν χρήσιμο υλικό για τη δημιουργία εφαρμογών αργότερα. Επιπλέον, για να μη δημιουργηθεί σύγχηση, όπου κρίνεται απαραίτητο θα αναλύονται και οι συναρτήσεις/εντολές των βιβλιοθηκών του Qt.

Σε αυτό το σημείο κρίνεται σκόπιμο να ειπωθεί, πως το παρόν εγχειρίδιο θεωρεί δεδομένη τη βασική εξοικείωση του χρήση με τη γλώσσα προγραμματισμού C/C++.

Τέλος, μπορεί κανείς να βρει τα παραδείγματα που περιγράφονται στο παρόν εγχειρίδιο, μαζί με τον οδηγό, κάθως κι άλλο χρήσιμο υλικό, στη σελίδα μου στο github.

https://github.com/JohnCrabs

1.2 Εγκατάσταση των Απαραίτητων Βιβλιοθηκών

Προκειμένου να μην υπάρξουν τεχνικές δυσκολίες, είναι αναγκαίο να εγκατασταθούν οι παρακάτω βιβλιοθήκες και προγράμματα:

- gl.h
- glu.h
- glut.h
- glew.h
- QtCreator

Οι τέσσερις βιβλιοθήκες που σχετίζονται με την OpenGL είναι απαραίτητες να υπάρχουν στο σύστημα. Η εγκατάσταση του QtCreator δεν είναι υποχρεωτική, ωστόσο η μη εγκατάσταση του θέτει αδύνατη τη χρήση των παραδειγμάτων που ακολουθούν το δεύτερο τρόπο έκφρασης.

Ο τρόπος εγκατάστασης των παραπάνω βιβλιοθηκών, διαφέρει ανάλογα το λειτουργικό σύστημα κι επειδή υπάρχει μπόλικο υλικό στο διαδίκτυο σχετικά με το πως μπορούν να εγκατασταθούν, δε θα αναφερθεί κάποια μέθοδος εγκατάστασης.

1.3 Άδεια

Κεφάλαιο 2

Μία πρώτη ματιά στον κόσμο της OpenGL

2.1 Βασική Θεωρεία

2.1.1 Τι είναι η OpenGL

Η OpenGL αποτελεί τη διεπαφή μεταξύ ενός λογισμικού και του υλικού συστήματος γραφικών του υπολογιστή. Η διεπαφή αυτή αποτελείται από περίπου 150 διακριτές εντολές, οι οποίες χρησιμοποιούνται για τον προσδιορισμό ενός αντικειμένου και των διαδικασιών που απαιτούνται ώστε να δημιουργηθούν διαδραστικές τριδιάστατες εφαρμογές.

Η OpenGL λειτουργεί ανεξαρτήτως υλικού συστήματος και είναι συμβατή με πολλές πλατφόρμες. Για να πραγματοποιηθεί αυτό, η OpenGL δε διαθέτει καμία εντολή διαχείρισης παραθύρων ή αλληλεπίδρασης του χρήστη με το πρόγραμμα, αλλά αντιθέτως μπορεί να συνεργαστεί με το οποιοδήποτε σύστημα διαχείρισης παραθύρων και περιφερειακού υλικού (στο παρόν εγχειρίδιο τα συστήματα αυτά είναι η βιβλιοθήκη GLUT και το λογισμικό QCreator). Τέλος η OpenGL δεν παρέχει υψηλού επιπέδου εντολές για την περιγραφή μοντέλων ή τριδιάστατων αντικειμένων. Η δημιουργία τέτοιων μοντέλων πραγματοποιείται με τη χρήση ενός μικρού σετ από γεωμετρικά πρότυπα (σημεία, ευθείες και πολύγωνα), τα οποία μπορούν να χρησιμοποιηθούν για να περιγράψουν οποιοδήποτε σύνθετο μοντέλο.

2.1.2 Η OpenGL ως μηχανή κατάστασης

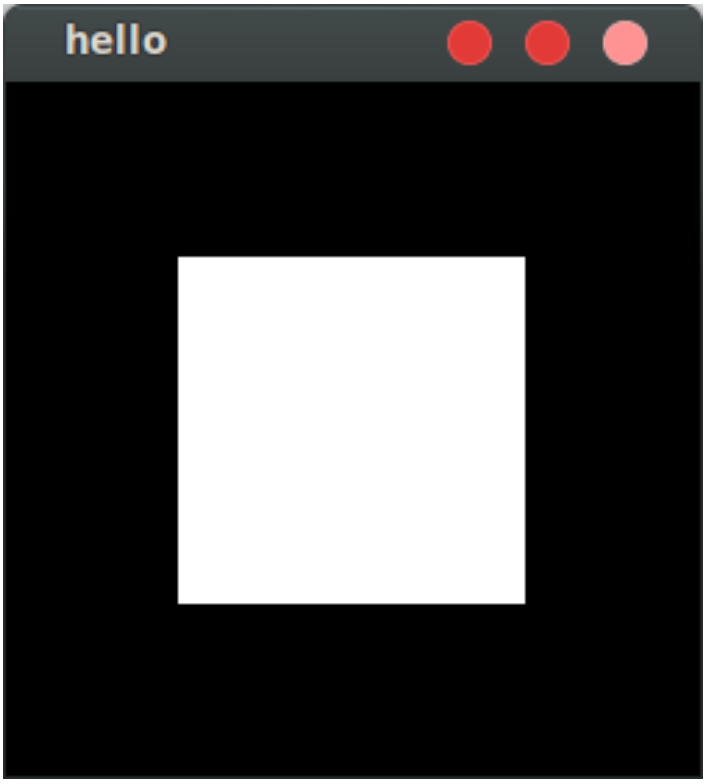
Η OpenGL είναι μία μηχανή κατάστασης. Αυτό σημαίνει ότι κατά τη δημιουργεία ενός προγράμματος, ο προγραμματιστής ορίζει διάφορες καταστάσεις, οι οποίες παραμένουν ενεργές έως ότου τροποποιηθούν. Ένα παράδειγμα είναι ο ορισμός του χρώματος σχεδιασμού ως άσπρο, κόκκινο κλπ. Όταν οριστεί ένα χρώμα, τότε κάθε φορά που σχεδιάζεται στην οθόνη ένα μοντέλο, θα εμφανιστεί με αυτό το χρώμα έως ότου οριστεί ένα νέο. Ένα άλλο χαρακτηριστικό παράδειγμα είναι τα προβολικά συστήματα και οι μετασχηματισμοί, οι οποίοι καθορίζουν το πως θα απεικονιστούν τα σχεδιασμένα αντικείμενα στην οθόνη. Σε επόμενα παραδείγματα θα γίνει περαιτέρω ανάλυση αυτών των καταστάσεων.

2.2 Δημιουργία Παραθύρου με Χρήση της GLUT

main.cpp 1

```
#include <GL/gl.h>
#include <GL/glut.h>

#define MY_WIN_WIDTH 250
#define MY_WIN_HEIGHT 250
#define MY_WIN_POS_X 100
#define MY_WIN_POS_Y 100
#define MY_WIN_TITLE "hello"

void display(void) {
    //clear all pixels
    glClear(GL_COLOR_BUFFER_BIT);

    /* draw white polygon (rectangle) with corners at
     * (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)
     */
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f(0.25, 0.25, 0.0);
        glVertex3f(0.75, 0.25, 0.0);
        glVertex3f(0.75, 0.75, 0.0);
        glVertex3f(0.25, 0.75, 0.0);
    glEnd();

    /* don't wait!
     * start processing buffered OpenGL routines
     */
    glFlush();
}

void init(void) {
    /* select clearing (background) color */
    glClearColor(0.0, 0.0, 0.0, 0.0);

    /* initialize viewing values */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

void winCreate(void) {
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(MY_WIN_WIDTH, MY_WIN_HEIGHT);
    glutInitWindowPosition(MY_WIN_POS_X, MY_WIN_POS_Y);
    glutCreateWindow(MY_WIN_TITLE);
}

/*
 * Declares initial window size, position and display mode
 * (single buffer and RGB). Open window with "hello"
 * in its title bar. Call initialization routines.
 * Register callback function to display graphics.
 * Enter main loop and process events.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    winCreate();
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Στην προηγούμενη σελίδα φαίνεται ο κώδικας ενός απλού προγράμματος, που ανοίγει ένα παράθυρο και ζωγραφίζει σε αυτό ένα άσπρο ορθογώνιο, όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 2.1: Δημιουργία Παραθύρου με χρήση της GLUT

2.2.1 Ανάλυση των Συναρτήσεων

Με μία πρώτη ματιά στον παραπάνω κώδικα παρατηρούνται τρεις βασικές συναρτήσεις.

- display(void)
- init(void)
- winCreate(void)

2.2.1.a display(void)

Η συνάρτηση αυτή αποτελεί μία από τις τρεις βασικές και απαραίτητες συναρτήσεις που πρέπει να έχει ένα πρόγραμμα που χρησιμοποιεί τη βιβλιοθήκη OpenGL. Οι άλλες δύο συναρτήσεις είναι οι:

- init(void)
- resize(int w, int h)

Η συνάστηρη display(void) διαχειρίζεται το τι πρέπει να σχεδιαστεί στην οθόνη και με ποια σειρά. Αυτό επιτυγχάνεται με την εντολή glutDisplayFunc(display) που καλείται στη συνάρτηση main(...).

2.2.1.b init(void)

Η συνάρτηση αυτή αποτελεί μία από τις τρεις βασικές και απαραίτητες συναρτήσεις που πρέπει να έχει ένα πρόγραμμα που χρησιμοποιεί τη βιβλιοθήκη OpenGL. Οι άλλες δύο συναρτήσεις είναι οι:

- display(void)
- resize(int w, int h)

Η συνάστηρη init(void) αρχικοποιεί τις βασικές καταστάσεις της OpenGL, όπως είναι το χρώμα με το οποίο θα καθαρίζεται φόντο, χρησιμοποιώντας την εντολή glClearColor(...). Επιπλέον η συνάρτηση αυτή περιέχει και το τι είδους προσανατολισμούς και μαθηματικά μοντέλα θα χρησιμοποιηθούν για την απεικόνιση.

Στο συγκεκριμένο παράδειγμα αυτά που αρχικοποιεί η συνάρτηση αυτή είναι ελάχιστα, σε πιο προχωρημένα παραδείγματα, ωστόσο, η συνάρτηση θα αρχικοποιεί παραπάνω παραμέτρους, όπως είναι το βάθος, η φωτεινότητα και άλλες χρήσιμες καταστάσεις, ώστε να δημιουργηθεί ένα πιο ρεαλιστικό γραφικό περιβάλλον.

2.2.1.c winCreate(void)

Η συνάρτηση αυτή περιέχει τις βασικές συναρτήσεις για τη δημιουργία ενός παραθύρου GLUT. Οι συναρτήσεις αυτές θα μπορούσαν κάλλιστα να βρίσκονταν απευθείας στη main(...), ωστόσο, είναι καλύτερα προγραμματιστικά εάν η συνάρτηση main(...) είναι μικρή σε έκταση. Οι συναρτήσεις της βιβλιοθήκης GLUT που χρησιμοποιούνται εντός της συνάρτησης είναι οι εξής:

- **glutInitDisplayMode**(unsigned int mode): Η συνάρτηση αρχικοποίησης λειτουργιών εμφάνισης χρησιμοποιείται όταν δημιουργούνται παράθυρα, δευτερεύοντα παράθυρα και επικαλύψεις ανωτέρου επιπέδου, για να καθορίζεται η λειτουργία απεικόνισης της OpenGL, για το παράθυρο ή την επικάλυψη που πρόκειται να δημιουργηθεί.
- **glutInitWindowSize**(int width, int height): Καθορίζει τις αρχικές διαστάσεις του παραθύρου.
- **glutInitWindowPosition**(int x, int y): Καθορίζει την αρχική θέση του παραθύρου στην οθόνη.
- **glutCreateWindow**(char* title): Δημιουργεί ένα παράθυρο υψηλού επιπέδου με τίτλο την εισαχθέα ροή.

2.2.1.d OpenGL/GLUT Functions

- **glClear**(GLbitfield mask): Καθαρίζει τα pixel της οθόνης με τις προκαθορισμένες τιμές που έχουν οριστεί πρωτίτερα με την αρχικοποίηση των αντίστοιχων καταστάσεων.
- **glColor3f**(GLfloat red, GLfloat green, GLfloat blue): Είναι μία από τις πολλές παραλλαγές της συνάρτησης που καθορίζει την κατάσταση του χρώματος σχεδίασης.
- **glBegin**(GLenum mode), glEnd(): Οτιδήποτε βρίσκεται εντός των δύο αυτών συναρτήσεων θα σχεδιαστεί στην οθόνη. Η συνάστηση glBegin(...) δέχεται σαν όρισμα την πρωτεύουσα γεωμετρική δομή σχεδίασης.
- **glVertex3f**(GLfloat x, GLfloat y, GLfloat z): Είναι μία από τις πολλές παραλλαγές της συνάρτησης που ορίζει τη θέση ενός κόμβου σχεδίασης.

- **glFlush**(void): Αδειάζει τα buffer, προκειμένου να δημιουργήσει χώρο και να επιταγχύνει τη διαδικασία σχεδίασης σε πεπερασμένο χρόνο.
- **glClearColor**(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha): Προσδιορίζει το ποσοστό συμμετοχής του κόκκινου, πράσινου, μπλε και αδειαφάνειας με το οποίο θα καθαρίζεται ο ρυθμιστής χρώματος. Οι προκαθορισμένες τιμές των παραμέτρων είναι 0.
- **glMatrixMode**(GLenum mode): Καθορίζει ποια στοίβας μήτρας είναι ο στόχος για τις επόμενες λειτουργίες μήτρας. Οι επιτρεπόμενες τιμές είναι οι GL_PROJECTION, GL_MODELVIEW, GL_TEXTURE. Η προκαθορισμένη τιμή είναι GL_MODELVIEW.
- **glLoadIdentity**(void): Αντικαθιστά την υφιστάμενη μήτρα με τη μήτρα που καθορίστηκε από τη συνάρτηση glMatrixMode(...).
- **glOrtho**(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal): Καθορίζει τις συντεταγμένες του αριστερού και δεξιού κατακόρυφων επιπέδων, του κάτω και πάνω οριζόντιων επιπέδων και των επιπέδων του βάθους.
- **glutInit**(int *argcp, char **argv): Αρχικοποιεί τα συστήματα και υποσυστήματα της βιβλιοθήκης GLUT.
- **glutMainLoop**(void): Εισέρχεται στον κύριο ατέρμονο βρόγχο κι εκτελεί όλες τις παραπάνω συναρτήσεις. Δίχως αυτή τη συνάρτηση, δεν εμφανίζεται τίποτα στην οθόνη.

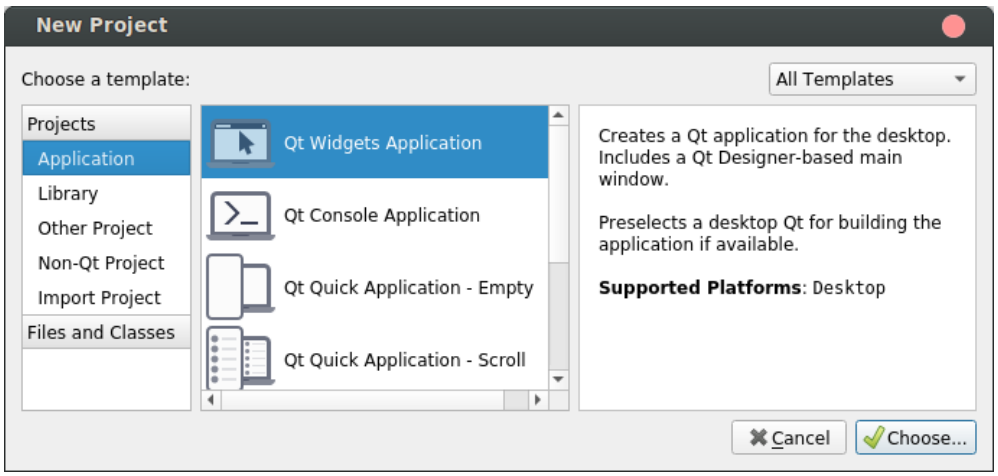
Σε αυτό το σημείο αξίζει να σημειώθεί, ότι μπορούν να βρεθούν εκτενέστερες περιγραφές για τις παραπάνω συναρτήσεις της OpenGL και GLUT στους παρακάτω ιστοχώρους.

- **https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/**
- **https://www.opengl.org/resources/libraries/glut/spec3/node1.html**

2.3 Δημιουργία Παραθύρου με Χρήση του QtCreator

Η Δημιουργία παραθύρου με χρήση του περιβάλλοντος QtCreator, απαιτεί κάποια στάδια, τα οποία διευκολύνουν τη διαδικασία. Τα στάδια είναι τα εξής:

New Project → Application → Qt Widgets Application



Σχήμα 2.2: Δημιουργία Qt Widgets Application

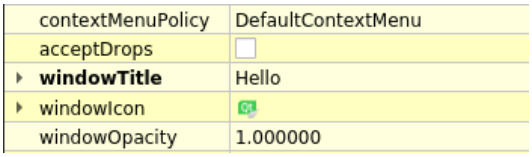
Στη συνέχεια πραγματοποιούνται οι κατάλληλες επιλογές και δημιουργούνται τα αρχεία **main.cpp**, **mainwindow.h**, **mainwindow.cpp** και **mainwindow.ui**.

Πριν ωστόσο αρχίσει το βασικό κομμάτι του προγραμματισμού, πρέπει να πραγματοποιηθούν δύο ακόμα στάδια.

Το πρώτο στάδιο είναι δημιουργία της κλάσης **GLWidget**, η οποία πρέπει να συνοδεύεται από τη βιβλιοθήκη **QGLWidget**. Η δημιουργία της κλάσης δημιουργεί τα αρχεία **glwidget.h** και **glwidget.cpp**.

Το δεύτερο στάδιο είναι η τροποποίηση του αρχείο ProjectName.pro, ώστε να περιέχει τους φακέλους και τις κατάλληλες σημαίες, για να αναγνωρίζει τις συναρτήσεις της βιβλιοθήκης OpenGL και τη διαδικασία του compile.

Τέλος χρειάζεται να σημειωθεί ότι για στο αρχείο mainwindow.ui πρέπει να δημιουργηθεί ένα Widget, το οποίο στη συνεχεία θα συνδεθεί με την κλάση GLWidget, χρησιμοποιώντας της επιλογής **Promote to**. Όσον αφορά την αλλαγή του ονόματος του παραθύρου, στο αρχείο mainwindow.ui και στις ιδιότητες του MainWindow αρκεί να αλλάξει το όνομα, όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 2.3: Αλλαγή του Ονόματος Παραθύρου

Στις επόμενες σελίδες φαίνονται τα αρχεία κώδικα.

```
#-------------------------------------------------
#
# Project created by QtCreator 2018-02-05T17:43:03
#
#-------------------------------------------------

QT       += core gui opengl

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = 03_Qt_version
TEMPLATE = app

# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before
Qt 6.0.0


SOURCES += \
        main.cpp \
        mainwindow.cpp \
    glwidget.cpp

HEADERS += \
        mainwindow.h \
    glwidget.h

FORMS += \
        mainwindow.ui

LIBS += -lGL -lGLEW -lglut -lGLU
```

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

```
#ifndef GLWIDGET_H
#define GLWIDGET_H

#include <QGLWidget>
#include <QTimer>

class GLWidget : public QGLWidget
{
    Q_OBJECT
public:
    explicit GLWidget(QWidget *parent = nullptr);

    void initializeGL(void);

    void paintGL(void);

    void resizeGL(int w, int h);

private:
    QTimer GLtimer;
};

#endif // GLWIDGET_H
```

```
#include "glwidget.h"
#include <GL/glu.h>

GLWidget::GLWidget(QWidget *parent) : QGLWidget(parent)
{
    //Update Window every 30ms
    connect(&GLtimer, SIGNAL(timeout()), this, SLOT(updateGL()));
    GLtimer.start(30);
}

/* Same as init(void) */
void GLWidget::initializeGL(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0); //Set Clear Color State

}

/* Same as display(void) */
void GLWidget::paintGL(void) {
    glClear(GL_COLOR_BUFFER_BIT); //Clear Window Background

    glColor3f(1.0, 1.0, 1.0); //Set Draw Color to WHITE

    //Draw rectangle
    glBegin(GL_POLYGON);
        glVertex3f(0.25, 0.25, 0.0);
        glVertex3f(0.75, 0.25, 0.0);
        glVertex3f(0.75, 0.75, 0.0);
        glVertex3f(0.25, 0.75, 0.0);
    glEnd();
}

/* Same as reshape(int w, int h) */
void GLWidget::resizeGL(int w, int h) {
    glViewport(0, 0, (GLint)w, (GLint)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
```

Αυτά που πρέπει να σημειωθούν είναι ότι το αρχείο ProjectName.pro πρέπει να είναι όπως φαίνεται παραπάνω, ενώ τα αρχεία main.cpp, mainwindow.h και mainwindow.cpp όσον αφορά τη διαχείριση της OpenGL μένουν ως έχουν. Τα αρχεία mainwindow.h και mainwindow.cpp αποτελούν το συνδετικό κρίκο όλων των διαδικάσιών που πραγματοποιούνται από την εφαρμογή, ωστόσο αυτό έχει να κάνει καθαρά με τη διαχείριση Qt παραθύρων κι έτσι δε θα πραγματοποιηθούν εκτενείς αναλύσεις (στο παρόν εγχειρίδιο θα αναλύονται μόνον ότι έχει να κάνει με την OpenGL και το πως αυτή μπορεί να συνεργαστεί με το Qt).

2.3.1 Ανάλυση των Συναρτήσεων

Αυτό που πρέπει να επισημανθεί σε αυτή την ενότητα είναι ότι στο περιβάλλον Qt, εφόσον στο αρχείο glwidget.h περιέχεται η βιβλιοθήκη **QGLWidget**, είναι πως δε χρειάζεται να γίνει τίποτε περισσότερο από τη δημιουργία των τριών βασικών συναρτήσεων της βιβλιοθήκης OpenGL με συγκεκριμένο όνομα. Επιπλέον χρειάζεται να δημιουργηθεί κι ένα χρονόμετο QTimer, το οποίο θα ανανεώνει ανά τακτά χρονικά διαστήματα το παράθυρο.

Αυτό επιτυγχάνεται με τον εξής τρόπο:

- Στην κλάση GLWidget δημιουργείται μία ιδιωτική μεταβλητή QTimer GLtimer.
- Στο constructor της κλάσης στο αρχείο glwidget.cpp προστίθενται οι εξής γραμμές:
  connect(&GLtimer, SIGNAL(timeout()), this, SLOT(UpdateGL()));
  GLtimer.start(30);

Τέλος οι τρεις βασικές συναρτήσεις που πρέπει να περιέχονται στην κλάση είναι οι εξής:

- initializeGL(void)
- paintGL(void)
- resizeGL(int w, int h)

2.3.1.a initializeGL(void)

Η συνάρτηση αυτή είναι η ίδια με την init(void) που χρησιμοποιείται στη δημιουργία παραθύρου μέσω GLUT. Αυτό που πρέπει να παρατηρηθεί είναι πως στη συνάρτηση αυτή τοποθετούνται μόνον οι συναρτήσεις που δίνουν ορίζουν αρχικές τιμές. Οι συναρτήσεις που διαχειρίζονται τους προσανατολισμούς και γενικά το πως θα προβληθούν στην οθόνη τα γεωμετρικά στοιχεία, τοποθετούνται στη συνάρτηση resizeGL(int w, int h).

2.3.1.b paintGL(void)

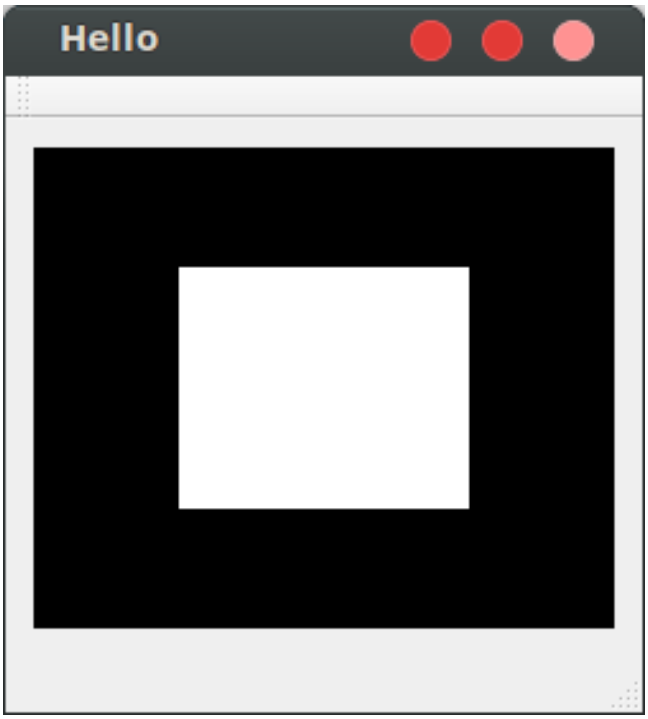
Η συνάρτηση αυτή είναι η ίδια με την display(void) που χρησιμοποιείται στη δημιουργία παραθύρου μέσω GLUT. Οτιδήποτε χρειάζεται να σχεδιαστεί στην οθόνη, τοποθετείται σε αυτή τη συνάρτηση.

2.3.1.c resizeGL(int w, int h)

Η συνάρτηση αυτή απότελεί στην τρίτη βασική συνάρτηση που απαιτείται να έχει ενα πρόγραμμα που χρησιμοποιεί τη βιβλιοθήκη OpenGL και αντιστοιχεί στη συνάρτηση reshape(int w, int h) που θα χρησιμοποιηθεί στο επόμενο παράδειγμα. Η συνάρτηση αυτή περιέχει όλα εκείνα τα στοιχεία που απαιτούνται ώστε να προσδιοριστεί το πως θα απεικονιστούν τα γεωμετρικά στοιχεία στην οθόνη.

2.3.1.d OpenGL Functions

- **glViewport**(GLint x, GLint y, GLsizei width, GLsizei height): Καθορίζει τα στοιχεία του αφινικού μετασχηματισμού των x και y από κανονικοποιημένες συντεταγμένες σε συντεταγμένες παραθύρου.



Σχήμα 2.4: Το απεικονιζόμενο παράθυρο με χρήση του QtCreator

Κεφάλαιο 3

Απεικόνιση Απλής Κίνησης

3.1 Βασική Θεωρεία

3.1.1 Ο Μηχανισμός της Κίνησης

Ένα από τα πιο ενδιαφέροντα πράγματα που μπορούν να συμβούν με τα γραφικά υπολογιστών είναι ο σχεδιασμός κινούμενων εικόνων. Το αντικείμενο αυτό έχει ευρεία χρήση και μπορεί να απασχολήσει, είτε μηχανικούς που προσπαθούν να μελετήσουν όλες τις πλευρές ενός μηχανικού αντικειμένου, είτε από πιλότους που εκπαιδεύονται μέσω ενός προγράμματος προσομοίωσης, είτε απλά από προγραμματιστές βιντεοπαιχνιδιών. Γενικότερα η κίνηση είναι ένα από τα σημαντικότερα ζητήματα στα γραφικά υπολογιστών.

Σε μία κινηματογραφική ταινία, η κίνηση πραγματοποιείται από τη μία διαδοχική προβολή εικόνων, οι οποίες προβάλονται με ταχύτητα 24 καρέ/δευτερόλεπτο στην οθόνη. Κάθε καρέ μετακινείται πίσω από το φακό και το κλείστρο της μηχανής ανοίγει ώστε να προβληθεί το καρέ. Στη συνέχεια το κλείστο ξανακλείνει καθώς η ταινία μεταβάλεται στο επόμενο καρέ και η διαδικασία αυτή επαναλαμβάνεται. Αν και οι θεατές βλέπουν 24 διαφορετικά καρέ/δευτερόλεπτο, το μυαλό του ανθρώπου τα αναμειγνύει, δημιουργώντας κατά αυτό τον τρόπο την αίσθηση μίας απαλής κίνησης.

Τα σύγχρονα συστήματα προβολής προβάλουν κάθε καρέ δύο φορές με ταχύτητα 48 καρέ/δευτερόλεπτο προκειμένου να μειώσουν το θόρυβο που προκαλείται από τη γρήγορη αλληλουχία των εικόνων που δίνουν την αίσθηση πως οι εικόνες τρεμοπαίζουν στην οθόνη. Οι οθόνες γραφικών των υπολογιστών τυπικά ανανεώνονται (επανασχεδιάζουν την εικόνα) με ταχύτητα που προσεγγίζει τα 60 καρέ/δευτερόλεπτο έως 76 καρέ/δευτερόλεπτο, ενώ υπάρχουν και περιπτώσεις που μπορούν να φτάσουν έως και ταχύτητα 120 καρέ/δευτερόλεπτο. Είναι φανερό πως ταχύτητα 60 καρέ/δευτερόλεπτο είναι πιο ομαλή από ταχύτητα 30 καρέ/δευτερόλεπτο και ταχύτητα 120 καρέ/δευτερόλεπτο είναι ελαφρώς καλύτερη από ταχύτητα 60 καρέ/δευτερόλεπτο. Ωστόσο δεν έχει νόημα να γίνεται συζήτηση για ταχύτητα πάνω από 120 καρέ/δευτερόλεπτο, αφότου αυτή είναι η μέγιστη αντιληπτή από τον άνθρωπο ταχύτητα.

Προκειμένου η OpenGL να πετύχει τα παραπάνω αποτελέσματα, χρησιμοποιεί διπλές διεπαφές. Η διεπαφή D_1 προβάλεται στην οθόνη και είναι αυτό που φαίνεται κάποια χρονική στιγμή t_0, ενώ η διεπαφή D_2 βρίσκεται στην προσωρινή μνήμη και λειτουργεί ως καμβάς σχεδίασης της επόμενης εικόνας. Έτσι τη χρονική στιγμή $t_1 = t_0 + dt$ η διεπαφή D_2 είναι αυτή που προβάλεται στην οθόνη, ενώ η διεπαφή D_1 λειτουργεί ως καμβάς σχεδίασης και η διαδικασία αυτή επαναλμβάνεται έως ότου τερματιστεί η εφαρμογή. Το βήμα εναλλαγής dt των καρέ ισούται με το αντίστροφο της ταχύτητας εναλλαγής (fps = frame per second) των είκόνων, δηλαδή $dt = 1/fps$.

3.1.2 Ανανέωση με Παύσεις

Σε μερικές εκτελέσεις της OpenGL, εκτός από την απλή αλλαγή της εικονιζόμενης και της σχεδιαστικής διεπαφής, της ρουτίνας swap_the_buffers(), το σύστημα περιμένει μέχρι ο υφιστάμενος χρόνος ανανέωσης της οθόνης τελειώσει, ώστε η προηγούμενη διεπαφή να προβληθεί πλήρως. Η ρουτίνα αυτή επιπλέον επιτρέπει στη νέα διεπαφή να προβληθεί ολοκληρωμένη, ξεκινώντας από την αρχή.

Για παράδειγμα, ας γίνει η υπόθεση ότι ένα σύστημα ανανεώνεται 60 φορές/δευτερόλεπτο, αυτό σημαίνει ότι η μέγιστη ταχύτητα που μπορεί να επιτευχθεί είναι 60fps και εάν όλα τα καρέ καθαρίζονται κι επανασχεδιάζονται σε λιγότερο από $\frac{1}{60}sec$, τότε η κίνηση θα προβάλεται ομαλά σε αυτό το ρυθμό. Αυτό όμως που συμβαίνει σε ένα πραγματικό σύστημα είναι ότι τα καρέ είναι αρκετά πολύπλοκά ώστε να σχεδιάζονται σε χρόνο $\frac{1}{60}sec$, με αποτέλεσμα κάθε καρέ να προβάλεται περισσότερες από μία φορές. Εάν για παράδειγμα, το κάθε καρέ χρειάζεται $\frac{1}{45}sec$ για να σχεδιαστεί, τότε η ταχύτητα μειώνεται στα 30fps και τα γραφικά προβάλονται ιδανικά για χρόνο $\frac{1}{30} - \frac{1}{45} = \frac{1}{90}spf$(second per frame), ή αλλιώς το 1/3 του χρόνου.

Επιπλέον, ο χρόνος ανανέωσης ενός βίντεο είναι σταθερός, το οποίο μπορεί να έχει προβλήματα στην απόδοση. Για παράδειγμα με $\frac{1}{60}spr$ (second per refresh) monitor και ένα σταθερό χρόνο εναλλαγής των καρέ, το σύστημα μπορεί να τρέχει με ταχύτητες 60fps, 30fps, 20fps, 15fps, 12fps και λοιπά ($\frac{60}{1}$, $\frac{60}{2}$, $\frac{60}{3}$, $\frac{60}{4}$, $\frac{60}{5}$, ..., $\frac{60}{n}$). Αυτό σημαίνει πως όταν προγραμματίζεται μία εφαρμογή και προστίθενται σταδιακά διάφορα χαρακτηριστικά, αρχικά κάθε χαρακτηριστικό που προστίθεται δεν έχει κανένα ίχνος χρονικής καθυστέρησης ως προς τη συνολική απόδοση, ενώ κάποια στιγμή, με την προσθήκη ενός ακόμα χαρακτηριστικού, το σύστημα δε μπορεί να σχεδιάσει τη σκηνή του καρέ σε χρόνο $\frac{1}{60}sec$ κι έτσι η κίνηση μειώνεται από 60fps στα 30fps, επειδή χάνει το πρώτο χρονικό περιθώρειο της πρώτης εναλλαγής της διεπαφής. Αντίστοιχα συμβαίνει και όταν ο χρόνος σχεδίασης αυξάνεται περισσότερο από $\frac{1}{30}sec$ (η κίνησει μειώνεται από 30έν φπς στα 20fps).

Στην περίπτωση όπου η χρονική καθυστέρηση μεταξύ των καρέ είναι οριακή, δηλαδή έστω ότι το ένα καρέ μεταβάλλεται με ταχύτητα 60fps, ενώ το επόμενο με ταχύτητα 30fps και μετά πάλι μεταβολή με ταχύτητα 60fps, τότε το αποτέλεσμα είναι ενοχλητικό και συνίσταται η προσθήκη μίας επιπλέον καθυστέρησης, ώστε όλα τα καρέ να μεταβάλλονται σε χρόνο $\frac{1}{30}sec$, ώστε το τελικό αποτέλεσμα να φαίνεται ομαλότερο. Εάν η κατάσταση τείνει να είναι πιο πολύπλοκη, τότε χρειάζεται μία πιο σοφιστική μέθοδος προσέγγισης.

Γενικά αυτό που χρειάζεται να έχει υπόψιν ο προγραμματιστής είναι πως ο χρόνος μεταβολής της κίνησης υπολογίζεται από την παρακάτω σχέση:

$$Motion = Redraw_{scene} + Swap_{buffers} \tag{3.1}$$

3.2 Απεικόνιση Κίνησης σε Περιβάλλον GLUT

main.cpp 1

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdlib.h>

#define MY_WIN_WIDTH 250
#define MY_WIN_HEIGHT 250
#define MY_WIN_POS_X 100
#define MY_WIN_POS_Y 100
#define MY_WIN_TITLE "Motion"

static GLfloat spin =0.0;

void init(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glRectf(-25.0, -25.0, 25.0, 25.0);
    glPopMatrix();
    glutSwapBuffers();
}

void reshape(int w, int h) {
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void spinDisplay(void) {
    spin = spin + 2.0;
    if(spin > 360)
        spin -= 360;
    glutPostRedisplay();
}

void mouse(int button, int state, int x, int y) {
    switch(button) {
    case GLUT_LEFT_BUTTON: {
        if(state == GLUT_DOWN)
            glutIdleFunc(spinDisplay);
        break;
    } case GLUT_MIDDLE_BUTTON: {
        if(state == GLUT_DOWN)
            glutIdleFunc(NULL);
        break;
    } default:
        break;
    }
}

void initializeGL(void) {
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(MY_WIN_WIDTH, MY_WIN_HEIGHT);
    glutInitWindowPosition(MY_WIN_POS_X, MY_WIN_POS_Y);
    glutCreateWindow(MY_WIN_TITLE);
}
```

```
/*
 * Request double buffer display mode.
 * Register mouse input callback functions.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    initializeGL();

    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutMainLoop();

    return 0;
}
```

Το παράδειγμα αυτό είναι συνέχεια του προηγούμενου. Ο κώδικας που παρουσιάζεται παραπάνω ανοίγει ένα παράθυρο, στο κέντρο του οποίου εμφανίζεται ένα άσπρο ορθογώνιο. Τα νέα χαρακτηριστικά που εισάγει ο κώδικας είναι τα εξής:

- Τη συνάρτηση **reshape**(int w, int h), η οποία όπως ειπώθηκε στην προηγούμενη ενότητα διαχειρίζεται τους πίνακες προσανατολισμού και γενικότερα όλες εκείνες τις εξισώσεις που προσδιορίζουν την απεικόνιση των γεωμετρικών στοιχείων στην οθόνη. Η συνάρτηση αυτή εκτελείται κάθε φορά που αλλάζει διαστάσεις το παράθυρο.
- Απλή αλληλεπίδραση του χρήστη με το περιεχόμενο του παραθύρου, με τη χρήση του ποντικιού.
- Απλή στροφική κίνηση του ορθογωνίου, γύρω από το κέντρο μάζας του.

3.2.1 Ανάλυση των Συναρτήσεων

Σε αυτό το παράδειγμα εισήχθησαν οι εξής συναρτήσεις:

- reshape(int w, int h)
- mouse(int button, int state, int x, int y)

3.2.1.a reshape(int w, int h)

Περιέχει τις συναρτήσεις της OpenGL, οι οποίες προσδιορίζουν το πως θα προβληθούν τα γεωμετρικά χαρακτηριστικά στην οθόνη. Η συνάρτηση αυτή αποτελεί την τρίτη βασική συνάρτηση που πρέπει να έχει ένα πρόγραμμα OpenGL, για να δημιουργεί χρήσιμες εφαρμογές. Καλείται από τη συνάρτηση **glutReshapeFunc(...)** κι εκτελείται κάθε φορά που το παράθυρο αλλάζει διαστάσεις.

3.2.1.b mouse(int button, int state, int x, int y)

Διαχειρίζεται τα σήματα που δέχεται το πρόγραμμα από το ποντίκι. Τα ορίσματά της πρέπει αυστηρά να είναι τα παραπάνω, διότι καλείται αργότερα από τη συνάρτηση **glutMouseFunc(...)**, ώστε να γίνει η διεπαφή μεταξύ της εφαρμογής και του ποντικιού (σύνδεση μεταξύ hardware-software).

3.2.1.c OpenGL/GLUT Functions

- **glShadeModel**(GLenum mode): Τα γεωμετρικά πρωτεύοντα μπορούν να έχουν είτε επίπεδο, είτε ομαλό τρόπο σκίασης (shading). Η ομαλή σκίαση, το προκαθορισμένο, επιβάλει στον υπολογισμό του χρώματος των κόμβων την παρεμβολή κατά τη διαδικασία ραστεροποίησης, τυπικά καθορίζει ένα διαφορετικό χρώμα για κάθε ένα pixel. Η επίπεδη σκίαση επιλέγει το υπολογιζόμενο χρώμα ενός κόμβου και με αυτό χρωματίζει όλα τα pixel κατά τη ραστεροποίηση ως εννιαίο πρωτεύον.

  Οι τιμές αυτές επιλέγονται με τη χρήση των σημαιών GL_SMOOTH και GL_FLAT.

- **glPushMatrix**(void): Υπάρχει μία στοίβα από πίνακες για κάθε σύστημα πινάκων. Στο GL_MODELVIEW σύστημα, το μέγεθος της στοίβας είναι τουλάχιστον 32. Στα άλλα συστήματα GL_COLOR, GL_PROJECTION και GL_TEXTURE, το μέγεθος της στοίβαε είναι τουλάχιστον 2. Ο υφιστάμενος πίνακας σε οποιοδήποτε σύστημα είναι ο πίνακας που βρίσκεται στην κορυφή της στοίβας του συστήματος.

  Η συνάρτηση αυτή λοιπόν, προωθεί τον υφιστάμενο πίνακα της στοίβας, έναν πίνακα κάτω, διπλασιάζοντας τον υφιστάμενο πίνακα. Έτσι λοιπόν, μετά το κάλεσμα της συνάρτησης glPushMatrix(void), ο πίνακας στην κορυφή της στοίβας είναι ίδιος με τον πίνακα που βρίσκεται ακριβώς από κάτω του.

- **glutPopMatrix**(void): Η συνάρτηση αυτή αντικαθιστά τον υφιστάμενο πίνακα που βρίσκεται στην κορυφή της στοίβας με τον ακριβώς από κάτω του πίνακα.

- **glRectf**(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2): Υποστηρίζει έναν ιδανικό τρόπο για τη δημιουργία ενός τετραγώνου, προσδιορίζοντας τα σημεία δύο αντιδιαμετρικών γωνιών. Το τελικό ορθογώνιο ορίζεται πάνω στο επίπεδο με z = 0.

- **glutSwapBuffers**(void): Πραγματοποιεί την αλλαγή των διεπαφών, εάν το παράθυρο έχει οριστεί ως παράθυρο διπλής διεπαφής (double buffered window).

- **glutPostRedisplay**(void): Μαρκάρει το υφιστάμενο παράθυρο έτσι όπως χρειάζεται ώστε να επαναπαρουσιαστεί.

- **glutIdleFunc**(void (*func)(void): Θέτει ένα καθολικό κάλεσμα, ώστε το πρόγραμμα που χρησιμοποιεί διαχείριση παραθύρων GLUT να πραγματοποιεί επεξεργασία παρασκήνιου ή να συνεχίζει την κίνηση όταν η δραστηριότητα στο σύστημα παραθύρου δεν έχει ακόμα ληφθεί.

3.3 Συνοπτική Περιγραφή του Αλγορίθμου Κίνησης

Όπως φαίνεται παραπάνω, ο αλγόριθμος ακολουθεί μερικά απλά βήματα. Συνοπτικά τα βήματα είναι τα εξής:

- Δημιουργία-Άνοιγμα Παραθύρου: Σε αυτό το στάδιο συμπεριλαμβάνονται οι συναρτήσεις αρχικοποίησης και προσδιορισμού των γεωμετρικών και ποιοτικών χαρακτηριστικών του παραθύρου.
- Επιλογή Προβολικού Συστήματος: Συμπεριλαμβάνεται η συνάρτηση reshape(int w, int h), η οποία προσδιορίζει το πως θα προβληθούν τα γεωμετρικά στοιχεία. Το βήμα αυτό εκτελείται κάθε φορά που αλλάζουν οι διαστάσεις του παραθύρου.
- Υπολογισμός των Συντεταγμένων: Σε κάθε επανάληψη επαναπροσδιορίζονται οι συντεταγμένες των γεωμετρικών στοιχείων. Εάν οι συντεταγμένες είναι ίδιες, τότε το σημείο θεωρείται σταθερό. Εάν αλλάζουν, τότε το αντικείμενο κινείται. Σημειώνεται πως ως κίνηση μπορεί να θεωρηθεί και η αλλάγή θέσης της κάμερας. Το πως θα πραγματοποιηθεί η κίνηση σε μία εφαρμογή, εξαρτάται καθαρά από την προβαλόμενη σκηνή και από την αίσθηση που πρέπει να δοθεί.
- Προβολή της Σκηνής (Εναλλαγή των Διεπαφών): Σε αυτό το στάδιο προβάλεται η σκηνή στην οθόνη, εναλλάσσοντας τις διεπαφές.
- Επανάληψη των δύο τελευταίων σταδίων.

3.4 Απεικόνιση Κίνησης σε Περιβάλλον QtCreator

mainwindow.h 1

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_SpinIt_clicked();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_SpinIt_clicked() //When Button Clicked
{
    ui->glWidget->changeSpinState();
}
```

```
#ifndef GLWIDGET_H
#define GLWIDGET_H

#include <QGLWidget>
#include <QMouseEvent>
#include <QTimer>

class GLWidget : public QGLWidget
{
public:
    GLWidget(QWidget *parent);

    void initializeGL(void); //init(void)

    void resizeGL(int w, int h); //reshape(int h, int w)

    void paintGL(void); //display(void)

    void mousePressEvent(QMouseEvent* event); //Check Mouse Events

    void changeSpinState();

private:
    QTimer glTimer;
    GLdouble spin;
    bool spinState;

    void spinDisplay(void);
};

#endif // GLWIDGET_H
```

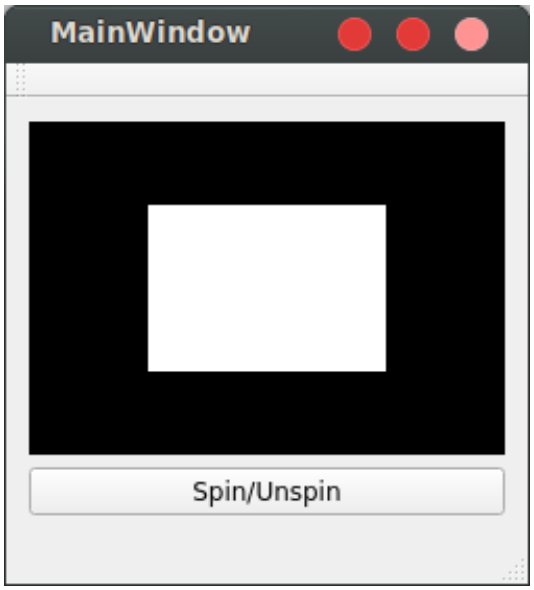
```
#include "glwidget.h"

GLWidget::GLWidget(QWidget* parent) :
    QGLWidget(parent)
{
    connect(&glTimer, SIGNAL(timeout()), this, SLOT(updateGL()));
    glTimer.start(30);

    spin = 0.0;
    spinState = false;
}

/* Same as
 *  void init(void){...}
 * function.
 */
void GLWidget::initializeGL(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
}

/* Same as
 *  void reshape(int w, int h){...}
 * function. There isn't any need to call
 *  glutReshapeFunc(...)
 * to run this function.
 */
void GLWidget::resizeGL(int w, int h) {
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

/* Same as
 *  void display(void){...}
 * function. There isn't any need to call
 *  glutDisplayFunc(...)
 * to run this function.
 */
void GLWidget::paintGL(void) {
    if(spinState)
        spinDisplay();
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glRectf(-25.0, -25.0, 25.0, 25.0);
    glPopMatrix();
}

//Mouse Event Handling on QT
void GLWidget::mousePressEvent(QMouseEvent* event) {
    if(event->button() == Qt::LeftButton) {
        spinState = true;
    } else if (event->button() == Qt::MiddleButton) {
        spinState = false;
    } else {
        ;
    }
}
```

```
void GLWidget::spinDisplay(void) {
    spin += 2.0;
    if(spin >360)
        spin -= 360;
}

void GLWidget::changeSpinState() {
    spinState = (spinState == true) ? false : true;
}
```

Από αυτό το παράδειγμα και μετά, στην ενότητα αυτή θα προστίθενται μόνο τα τμήματα κώδικα που διαφοροποιούνται από το αρχικό. Δηλαδή το αρχείο κώδικα **main.cpp**, είναι το ίδιο με το πρώτο παράδειγμα κώδικα, οπότε δεν υπάρχει λόγος υπόδειξής του. Μία δεύτερη επισήμανση που πρέπει να γίνει σε αυτό το σημείο, είναι ότι ο κώδικας στο περιβάλλον QtCreator θα έχει επιπλέον προσθήκες, προκειμένου να προσεγγίζει όσο το δυνατόν γίνεται μία εμφανίσημη εφαρμογή. Στο συγκεκριμένο παράδειγμα, εκτός από την αλληλεπίδραση του χρήστη με το παράθυρο μέσω του ποντικιού, προστίθεται κι ένα PushButton (κουμπί), το οποίο κάνει ακριβώς το ίδιο, αλλά με πιο κομψό τρόπο.

Οι διαφορές του περιβάλλοντος QtCreator με το περιβάλλον GLUT είναι οι εξής:

- **mousePressEvent**(QMouseEvent* event): Η αντίστοιχη συνάρτηση που διαχειρίζεται τα σήματα του ποντικιού.
- **changeSpinState**(void): Προκειμένου να ελεγχθεί το αν πρέπει η όχι να πραγματοποιηθεί η κίνηση, ορίζεται μία μεταβλητή **bool**, η οποία έχει τιμή **true** όταν πρέπει να γίνει κίνηση και τιμή **false** όταν δεν πρέπει να γίνει κίνηση. Η συνάρτηση αυτή χρησιμοποιείται από το PushButton.
- **on_SpinIt_Clicked**(void): Είναι ένα Slot του QtCreator, το οποίο σημαίνει πως όταν γίνει κλικ στο PushButton με όνομα SpinIt, θα εκτελεστεί το αντίστοιχο τμήμα κώδικα που υπάρχει στο αρχείο mainwindow.cpp και αντιστοιχεί στη συγκεκριμένη συνάρτηση.



Σχήμα 3.1: Το απεικονιζόμενο παράθυρο με χρήση του QtCreator. Εάν παρηθεί το κουμπί Spin/Unspin το ορθογώνιο αρχίζει να στρέφεται ή σταματά την κίνηση. Το τετράγωνο κινείται επίσης με αριστερό κλικ στο παράθυρο και σταματάει πιέζοντας τη ροδέλα.