

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Кафедра ПЗ

**МЕТОДИЧНІ ВКАЗІВКИ**

до лабораторних робіт  
з дисципліни «Комп'ютерна графіка та обробка зображень» для  
студентів ОР «бакалавр»  
за спеціальностями 121 «Інженерія програмного забезпечення»  
та 122 «Комп'ютерні науки»

Методичні вказівки до лабораторних робіт з дисципліни «Комп'ютерна графіка та обробка зображень» для студентів ОР «бакалавр» за спеціальностями 121 «Інженерія програмного забезпечення» та 122 «Комп'ютерні науки» / Укл.: А.В. Пархоменко, Ж.К. Камінська, М.В. Калініна – Запоріжжя: НУЗП, 2023. – 98 с.

Укладачі: А.В. Пархоменко, к.т.н., доцент кафедри ПЗ,  
Ж.К. Камінська, асистент кафедри ПЗ,  
М.В. Калініна, асистент кафедри ПЗ.

Рецензент: Н.О. Миронова, к.т.н., доц. каф. ІТЕЗ.

Відповідальний  
за випуск: С.О. Субботін, зав. каф. ПЗ, д.т.н., професор.

Затверджено  
на засіданні кафедри  
Програмних засобів  
Протокол № 12  
від 09.06.2023 р.

## ЗМІСТ

<b>1. ЛАБОРАТОРНА РОБОТА № 1 .....</b>	<b>5</b>
<b>1.1 Мета роботи .....</b>	<b>5</b>
<b>1.2 Стислі теоретичні відомості .....</b>	<b>5</b>
<b>1.3 Методика виконання лабораторної роботи ....</b>	<b>6</b>
<i>1.3.1 Створення простого зображення .....</i>	<i>6</i>
<b>1.4 Завдання на виконання лабораторної роботи .</b>	<b>29</b>
<b>1.5 Зміст звіту .....</b>	<b>31</b>
<b>1.6 Контрольні питання.....</b>	<b>31</b>
<b>2. ЛАБОРАТОРНА РОБОТА № 2 .....</b>	<b>32</b>
<b>2.1 Мета роботи .....</b>	<b>32</b>
<b>2.2 Стислі теоретичні відомості .....</b>	<b>32</b>
<i>2.2.1 Створення тривимірних об'єктів .....</i>	<i>32</i>
<i>2.2.2 Функції роботи з об'ктом.....</i>	<i>33</i>
<b>2.3 Методика виконання лабораторної роботи ..</b>	<b>34</b>
<i>2.3.1 Створення літер .....</i>	<i>34</i>
<b>2.4 Завдання на виконання лабораторної роботи .</b>	<b>46</b>
<b>2.5 Зміст звіту .....</b>	<b>46</b>
<b>2.6 Контрольні питання.....</b>	<b>46</b>
<b>3. ЛАБОРАТОРНА РОБОТА № 3 .....</b>	<b>47</b>
<b>3.1 Мета роботи .....</b>	<b>47</b>
<b>3.2 Стислі теоретичні відомості .....</b>	<b>47</b>
<b>3.2.1 Текстури .....</b>	<b>47</b>
<b>3.2.2 Освітлення .....</b>	<b>49</b>

3.2.2.1 Джерела спрямованого світла .....	50
3.2.2.2 Матеріал .....	51
3.2.2.3 Функції згасання .....	52
3.2.2.4 Прожектори .....	52
<b>3.3 Методика виконання лабораторної роботи ..</b>	<b>53</b>
3.3.1 Приклад накладення текстури .....	53
3.3.2 Приклад встановлення світла .....	55
3.3.3 Накладання текстури на 3d- об'єкт .....	62
<b>3.4 Завдання на виконання лабораторної роботи .</b>	<b>70</b>
<b>3.5 Зміст звіту .....</b>	<b>70</b>
<b>3.6 Контрольні питання.....</b>	<b>70</b>
<b>4. ЛАБОРАТОРНА РОБОТА № 4.....</b>	<b>71</b>
4.1 Мета роботи .....	71
4.2 Стислі теоретичні відомості .....	71
4.2.1 Створення ефекту мерехтіння .....	71
4.2.2 Створення ефекту туману .....	72
<b>4.3 Методика виконання лабораторної роботи ..</b>	<b>74</b>
4.3.1 Приклад програми (мерехтіння) .....	74
4.3.2 Приклад програми (ефект туману) .....	80
4.3.3 Приклад програми (розщеплення) .....	87
<b>4.4 Завдання на виконання лабораторної роботи .</b>	<b>97</b>
<b>4.5 Зміст звіту .....</b>	<b>97</b>
<b>4.6 Контрольні питання.....</b>	<b>97</b>
<b>5. РЕКОМЕНДОВАНА ЛІТЕРАТУРА .....</b>	<b>98</b>

## 1. ЛАБОРАТОРНА РОБОТА № 1

### «Створення простого зображення за допомогою бібліотеки Open GL у середовищі розробки Qt»

#### 1.1 Мета роботи

Ознайомитися з бібліотекою Open GL у середовищі розробки Qt та створити просте зображення.

#### 1.2 Стислі теоретичні відомості

OpenGL (Open Graphics Library – відкрита графічна бібліотека, графічний API) – специфікація, що визначає незалежний від мови програмування платформо-незалежний програмний інтерфейс для написання прикладних програм, що використовують двовимірну та тривимірну комп'ютерну графіку

На базовому рівні OpenGL – це просто специфікація, тобто документ, що описує набір функцій і їх точну поведінку. Виробники обладнання на основі цієї специфікації створюють реалізації – бібліотеки функцій, що відповідають набору функцій специфікації. Реалізація слугує для ефективного використання можливостей устаткування. Якщо апаратура не дозволяє реалізувати будь-яку можливість, вона повинна бути земульована програмно. Виробники повинні пройти специфічні тести (conformance tests – тести на відповідність) перед тим, як реалізація буде класифікована як OpenGL-реалізація. Таким чином, розробникам програмного забезпечення достатньо навчитися використовувати функції, що описані в специфікації, залишивши ефективну реалізацію останніх розробникам апаратного забезпечення.

Існує низка бібліотек створених поверх або як додаток до OpenGL. Наприклад, бібліотека GLU, що практично є стандартним додатком OpenGL і завжди її супроводжує, побудована поверх останньої, тобто, використовує її функції для реалізації своїх можливостей. Інші бібліотеки, такі як GLUT і SDL, створені для впровадження додаткових можливостей, недоступних в OpenGL. До таких можливостей відносяться створення інтерфейсу користувача (вікна, кнопки, меню тощо.), налаштування контексту малювання

(область малювання, що використовується OpenGL), обробка повідомлень від пристроїв вводу/виводу (клавіатура, миша тощо.), а також робота з файлами. Як правило, кожен віконний менеджер має власну бібліотеку-розширення для реалізації можливостей, що наведені вище, наприклад, WGL у Windows або GLX у X Window System, але бібліотеки GLUT та SDL є крос-платформними, що полегшує перенесення написаних прикладних програм на інші платформи.

## 1.3 Методика виконання лабораторної роботи

### 1.3.1 Створення простого зображення

Створюємо новий проєкт в QT 5.1.1:

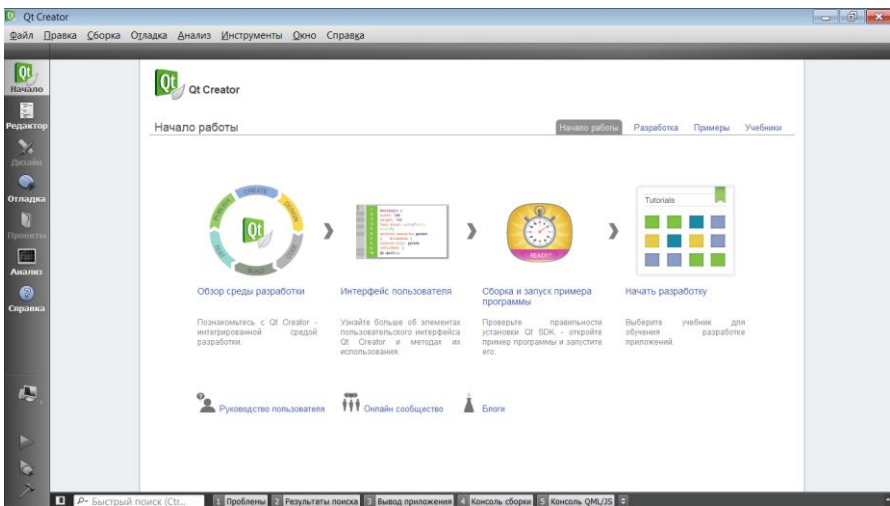


Рисунок 1.1 – Головне меню QT

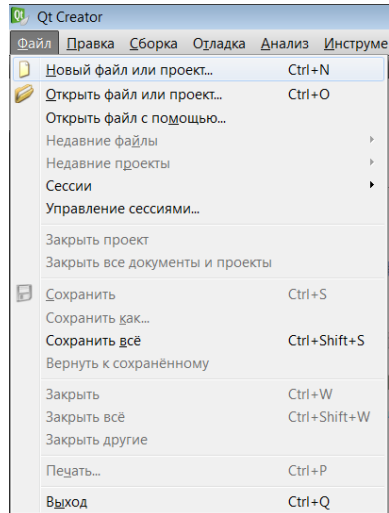


Рисунок 1.2 – Створення нового проекту

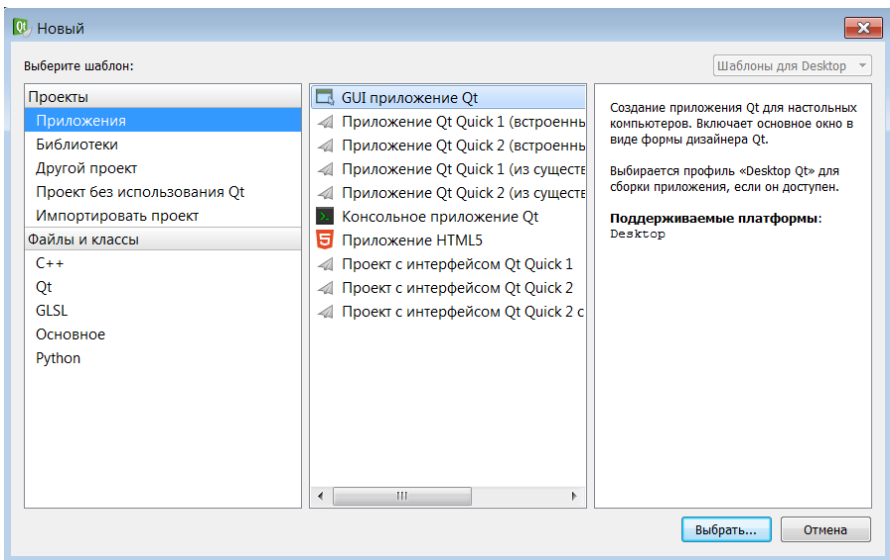


Рисунок 1.3 – Вибір шаблону

Обираємо – Прикладні програми(Приложения) → GUI програма Qt → Обрати. Вводимо ім'я проєкту, наприклад **Test\_OpenGL\_Lab1**, обираємо шлях до розташування проєкту і натискаємо Далі:

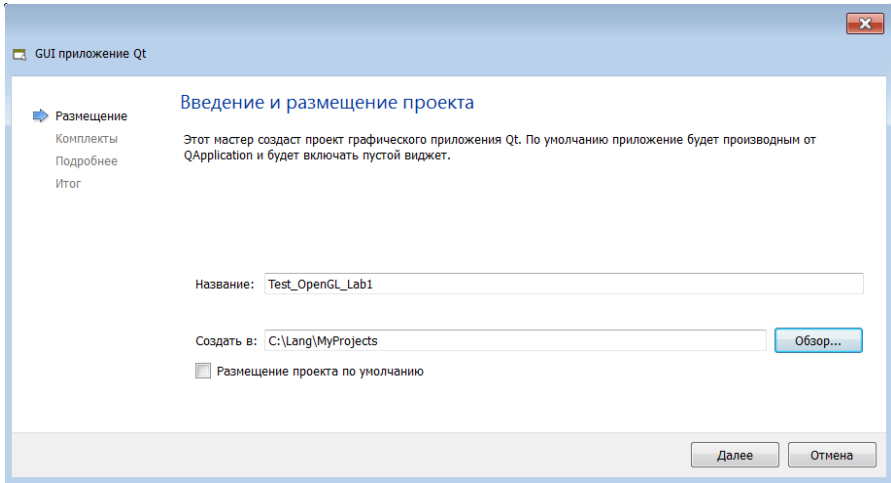


Рисунок 1.4 – Розміщення проєкту

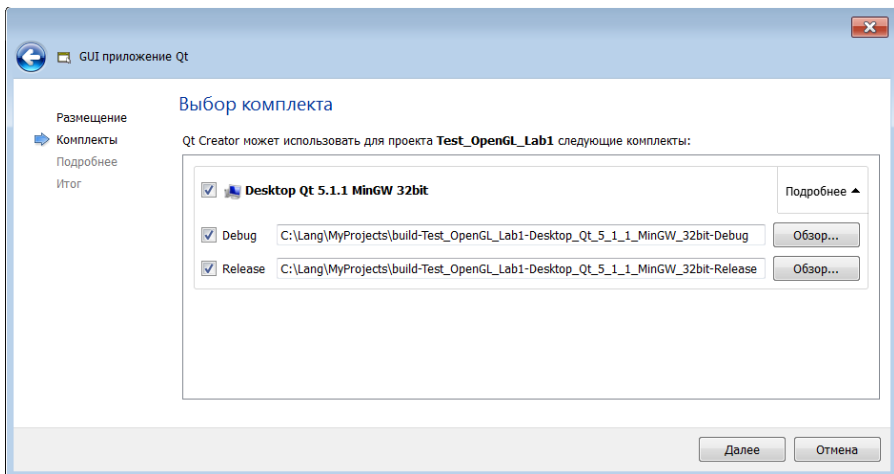


Рисунок 1.5 –Вибір комплекта



Тут за бажанням можна задати нове ім'я класу або залишити стандартне.

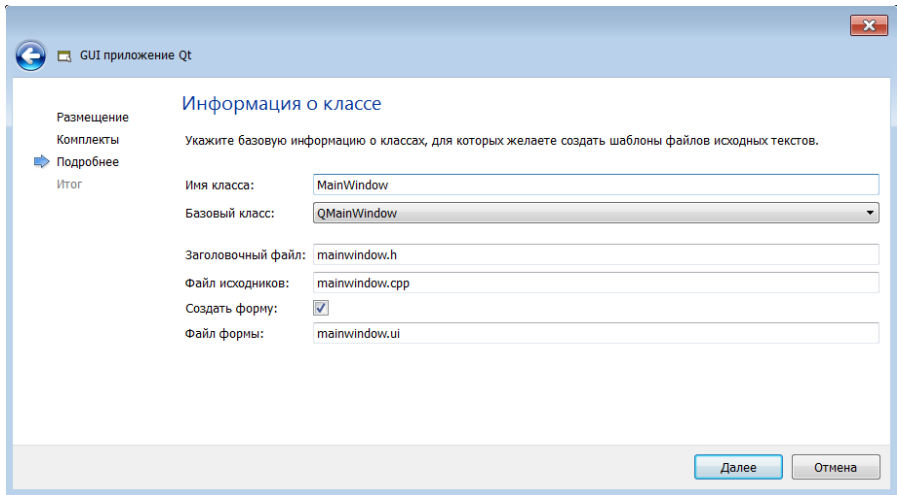


Рисунок 1.6 – Створення класу

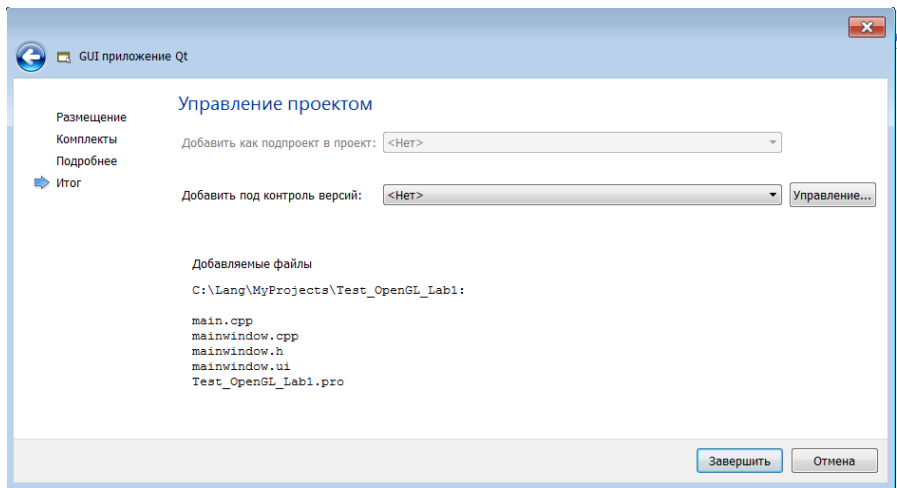


Рисунок 1.7 – Завершення створення проекту.

Отримуємо наступний проект з файлами:

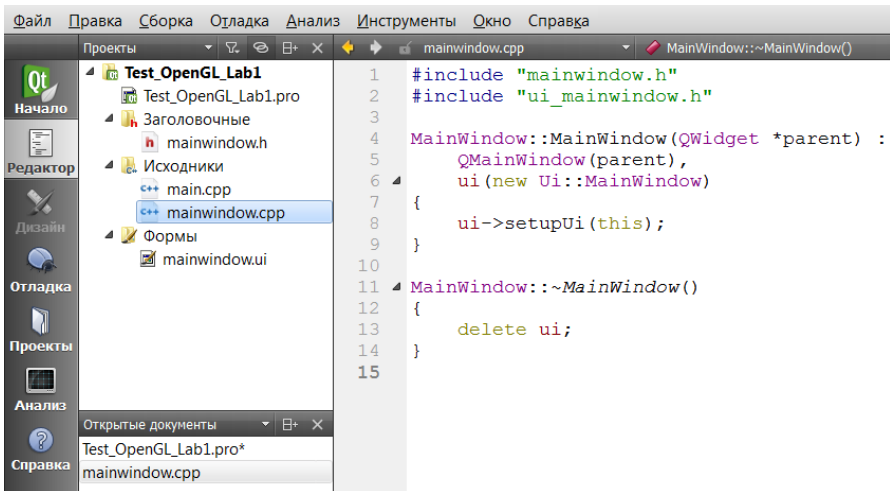


Рисунок 1.8 – Створений проект

Враховуючи, що ми працюємо з OpenGL, нам необхідно прописати в файлі *Test\_OpenGL\_Lab1.pro* наступне:

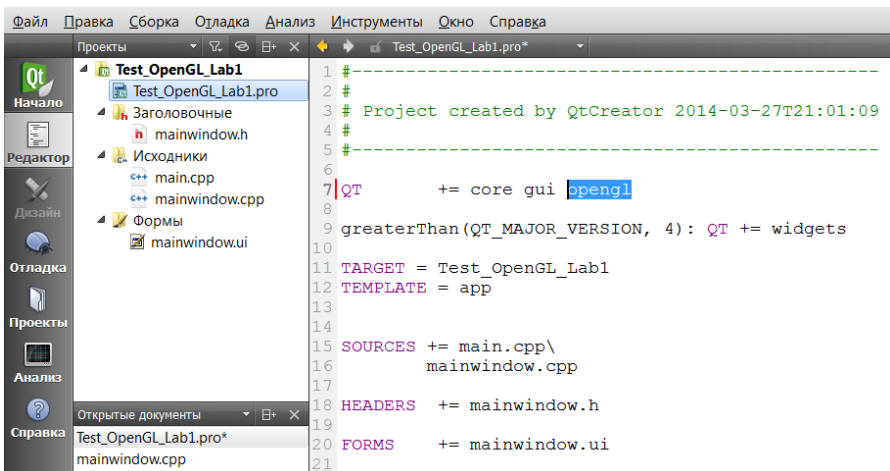


Рисунок 1.9 – Підключення OpenGL

Нам знадобиться додатковий клас, тому створимо його:

Натискаємо правою клавішею миші на *Test\_OpenGL\_Lab1.pro* та вибираємо *Додати новий...*:

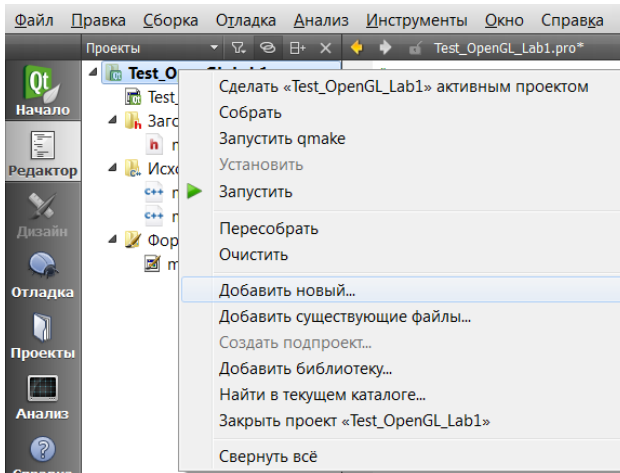


Рисунок 1.10 – Додавання класу

Обираємо: *C++*→*Клас C++*.

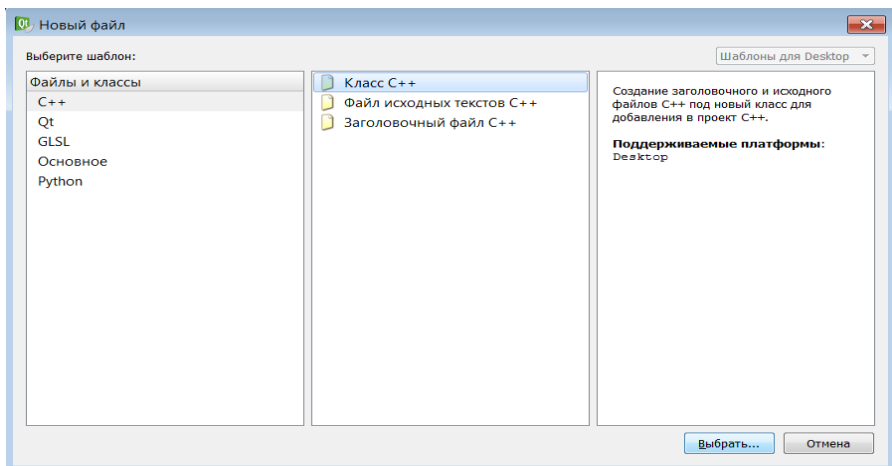


Рисунок 1.11 – Вибір класу

Вводимо дані – назва нашого класу буде *My\_Paint*; назва базового класу – *QGLWidget*.

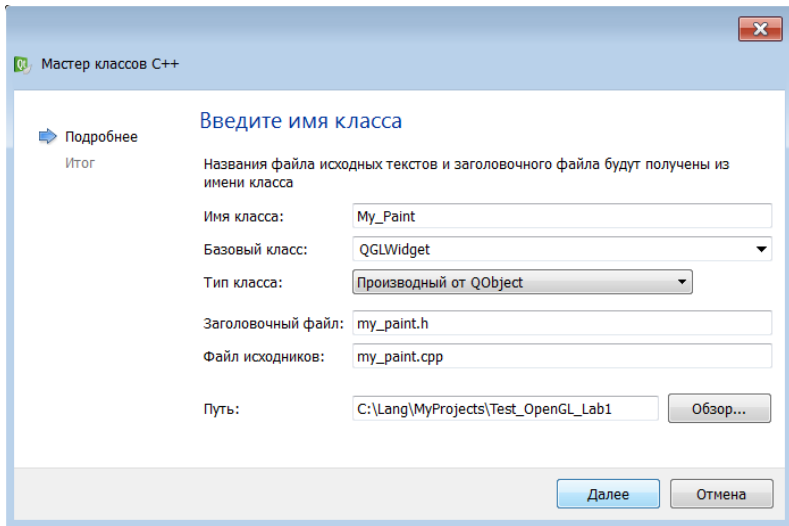


Рисунок 1.12 – Налаштування класу

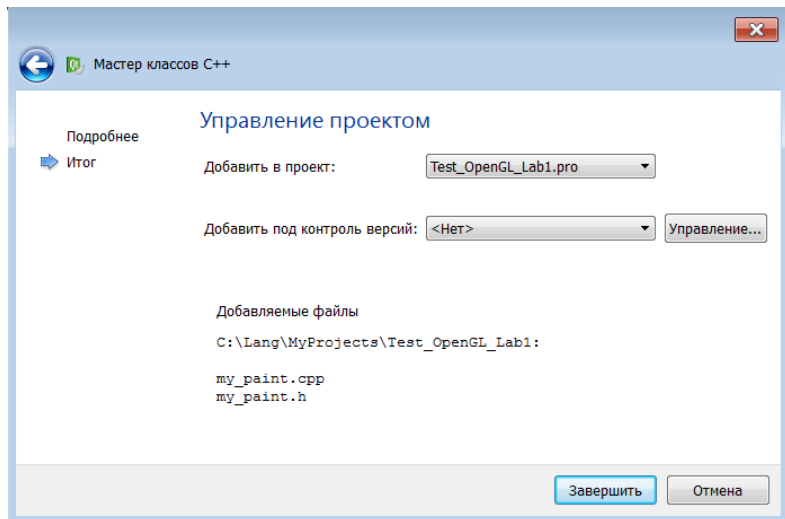


Рисунок 1.13 – Налаштування класу

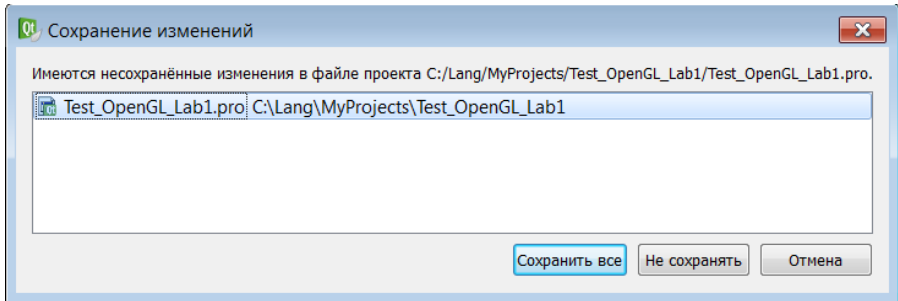


Рисунок 1.14 – Збереження змін в проєкті

Після збереження змін додалися дані у файлі *Test\_OpenGL\_Lab1.pro*.

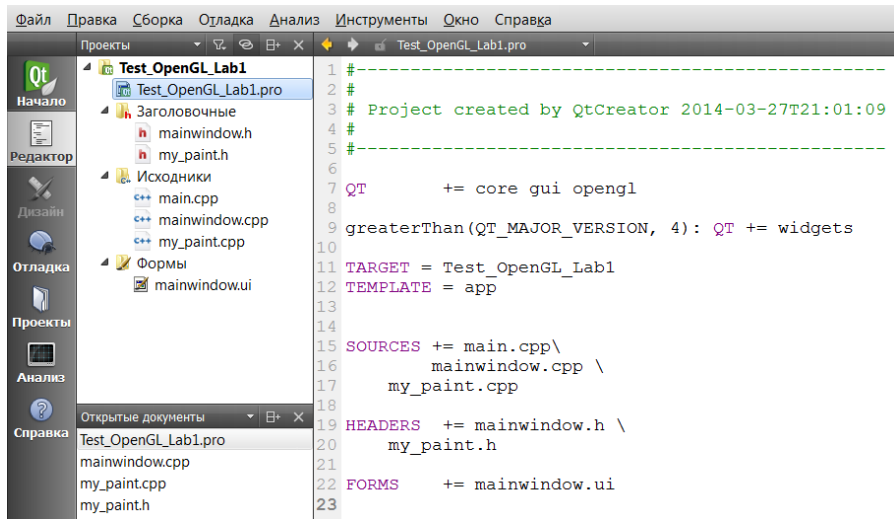


Рисунок 1.15 –Результат додавання класу

Додавши клас, заходимо у файл *my\_paint.h* та замість виділеного тексту на малюнку:

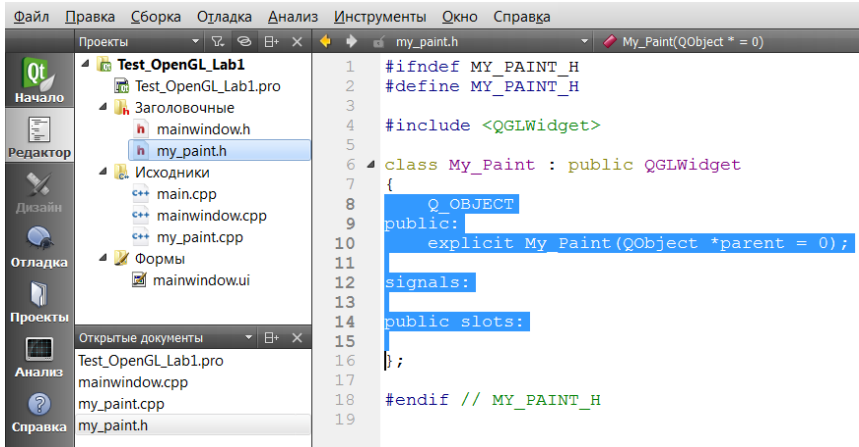


Рисунок 1.16 – Змінюємо текст

Заміняємо на наступне:

```

public:
    My_Paint();
    void initializeGL();
    void paintGL();
    void resizeGL(int w, int h);

```

Отримуємо наступне:

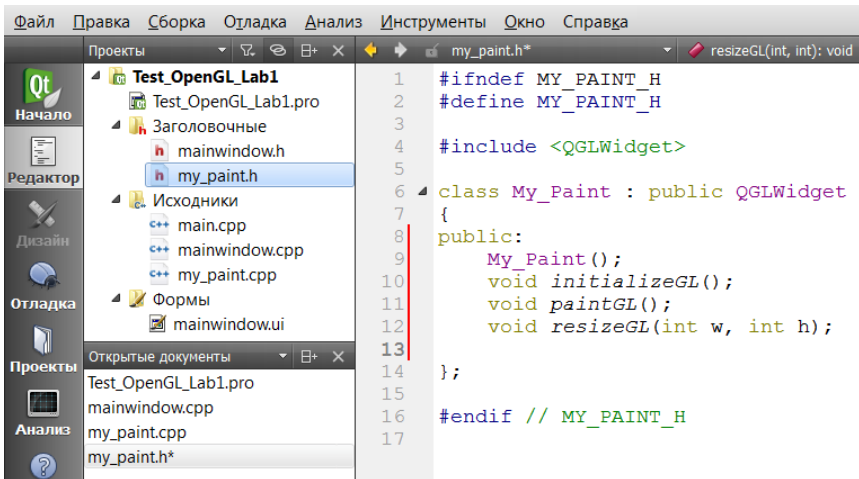


Рисунок 1.17 – Результат зміни тексту

Далі переходимо в *my\_paint.cpp*:

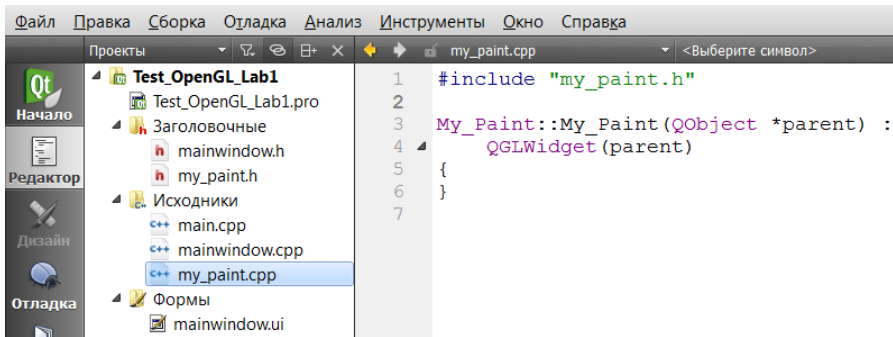


Рисунок 1.18 – Відображення my\_paint.cpp

Підключаємо бібліотеки:

```

#include <QtOpenGL>
#include <GL/gl.h>

```

Змінюємо опис класу на:

```
My_Paint::My_Paint() {}
```

І прописуємо наступне:

```

void My_Paint::initializeGL() {}
void My_Paint::paintGL() {}
void My_Paint::resizeGL(int w, int h) {}

```

Отримуємо:

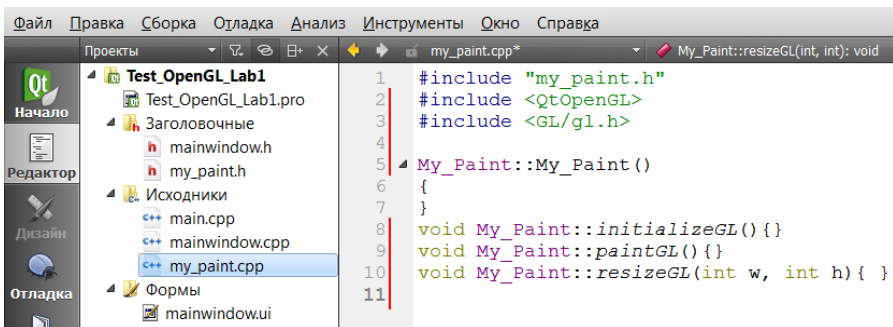


Рисунок 1.19 – Вносимо зміни

Далі заходимо на **Форми** → **mainwindow.ui**:

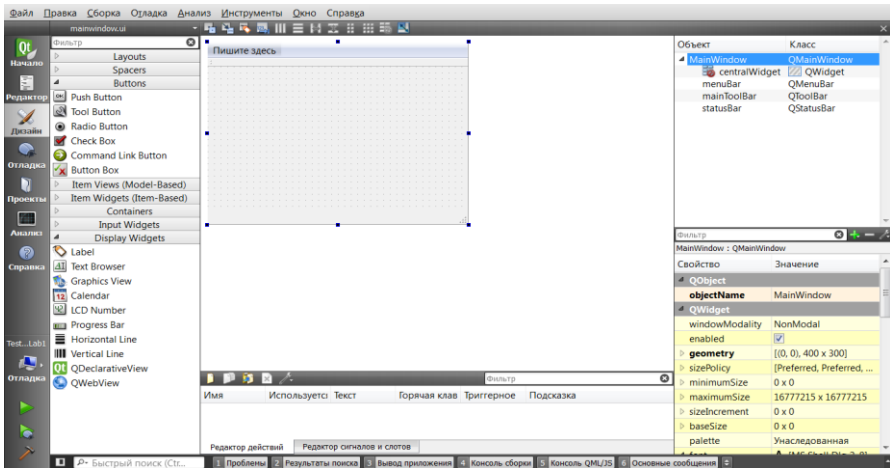


Рисунок 1.20 – Меню Форми → **mainwindow.ui**:

Видаляємо зайві об'єкти: панель меню – **menuBar**, панель інструментів – **mainToolBar** та **statusBar**.

Залишається лише **QWidget**:

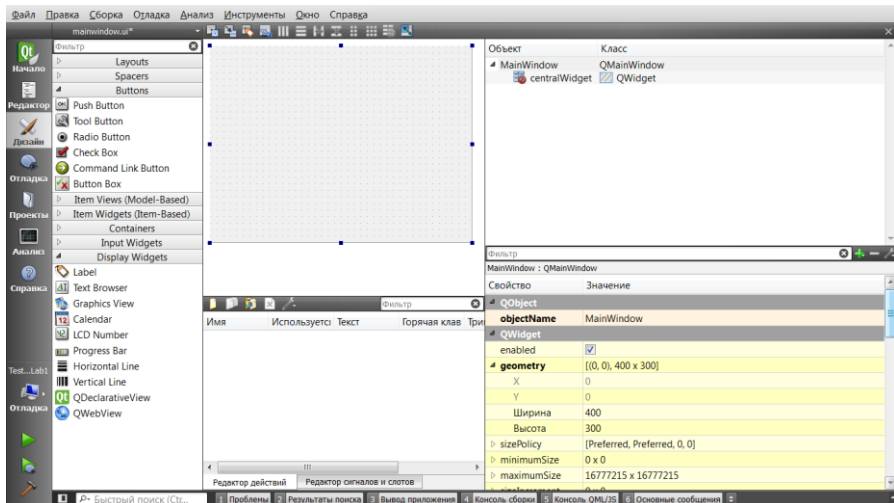


Рисунок 1.21 – Вигляд форми після змін



Далі додамо дві кнопки (**Push Button**): одна потрібна для виходу, інша – для запуску програми на малювання. Також додамо текст (**Label**), змінимо розмір, шрифт та отримаємо наступне:

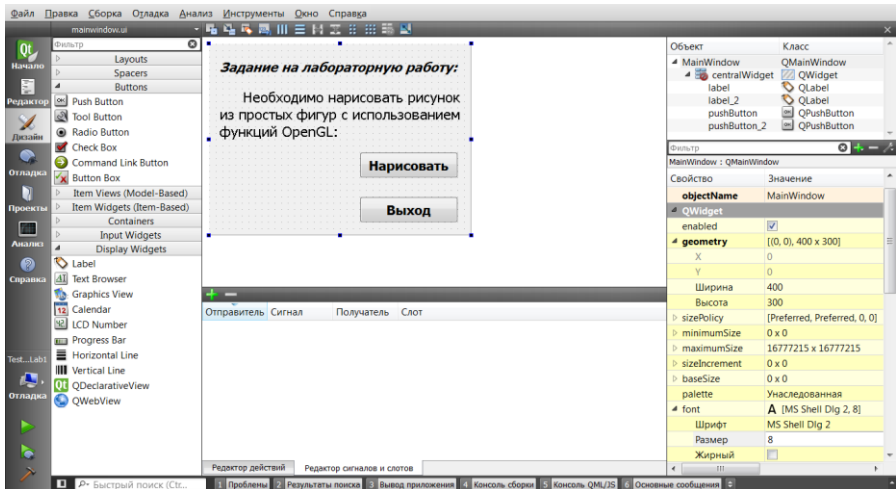


Рисунок 1.22 – Форма завдання

Далі для кнопки «Вихід» обираємо Змінення сигналів/слотів (F4) та проводимо стрілку:

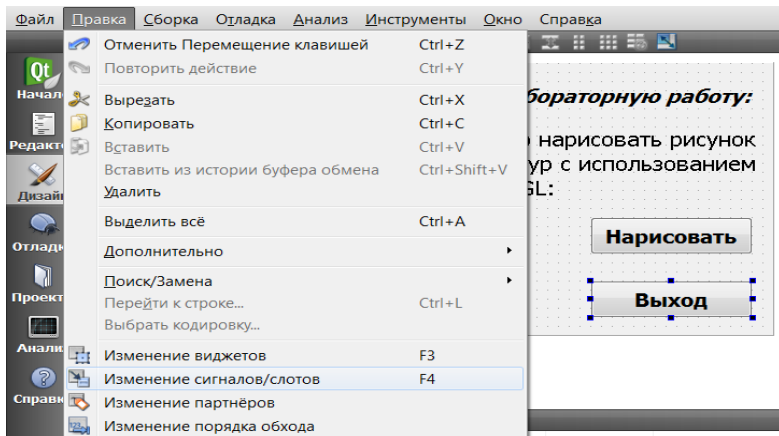


Рисунок 1.23 – Створення кнопки «Вихід»

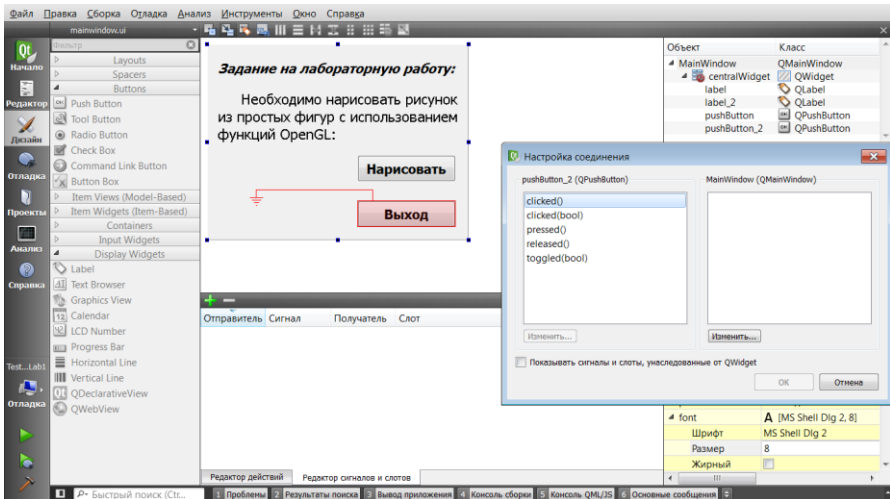


Рисунок 1.24 – Створення кнопки «Вихід»

Після цього відкриється вікно «Налаштування з'єднання», на якому необхідно вибрати *clicked()* та *close()*, ввімкнути Показувати сигнали та слоти, успадковані від QWidget та натиснути **OK**.

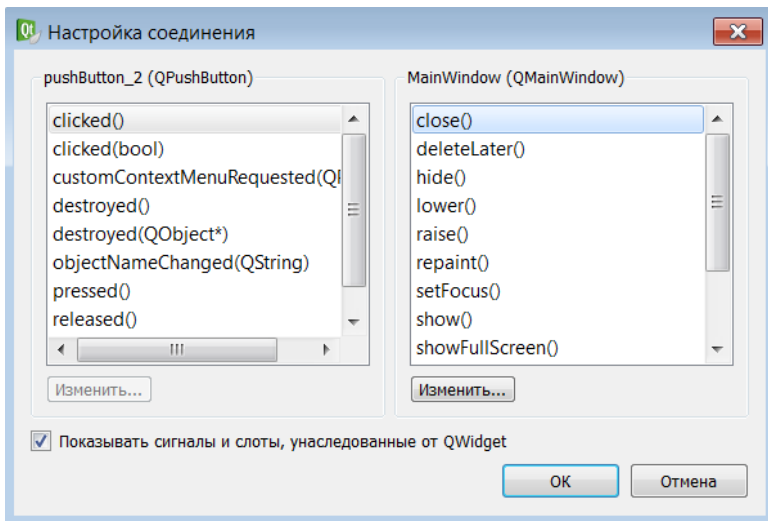


Рисунок 1.25 – Вікно «Налаштування з'єднання»

Отримаємо наступне:

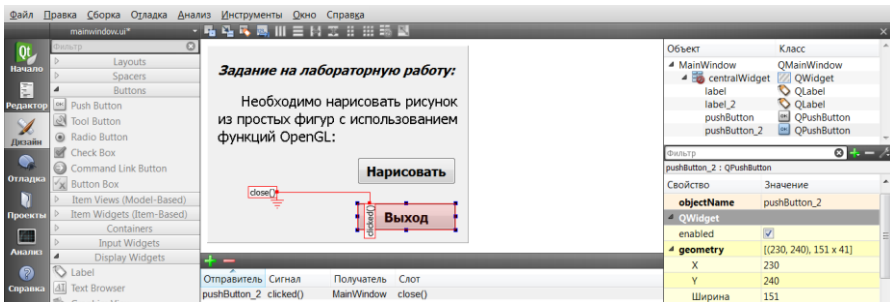


Рисунок 1.26–Результати налаштування

Потім натискаємо Змінення віджетів (F3).

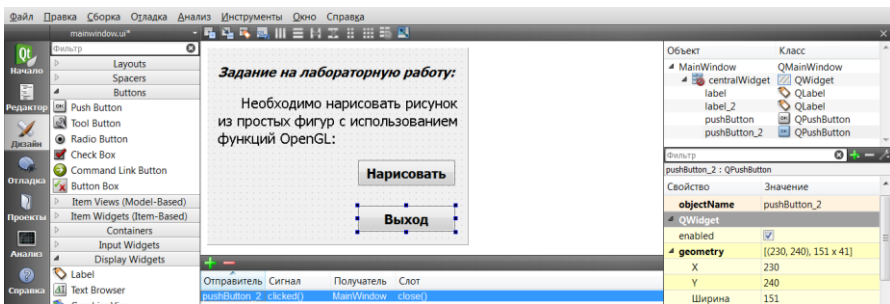


Рисунок 1.27 – Результаты налаштування

Запускаємо програму Ctrl+R та отримуємо:

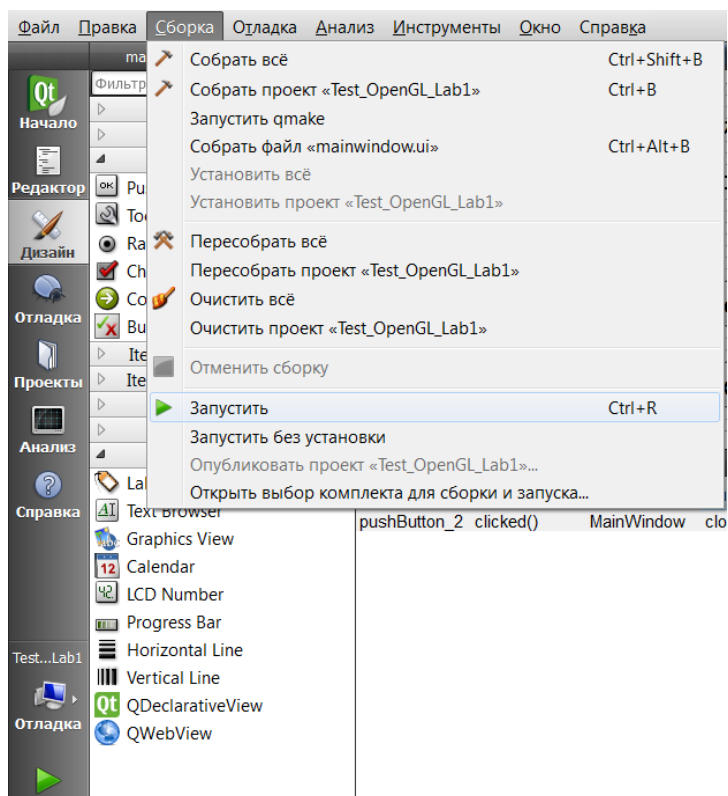


Рисунок 1.28– Запуск программы

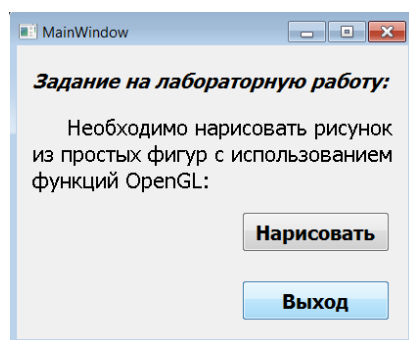


Рисунок 1.29– Работа программы

Натискаємо «Вихід» та для активування кнопки «Намалювати» робимо наступне:

Заходимо у *mainwindow.h*:

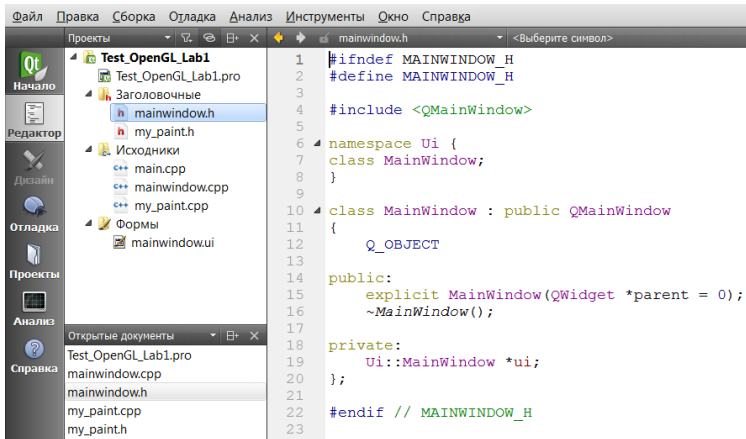


Рисунок 1.30 –Заголовочний файл mainwindow.h

Додаємо:

private slots:  
void on\_pushButton\_clicked();

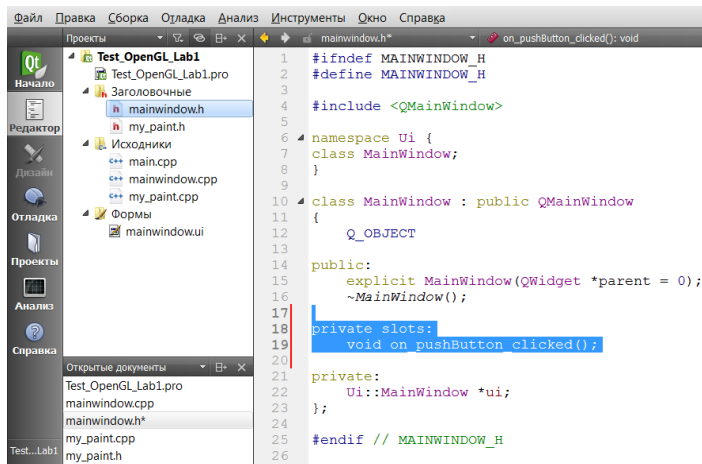


Рисунок 1.31 – Заміна рядків в файлі

Заходимо в *mainwindow.cpp*:

Додаємо:

```
#include "my_paint.h"
```

Та прописуємо функцію у вільному місці коду:

```
void MainWindow::on_pushButton_clicked()

{
    My_Paint *opengl_window = new My_Paint;
    opengl_window->show();
}
```

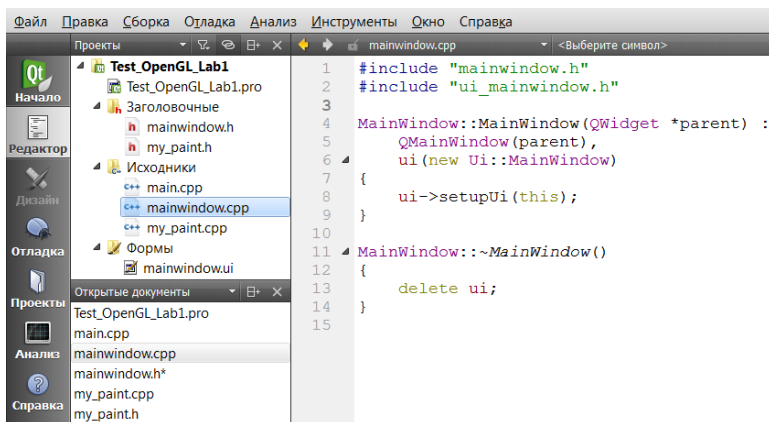


Рисунок 1.32 – Вигляд файлу mainwindow.cpp

Отримуємо наступне:

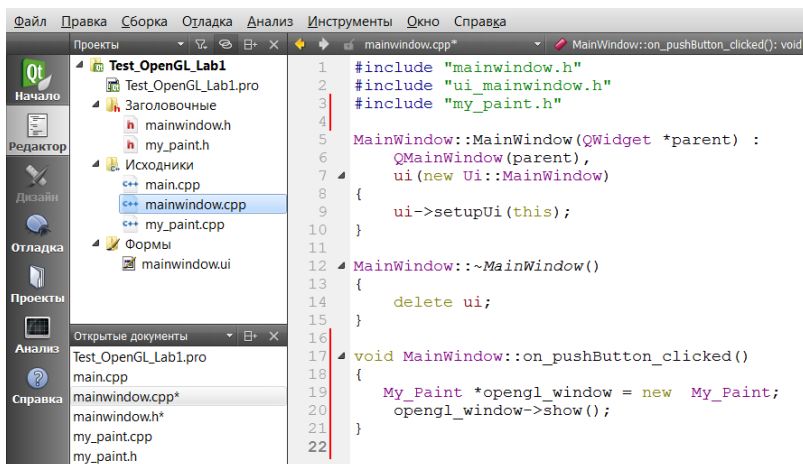


Рисунок 1.33 – Вигляд файлу mainwindow.cpp після внесених змін

Далі додаємо функцію в *my\_paint.h*, в клас *My\_Paint*:

```
void scene();
```

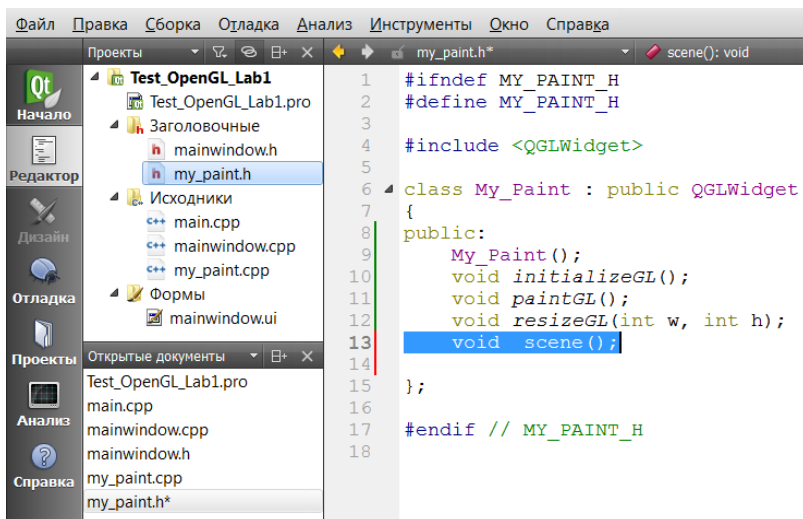


Рисунок 1.34 – Додавання функції

Заносимо код в функції (усе робиться в файлі *my\_paint.cpp*):  
Переходимо в *my\_paint.cpp*:

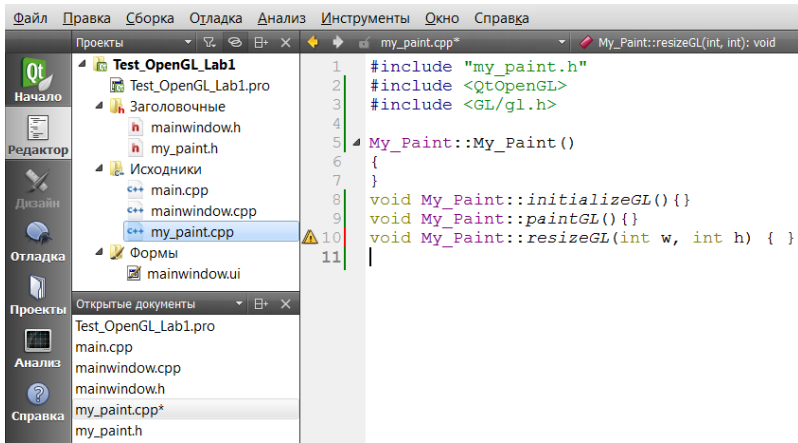


Рисунок 1.35 – Вигляд файлу *my\_paint.cpp*

Для цього в файлі *my\_paint.cpp* виділяємо та виконуємо зміну на код, що приведено нижче (усе робиться в файлі):

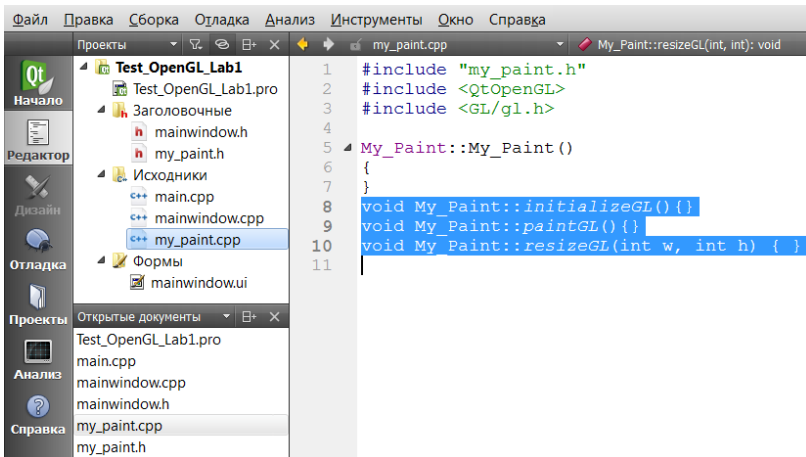


Рисунок 1.36 – Заміна коду в файлі *my\_paint.cpp*



```

void My_Paint::initializeGL()
{
    //Обрати фоновий (очищувачий) колір
    glClearColor(1.0,0.84,0.0,1.0);
    //glClearColor(r,g,b,t); Де, r-червоний колір, g-зелений, b- голубий, t-
    прозорість.

    //Задаємо режим обробки полігонів - передню та задню частини,
    //полігони повністю зафарбовані
    //(можна просто відображувати обрамлення)
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
}

void My_Paint::resizeGL(int nWidth, int nHeight)
{
    //Встановлюємо точку огляду. Останні два параметра однакові -
    // щоб не порушувати пропорції у широких екранів
    glViewport(0, 0, nWidth, nHeight);

    //Встановлюємо режим матриці
    glMatrixMode(GL_PROJECTION);

    //Завантажуємо матрицю
    glLoadIdentity();
}

void My_Paint::paintGL()
{
    struct MyThread : public QThread {using QThread::msleep;};

    //Очищуємо екран
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //Встановити проекцію:

    //Задаємо режим матриці
    glMatrixMode(GL_PROJECTION);
    //Завантажуємо матрицю
    glLoadIdentity();
    glOrtho(0.0,600.0,0.0,400.0,-1.0,1.0);

    //Тут малюємо - для зручності в окремій функції
    scene();

    //выводимо на екран
    swapBuffers();

    MyThread::msleep(100000);
}

void My_Paint::scene()
{

```

```

//Задаємо колір зображення
qglColor(Qt::red);
//Починаємо відрисовку, аргумент означає відрисовку прямокутника.
//Кожний виклик glVertex3f задає одну вершину прямокутника
glBegin(GL_POLYGON);
glVertex3f(50.0,350.0,0.0); //Координати квадрата
glVertex3f(250.0,350.0,0.0);
glVertex3f(250.0,150.0,0.0);
glVertex3f(50.0,150.0,0.0);
glEnd();

//Стрілка вгору
glColor3f(0.0,0.75,1.0); //Обираємо голубий колір
glBegin(GL_POLYGON);
glVertex3f(400.0,275.0,0.0); //Координати трикутника
glVertex3f(475.0,375.0,0.0);
glVertex3f(550.0,275.0,0.0);
glEnd();

glBegin(GL_POLYGON);
glVertex3f(435.0,275.0,0.0); //Координати квадрата
glVertex3f(515.0,275.0,0.0);

glVertex3f(515.0,175.0,0.0);
glVertex3f(435.0,175.0,0.0);
glEnd();

//Стрілка ліворуч
glColor3f(0.0,0.75,1.0); //Обираємо голубий колір
glBegin(GL_POLYGON);
glVertex3f(350.0,25.0,0.0); //Координати трикутника
glVertex3f(250.0,100.0,0.0);
glVertex3f(350.0,175.0,0.0);
glEnd();

glBegin(GL_POLYGON);
glVertex3f(350.0,140.0,0.0); //Координати квадрата
glVertex3f(525.0,140.0,0.0);
glVertex3f(525.0,70.0,0.0);
glVertex3f(350.0,70.0,0.0);
glEnd();
}

```

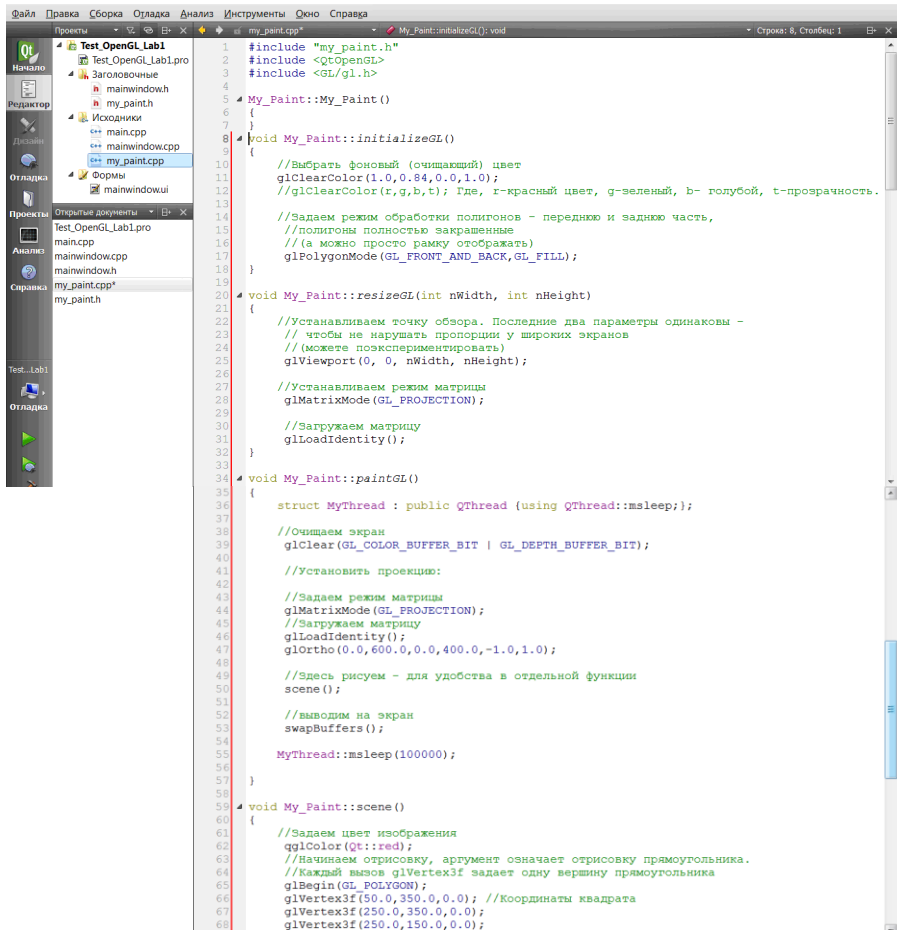


Рисунок 1.37 – Результат

Тепер запускаємо програму:

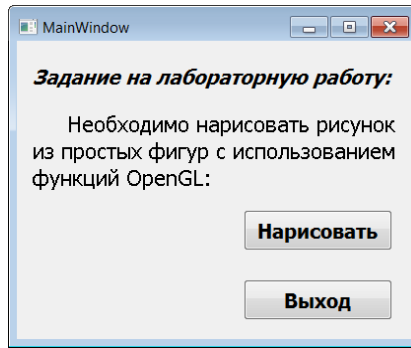


Рисунок 1.38 – Вигляд головного вікна програми

Натискаємо «Намалювати» та отримуємо ще одне віконце (*Test\_OpenGL\_Lab1*):

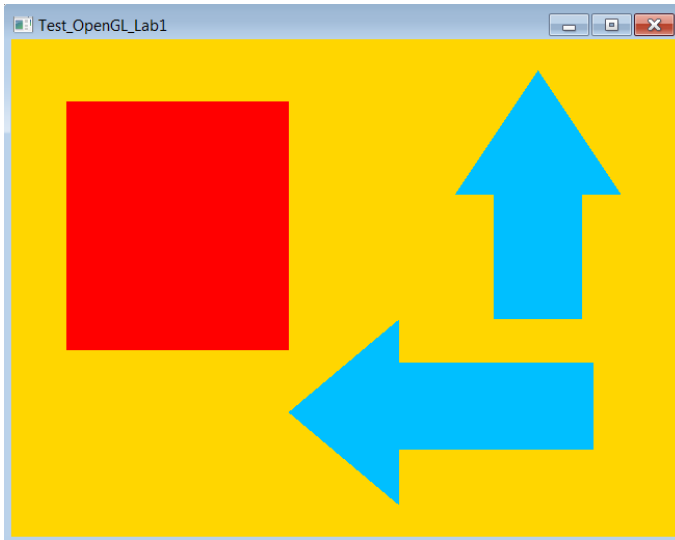


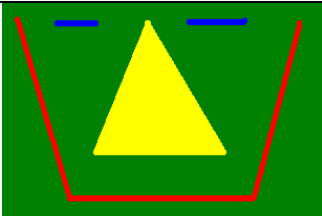
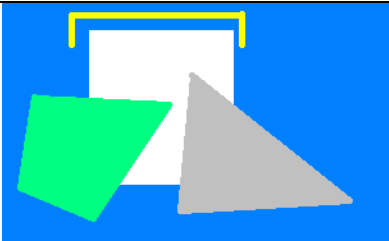
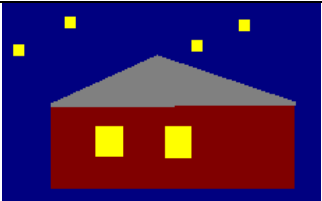
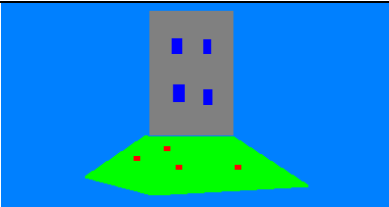

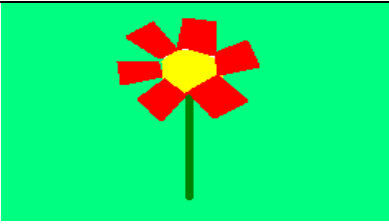
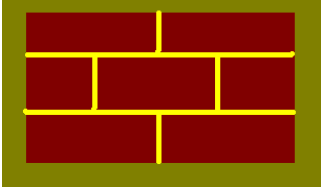

Рисунок 1.39 – Результат роботи програми

Закриваємо вікно *Test\_OpenGL\_Lab1*, після чого натискаємо «Вихід».


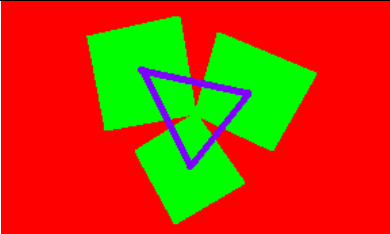
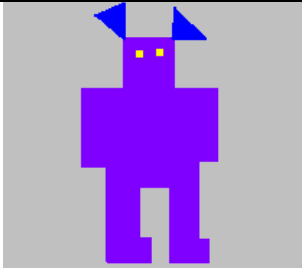
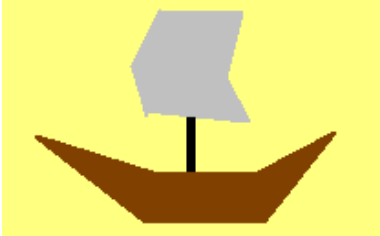
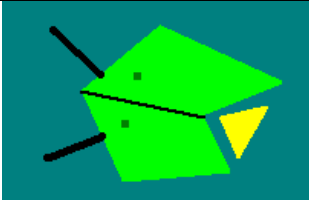

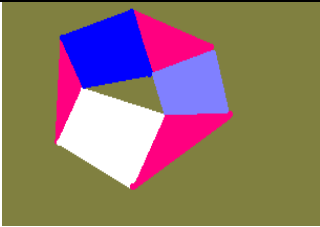

### 1.4 Завдання на виконання лабораторної роботи

Використовуючи примітиви «точка», «лінія», «трикутник», «чотирикутник» та «многокутник» зобразити фігуру, вказану у варіанті.

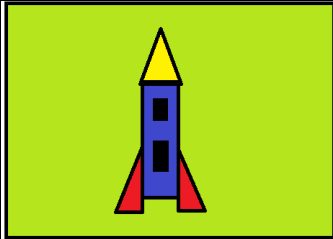
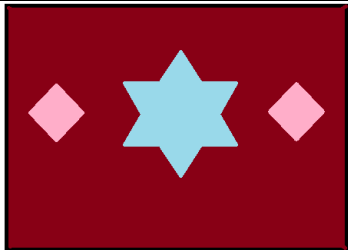
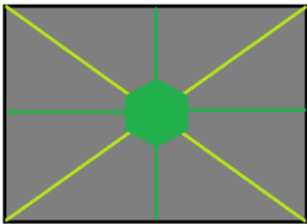
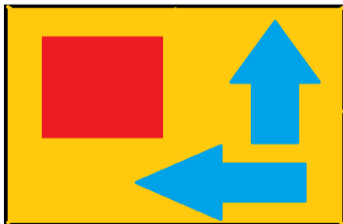
Таблиця 1.1 – Таблиця варіантів

1.		9.	
2.		10.	
3.		11.	
4.		12.	

Продовження таблиці 1.1

5.		13.	
6.		14.	
7.		15.	
8.		16.	

Продовження таблиці 1.1

17.		18.	
19.		20.	

### 1.5 Зміст звіту

У звіті мають бути відображені такі питання: мета роботи; завдання до лабораторної роботи; файли програми; результати роботи програми; висновок.

### 1.6 Контрольні питання

1.6.1 Опишіть склад бібліотек OpenGL.

1.6.2 Яким чином OpenGL інтегрується в обране середовище розробки?

1.6.3 Як програмно описується відрисовка примітиву?

1.6.4 Опишіть функцію вказання координат вершини фігури. Які параметри вона приймає?

1.6.5 Опишіть функцію встановлення кольору. Які параметри вона приймає?

## 2. ЛАБОРАТОРНА РОБОТА № 2

### «Малювання тривимірних об'єктів. Обертання»

#### 2.1 Мета роботи

Навчитися працювати з тривимірними фігурами та їх положенням.

#### 2.2 Стислі теоретичні відомості

##### 2.2.1 Створення тривимірних об'єктів

Розглянемо невеликий приклад, попередньо ознайомившись з необхідними командами.

**glColor3f** вимагає три речових (float) чисел, а **glColor3i** - вимагає три цілих (int) числа. Аналогічний синтаксис мають і інші команди OpenGL. При запису функцій в речовій формі аргументи лежать в інтервалі [0,1], а в цілочисельній формі - лінійно відображаються на цей інтервал, тобто для завдання білого кольору цілочисельна форма буде виглядати:

**glColor3i(2147483647, 2147483647, 2147483647)** – колір примітивів.

**glBegin(GL\_POLYGON)** – визначає точку входу в процес малювання графічного примітиву.

**glVertex3f(0.25, 0.25, 0.0)** – має формат по аналогії з **glColor**, однак у якості аргументів приймає координати точок трьох осей координат. У випадку з малюванням двовимірних фігур координата по осі простору залишається нульовою.

**glEnd ()** – визначає закінчення малювання примітиву, заданого в **glBegin**.

**glMatrixMode(GL\_PROJECTION)** – встановлює проекцію відображення. За замовчуванням – ортогональна.

**glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB)** – встановлює властивості вікна відрисовки і колірну модель.

**glutInitWindowSize (600600)** – встановлює розмір вікна в пікселях.



**glutInitWindowPosition (100100)** – встановлює позицію вікна на екрані.

**glutCreateWindow ( " Polygon ")** – створює вікно з заданим ім'ям.

**glLoadIdentity ();** – скидання буферу перегляду.

**gluPerspective( 45.0f, (GLfloat)width/(GLfloat)height, 0.1f, 100.0f );** – вибирає співвідношення розмірів вікна прорисовки

**glRotatef (RC1, 1.0f, 1.0f, 0.0f);** – визначає поворот фігури за заданим в змінній кутом.

### 2.2.2 Функції роботи з об'єктом

Керування камерою за допомогою миші:

**# mainscene.h**

```
...
class MainScene : public QGLWidget
{
    Q_OBJECT
private:
    QPoint pressPosition;
    QPoint releasePosition;
    GLfloat xAxisRotation;
    GLfloat yAxisRotation;
    GLfloat zAxisRotation;
    GLfloat currentWidth;
    GLfloat currentHeight;
...

```

**#mainscene.cpp**

```
void MainScene::mousePressEvent(QMouseEvent *event)
{
    pressPosition = event->pos();
}

void MainScene::mouseMoveEvent(QMouseEvent *event)
{
    xAxisRotation += (180 * ((GLfloat)event->y() - (GLfloat)pressPosition.y())) / (currentHeight);
    yAxisRotation += (180 * ((GLfloat)event->x() - (GLfloat)pressPosition.x())) / (currentWidth);
    pressPosition = event->pos();
    updateGL();
}

```

Самостійне обертання об'єкта:

```
...
glRotatef(rc, 0.0, 1.0, 0.0f); // функція обертання
...
rc -= 0.15f; // кут і напрямок повороту
...
```

## 2.3 Методика виконання лабораторної роботи

### 2.3.1 Створення літер

Намалюємо дві обертові літери (можна використовувати будь-які методи).

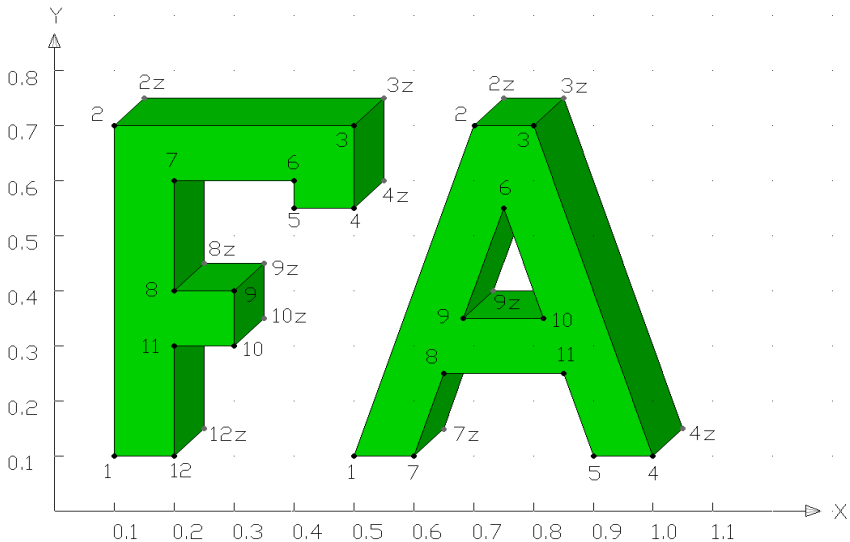


Рисунок 2.1 – Схематичне зображення літер

Літера “F”

(0.1, 0.1, 0.2) – 1  
 (0.1, 0.7, 0.2) – 2  
 (0.5, 0.7, 0.2) – 3  
 (0.5, 0.55, 0.2) – 4  
 (0.4, 0.55, 0.2) – 5  
 (0.4, 0.6, 0.2) – 6  
 (0.2, 0.6, 0.2) – 7

(0.2, 0.4, 0.2) – 8  
 (0.3, 0.4, 0.2) – 9  
 (0.3, 0.3, 0.2) – 10  
 (0.2, 0.3, 0.2) – 11  
 (0.2, 0.1, 0.2) – 12  
 Літера “A”  
 (0.5, 0.1, 0.2) – 1  
 (0.7, 0.7, 0.2) – 2

(0.8, 0.7, 0.2) – 3	(0.65, 0.25, 0.2) – 8
(1.0, 0.1, 0.2) – 4	(0.675, 0.35, 0.2) – 9
(0.9, 0.1, 0.2) – 5	(0.825, 0.35, 0.2) – 10
(0.75, 0.55, 0.2) – 6	(0.85, 0.25, 0.2) – 11
(0.5, 0.1, 0.2) – 7	

Z – це координати задньої стінки літери

Перейдемо безпосередньо до коду програми

```
#include "figuregl.h"
#include <QtOpenGL>
#include <GL/gl.h>
GLfloat rc; // Кут повороту літер

figureGL::figureGL()
{
    r = 0; g = 0; b = 0;
}

void figureGL::initializeGL()
{
    //Задаємо колір фону в OpenGL вікні
    glClearColor(Qt::black);
}

void figureGL::initializeGL(QColor rgb)
{
    glClearColor(rgb); // Вибираємо колірну палітру типу RGB
}

void figureGL::resizeGL(int nWidth, int nHeight)
{
    //Встановлюємо точку огляду.
    glViewport(0, 0, nWidth, nHeight);
    //Встановлюємо режим матриці.
    glMatrixMode(GL_PROJECTION);

    //Завантажуємо матрицю
    glLoadIdentity();

    glClearDepth(1.0f); // Дозволяємо очищення буфера
    глибини
    glEnable (GL_DEPTH_TEST); //Дозволяємо тест глибини
    glDepthFunc (GL_LEQUAL); // Тип тесту глибини
    glHint (GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
    // Поліпшення в обчисленні перспективи
    glEnable (GL_BLEND); // Дозволяємо змішування
```

```

glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
// Вказуємо спосіб змішування

}

void figureGL::paintGL()
{
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Очистити екран і буфер глибини

    // Тут малюємо (для зручності, в окремій функції)
    scene();
    //виводимо на екран
    swapBuffers();
}
void figureGL::scene()
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT ); //
    Очистити екран і буфер глибини

    glLoadIdentity();
    glTranslatef(0.0, 0.0,0.0);
    glRotatef(rc, 0.0, 1.0, 0.0f); // функція обертання
    glBegin(GL_QUADS); // створюємо букву квадратними
    полігонами

    //F - передня частина
    glColor(Qt::green); // заливаємо зеленим кольором

    glVertex3f(0.1, 0.1, 0.2);//1
    glVertex3f(0.1, 0.7, 0.2);//2
    glVertex3f(0.2, 0.6, 0.2);//7
    glVertex3f(0.2, 0.1, 0.2);//1

    glVertex3f(0.1, 0.7, 0.2);//2
    glVertex3f(0.5, 0.7, 0.2);//3
    glVertex3f(0.4, 0.6, 0.2);//6
    glVertex3f(0.2, 0.6, 0.2);//7

    glVertex3f(0.5, 0.7, 0.2);//3
    glVertex3f(0.5, 0.55, 0.2);//4
    glVertex3f(0.4, 0.55, 0.2);//5
    glVertex3f(0.4, 0.6, 0.2);//6

    glVertex3f(0.2, 0.4, 0.2);//8
    glVertex3f(0.3, 0.4, 0.2);//9

```

```

    glVertex3f(0.3, 0.3, 0.2); //10
    glVertex3f(0.2, 0.3, 0.2); //11
//F-задня частина
    glVertex3f(0.1, 0.1, 0.3); //1z
    glVertex3f(0.1, 0.7, 0.3); //2z
    glVertex3f(0.2, 0.6, 0.3); //7z
    glVertex3f(0.2, 0.1, 0.3); //12z

```

```

    glVertex3f(0.1, 0.7, 0.3); //2z
    glVertex3f(0.5, 0.7, 0.3); //3z
    glVertex3f(0.4, 0.6, 0.3); //6z
    glVertex3f(0.2, 0.6, 0.3); //7z

```

```

    glVertex3f(0.5, 0.7, 0.3); //3z
    glVertex3f(0.5, 0.55, 0.3); //4z
    glVertex3f(0.4, 0.55, 0.3); //5z
    glVertex3f(0.4, 0.6, 0.3); //6z

```

```

    glVertex3f(0.2, 0.4, 0.3); //8z
    glVertex3f(0.3, 0.4, 0.3); //9z
    glVertex3f(0.3, 0.3, 0.3); //10z
    glVertex3f(0.2, 0.3, 0.3); //11z

```

```

//F- бокові частини

```

```

    qglColor(Qt::green);
    glVertex3f(0.1, 0.1, 0.2); //1
    glVertex3f(0.1, 0.7, 0.2); //2
    glVertex3f(0.1, 0.7, 0.3); //2z
    glVertex3f(0.1, 0.1, 0.3); //1z

```

```

    qglColor(Qt::green);
    glVertex3f(0.1, 0.7, 0.2); //2
    glVertex3f(0.5, 0.7, 0.2); //3
    glVertex3f(0.5, 0.7, 0.3); //3z
    glVertex3f(0.1, 0.7, 0.3); //2z
    qglColor(Qt::green);
    glVertex3f(0.5, 0.7, 0.2); //3
    glVertex3f(0.5, 0.55, 0.2); //4
    glVertex3f(0.5, 0.55, 0.3); //4z
    glVertex3f(0.5, 0.7, 0.3); //3z

```

```

    qglColor(Qt::green);
    glVertex3f(0.5, 0.55, 0.2); //4
    glVertex3f(0.4, 0.55, 0.2); //5
    glVertex3f(0.4, 0.55, 0.3); //5z
    glVertex3f(0.5, 0.55, 0.3); //4z

```

```

    qglColor(Qt::green);
    glVertex3f(0.4, 0.55, 0.2); //5
    glVertex3f(0.4, 0.6, 0.2); //6
    glVertex3f(0.4, 0.6, 0.3); //6z
    glVertex3f(0.4, 0.55, 0.3); //5z

    qglColor(Qt::green);
    glVertex3f(0.4, 0.6, 0.2); //6
    glVertex3f(0.2, 0.6, 0.2); //7
    glVertex3f(0.2, 0.6, 0.3); //7z
    glVertex3f(0.4, 0.6, 0.2); //6z

    qglColor(Qt::green);
    glVertex3f(0.2, 0.6, 0.2); //7
    glVertex3f(0.2, 0.4, 0.2); //8
    glVertex3f(0.2, 0.4, 0.3); //8z
    glVertex3f(0.2, 0.6, 0.3); //7z

    qglColor(Qt::green);
    glVertex3f(0.2, 0.4, 0.2); //8
    glVertex3f(0.3, 0.4, 0.2); //9
    glVertex3f(0.3, 0.4, 0.3); //9z
    glVertex3f(0.2, 0.4, 0.3); //8z

    qglColor(Qt::green);
    glVertex3f(0.3, 0.4, 0.2); //9
    glVertex3f(0.3, 0.3, 0.2); //10
    glVertex3f(0.3, 0.3, 0.3); //10z
    glVertex3f(0.3, 0.4, 0.3); //9z

    qglColor(Qt::green);
    glVertex3f(0.3, 0.3, 0.2); //10
    glVertex3f(0.2, 0.3, 0.2); //11
    glVertex3f(0.2, 0.3, 0.3); //11z
    glVertex3f(0.3, 0.3, 0.3); //10z
    qglColor(Qt::green);
    glVertex3f(0.2, 0.3, 0.2); //11
    glVertex3f(0.2, 0.1, 0.2); //12
    glVertex3f(0.2, 0.1, 0.3); //12z
    glVertex3f(0.2, 0.3, 0.3); //11z
    glEnd();

    glLineWidth(2.0); //координати меж літери F
    glColor3f(1,1,1);

    glBegin(GL_LINE_STRIP);

```

```

glVertex3f(0.1, 0.1, 0.2); //1
glVertex3f(0.1, 0.7, 0.2); //2
glVertex3f(0.5, 0.7, 0.2); //3
glVertex3f(0.5, 0.55, 0.2); //4
glVertex3f(0.4, 0.55, 0.2); //5
glVertex3f(0.4, 0.6, 0.2); //6
glVertex3f(0.2, 0.6, 0.2); //7
glVertex3f(0.2, 0.4, 0.2); //8
glVertex3f(0.3, 0.4, 0.2); //9
glVertex3f(0.3, 0.3, 0.2); //10
glVertex3f(0.2, 0.3, 0.2); //11
glVertex3f(0.2, 0.1, 0.2); //12
glVertex3f(0.1, 0.1, 0.2); //1
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.1, 0.1, 0.3); //1z
glVertex3f(0.1, 0.7, 0.3); //2z
glVertex3f(0.5, 0.7, 0.3); //3z
glVertex3f(0.5, 0.55, 0.3); //4z
glVertex3f(0.4, 0.55, 0.3); //5z
glVertex3f(0.4, 0.6, 0.3); //6z
glVertex3f(0.2, 0.6, 0.3); //7z
glVertex3f(0.2, 0.4, 0.3); //8z
glVertex3f(0.3, 0.4, 0.3); //9z
glVertex3f(0.3, 0.3, 0.3); //10z
glVertex3f(0.2, 0.3, 0.3); //11z
glVertex3f(0.2, 0.1, 0.3); //12z
glVertex3f(0.1, 0.1, 0.3); //1z
glEnd();

glBegin(GL_LINE_STRIP); // (GL_LINE_STRIP)-лінія з
2x точок
glVertex3f(0.1, 0.1, 0.2); //1
glVertex3f(0.1, 0.1, 0.3); //1z
glEnd();
glBegin(GL_LINE_STRIP);
glVertex3f(0.1, 0.7, 0.2); //2
glVertex3f(0.1, 0.7, 0.3); //2z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.5, 0.7, 0.2); //3
glVertex3f(0.5, 0.7, 0.3); //3z
glEnd();

glBegin(GL_LINE_STRIP);

```

```
glVertex3f(0.5, 0.55, 0.2); //4
glVertex3f(0.5, 0.55, 0.3); //4z
glEnd();
```

```
glBegin(GL_LINE_STRIP);
glVertex3f(0.4, 0.55, 0.2); //5
glVertex3f(0.4, 0.55, 0.3); //5z
glEnd();
```

```
glBegin(GL_LINE_STRIP);
glVertex3f(0.4, 0.6, 0.2); //6
glVertex3f(0.4, 0.6, 0.3); //6z
glEnd();
```

```
glBegin(GL_LINE_STRIP);
glVertex3f(0.2, 0.6, 0.2); //7
glVertex3f(0.2, 0.6, 0.3); //7z
glEnd();
```

```
glBegin(GL_LINE_STRIP);
glVertex3f(0.2, 0.4, 0.2); //8
glVertex3f(0.2, 0.4, 0.3); //8z
glEnd();
```

```
glBegin(GL_LINE_STRIP);
glVertex3f(0.3, 0.4, 0.2); //9
glVertex3f(0.3, 0.4, 0.3); //9z
glEnd();
```

```
glBegin(GL_LINE_STRIP);
glVertex3f(0.3, 0.3, 0.2); //10
glVertex3f(0.3, 0.3, 0.3); //10z
glEnd();
```

```
glBegin(GL_LINE_STRIP);
glVertex3f(0.2, 0.3, 0.2); //11
glVertex3f(0.2, 0.3, 0.3); //11z
glEnd();
```

```
glBegin(GL_LINE_STRIP);
glVertex3f(0.2, 0.1, 0.2); //12
glVertex3f(0.2, 0.1, 0.3); //12z
glEnd();
```

```
//A - передняя часть
glBegin(GL_QUADS);
```



```

qglColor(Qt::green);
glVertex3f(0.5, 0.1, 0.2); // 1
glVertex3f(0.7, 0.7, 0.2); //2
glVertex3f(0.8, 0.7, 0.2); //3
glVertex3f(0.6, 0.1, 0.2); //7

qglColor(Qt::green);
glVertex3f(0.75, 0.55, 0.2); //6
glVertex3f(0.8, 0.7, 0.2); //3
glVertex3f(1, 0.1, 0.2); //4
glVertex3f(0.9, 0.1, 0.2); //5

qglColor(Qt::green);
glVertex3f(0.65, 0.25, 0.2); //8
glVertex3f(0.675, 0.35, 0.2); //9
glVertex3f(0.825, 0.35, 0.2); //10
glVertex3f(0.85, 0.25, 0.2); //11

//А-задня частина
qglColor(Qt::green);
glVertex3f(0.5, 0.1, 0.3); // 1z
glVertex3f(0.7, 0.7, 0.3); //2z
glVertex3f(0.8, 0.7, 0.3); //3z
glVertex3f(0.6, 0.1, 0.3); //7z

qglColor(Qt::green);
glVertex3f(0.75, 0.55, 0.3); //6z
glVertex3f(0.8, 0.7, 0.3); //3z
glVertex3f(1, 0.1, 0.3); //4z
glVertex3f(0.9, 0.1, 0.3); //5z

qglColor(Qt::green);
glVertex3f(0.65, 0.25, 0.3); //8z
glVertex3f(0.675, 0.35, 0.3); //9z
glVertex3f(0.825, 0.35, 0.3); //10z
glVertex3f(0.85, 0.25, 0.3); //11z

//А-бокові частини
glVertex3f(0.5, 0.1, 0.2); // 1
glVertex3f(0.7, 0.7, 0.2); //2
glVertex3f(0.7, 0.7, 0.3); //2z
glVertex3f(0.5, 0.1, 0.3); // 1z

glVertex3f(0.8, 0.7, 0.2); //3
glVertex3f(1, 0.1, 0.2); //4

```

```

glVertex3f(1, 0.1, 0.3); //4z
glVertex3f(0.8, 0.7, 0.3); //3z

glVertex3f(1, 0.1, 0.2); //4
glVertex3f(0.9, 0.1, 0.2); //5
glVertex3f(0.9, 0.1, 0.3); //5z
glVertex3f(1, 0.1, 0.3); //4z

glVertex3f(0.9, 0.1, 0.2); //5
glVertex3f(0.85, 0.25, 0.2); //11
glVertex3f(0.85, 0.25, 0.3); //11z
glVertex3f(0.9, 0.1, 0.3); //5z

glVertex3f(0.85, 0.25, 0.2); //11
glVertex3f(0.65, 0.25, 0.2); //8
glVertex3f(0.65, 0.25, 0.3); //8z
glVertex3f(0.85, 0.25, 0.3); //11z

glVertex3f(0.5, 0.1, 0.2); //7
glVertex3f(0.65, 0.25, 0.2); //8
glVertex3f(0.65, 0.25, 0.3); //8z
glVertex3f(0.5, 0.1, 0.3); //7z

glVertex3f(0.5, 0.1, 0.2); // 1
glVertex3f(0.5, 0.1, 0.2); //7
glVertex3f(0.5, 0.1, 0.3); //7z
glVertex3f(0.5, 0.1, 0.3); // 1z

glVertex3f(0.675, 0.35, 0.2); //9
glVertex3f(0.75, 0.55, 0.2); //6
glVertex3f(0.75, 0.55, 0.3); //6z
glVertex3f(0.675, 0.35, 0.3); //9z

glVertex3f(0.75, 0.55, 0.2); //6
glVertex3f(0.825, 0.35, 0.2); //10
glVertex3f(0.825, 0.35, 0.3); //10z
glVertex3f(0.75, 0.55, 0.3); //6z

glVertex3f(0.675, 0.35, 0.2); //9
glVertex3f(0.825, 0.35, 0.2); //10
glVertex3f(0.825, 0.35, 0.3); //10z
glVertex3f(0.675, 0.35, 0.3); //9z
glEnd();

glLineWidth(2.0); //координати меж літери А
glColor3f(1,1,1);

```

```

glBegin(GL_LINE_STRIP);
glVertex3f(0.5, 0.1, 0.2); // 1
glVertex3f(0.7, 0.7, 0.2); //2
glVertex3f(0.8, 0.7, 0.2); //3
glVertex3f(1, 0.1, 0.2); //4
glVertex3f(0.9, 0.1, 0.2); //5
glVertex3f(0.85, 0.25, 0.2); //11
glVertex3f(0.65, 0.25, 0.2); //8
glVertex3f(0.6, 0.1, 0.2); //7
glVertex3f(0.5, 0.1, 0.2); // 1
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.675, 0.35, 0.2); //9
glVertex3f(0.825, 0.35, 0.2); //10
glVertex3f(0.75, 0.55, 0.2); //6
glVertex3f(0.675, 0.35, 0.2); //9
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.5, 0.1, 0.3); // 1z
glVertex3f(0.7, 0.7, 0.3); //2z
glVertex3f(0.8, 0.7, 0.3); //3z
glVertex3f(1, 0.1, 0.3); //4z
glVertex3f(0.9, 0.1, 0.3); //5z
glVertex3f(0.85, 0.25, 0.3); //11z
glVertex3f(0.65, 0.25, 0.3); //8z
glVertex3f(0.6, 0.1, 0.3); //7z
glVertex3f(0.5, 0.1, 0.3); // 1z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.675, 0.35, 0.3); //9z
glVertex3f(0.825, 0.35, 0.3); //10z
glVertex3f(0.75, 0.55, 0.3); //6z
glVertex3f(0.675, 0.35, 0.3); //9z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.5, 0.1, 0.2); // 1
glVertex3f(0.5, 0.1, 0.3); // 1z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.7, 0.7, 0.2); //2
glVertex3f(0.7, 0.7, 0.3); //2z

```

```

glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.8, 0.7, 0.2); //3
glVertex3f(0.8, 0.7, 0.3); //3z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(1, 0.1, 0.2); //4
glVertex3f(1, 0.1, 0.3); //4z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.9, 0.1, 0.2); //5
glVertex3f(0.9, 0.1, 0.3); //5z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.75, 0.55, 0.2); //6
glVertex3f(0.75, 0.55, 0.3); //6z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.5, 0.1, 0.2); //7
glVertex3f(0.5, 0.1, 0.3); //7z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.65, 0.25, 0.2); //8
glVertex3f(0.65, 0.25, 0.3); //8z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.675, 0.35, 0.2); //9
glVertex3f(0.675, 0.35, 0.3); //9z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.825, 0.35, 0.2); //10
glVertex3f(0.825, 0.35, 0.3); //10z
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(0.85, 0.25, 0.2); //11
glVertex3f(0.85, 0.25, 0.3); //11z

```

```

        glEnd();

        rc -= 0.15f; // кут і напрямок повороту
    }
    void figureGL::timer()
    {
        for (int i=0; i<1; i--)
        {
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            // Очистити екран і буфер глибини
            paintGL();
        }
    }
    void figureGL::mouseMoveEvent(QMouseEvent*me)
    { timer(); }
    void figureGL::keyPressEvent(QKeyEvent *event)
    {
        timer();
    }

```

Результати роботи представлені на рис. 2.1

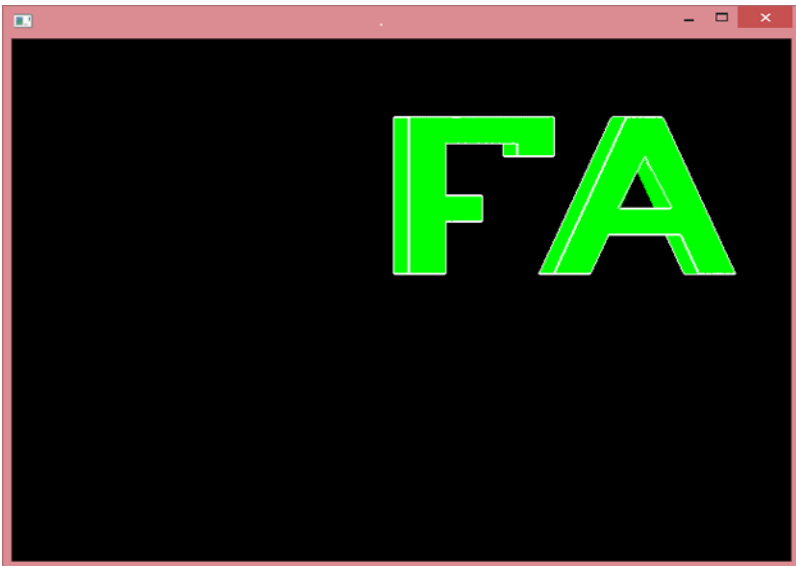


Рисунок 2.1 – Робота програми

## **2.4 Завдання на виконання лабораторної роботи**

Виконати побудову перших букв імені і прізвища (англійською) у тривимірному вигляді, використовуючи функції повороту примітиву.

## **2.5 Зміст звіту**

У звіті мають бути відображені такі питання: мета роботи; завдання до лабораторної роботи; файли програми; результати роботи програми; висновок.

## **2.6 Контрольні питання**

- 2.6.1 Що виконує функція `glClear`?
- 2.6.2 Як вибирати колірну палітру типу RGB?
- 2.6.3 Якими полігонами можна створювати 3d-об'єкт?
- 2.6.4 Що виконує функція `glRotatef`?
- 2.6.5 Навіщо потрібен буфер глибини?

### 3. ЛАБОРАТОРНА РОБОТА № 3

#### «Накладення текстури та освітлення»

#### 3.1 Мета роботи

Навчитися накладати текстури та освітлення на об'єкти

#### 3.2 Стислі теоретичні відомості

##### 3.2.1 Текстури

Розглянемо невеликий приклад, попередньо ознайомившись з необхідними командами. **glColor3f** вимагає три дійсних (float) числа, а **glColor3i** – три цілих (int) числа. Аналогічний синтаксис мають і інші команди OpenGL. Записи функцій в дійсній формі аргументи лежать в інтервалі [0,1], а в цілочисловій формі - лінійно відображаються на цей інтервал, тобто для завдання білого кольору цілочислова форма матиме вигляд:

**glColor3i (2147483647, 2147483647, 2147483647);** // колір примітивів

**glBegin(GL\_POLYGON)** – визначає точку входу в процес малювання графічного примітиву.

**glVertex2f(0.25,0.25,0.0)** – має формат аналогічний glColor, проте приймає в якості аргументів – координати точок по двом осям координат.

**glEnd()** – визначає закінчення малювання примітиву, заданого в glBegin.

**glMatrixMode(GL\_PROJECTION)** – встановлює проекцію відображення. За замовчуванням – ортогональна.

**glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB)** – встановлює властивості вікна відтворення і колірну модель.

**glutInitWindowSize(600,600)** – встановлює розмір вікна в пікселях.

**glutInitWindowPosition(100,100)** – встановлює позицію вікна на екрані.

**glutCreateWindow("Polygon")** - створює вікно з заданим ім'ям.

**glTexCoord2f(0.0,0.0)** - показує координати накладення завантаженої текстури (про це – трохи нижче).

А тепер – найголовніше: команди завантаження текстур.

**AUX\_RGBImageRec \*texture1** задає покажчик на структуру для зберігання першої картинки, яку ми завантажимо і використаємо як текстуру. Структура містить червону, зелену і синю компоненти кольору, які використовуються при створенні зображення. Зазвичай так розміщується в пам'яті завантажена картинка. Структура **AUX\_RGBImageRec** визначена в бібліотеці **glAux**, і робить можливим завантаження картинки в пам'ять. У наступному рядку відбувається безпосереднє завантаження.

**auxDIBImageLoadA("ім'я\_файла");** - завантажує текстуру за вказаним шляхом.

У другому рядку виклик **glBindTexture(GL\_TEXTURE\_2D, texture[0])** говорить OpenGL, що **texture[0]** (перша текстура) буде 2D текстурою. 2D текстури мають і висоту (по осі Y) і ширину (по осі X). Основна задача **glGenTexture** вказати OpenGL на доступну пам'ять. В цьому випадку ми говоримо OpenGL, що пам'ять доступна в **&texture[0]**. Потім ми створюємо текстуру, і вона буде збережена в цій пам'яті. Далі, якщо ми прив'язуємося до пам'яті, в якій вже знаходиться текстура, ми говоримо OpenGL захопити дані текстури з цієї області пам'яті. Зазвичай це покажчик на вільну пам'ять, або пам'ять, в якій міститься текстура.

**glGenTextures(1, &texture[0]);**

**glBindTexture(GL\_TEXTURE\_2D, texture[0]);**

У наступних двох рядках ми повідомимо OpenGL який тип фільтрації треба використовувати, коли зображення більше на екрані, ніж оригінальна текстура (**GL\_TEXTURE\_MAG\_FILTER**), або коли воно менше на екрані, ніж текстура (**GL\_TEXTURE\_MIN\_FILTER**). При цьому текстура виглядає згладженою на відстані і зблизька. Використання **GL\_LINEAR** вимагає багато роботи для процесора/відеокарти, тому якщо ваша система повільна, ви можете захотіти використовувати **GL\_NEAREST**. Текстура, яка фільтрується з **GL\_NEAREST** складається з добре видимих кольорових прямокутників, коли вона наближена.

**lTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);**



**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);**

На завершення створюємо фактичну текстуру. В наступному рядку заявляємо OpenGL, що текстура буде двовимірною (**GL\_TEXTURE\_2D**). Нуль задає рівень деталізації, це зазвичай нуль. Три - число компонент колірних даних, так як зображення зроблено з трьох колірних компонент (червоний, зелений, синій). **texture1->sizeX** - це ширина текстури, автоматично. Якщо ви знаєте ширину, ви можете вказати тут, але простіше дати комп'ютеру зробити це за вас. **texture1->sizeY** - висота текстури. Нуль - це бордюр. Він зазвичай залишається нулем. **GL\_RGB** повідомляє OpenGL, дані зображення представлені у порядку слідування червоних, зелених і блакитних компонент кольору. **GL\_UNSIGNED\_BYTE** означає, що дані з яких складається зображення мають розмір байта і всі числа без знака, і в кінці **texture1->data** повідомляє OpenGL, де брати самі дані. В цьому випадку покажчик на дані в записі **texture1**.

**glTexImage2D(GL\_TEXTURE\_2D, 0, 3, texture1->sizeX, texture1->sizeY, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, texture1->data);**

### 3.2.2 Освітлення

Освітлення будь-якого простору - це процес, завдяки якому цей простір наповнюється світлом і присутні в ньому предмети робляться видимими.

Освітлення будь-якого об'єкту залежить від двох чинників:

- матеріал, з якого зроблений об'єкт;
- світло, яким він освітлений.

Залежно від реалізації OpenGL, на сцені можуть бути присутніми вісім і більше джерел світла. За замовчуванням освітлення вимкнене. Ввімкнути нульове джерело світла можна командою:

**glEnable (GL\_LIGHT0);**

Решта вмикається аналогічним способом, де замість **GL\_LIGHT0** вказується **GL\_LIGHTi**. Після того, як джерело ввімкнене, необхідно задати його параметри. Якщо монотонне тіло у вас рівномірно освітлене, ви не можете побачити його рельєфу. Тому нам потрібно використовувати джерела світла.

У OpenGL існує три типи джерел світла:

- точкове джерело світла: розташоване в конкретній точці простору і світить рівномірно у всіх напрямках. Для нього можна задати ефект згасання світла з відстанню;

- джерело спрямованого світла: розташоване на нескінченності і має виділений напрям освітлення;

- прожектор: є окремим випадком точкового джерела, але світло від нього поширюється тільки всередині обмежуючого конусу, а не в усіх напрямках.

Для управління властивостями джерела світла використовуються команди `glLight *`:

**`glLightf (GLenum light, GLenum pname, GLfloat param);`**

**`glLightfv (GLenum light, GLenum pname, const GLfloat *param);`**

Параметр **`light`** вказує OpenGL для якого джерела світла задаються параметри. Команда **`glLightf`** використовується для задання скалярних параметрів, а **`glLightfv`** використовується для задання векторних характеристик джерел світла.

Спочатку розглянемо функцію, яка встановлює базові налаштування. Коли ви дозволили освітлення, ви можете вже встановлювати фонову освітленість. За замовчуванням, значення фонові освітленості дорівнює **`(0.2, 0.2, 0.2, 1)`**.

### *3.2.2.1 Джерела спрямованого світла*

Джерела світла такого типу знаходиться на нескінченності і світло від них поширюється в заданому напрямку. Ідеально підходить для створення рівномірного освітлення. Хорошим прикладом джерела спрямованого світла може служити Сонце. Для джерела спрямованого світла, крім компонент випромінювання, можна задати тільки напрямки.

**`GL_POSITION (0.0, 0.0, 1.0, 0.0)`** //  $(x, y, z, w)$  напрямки джерела спрямованого світла

Перші три компоненти **`(x, y, z)`** задають вектор напрямку, а компонента `w` завжди дорівнює нулю (інакше джерело перетвориться в точковий).

### 3.2.2.2 Матеріал

Матеріал може розсіювати, відображати і випромінювати світло. Властивості матеріалу встановлюються за допомогою функції

**glMaterialfv (GLenum face, GLenum pname, GLtype \* params)**

Перший параметр визначає межу, для якої встановлюються властивості. Він може приймати одне з наступних значень:

- **GL\_BACK** задня грань
- **GL\_FRONT** передня грань
- **GL\_FRONT\_AND\_BACK** обидві грані
- Другий параметр функції **glMaterialfv** визначає властивість матеріалу, яку буде встановлено, і може набувати таких значень.

- **GL\_AMBIENT** розсіяне світло
- **GL\_DIFFUSE** теж розсіяне світло
- **GL\_SPECULAR** відбите світло
- **GL\_EMISSION** випромінюється світло
- **GL\_SHININESS** ступінь відбитого світла
- **GL\_AMBIENT\_AND\_DIFFUSE** обидва розсіяних світла

Колір задається у вигляді масиву з чотирьох елементів - **RGBA**. У разі **GL\_SHININESS** params вказує на число типу **float**, яке повинно бути в діапазоні від 0 до 128.

Вам треба всього лише модифікувати функцію **display**.

```
void CALLBACK display (void)
{
    GLUquadricObj * quadObj;
    GLfloat front_color [] = {0,1,0,1};
    GLfloat back_color [] = {0,0,1,1};
    quadObj = gluNewQuadric ();
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMaterialfv (GL_FRONT, GL_DIFFUSE, front_color);
    glMaterialfv (GL_BACK, GL_DIFFUSE, back_color);
    glPushMatrix ();
    glRotated (110, -1,1,0);
    gluCylinder (quadObj, 1, 0.5, 2, 10, 10);
    glPopMatrix ();
    gluDeleteQuadric (quadObj);
    auxSwapBuffers ();
}
```

І ви повинні дозволити режим освітленості для двох граней. За замовчуванням він заборонений. Додайте в функцію main наступну сходянку.

**glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);**

### *3.2.2.3 Функції згасання*

Це функція зміни інтенсивності освітлення (інтенсивність світла не зменшується з відстанню), використовується разом з точковим освітленням

- **GL\_POSITION (0.0, 0.0, 1.0, 0.0)** // позиція джерела світла (за замовчуванням джерело світла спрямоване)
- **GL\_CONSTANT\_ATTENUATION 1.0** // постійна  $k_{const}$  в функції згасання  $f(d)$
- **GL\_LINEAR\_ATTENUATION 0.0** // коефіцієнт  $k_{linear}$  при лінійному члені в функції згасання  $f(d)$
- **GL\_QUADRATIC\_ATTENUATION 0.0** // коефіцієнт  $k_{quadratic}$  при квадраті відстані в функції згасання  $f(d)$

### *3.2.2.4 Прожектори*

Одним з різновидів точкового джерела є прожектор. Для нього застосовні всі параметри, що і для точкового джерела, але крім цього прожектор дозволяє обмежити поширення світла конусом. Для цього конуса можна задати коефіцієнт зменшення інтенсивності залежно від кута між віссю конуса і променем поширення світла.

- **GL\_SPOT\_DIRECTION (0.0, 0.0, -1.0)** // (x, y, z) - напрям прожектора (вісь обмежуючого конусу)
- **GL\_SPOT\_CUTOFF 180.0** // кут між віссю і стороною конуса (він же половина кута при вершині)
- **GL\_SPOT\_EXPONENT 0.0** // експонента убавання інтенсивності

### 3.3 Методика виконання лабораторної роботи

#### 3.3.1 Приклад накладення текстури

```
#include <windows.h>
#include <glut.h>
#include <glaux.h>
#pragma comment (lib, "glaux.lib")

unsigned int textures[1];

void LoadTextures()
{
    AUX_RGBImageRec *texture1=auxDIBImageLoadA("egypt.bmp");
    glGenTextures(1, &textures[0]);
    glBindTexture(GL_TEXTURE_2D, textures[0]);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, texture1->sizeX, texture1->sizeY, 0,
GL_RGB, GL_UNSIGNED_BYTE, texture1->data);
}

void Draw()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUADS);
    glColor3f (0.3, 0.3, 0.6);
    glTexCoord2f(0.0,0.0); glVertex2f(-3.0, -3.0);
    glColor3f (0.0, 0.9, 1.0); glTexCoord2f(0.0,1.0); glVertex2f(-3.0, 3.0);
    glColor3f (1.0, 1.0, 0.0); glTexCoord2f(1.0,1.0); glVertex2f(3.0, 3.0);
    glColor3f (0.0, 0.9, 1.0); glTexCoord2f(1.0,0.0); glVertex2f(3.0, -3.0);

    glEnd();

    glFlush();
}

void Initialize()
{
    LoadTextures();
    glEnable(GL_TEXTURE_2D);
    glClearColor(0.0,0.0,0.0,1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-4.0,4.0,-4.0,4.0,-10.0,10.0);
    glMatrixMode(GL_MODELVIEW);
}
```

```

}

int main (int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(480,480);
    glutInitWindowPosition(10,10);
    glutCreateWindow("Square_egypt_hieroglyphs");
    glClearColor(0.0,0.0,0.0,0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    Initialize();
    glutDisplayFunc(Draw);
    glutMainLoop();
}

```



Рисунок 3.1 – Робота програми

Відповідно, сама текстура виглядала так:



Рисунок 3.2 – Оригінальна текстура

### 3.3.2 Приклад встановлення світла

```
#include <GLUT/glut.h>
#include <math.h>
#define PI 3.141592653
int light_sample = 1;

// ініціалізація
void init (void) {
    // кольор фону
    glClearColor (0.3, 0.3, 0.3, 0.0);
    // розрахунок освітлення
    glEnable(GL_LIGHTING);
    // двухсторонній розрахунок освітлення
    glLightModel(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
    // автоматичне приведення нормалей до
    // одиничної довжини
    glEnable(GL_NORMALIZE);
}

void reshape(int width, int height) {
    // двувимірне вікно виводу
    glViewport(0, 0, width, height);
}
```

```

// ортогональна проєкція
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.2, 1.2, -1.2, 1.2, -1, 1);
// модельна матриця одинична
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}

void display(void) {
    // очищаємо буфер кадра та глибини
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // властивості матеріалу
    GLfloat material_diffuse[] = {1.0, 1.0, 1.0, 1.0};
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse);
    // встановлення джерела світла
    if (light_sample == 1) {
        // направлене джерело світла
        GLfloat light0_diffuse[] = {0.4, 0.7, 0.2};
        GLfloat light0_direction[] = {0.0, 0.0, 1.0, 0.0};
        glEnable(GL_LIGHT0);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
        glLightfv(GL_LIGHT0, GL_POSITION, light0_direction);
    }
    if (light_sample == 2) {
        // точкове джерело світла
        // спадання інтенсивності з відстанню відключено (за
        // замовчуванням)
        GLfloat light1_diffuse[] = {0.4, 0.7, 0.2};
        GLfloat light1_position[] = {0.0, 0.0, 1.0, 1.0};
        glEnable(GL_LIGHT1);
        glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
        glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
    }
    if (light_sample == 3) {
        // точкове джерело світла
        // спадання інтенсивності з відстанню задано функцією f(d)
        = 1.0 / (0.4 * d * d + 0.2 * d)
        GLfloat light2_diffuse[] = {0.4, 0.7, 0.2};
        GLfloat light2_position[] = {0.0, 0.0, 1.0, 1.0};
        glEnable(GL_LIGHT2);
        glLightfv(GL_LIGHT2, GL_DIFFUSE, light2_diffuse);
        glLightfv(GL_LIGHT2, GL_POSITION, light2_position);
        glLightf(GL_LIGHT2, GL_CONSTANT_ATTENUATION, 0.0);
        glLightf(GL_LIGHT2, GL_LINEAR_ATTENUATION, 0.2);
        glLightf(GL_LIGHT2, GL_QUADRATIC_ATTENUATION, 0.4);
    }
}

```



```

    if (light_sample == 4) {
        // прожектор
        // спадання інтенсивності з відстанню відключено (за
замовчуванням)
        // половина кута при вершині 30 градусів
        // напрямок на центр поверхні
        GLfloat light3_diffuse[] = {0.4, 0.7, 0.2};
        GLfloat light3_position[] = {0.0, 0.0, 1.0, 1.0};
        GLfloat light3_spot_direction[] = {0.0, 0.0, -1.0};
        glEnable(GL_LIGHT3);
        glLightfv(GL_LIGHT3, GL_DIFFUSE, light3_diffuse);
        glLightfv(GL_LIGHT3, GL_POSITION, light3_position);
        glLightf(GL_LIGHT3, GL_SPOT_CUTOFF, 30);
        glLightfv(GL_LIGHT3, GL_SPOT_DIRECTION,
light3_spot_direction);
    }
    if (light_sample == 5) {
        // прожектор
        // спадання інтенсивності з відстанню відключено (за
замовчуванням)
        // половина кута при вершині 30 градусів
        // напрям на центр поверхні
        // включений розрахунок зменшення інтенсивності для
прожектора
        GLfloat light4_diffuse[] = {0.4, 0.7, 0.2};
        GLfloat light4_position[] = {0.0, 0.0, 1.0, 1.0};
        GLfloat light4_spot_direction[] = {0.0, 0.0, -1.0};
        glEnable(GL_LIGHT4);
        glLightfv(GL_LIGHT4, GL_DIFFUSE, light4_diffuse);
        glLightfv(GL_LIGHT4, GL_POSITION, light4_position);
        glLightf(GL_LIGHT4, GL_SPOT_CUTOFF, 30);
        glLightfv(GL_LIGHT4, GL_SPOT_DIRECTION,
light4_spot_direction);
        glLightf(GL_LIGHT4, GL_SPOT_EXPONENT, 15.0);
    }
    if (light_sample == 6) {
        // декілька джерел світла
        GLfloat light5_diffuse[] = {1.0, 0.0, 0.0};
        GLfloat light5_position[] = {0.5 * cos(0.0), 0.5 *
sin(0.0), 1.0, 1.0};
        glEnable(GL_LIGHT5);
        glLightfv(GL_LIGHT5, GL_DIFFUSE, light5_diffuse);
        glLightfv(GL_LIGHT5, GL_POSITION, light5_position);
        glLightf(GL_LIGHT5, GL_CONSTANT_ATTENUATION, 0.0);
        glLightf(GL_LIGHT5, GL_LINEAR_ATTENUATION, 0.4);
        glLightf(GL_LIGHT5, GL_QUADRATIC_ATTENUATION, 0.8);
        GLfloat light6_diffuse[] = {0.0, 1.0, 0.0};
    }

```

```

    GLfloat light6_position[] = {0.5 * cos(2 * PI / 3), 0.5 *
sin(2 * PI / 3), 1.0, 1.0};
    glEnable(GL_LIGHT6);
    glLightfv(GL_LIGHT6, GL_DIFFUSE, light6_diffuse);
    glLightfv(GL_LIGHT6, GL_POSITION, light6_position);
    glLightf(GL_LIGHT6, GL_CONSTANT_ATTENUATION, 0.0);
    glLightf(GL_LIGHT6, GL_LINEAR_ATTENUATION, 0.4);
    glLightf(GL_LIGHT6, GL_QUADRATIC_ATTENUATION, 0.8);
    GLfloat light7_diffuse[] = {0.0, 0.0, 1.0};
    GLfloat light7_position[] = {0.5 * cos(4 * PI / 3), 0.5 *
sin(4 * PI / 3), 1.0, 1.0};
    glEnable(GL_LIGHT7);
    glLightfv(GL_LIGHT7, GL_DIFFUSE, light7_diffuse);
    glLightfv(GL_LIGHT7, GL_POSITION, light7_position);
    glLightf(GL_LIGHT7, GL_CONSTANT_ATTENUATION, 0.0);
    glLightf(GL_LIGHT7, GL_LINEAR_ATTENUATION, 0.4);
    glLightf(GL_LIGHT7, GL_QUADRATIC_ATTENUATION, 0.8);
}
// поверхня
GLfloat x, y;
glBegin(GL_QUADS);
glNormal3f(0.0, 0.0, -1.0);
for (x = -1.0; x < 1.0; x += 0.005) {
    for (y = -1.0; y < 1.0; y += 0.005) {
        glVertex3f(x, y, 0.0);
        glVertex3f(x, y + 0.005, 0.0);
        glVertex3f(x + 0.005, y + 0.005, 0.0);
        glVertex3f(x + 0.005, y, 0.0);
    }
}
glEnd();
// відключення джерел світла
glDisable(GL_LIGHT0);
glDisable(GL_LIGHT1);
glDisable(GL_LIGHT2);
glDisable(GL_LIGHT3);
glDisable(GL_LIGHT4);
glDisable(GL_LIGHT5);
glDisable(GL_LIGHT6);
glDisable(GL_LIGHT7);
// елемент подвійної буферизації
glutSwapBuffers();
}
void keyboard_function(unsigned char key, int x, int y) {
    if (key == '1') light_sample = 1;
    if (key == '2') light_sample = 2;
    if (key == '3') light_sample = 3;
}

```

```

    if (key == '4') light_sample = 4;
    if (key == '5') light_sample = 5;
    if (key == '6') light_sample = 6;
    glutPostRedisplay();
}
int main (int argc, char** argv) {
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition (50, 100);
    glutInitWindowSize (500, 500);
    glutCreateWindow ("Приклад встановлення джерел світла в
OpenGL");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard_function);
    glutMainLoop ();
}

```

За допомогою клавіш 1, 2, 3, 4, 5, 6 ви зможете переключати різні варіанти освітлення



Рисунок 3.3 – Направлене джерело світла

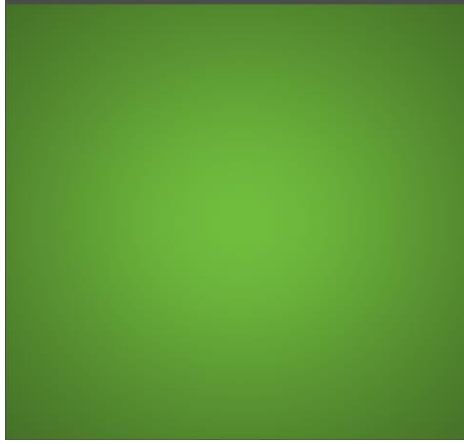


Рисунок 3.4 – Точкове джерело світла, спадання інтенсивності з відстанню  
вимкнено

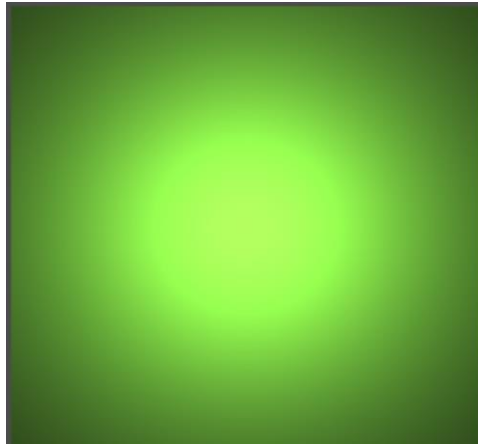


Рисунок 3.5 – Точкове джерело світла, спадання інтенсивності з відстанню  
увімкнено

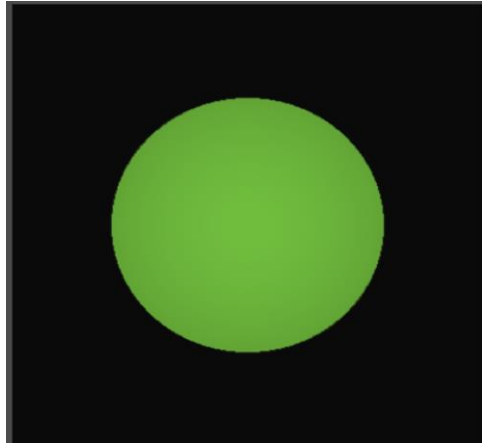


Рисунок 3.6 – Проектор

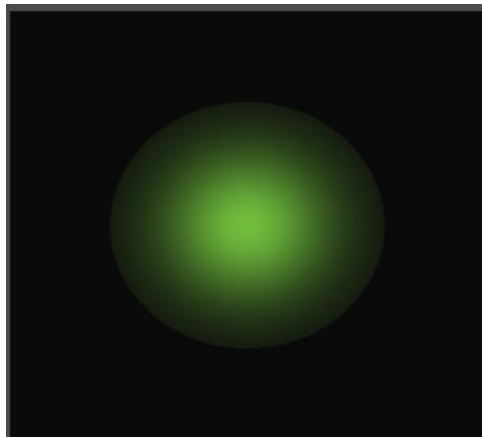


Рисунок 3.7 – Проектор, включений розрахунок зменшення інтенсивності  
для прожектора

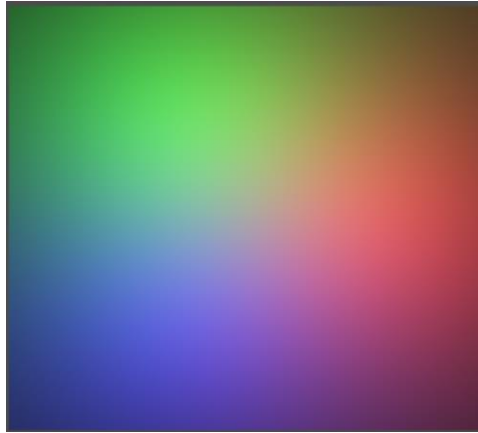


Рисунок 3.8 – Кілька джерел світла

### 3.3.3 Накладання текстури на 3d- об'єкт

```
#include <QtGui>
#include "mainscene.h"
#include <math.h>
#include <glxt.h>
#include <gl/gl.h>

MainScene::MainScene(QWidget *parent):QGLWidget(parent)
{
    xAxisRotation = yAxisRotation = 0;
}

MainScene::~MainScene()
{
}

void MainScene::initializeGL()
{
    glClearColor(0.0, 0.0, 0.0, 1.0);

    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_FLAT);
    glEnable(GL_CULL_FACE);
    glEnable(GL_TEXTURE_2D);
    // glEnable(GL_MULTISAMPLE);

    generateTextures();
}
```

```

    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
}

void MainScene::resizeGL(int nWidth, int nHeight)
{
    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();

    glOrtho(-0.5, 1.5, -0.5, 1.5, -10.0, 10.0);

    glViewport(0, 0, (GLint)nWidth, (GLint)nHeight);

    currentWidth = nWidth;
    currentHeight = nHeight;
}

void MainScene::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);

    glLoadIdentity();

    glRotatef(yAxisRotation, 0.0, 1.0, 0.0);
    glRotatef(xAxisRotation, 1.0, 0.0, 0.0);

    glBindTexture(GL_TEXTURE_2D, textures[0]);

    cubeVertexArray[0][0] = 0.0;
    cubeVertexArray[0][1] = 0.0;
    cubeVertexArray[0][2] = 1.0;

    cubeVertexArray[1][0] = 0.0;
    cubeVertexArray[1][1] = 1.0;
    cubeVertexArray[1][2] = 1.0;

    cubeVertexArray[2][0] = 1.0;
    cubeVertexArray[2][1] = 1.0;
    cubeVertexArray[2][2] = 1.0;

    cubeVertexArray[3][0] = 1.0;
    cubeVertexArray[3][1] = 0.0;
    cubeVertexArray[3][2] = 1.0;

```

```

cubeVertexArray[4][0] = 0.0;
cubeVertexArray[4][1] = 0.0;
cubeVertexArray[4][2] = 0.0;

cubeVertexArray[5][0] = 0.0;
cubeVertexArray[5][1] = 1.0;
cubeVertexArray[5][2] = 0.0;

cubeVertexArray[6][0] = 1.0;
cubeVertexArray[6][1] = 1.0;
cubeVertexArray[6][2] = 0.0;

cubeVertexArray[7][0] = 1.0;
cubeVertexArray[7][1] = 0.0;
cubeVertexArray[7][2] = 0.0;

cubeTextureArray[0][0] = 0.0;
cubeTextureArray[0][1] = 0.0;

cubeTextureArray[1][0] = 1.0;
cubeTextureArray[1][1] = 0.0;

cubeTextureArray[2][0] = 1.0;
cubeTextureArray[2][1] = 1.0;

cubeTextureArray[3][0] = 0.0;
cubeTextureArray[3][1] = 1.0;

cubeIndexArray[0][0] = 0;
cubeIndexArray[0][1] = 3;
cubeIndexArray[0][2] = 2;
cubeIndexArray[0][3] = 1;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);

glBindTexture(GL_TEXTURE_2D, textures[1]);

cubeTextureArray[0][0] = 0.0;
cubeTextureArray[0][1] = 0.0;

cubeTextureArray[1][0] = 1.0;
cubeTextureArray[1][1] = 0.0;

cubeTextureArray[5][0] = 1.0;

```



```

cubeTextureArray[5][1] = 1.0;

cubeTextureArray[4][0] = 0.0;
cubeTextureArray[4][1] = 1.0;

cubeIndexArray[0][0] = 0;
cubeIndexArray[0][1] = 1;
cubeIndexArray[0][2] = 5;
cubeIndexArray[0][3] = 4;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);

glBindTexture(GL_TEXTURE_2D, textures[2]);

cubeTextureArray[7][0] = 0.0;
cubeTextureArray[7][1] = 0.0;

cubeTextureArray[4][0] = 1.0;
cubeTextureArray[4][1] = 0.0;

cubeTextureArray[5][0] = 1.0;
cubeTextureArray[5][1] = 1.0;

cubeTextureArray[6][0] = 0.0;
cubeTextureArray[6][1] = 1.0;

cubeIndexArray[0][0] = 7;
cubeIndexArray[0][1] = 4;
cubeIndexArray[0][2] = 5;
cubeIndexArray[0][3] = 6;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);

glBindTexture(GL_TEXTURE_2D, textures[3]);

cubeTextureArray[3][0] = 0.0;
cubeTextureArray[3][1] = 0.0;

cubeTextureArray[7][0] = 1.0;
cubeTextureArray[7][1] = 0.0;

cubeTextureArray[6][0] = 1.0;

```

```

cubeTextureArray[6][1] = 1.0;

cubeTextureArray[2][0] = 0.0;
cubeTextureArray[2][1] = 1.0;

cubeIndexArray[0][0] = 3;
cubeIndexArray[0][1] = 7;
cubeIndexArray[0][2] = 6;
cubeIndexArray[0][3] = 2;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);

glBindTexture(GL_TEXTURE_2D, textures[4]);

cubeTextureArray[1][0] = 0.0;
cubeTextureArray[1][1] = 0.0;

cubeTextureArray[2][0] = 1.0;
cubeTextureArray[2][1] = 0.0;

cubeTextureArray[6][0] = 1.0;
cubeTextureArray[6][1] = 1.0;

cubeTextureArray[5][0] = 0.0;
cubeTextureArray[5][1] = 1.0;

cubeIndexArray[0][0] = 1;
cubeIndexArray[0][1] = 2;
cubeIndexArray[0][2] = 6;
cubeIndexArray[0][3] = 5;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);

glBindTexture(GL_TEXTURE_2D, textures[5]);

cubeTextureArray[0][0] = 0.0;
cubeTextureArray[0][1] = 0.0;

cubeTextureArray[4][0] = 1.0;
cubeTextureArray[4][1] = 0.0;

cubeTextureArray[7][0] = 1.0;

```

```

cubeTextureArray[7][1] = 1.0;

cubeTextureArray[3][0] = 0.0;
cubeTextureArray[3][1] = 1.0;

cubeIndexArray[0][0] = 0;
cubeIndexArray[0][1] = 4;
cubeIndexArray[0][2] = 7;
cubeIndexArray[0][3] = 3;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);
}

void MainScene::mousePressEvent(QMouseEvent *event)
{
    pressPosition = event->pos();
}

void MainScene::mouseMoveEvent(QMouseEvent *event)
{
    xAxisRotation += (180 * ((GLfloat)event->y() -
(GLfloat)pressPosition.y())) / (currentHeight);
    yAxisRotation += (180 * ((GLfloat)event->x() -
(GLfloat)pressPosition.x())) / (currentWidth);

    pressPosition = event->pos();

    updateGL();
}

void MainScene::generateTextures()
{
    glGenTextures(6, textures);

    QImage texture1;
    texture1.load(":/gran1.png");
    texture1 = QGLWidget::convertToGLFormat(texture1);
    glBindTexture(GL_TEXTURE_2D, textures[0]);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, (GLsizei)texture1.width(),
(GLsizei)texture1.height(), 0, GL_RGBA, GL_UNSIGNED_BYTE,
texture1.bits());
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
}

```

```

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

    QImage texture2;
    texture2.load(":/gran2.png");
    texture2 = QGLWidget::convertToGLFormat(texture2);
    glBindTexture(GL_TEXTURE_2D, textures[1]);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, (GLsizei)texture2.width(),
(GLsizei)texture2.height(), 0, GL_RGBA, GL_UNSIGNED_BYTE,
texture2.bits());
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

    QImage texture3;
    texture3.load(":/gran3.png");
    texture3 = QGLWidget::convertToGLFormat(texture3);
    glBindTexture(GL_TEXTURE_2D, textures[2]);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, (GLsizei)texture3.width(),
(GLsizei)texture3.height(), 0, GL_RGBA, GL_UNSIGNED_BYTE,
texture3.bits());
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

    QImage texture4;
    texture4.load(":/gran4.png");
    texture4 = QGLWidget::convertToGLFormat(texture4);
    glBindTexture(GL_TEXTURE_2D, textures[3]);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, (GLsizei)texture4.width(),
(GLsizei)texture4.height(), 0, GL_RGBA, GL_UNSIGNED_BYTE,
texture4.bits());

```

```

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

    QImage texture5;
    texture5.load(":/gran5.png");
    texture5 = QGLWidget::convertToGLFormat(texture5);
    glBindTexture(GL_TEXTURE_2D, textures[4]);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, (GLsizei)texture5.width(),
(GLsizei)texture5.height(), 0, GL_RGBA, GL_UNSIGNED_BYTE,
texture5.bits());
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

    QImage texture6;
    texture6.load(":/gran6.png");
    texture6 = QGLWidget::convertToGLFormat(texture6);
    glBindTexture(GL_TEXTURE_2D, textures[5]);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, (GLsizei)texture6.width(),
(GLsizei)texture6.height(), 0, GL_RGBA, GL_UNSIGNED_BYTE,
texture6.bits());
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
}

```

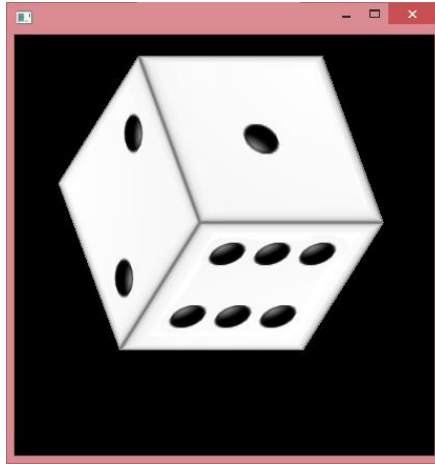


Рисунок 3.9 – Результат виконання програми

### 3.4 Завдання на виконання лабораторної роботи

Використовуючи різні графічні примітиви намалювати 3d-об'єкт призму або піраміду, в основі якої лежить багатокутник з кількістю вершин (`НОМЕР_ВАРІАНТА+2`) і накласти на нього довільну текстуру.

### 3.5 Зміст звіту

У звіті мають бути відображені такі питання: мета роботи; завдання до лабораторної роботи; файли програми; результати роботи програми; висновок.

### 3.6 Контрольні питання

3.6.1 Що таке текстура та для чого використовуються текстури?

3.6.2 Як задати конусне джерело світла?

3.6.3 Що виконує команда `glColorMaterial`?

3.6.4 Як задати фіксоване положення джерела світла? Чи можливо задавати положення джерела відносно локальних координат об'єкта?

3.6.5 Які буфери зображення використовуються в OpenGL та для чого?

## 4 ЛАБОРАТОРНА РОБОТА № 4

### «Моделювання фізичних явищ»

#### 4.1 Мета роботи

Навчитися моделювати прості фізичні явища в OpenGL

#### 4.2 Стислі теоретичні відомості

##### 4.2.1 Створення ефекту мерехтіння

**Slowdown** (гальмування) контролює, як швидко переміщаються частинки. Чим більше її значення, тим повільніше вони рухаються. Чим менше її значення, тим швидше вони рухаються. Якщо значення задано маленьке, частинки будуть рухатися дуже швидко. Швидкість, з якою частиночки переміщаються, буде задавати їх траєкторію руху по екрану. Більш повільні частки не будуть відлітати далеко.

Змінні **xspeed** і **yspeed** дозволяють нам контролювати напрямком хвоста потоку частинок.

**Xspeed** буде додаватися до поточної швидкості частинки по осі X. Якщо у **xspeed** - позитивне значення, то наша частка буде зміщуватися направо. Якщо у **xspeed** - від'ємне значення, то наша частка буде зміщуватися наліво. Чим вище значення, тим більше це зміщення у відповідному напрямку. **Yspeed** буде додаватися до поточної швидкості частинки по осі Y.

Нехай частинка має швидкість переміщення 5 по осі X і 0 по осі Y. Якщо ми зменшимо **xspeed** до  $-10$ , то швидкість переміщення буде дорівнює  $-10$  (**xspeed**) + 5 (початкова швидкість). Тому замість переміщення з темпом 10 вправо, частка буде переміщатися з темпом  $-5$  вліво

**glBegin (GL\_TRIANGLE\_STRIP);** // Побудова чотирикутника з трикутної смужки

Смужка з трикутників малюється як ряд трикутників (тристоронніх полігонів), використовуючи вершини V0, V1, V2, потім V2, V1, V3 (зверніть увагу на порядок), потім V2, V3, V4, і так далі. Порядок повинен гарантувати, що всі трикутники будуть виведені з тієї ж самої орієнтацією так, щоб смужка могла правильно формувати

частину поверхні. Дотримання орієнтації важливо для деяких операцій, типу відсікання. Повинні бути хоча б 3 точки, щоб було щось виведено.



Рисунок 4.1 – Смужка з двох трикутників

Тому перший трикутник виведений, використовуючи вершини 0, 1 і 2. Якщо Ви подивитесь на рисунок 4.1, Ви побачите, що точки вершин 0, 1 і 2 дійсно складають перший трикутник (верхня права, верхня ліва, нижня права). Другий трикутник виведений, використовуючи вершини 2, 1 і 3, вершини 2, 1 і 3 створюють другий трикутник (нижня права, верхня ліва, нижня права). Зауважте, що обидва трикутника виведені з тим же самим порядком обходу (проти годинникової стрілки). OpenGL буде міняти вершини, щоб гарантувати, що всі трикутники виведені тим же самим способом.

Є дві причини, для того щоб використовувати смужки з трикутників. По-перше, після визначення перших трьох вершин початкового трикутника, Ви повинні тільки визначати одну єдину точку для кожного іншого додаткового трикутника. Ця точка буде об'єднана з 2 попереднім вершинами для створення трикутника. По-друге, скорочуючи кількість даних, необхідних для створення трикутників ваша програма буде працювати швидше, і кількість коду або даних, необхідних для виведення об'єкта різко скоротиться.

#### 4.2.2 Створення ефекту туману

Розглянемо можливість OpenGL - створення ефекту туману. Легке затуманення сцени створює реалістичний ефект, а частенько може і приховати деякі артефакти, які з'являються, коли в сцені присутні віддалені об'єкти.



Туман в OpenGL реалізується шляхом зміни кольору об'єктів в сцені в залежності від їх глибини, тобто відстані до точки спостереження. Зміна кольору відбувається або для вершин примітивів, або для кожного пікселя на етапі растеризації в залежності від реалізації OpenGL. Цим процесом можна частково управляти.

Для включення ефекту затуманення необхідно викликати команду **glEnable (GL\_FOG)**.

Метод обчислення інтенсивності туману в вершині можна визначити за допомогою команд

```
void glFog [if] (enum pname, T param)
```

```
void glFog [if] v (enum pname, T params)
```

Аргумент pname може набувати таких значень:

**GL\_FOG\_MODE** аргумент param визначає формулу, по якій буде обчислюватися інтенсивність туману в точці.

В цьому випадку param може приймати значення

<b>GL_EXP</b>	Інтенсивність обчислюється за формулою $f = \exp(-d * z)$
<b>GL_EXP2</b>	Інтенсивність обчислюється за формулою $f = \exp(-(d * z)^2)$
<b>GL_LINEAR</b>	Інтенсивність обчислюється за формулою $f = e - z / e - s$ де z - відстань від вершини, в якій обчислюється інтенсивність туману, до точки спостереження. Коефіцієнти d, e, s задаються за допомогою наступних значень аргументу pname
<b>GL_FOG_DENSITY</b>	param визначає коефіцієнт d
<b>GL_FOG_START</b>	param визначає коефіцієнт s
<b>GL_FOG_END</b>	param визначає коефіцієнт e

Колір туману задається за допомогою аргументу pname, рівного

**GL\_FOG\_COLOR** params - покажчик на масив з 4-х компонент кольору

Наведемо приклад використання цього ефекту:

```
GLfloat FogColor[4]={0.5,0.5,0.5,1};
glEnable(GL_FOG);
glFogi(GL_FOG_MODE, GL_LINEAR);
glFogf(GL_FOG_START, 20.0);
glFogf(GL_FOG_END, 100.0);
glFogfv(GL_FOG_COLOR, FogColor);
```

## 4.3 Методика виконання лабораторної роботи

### 4.3.1 Приклад програми (мерехтіння)

```
#include "stdafx.h"
#include "Lesson19.h"
#include <windows.h> // Файл заголовка для Windows
#include <stdio.h> // Файл заголовка для стандартної бібліотеки
введення/виведення (HOBE)
#include <gl\gl.h> // Файли заголовка для бібліотеки OpenGL32
#include <gl\glu.h> // Файли заголовка для бібліотеки GLu32
#include <gl\glaux.h> // Файли заголовка для бібліотеки GLaux
#define MAX_PARTICLES 3000 // Число частинок для створення ( HOBE )

HGLRC hRC = NULL; // Постійний контекст візуалізації (рендеринга)
HDC hDC = NULL; // Приватний контекст пристрою GDI
HWND hWnd = NULL; // Тут буде зберігатися дескриптор вікна
HINSTANCE hInstance; // Тут буде зберігатися дескриптор додатка

bool keys[256]; // Масив, що використовується для операцій з
клавіатурою
bool active = true; // Флаг активності вікна, встановлений в true за
замовченням
bool fullscreen = true; // Флаг режиму вікна, встановлений в
повноекранний за замовченням
bool rainbow=true; // Режим райдуги? ( HOBE )
bool sp; // Пробіл натиснутий? ( HOBE )
bool rp; // Введення натиснуте? ( HOBE )

float slowdown=2.0f; // Гальмування частинок
float xspeed; // Основна швидкість по X (з клавіатури змінюється
напрямок хвоста)
float yspeed; // Основна швидкість по Y (з клавіатури змінюється
напрямок хвоста)
```

```

float zoom=-40.0f; // Масштаб пучка частинок

GLuint loop; // Змінна циклу
GLuint col; // Поточний вибраний колір
GLuint delay; // Затримка для ефекту райдуги
GLuint texture[1]; // Пам'ять для нашої текстури

typedef struct // Структура частинки
{
    bool active; // Активність (Так/ні)
    float life; // Життя
    float fade; // Швидкість згасання

    float r; // Червоне значення
    float g; // Зелене значення
    float b; // Синє значення

    float x; // X позиція
    float y; // Y позиція
    float z; // Z позиція

    float xi; // X напрямок
    float yi; // Y напрямок
    float zi; // Z напрямок

    float xg; // X гравітація
    float yg; // Y гравітація
    float zg; // Z гравітація
}
particles; // Структура частинок

particles particle[MAX_PARTICLES]; // Масив частинок (Місце для
інформації про частинки)

static GLfloat colors[12][3]= // Колірна райдуга
{
    {1.0f,0.5f,0.5f},{1.0f,0.75f,0.5f},{1.0f,1.0f,0.5f},{0.75f,1.0f,0.5f},
    },
    {0.5f,1.0f,0.5f},{0.5f,1.0f,0.75f},{0.5f,1.0f,1.0f},{0.5f,0.75f,1.0f},
    },
    {0.5f,0.5f,1.0f},{0.75f,0.5f,1.0f},{1.0f,0.5f,1.0f},{1.0f,0.5f,0.75f},
    },
    };

```

```
LRESULT CALLBACK WndProc( HWND, UINT, WPARAM, LPARAM ); // Прототип
функції WndProc
```

```
int InitGL(GLvoid) // Всі налаштування для OpenGL робляться тут
{
    if (!LoadGLTextures()) // Перехід на процедуру завантаження текстури
    {
        return FALSE;      // Якщо текстура не завантажена повертаємо FALSE
    }

    glShadeModel(GL_SMOOTH); // Дозвіл згладженого зафарбовування
    glClearColor(0.0f, 0.0f, 0.0f, 0.5f); // Чорний фон
    glClearDepth(1.0f); // Встановлення буфера глибини
    glDisable(GL_DEPTH_TEST); // Заборона тексту глибини
    glEnable(GL_BLEND); // Дозволяємо змішування
    glBlendFunc(GL_SRC_ALPHA, GL_ONE); // Тип змішування
    // Покращені обчислення перспективи
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
    glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);
    // Покращене точкове змішування
    glEnable(GL_TEXTURE_2D);
    // Дозвіл накладання текстури
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    // Вибір нашої текстури
    for (loop=0; loop<MAX_PARTICLES; loop++)
    // Ініціалізація всіх частинок
    {
        particle[loop].active=true; // Зробити всі частинки активними
        particle[loop].life=1.0f; // Зробити вся частинки з повним життям
        //Випадкова швидкість згасання
        particle[loop].fade=float(rand()%100)/1000.0f+0.003f;
        particle[loop].r=colors[loop*(12/MAX_PARTICLES)][0]; // Вибір
        червоного кольору райдуги
        particle[loop].g=colors[loop*(12/MAX_PARTICLES)][1]; // Вибір
        зеленого кольору райдуги
        particle[loop].b=colors[loop*(12/MAX_PARTICLES)][2]; // Вибір
        синього кольору райдуги

        particle[loop].xi=float((rand()%50)-26.0f)*10.0f; // Випадкова
        швидкість по осі X
        particle[loop].yi=float((rand()%50)-25.0f)*10.0f; // Випадкова
        швидкість по осі Y
        particle[loop].zi=float((rand()%50)-25.0f)*10.0f; // Випадкова
        швидкість по осі Z

        particle[loop].xg=0.0f; // Задамо горизонтальне притягання в нуль
        particle[loop].yg=-0.8f; // Задамо вертикальне притягання донизу
    }
}
```

```

    particle[loop].zg=0.0f; // Задамо притягання по осі Z в нуль
}
return true; // Ініціалізація пройшла ОК
}

int DrawGLScene( GLvoid ) // Тут відбуватиметься все промальовування
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT ); // Очистити
    екран і буфер глибини
    glLoadIdentity(); // Скинути поточну матрицю

    for (loop=0;loop<MAX_PARTICLES;loop++)
// Цикл по всім частинкам
    {
        if (particle[loop].active) // Якщо частинки активні
        {
            float x=particle[loop].x; // Захопимо позицію X нашої частинки
            float y=particle[loop].y; // Захопимо позицію Y нашої частинки
            float z=particle[loop].z+zoom;
// Позиція частинки по Z + Zoom

            // Виведення частинки, використовуючи наші RGB значення, згасання
            частинки відповідно до її життя

            glColor4f(particle[loop].r,particle[loop].g,particle[loop].b,particle[loop].life);

            glBegin(GL_TRIANGLE_STRIP);
// Побудова чотирикутника з трикутної площини
            glTexCoord2d(1,1); glVertex3f(x+0.5f,y+0.5f,z); // Верхня права
            glTexCoord2d(0,1); glVertex3f(x-0.5f,y+0.5f,z); // Верхня ліва
            glTexCoord2d(1,0); glVertex3f(x+0.5f,y-0.5f,z); // Нижня права
            glTexCoord2d(0,0); glVertex3f(x-0.5f,y-0.5f,z); // Нижня ліва
            glEnd(); // Завершення побудови смуги трикутників

            // Пересування по осі X на швидкість по X
            particle[loop].x+=particle[loop].xi/(slowdown*1000);
            // Пересування по осі Y на швидкість по Y
            particle[loop].y+=particle[loop].yi/(slowdown*1000);
            // Пересування по осі Z на швидкість по Z
            particle[loop].z+=particle[loop].zi/(slowdown*1000);

            particle[loop].xi+=particle[loop].xg; // Притягання по X для
            цього запису
            particle[loop].yi+=particle[loop].yg; // Притягання по Y для
            цього запису

```

```

    particle[loop].zi+=particle[loop].zg;    // Притягання по Z для
цього запису

    particle[loop].life-=particle[loop].fade; // Зменшити життя
частинки на 'згасання'

    if (particle[loop].life<0.0f)           // Якщо частинка згасла
    {
        particle[loop].life=1.0f;          // Дати нове життя
        // Випадкове значення згасання
        particle[loop].fade=float(rand()%100)/1000.0f+0.003f;

        particle[loop].x=0.0f;             // На центр осі X
        particle[loop].y=0.0f;             // На центр осі Y
        particle[loop].z=0.0f;             // На центр осі Z

        particle[loop].xi=xspeed+float((rand()%60)-32.0f); // Швидкість
і напрямок по осі X
        particle[loop].yi=yspeed+float((rand()%60)-30.0f); // Швидкість
і напрямок по осі Y
        particle[loop].zi=float((rand()%60)-30.0f); // Швидкість і
напрямок по осі Z

        particle[loop].r=colors[col][0]; // Вибір червоного з таблиці кольорів
        particle[loop].g=colors[col][1]; // Вибір зеленого з таблиці кольорів
        particle[loop].b=colors[col][2]; // Вибір синього з таблиці кольорів
    }
    // Якщо клавіша 8 на цифровій клавіатурі натиснута й гравітація
менше ніж 1.5
    // тоді збільшимо притягання вгору
    if (keys[VK_NUMPAD8] && (particle[loop].yg<1.5f))
particle[loop].yg+=0.01f;
    // Якщо клавіша 2 на цифровій клавіатурі натиснута й гравітація
більше ніж -1.5
    // тоді збільшимо притягання донизу
    if (keys[VK_NUMPAD2] && (particle[loop].yg>-1.5f))
particle[loop].yg-=0.01f;
    // Якщо клавіша 6 на цифровій клавіатурі натиснута й гравітація
менше ніж 1.5
    // тоді збільшимо притягання праворуч
    if (keys[VK_NUMPAD6] && (particle[loop].xg<1.5f))
particle[loop].xg+=0.01f;
    // Якщо клавіша 4 на цифровій клавіатурі натиснута й гравітація
більше ніж -1.5
    // тоді збільшимо притягання ліворуч
    if (keys[VK_NUMPAD4] && (particle[loop].xg>-1.5f))
particle[loop].xg-=0.01f;

```

```

if (keys[VK_TAB])          // Клавіша табуляції викликає вибух
{
    particle[loop].x=0.0f;    // Центр по осі X
    particle[loop].y=0.0f;    // Центр по осі Y
    particle[loop].z=0.0f;    // Центр по осі Z
    particle[loop].xi=float((rand()%50)-26.0f)*10.0f; // Випадкова
швидкість по осі X
    particle[loop].yi=float((rand()%50)-25.0f)*10.0f; // Випадкова
швидкість по осі Y
    particle[loop].zi=float((rand()%50)-25.0f)*10.0f; // Випадкова
швидкість по осі Z
}
}
return true;                // Промальовування пройшло успішно
}

```

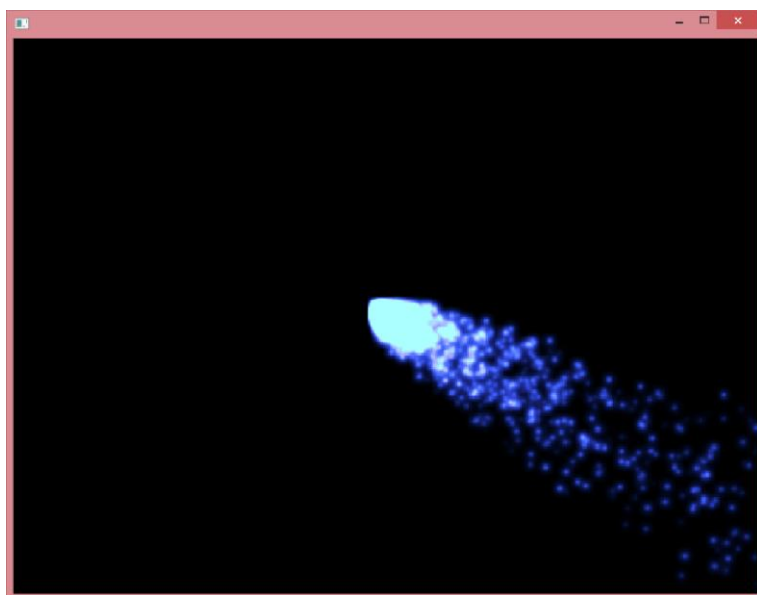


Рисунок 4.2 – Результат роботи програми (мерехтіння)

### 4.3.2 Приклад програми (ефект туману)

#### #mainwindow.h

```
#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
```

#### #mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QGLWidget>
#include <QtOpenGL>
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}
MainWindow::~MainWindow()
{
    delete ui;
}
```

#### #mainscene.h

```
#ifndef MAINSCENE_H
#define MAINSCENE_H
#include <QTimer>
#include <QGLWidget>
#include <QPoint>
#include <QDebug>
#include <QMouseEvent>
class MainScene : public QGLWidget
```



```

{
    Q_OBJECT
private:
    GLfloat cubeVertexArray[8][3];
    GLfloat cubeColorArray[8][3];
    GLubyte cubeIndexArray[6][4];
    GLfloat cubeTextureArray[8][2];
    QPoint pressPosition;
    QPoint releasePosition;
    GLfloat xAxisRotation;
    GLfloat yAxisRotation;
    GLfloat currentWidth;
    GLfloat currentHeight;
    GLuint textures[1];
    void generateTextures();
private slots:
protected:
    void initializeGL();
    void resizeGL(int nWidth, int nHeight);
    void paintGL();
    void mousePressEvent(QMouseEvent* event);
    void mouseMoveEvent(QMouseEvent* event);
public:
    MainScene(QWidget *parent = 0);
    ~MainScene();
};
#endif // MAINSCENE_H

```

## #mainscene.cpp

```

#include <QtGui>
#include "mainscene.h"
#include <math.h>
#include <glext.h>

MainScene::MainScene(QWidget *parent):QGLWidget(parent)
{
    xAxisRotation = yAxisRotation = 0;
}

MainScene::~MainScene()
{}

void MainScene::initializeGL()
{
    glClearColor(0.4f, 0.36f, 0.88f, 1.0f);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_FLAT);
}

```

```

    glEnable(GL_CULL_FACE);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_MULTISAMPLE);
    generateTextures();
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
}

void MainScene::resizeGL(int nWidth, int nHeight)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -10.0, 10.0);
    glViewport(0, 0, (GLint)nWidth, (GLint)nHeight);

    currentWidth = nWidth;
    currentHeight = nHeight;
}

void MainScene::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(yAxisRotation, 0.0, 1.0, 0.0);
    glRotatef(xAxisRotation, 1.0, 0.0, 0.0);

    GLuint fogMode[] = { GL_EXP, GL_EXP2, GL_LINEAR };
    GLuint fogfilter = 1;
    GLfloat fogColor[4] = {0.4f, 0.36f, 0.88f, 1.0f};

    glEnable(GL_FOG);
    glFogi(GL_FOG_MODE, fogMode[fogfilter]);
    glFogfv(GL_FOG_COLOR, fogColor);
    glFogf(GL_FOG_DENSITY, 1.5f);
    glHint(GL_FOG_HINT, GL_DONT_CARE);
    glFogf(GL_FOG_START, 0.0f);
    glFogf(GL_FOG_END, 100.0f);

    glBindTexture(GL_TEXTURE_2D, textures[0]);

    cubeVertexArray[0][0] = 0.0;
    cubeVertexArray[0][1] = 0.0;
    cubeVertexArray[0][2] = 1.0;

```

```

cubeVertexArray[1][0] = 0.0;
cubeVertexArray[1][1] = 1.0;
cubeVertexArray[1][2] = 0.5;

cubeVertexArray[2][0] = 1.0;
cubeVertexArray[2][1] = 1.0;
cubeVertexArray[2][2] = 0.5;

cubeVertexArray[3][0] = 1.0;
cubeVertexArray[3][1] = 0.0;
cubeVertexArray[3][2] = 1.0;

cubeVertexArray[4][0] = 0.0;
cubeVertexArray[4][1] = 0.0;
cubeVertexArray[4][2] = 0.0;

cubeVertexArray[5][0] = 0.0;
cubeVertexArray[5][1] = 1.0;
cubeVertexArray[5][2] = 0.5;

cubeVertexArray[6][0] = 1.0;
cubeVertexArray[6][1] = 1.0;
cubeVertexArray[6][2] = 0.5;

cubeVertexArray[7][0] = 1.0;
cubeVertexArray[7][1] = 0.0;
cubeVertexArray[7][2] = 0.0;

cubeTextureArray[0][0] = 0.0;
cubeTextureArray[0][1] = 0.0;

cubeTextureArray[1][0] = 1.0;
cubeTextureArray[1][1] = 0.0;

cubeTextureArray[2][0] = 1.0;
cubeTextureArray[2][1] = 1.0;

cubeTextureArray[3][0] = 0.0;
cubeTextureArray[3][1] = 1.0;

cubeIndexArray[0][0] = 0;
cubeIndexArray[0][1] = 3;
cubeIndexArray[0][2] = 2;
cubeIndexArray[0][3] = 1;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);

```

```

glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);
glBindTexture(GL_TEXTURE_2D, textures[0]); //11111111111111

cubeTextureArray[0][0] = 0.0;
cubeTextureArray[0][1] = 0.0;

cubeTextureArray[1][0] = 1.0;
cubeTextureArray[1][1] = 0.0;

cubeTextureArray[5][0] = 1.0;
cubeTextureArray[5][1] = 1.0;

cubeTextureArray[4][0] = 0.0;
cubeTextureArray[4][1] = 1.0;

cubeIndexArray[0][0] = 0;
cubeIndexArray[0][1] = 1;
cubeIndexArray[0][2] = 5;
cubeIndexArray[0][3] = 4;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);
glBindTexture(GL_TEXTURE_2D, textures[0]); //задняя

cubeTextureArray[7][0] = 0.0;
cubeTextureArray[7][1] = 0.0;

cubeTextureArray[4][0] = 1.0;
cubeTextureArray[4][1] = 0.0;

cubeTextureArray[5][0] = 1.0;
cubeTextureArray[5][1] = 1.0;

cubeTextureArray[6][0] = 0.0;
cubeTextureArray[6][1] = 1.0;

cubeIndexArray[0][0] = 7;
cubeIndexArray[0][1] = 4;
cubeIndexArray[0][2] = 5;
cubeIndexArray[0][3] = 6;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);
glBindTexture(GL_TEXTURE_2D, textures[0]); //правая

```

```

cubeTextureArray[3][0] = 0.0;
cubeTextureArray[3][1] = 0.0;

cubeTextureArray[7][0] = 1.0;
cubeTextureArray[7][1] = 0.0;

cubeTextureArray[6][0] = 1.0;
cubeTextureArray[6][1] = 1.0;

cubeTextureArray[2][0] = 0.0;
cubeTextureArray[2][1] = 1.0;

cubeIndexArray[0][0] = 3;
cubeIndexArray[0][1] = 7;
cubeIndexArray[0][2] = 6;
cubeIndexArray[0][3] = 2;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);
glBindTexture(GL_TEXTURE_2D, textures[0]); //нижняя

cubeTextureArray[0][0] = 0.0;
cubeTextureArray[0][1] = 0.0;

cubeTextureArray[4][0] = 1.0;
cubeTextureArray[4][1] = 0.0;

cubeTextureArray[7][0] = 1.0;
cubeTextureArray[7][1] = 1.0;

cubeTextureArray[3][0] = 0.0;
cubeTextureArray[3][1] = 1.0;

cubeIndexArray[0][0] = 0;
cubeIndexArray[0][1] = 4;
cubeIndexArray[0][2] = 7;
cubeIndexArray[0][3] = 3;

glVertexPointer(3, GL_FLOAT, 0, cubeVertexArray);
glTexCoordPointer(2, GL_FLOAT, 0, cubeTextureArray);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_BYTE, cubeIndexArray);
}
void MainScene::mousePressEvent(QMouseEvent *event)
{
    pressPosition = event->pos();
}

```

```

void MainScene::mouseMoveEvent(QMouseEvent *event)
{
    xAxisRotation += (180 * ((GLfloat)event->y() -
(GLfloat)pressPosition.y())) / (currentHeight);
    yAxisRotation += (180 * ((GLfloat)event->x() -
(GLfloat)pressPosition.x())) / (currentWidth);
    pressPosition = event->pos();
    updateGL();
}
void MainScene::generateTextures()
{
    glGenTextures(1, textures);
    QImage texture1;
    texture1.load(":/4.jpg");
    texture1 = QGLWidget::convertToGLFormat(texture1);
    glBindTexture(GL_TEXTURE_2D, textures[0]);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, (GLsizei)texture1.width(),
(GLsizei)texture1.height(), 0, GL_RGBA, GL_UNSIGNED_BYTE,
texture1.bits());
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
}

```

## #main.cpp

```

#include <QApplication>
#include "mainwindow.h"
#include "mainscene.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainScene scene;
    scene.resize(600, 600);
    scene.show();
    return a.exec();
}

```

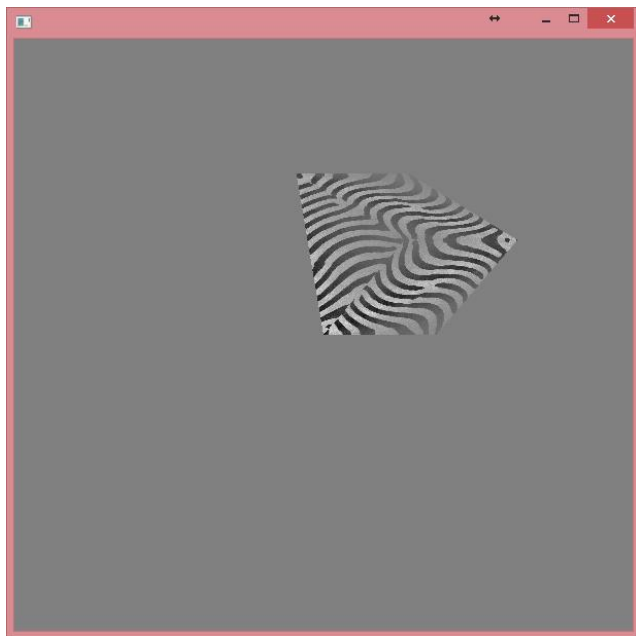


Рисунок 4.3 – Результат роботи програми (ефекту туману)

### 4.3.3 Приклад програми (розщеплення)

#### #mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QtWidgets/QMainWindow>
#include <qgridlayout.h>
#include <objectgl.h>
#include <QMenuBar>
#include <QMessageBox>

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    // Конструктор і деструктор
    MainWindow(QWidget *parent = 0, int w=600, int h=400);
    ~MainWindow();
};
```

```
protected slots:
    // Оновлення сцени
    void onTimer_UpdateDisplay();

    // Відкрийте діалогове вікно "about"
    void handleAbout();
protected:
    void resizeEvent(QResizeEvent *);
private:
    // Макет вікна
    QGridLayout *gridLayout;
    QWidget *gridLayoutWidget;
    // Центральний віджет (де намальовано вікно OpenGL)
    QWidget *centralWidget;
    // OpenGL object
    ObjectOpenGL *Object_GL;
};
#endif // MAINWINDOW_H
```

---

## #mainwindow.cpp

```
#include "mainwindow.h"

//Конструктор головного вікна
//Створить властивості вікна, меню тощо ...
MainWindow::MainWindow(QWidget *parent, int w, int h)
    : QMainWindow(parent)
{
    //Встановить розміри вікна
    this->resize(w,h);
    this->setWindowTitle("Object viewer");

    //Створить макет у головному вікні
    centralWidget = new QWidget (this);
    gridLayoutWidget = new QWidget (centralWidget);
    gridLayoutWidget->setGeometry(QRect(0,0,this->width(), this->
height()));
    gridLayout = new QGridLayout (gridLayoutWidget);

    //Створить екран OpenGL для карти
    Object_GL = new ObjectOpenGL (gridLayoutWidget);
    Object_GL -> setObjectName(QString::fromUtf8("ObjectOpenGL"));
    Object_GL -> setGeometry(QRect(0,0, this->width(), this->
height()));
};
```



```

// Вставте відображення openGL в макет
gridLayout->addWidget(Object_GL,0,0,1,1);
SetCentralWidget(centralWidget);

//Створить панель меню
QMain *FileMenu = menuBar()->addMenu("&File");
FileMenu->addSeparator();
FileMenu->addAction("Quit",qApp, SLOT
(quit()),QKeySequence(tr("Ctrl+Q")));

//Додати елементи меню
QMenu *ViewMenu = menuBar()->addMenu("&File");
FileMenu->addSeparator();
FileMenu->addAction("Front view", Object_GL, Slot (FrontView()),
QKeySequence(tr("Ctrl+f")));
FileMenu->addAction("Rear view", Object_GL, Slot (RearView()),
QKeySequence(tr("Ctrl+e")));
FileMenu->addAction("Left view", Object_GL, Slot (LeftView()),
QKeySequence(tr("Ctrl+l")));
FileMenu->addAction("Right view", Object_GL, Slot (RightView()),
QKeySequence(tr("Ctrl+r")));
FileMenu->addAction("Top view", Object_GL, Slot (TopView()),
QKeySequence(tr("Ctrl+t")));
FileMenu->addAction("Bottom view", Object_GL, Slot
(BottomView()), QKeySequence(tr("Ctrl+b")));
FileMenu->addSeparator();
ViewMenu->addAction("Isometric", Object_GL, Slot
(IsometricView()), QKeySequence(tr("Ctrl+i")));
QMenu *AboutMenu=menuBar()->addMenu("?");
AboutMenu->addAction("About
Convert_STL_2_Cube",this,SLOT(handleAbout()));

//Таймер (використовується для перемальовування вікон GL кожні 25
мсек)
QTimer *timer=new QTimer();
timer-
>connect(timer,SIGNAL(timeout()),this,SLOT(onTimer_UpdateDisplay()));
timer->start(25);

// Деструктор
MainWindow::~MainWindow()
{}

// У разі виклику події зміни розміру, розміри об'єктів у вікні змінюються
void MainWindow::resizeEvent(QResizeEvent *)
{
    Object_GL->resize(centralWidget->width(),centralWidget->height());
    gridLayoutWidget->setGeometry(QRect(0, 0, centralWidget->width(),
centralWidget->height()));
}

// Подія таймера: перефарбувати вікно OpenGL
void MainWindow::onTimer_UpdateDisplay()

```

```

{
    Object_GL->updateGL();
}

// Відкрийте діалогове вікно "about"
void MainWindow::handleAbout()
{
    QMessageBox::information(this, "About OpenGL Frame", "<H2>OpenGL
Frame</H2>2011<BR>Supported by the Cart-O-Matic project (ANR
CAROTTE)<BR>University of Angers (France, Europe)<BR>Designed by Philippe
Lucidarme <BR> Contact: philippe.lucidarme@univ-angers.fr. ");
}

```

## #objectgl.h

```

//using namespace std;

class ObjectOpenGL:public QGLWidget
{
    Q_OBJECT
public:
    ObjectOpenGL(QWidget *parent = 0);    //Конструктор
    ~ObjectOpenGL();                      //Деструктор

public slots:
    void    FrontView(void);              //Стандартний вид: front view
    void    RearView(void);               //Стандартний вид: read view
    void    LeftView(void);               //Стандартний вид: left view
    void    RightView(void);              //Стандартний вид: right view
    void    TopView(void);                //Стандартний вид: top view
    void    BottomView(void);              //Стандартний вид: bottom view
    void    IsometricView(void);           //Стандартний вид: isometric view

    void    SetXRotation(int angle);       //Оновити обертання навколо X
    void    SetYRotation(int angle);       //Оновити обертання навколо Y
    void    SetZRotation(int angle);       //Оновити обертання навколо Z

protected:
    void    initializeGL();                //Ініціалізація параметрів OpenGL
    void    paintGL();                     //Перемалювання сцени

    //Зміна розмірів відкритої GL сцени
    void    resizeGL(int width, int height);

    //Викликається при натисканні миші
    void    mouseMoveEvent(QMouseEvent *event);
}

```

```

//Викликається при переміщенні миші
void mousePressEvent(QMouseEvent *event);

//Викликається при прокручуванні колеса миші
void wheelEvent(QWheelEvent *event);

signals: //Сигнали для нових орієнтацій
void xRotationChanged(int angle);
void yRotationChanged(int angle);
void zRotationChanged(int angle);

private:
void Draw_Frame(); //Малювання ортонормального фрейму
void NormalizeAngle(int *angle); //Нормований кут між 0 і
360x16

QColor Background_Color; //Колір фону
QColor Axis_X_Color; //X осьовий колір
QColor Axis_Y_Color; //Y осьовий колір
QColor Axis_Z_Color; //Z осьовий колір
QColor Points_Color; //Колір точок

QSize WindowSize; //Розмір (у пікселях) вікна OpenGL
QPoint LastPos; //Остання позиція миші (у пікселях)
GLfloat dx; //Переміщення по осі X (для відображення)
GLfloat dy; //Переміщення по осі Y (для відображення)
GLfloat Zoom; //Маштаб об'єкта

int xRot; //Обертання навколо X
int yRot; //Обертання навколо Y
int zRot; //Обертання навколо Z
};

```

---

## #objectgl.cpp

```

#include "objectgl.h"
ObjectOpenGL::ObjectOpenGL(QWidget *parent) :
    QGLWidget(parent)
{
    Background_Color=QColor::fromRgb(50 ,50 ,100);
    Axis_X_Color=QColor::fromRgb(255,0 ,0 ,255);
    Axis_Y_Color=QColor::fromRgb(0 ,255,0 ,255);
    Axis_Z_Color=QColor::fromRgb(0 ,0 ,255,255);
    Points_Color=QColor::fromRgb(0 ,0 ,0 ,255);
    IsometricView();
}

```

```

ObjectOpenGL::~ObjectOpenGL()
{
    makeCurrent();
}

void ObjectOpenGL::initializeGL()
{
    qglClearColor(BackGround_Color);
    glShadeModel(GL_FLAT);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_BLEND);
    glEnable(GL_COLOR_MATERIAL);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glColorMaterial(GL_FRONT, GL_DIFFUSE);
    glEnable(GL_NORMALIZE);
}

void ObjectOpenGL::paintGL( )
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    GLfloat LightAmbient[]={0.4f,0.4f,0.4f,1.0f};
    GLfloat LightDiffuse[]={0.8f,0.8f,0.8f,1.0f};
    glLightfv(GL_LIGHT0, GL_AMBIENT, LightAmbient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, LightDiffuse);
    int LightPos[4]={0,0,10,1};
    glLightiv(GL_LIGHT0, GL_POSITION, LightPos);
    glRotated(xRot / 16.0, 1.0, 0.0, 0.0);
    glRotated(yRot / 16.0, 0.0, 1.0, 0.0);
    glRotated(zRot / 16.0, 0.0, 0.0, 1.0);
    glScalef(1,-1,1);
    Draw_Frame();
    glPushMatrix();
    glScalef(Zoom,Zoom,Zoom); // створення крапок
    glPointSize(4.0);
    for (double x=-0.25;x<=0.25;x+=0.05)
    {
        for (double y=-0.25;y<=0.25;y+=0.05)
        {
            for (double z=-0.25;z<=0.25;z+=0.05)
            {
                glBegin(GL_POINTS);
                qglColor(Points_Color);
                glVertex3d( x, y ,z);
                glEnd();
            }
        }
    }
}

```

```

    }
}
glPopMatrix();

glViewport(0, 0, WindowSize.width(), WindowSize.height());
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
GLfloat
Ratio=(GLfloat)WindowSize.width() / (GLfloat)WindowSize.height(
);
    glOrtho((-0.5+dx)*Ratio,
            ( 0.5+dx)*Ratio ,
            +0.5+dy,
            -0.5+dy,
            -1500.0, 1500.0);
    glMatrixMode(GL_MODELVIEW);
}

void ObjectOpenGL::Draw_Frame()
{
    glDisable(GL_LIGHTING);
    glLineWidth(10.0);
    glBegin(GL_LINES);
    glColor(Axis_X_Color);
    glVertex3d(0,0,0);
    glVertex3d(0.25, 0, 0);
    glEnd();
    glBegin(GL_LINES);
    glColor(Axis_Y_Color);
    glVertex3d(0,0,0);
    glVertex3d(0, 0.25, 0);
    glEnd();
    glBegin(GL_LINES);
    glColor(Axis_Z_Color);
    glVertex3d(0,0,0);
    glVertex3d(0, 0, 0.25);
    glEnd();
    glEnable(GL_LIGHTING);
}

void ObjectOpenGL::FrontView()
{
    SetXRotation(0);
    SetYRotation(0);
    SetZRotation(0);
    Zoom=1;
    dx=dy=0;
}

```

```

void ObjectOpenGL::RearView()
{
    SetXRotation(0);
    SetYRotation(180*16);
    SetZRotation(0);
    Zoom=1;
    dx=dy=0;
}

void ObjectOpenGL::LeftView()
{
    SetXRotation(0);
    SetYRotation(90*16);
    SetZRotation(0);
    Zoom=1;
    dx=dy=0;
}

void ObjectOpenGL::RightView()
{
    SetXRotation(0);
    SetYRotation(-90*16);
    SetZRotation(0);
    Zoom=1;
    dx=dy=0;
}

void ObjectOpenGL::TopView()
{
    SetXRotation(-90*16);
    SetYRotation(0);
    SetZRotation(0);
    Zoom=1;
    dx=dy=0;
}

void ObjectOpenGL::BottomView()
{
    SetXRotation(90*16);
    SetYRotation(0);
    SetZRotation(0);
    Zoom=1;
    dx=dy=0;
}

void ObjectOpenGL::IsometricView()

```

```

{
    SetXRotation(-45*16);
    SetYRotation(-45*16);
    SetZRotation(0);
    Zoom=1;
    dx=dy=0;
}

void ObjectOpenGL::resizeGL(int width, int height)
{
    WindowSize=QSize(width,height);
}

void ObjectOpenGL::NormalizeAngle(int *angle)
{
    while (*angle < 0)
        *angle += 360 * 16;
    while (*angle >= 360 * 16)
        *angle -= 360 * 16;
}

void ObjectOpenGL::SetXRotation(int angle)
{
    NormalizeAngle(&angle);
    if (angle != xRot)
    {
        xRot = angle;
        emit xRotationChanged(angle);
        updateGL();
    }
}

void ObjectOpenGL::SetYRotation(int angle)
{
    NormalizeAngle(&angle);
    if (angle != yRot)
    {
        yRot = angle;
        emit yRotationChanged(angle);
        updateGL();
    }
}

void ObjectOpenGL::SetZRotation(int angle)
{
    NormalizeAngle(&angle);

```

```

    if (angle != zRot)
    {
        zRot = angle;
        emit zRotationChanged(angle);
        updateGL();
    }
}

void ObjectOpenGL::mousePressEvent(QMouseEvent *event) {
    if(event->buttons() == Qt::RightButton)
        LastPos = event->pos();
    if(event->buttons() == Qt::LeftButton)
        LastPos = event->pos();
}

void ObjectOpenGL::wheelEvent(QWheelEvent *event)
{
    if(event->delta() < 0)
        Zoom /= 1 - (event->delta() / 120.0) / 10.0;
    if(event->delta() > 0)
        Zoom *= 1 + (event->delta() / 120.0) / 10.0;
}

void ObjectOpenGL::mouseMoveEvent(QMouseEvent *event)
{
    if(event->buttons() == Qt::LeftButton)
    {
        dx += -(event->x() - LastPos.x()) / (double)WindowSize.width();
        dy += -(event->y() - LastPos.y()) / (double)WindowSize.height();
        LastPos = event->pos();
    }

    if(event->buttons() == Qt::RightButton)
    {
        int dx_mouse = event->x() - LastPos.x();
        int dy_mouse = event->y() - LastPos.y();
        SetXRotation(xRot - 4 * dy_mouse);
        SetYRotation(yRot + 4 * dx_mouse);
        LastPos = event->pos();
    }
}

```



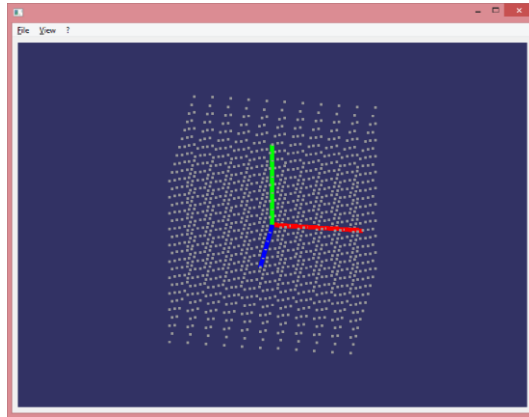


Рисунок 4.4 – Результат роботи програми (розщеплення)

#### 4.4 Завдання на виконання лабораторної роботи

Моделювання фізичних явищ – плавлення льоду, кипіння води, випаровування води, замерзання води, вивітрювання гірських порід, м'яч, який стрибає (різного матеріалу), м'яч, що котиться, перетин сфероїда по поверхні, зіткнення двох сферичних тіл, дзига, яка крутиться і не падає, накладення кольорів, вибух, маятник, повітряна куля, електричні заряди, переломлення на кордонах середовищ, дощ, сніг, туман, краплі на воді, мокра поверхня, м'яч, який стрибає з згасанням, об'єкт (різних матеріалів) кинутий в воду.

Гравітаційна модель сферичних тіл в безповітряному просторі (постріл з гармати, політ ракети)

Хімічні: горіння дерева – виходить зола, білок в спирті – відбувається денатурація, окислення металів (іржа), перетравлення їжі, бродіння.

#### 4.5 Зміст звіту

У звіті мають бути відображені такі питання: мета роботи; завдання до лабораторної роботи; файли програми; результати роботи програми; висновок.

#### 4.6 Контрольні питання

- 4.6.1 Що виконує команда glEnable(GL\_FOG)?
- 4.6.2 Якими бувають режими туману?
- 4.6.3 Для чого у OpenGL використовуються команди glEnable/gldisable?

## 5. РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Програмування комп'ютерної графіки та мультимедійні засоби : навч. посіб. / Л. М. Журавчак, О. М. Левченко. – Львів : Львівська політехніка, 2019. – 276 с. ([http://pdf.lib.vntu.edu.ua/books/2020/Zhuravchak\\_2019\\_276.pdf](http://pdf.lib.vntu.edu.ua/books/2020/Zhuravchak_2019_276.pdf))
2. Комп'ютерна графіка: конспект лекцій для студентів усіх форм навчання спеціальностей 122 «Комп'ютерні науки» та 123 «Комп'ютерна інженерія» з курсу «Комп'ютерна графіка» / Укладач: Скиба О.П. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2019. – 88 с. (<https://elartu.tntu.edu.ua/handle/lib/27541>).
3. Комп'ютерна графіка: лабораторний практикум / Я. Г. Скорюкова, О. В. Слободянюк, М. С. Гречанюк. – Вінниця: ВНТУ, 2020. – 93 с. ([http://pdf.lib.vntu.edu.ua/books/IRVC/Skorukova\\_2020\\_93.pdf](http://pdf.lib.vntu.edu.ua/books/IRVC/Skorukova_2020_93.pdf))
4. Комп'ютерна графіка: навчальний посібник / Є.В. Бородавка, О.О. Терентьев. Київ: КНУБА, 2023. 132 с. ([https://org2.knuba.edu.ua/pluginfile.php/16473/mod\\_resource/content/7/CG\\_Tutorial.pdf](https://org2.knuba.edu.ua/pluginfile.php/16473/mod_resource/content/7/CG_Tutorial.pdf))
5. М. Пічугін, І. Канкін, В. Воротніков Комп'ютерна графіка. Навчальний посібник, К.: ЦНЛ, 2019. – 346 с.
6. Г. Брюханова Комп'ютерні дизайн-технології. Навчальний посібник, К.: ЦНЛ, 2019. – 180 с.