МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА КРОСПЛАТФОРМНОГО ЗАСТОСУНКУ ВІЗУАЛІЗАЦІЇ ТРИВИМІРНИХ ЧИСЛОВИХ ДАНИХ»**

Виконав(ла): студент(ка) 2 курсу, групи 8.1219

спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

освітньої програми інженерія програмного забезпечення
(назва освітньої програми)

Кучменко І.А.
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н., Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент зав. кафедри фундаментальної маттемтики, доцент, д.т.н., Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії,к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

« ____ » _____ 2020 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТА

Кучменко Івана Андрійовича
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка кросплатформного застосунку візуалізації тривимірних числових даних

керівник роботи Гоменюк Сергій Іванович д.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » травня 2020 року № 576-с

2. Строк подання студентом роботи 14.12.2020

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Візуалізація даних за допомогою OpenGL

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	01.08.2020	виконано
2.	Збір вихідних даних.	01.09.2020	виконано
3.	Обробка методичних та теоретичних	01.10.2020	виконано
	джерел.		
4.	Розробка першого і другого розділу.	01.11.2020	виконано
5.	Розробка третього розділу.	01.12.2020	виконано
6.	Оформлення і нормоконтроль	10.12.2020	виконано
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	20.12.2020	виконано

Студент _____ І. А. Кучмено
(підпис) (ініціали та прізвище)

Керівник роботи _____ С. І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О. В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка кросплатформного застосунку візуалізації тривимірних числових даних»: 42 с., 11 рис., 12 джерел, 3 додатків.

ВІЗУАЛІЗАЦІЯ, C++, OPENGL, QT, QT CREATOR

Об’єкт дослідження – числові дані.

Предмет дослідження – візуалізація результатів числових розрахунків.

Мета роботи: розробити алгоритм візуалізації використовуючи мову програмування C++ та бібліотеку OpenGL.

Метод дослідження – аналітичний.

У кваліфікаційній роботі розглядаються методи візуалізації чисельних розрахунків. Розглянуто методи візуалізації, системи координат, методи перетворення координат, методи проектування тривимірних образів на екран, методи видалення невидимих ліній. Також для практичної частини розглянута мова програмування C++ та основні бібліотеки для візуалізації, такі як OpenGL та Qt.

На основі цієї інформації була розроблена програма для візуалізації чисельних даних із застосуванням мови програмування C++ та бібліотек OpenGL і Qt.

SUMMARY

Magister's qualifying paper « Development of the cross-platform application for visualization of three-dimensional numerical data»: 42 pages, 11 figures, 12 references, 3 supplements.

VIZUALISATION, C++, OPENGL, QT, QT CREATOR

The object of research is numerical data.

The subject of research is the visualization of the results of numerical calculations.

The aim of the study is developing a rendering algorithm using the C++ programming language and the OpenGL library.

The method of research is analytical.

In the qualifying work are considered methods of visualization of numerical calculations. The methods of visualization, coordinate systems, methods of transformation of coordinates, methods of designing three-dimensional images on the screen, methods of removing invisible lines are considered. Also, for practical part, the C++ programming language and the main visualization libraries such as OpenGL and Qt are considered.

Based on this information, a program for visualizing numerical data using the C++ language and the OpenGL and Qt libraries was developed.

ЗМІСТ

ВСТУП

У кваліфікаційній роботі будуть отримані результати, які у сукупності є розв’язанням задачі візуалізації великих масивів числової інформації, яка отримується або в результаті застосування певних чисельних методів (наприклад, методу скінченних елементів), або при зберіганні інформації з різноманітних сенсорів чи датчиків (наприклад, температури або швидкості вітру).

У роботі було виконано аналіз стану проблеми, зокрема огляд сучасних методів візуалізації просторових об’єктів. Було розглянуто системи координат, що використовуються в комп’ютерній графіці, а також методи їх перетворення (масштабування, поворот та зрушення).

Важливою задачею, що зазвичай постає в комп’ютерній графіці, є алгоритмізація перетворення координат тривимірних об’єктів у плоскі екранні. Разом з тим було розглянуто алгоритми візуалізації чисельних розрахунків за допомогою будування кольорових та напівтонових зображень із застосуванням режиму освітлення.

Під час програмної реалізації була застосована мова програмування C++. При описі цього розділу було наведено запропонований алгоритм візуалізації тривимірних геометричних областей і розподіл по ним досліджуваної функції. Разом з тим були описанні основні бібліотеки для візуалізації, такі як OpenGL і Qt.

Для практичної реалізації отриманих результатів було розроблено кросплатформний програмний продукт, який дозволяє візуалізувати дво- і тривимірні моделі.

1 ОГЛЯД МЕТОДІВ ВІЗУАЛІЗАЦІЇ

1.1 Методи візуалізації просторових об’єктів

У більшості випадків, видима поверхня геометричній області, досліджуваної методом скінченних елементів, являє собою кінцеву сукупність плоских непересічних геометричних фігур простої форми (найчастіше трикутників або чотирикутників) (рис. 1.1). Тому, зображення дискретного об’єкта зазвичай зводиться до візуалізації в просторі певної кількості плоских геометричних фігур – граней скінченних елементів, які мають назву граничних елементів.

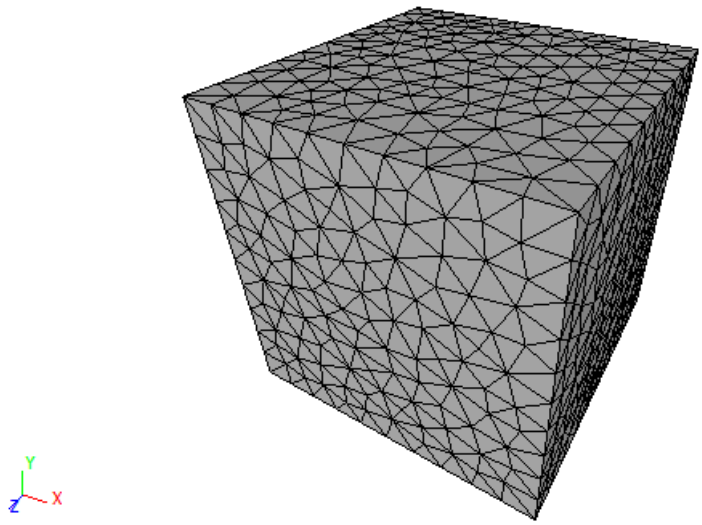


Рисунок 1.1 – Поверхня об’єкта як сукупність непересічних трикутників

У загальному випадку проблема візуалізації результатів чисельних розрахунків зводиться до вирішення двох наступних завдань:

– побудови математичного апарату проектування тривимірної геометричної області на екранну площину;

– зафарбовування вийшла проекції кольорами або відтінками, відповідними розподілу по вихідній геометричній області візуалізіруемой функції.

1.2 Системи координат в комп'ютерній графіці

У комп'ютерній графіці для завдання місця розташування 3D-об'єкта використовується тривимірна ортогональна декартова система координат. При цьому вісь X спрямована зліва на право, вісь Y направлена від низу до верху, а вісь Z спрямована в бік спостерігача. Така система координат називається правосторонньою (див рис. 1.2). Якщо вісь Z спрямована в бік від спостерігача, то така система координат називається лівосторонньої.

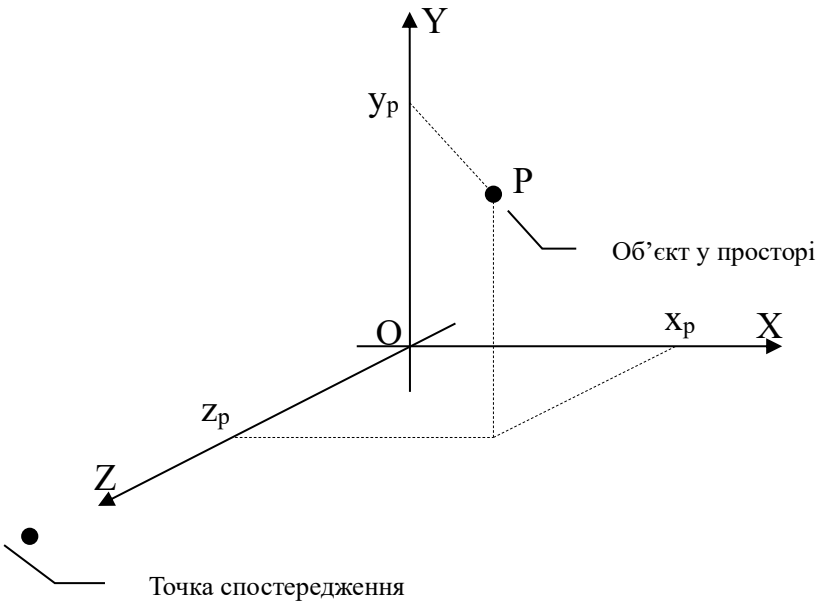


Рисунок 7.2 Правостороння система координат

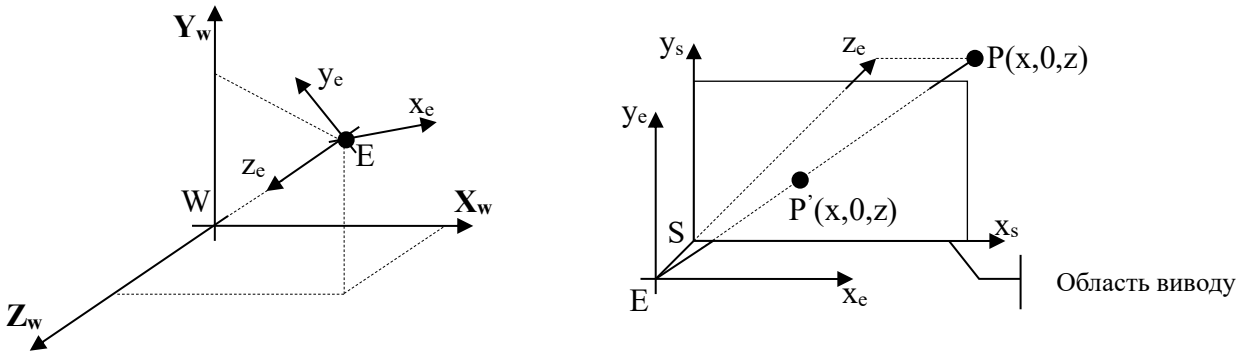
Сучасні графічні стандарти програмування тривимірної графіки, такі як OpenGL [1] і DirectX [2], оперують трьома видами систем координат:

– світової;

– видова;

– екранної.

Світова система координат - це правобічна декартова система координат, центр якої W збігається з положенням спостерігача сцени. Видова система координат - це аналогічна лівостороння система координат, центр якої E збігається з геометричним центром візуалізіруемого об’єкта. Екранна система координат - це плоска декартова система координат, центр якої S збігається з лівим нижнім кутом екрану, а осі направлені природним чином (див рис. 1.3).



Світова та видова системи координат

Видова та екранна системи координат

Рисунок 1.3 – Світова, видова і екранна системи координат

Головна проблема полягає в тому, щоб побудувати математичний апарат перетворення світових координат в екранні таким чином, щоб домогтися побудови на екрані монітора реалістичного образу. Дослідженню даної проблеми присвячена велика кількість робіт, найбільш відомими серед яких є роботи Л. Аммерала, Д. Роджерса та ін.

1.3 Перетворення координат

Базовими маніпуляціями над зображуваними об’єктами в комп’ютерній графіці є перетворення зсуву, масштабування і повороту. З метою уніфікації математичних співвідношень, що реалізують перетворення координат, і зручності їх використання в комп’ютерній графіці використовують однорідні координати, що мають в загальному випадку вид (x_w, y_w, z_w, W), де W – деякий довільний відмінний від нуля множник. Перехід від однорідних координат до декартових здійснюється наступним чином:

$$\begin{aligned} x &= x_w/W, \\ y &= y_w/W, \\ z &= z_w/W. \end{aligned} \tag{1.1}$$

У тривимірному випадку перетворення зсуву можна описати таким співвідношенням:

$$\begin{aligned} x_n &= x + T_x, \\ y_n &= y + T_y, \\ z_n &= z + T_z, \end{aligned} \tag{1.2}$$

або в матричної формі:

$$P_n = P \cdot T, \tag{1.3}$$

де (x, y, z) – поточні координати вершини в просторі;

$T_x,\ T_y,\ T_z$ – величини зсуву відповідно по осях X, Y і Z;

$P = \{x,\ y,\ z\}$ – вектор-рядок вихідних координат;

$P_n = \{x_n,\ y_n,\ z_n\}$ – вектор-рядок перетворених координат;

$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$ – матриця зсуву.

Перетворення масштабування визначається наступним співвідношенням:

$$\begin{aligned} x_n &= x \cdot S_x, \\ y_n &= y \cdot S_y, \\ z_n &= z \cdot S_z, \end{aligned} \tag{1.4}$$

або в матричної формі:

$$P_n = P \cdot S, \tag{1.5}$$

де S_x, S_y та S_z – коефіцієнти масштабування, а $S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ – матриця масштабування.

Перетворення повороту на кути $\varphi_x, \varphi_y, \varphi_z$ щодо осей X, Y і Z відповідно, має вигляд:

$$\begin{aligned} x_n &= x \cdot (1 + \cos\varphi_x + \cos\varphi_z) + y \cdot \sin\varphi_x + z \cdot \sin\varphi_y, \\ y_n &= x \cdot \sin\varphi_z + y \cdot (1 + \cos\varphi_x + \cos\varphi_z) - z \cdot \sin\varphi_x, \\ z_n &= -x \cdot \sin\varphi_y + y \cdot \sin\varphi_x + z \cdot (1 + \cos\varphi_x + \cos\varphi_y), \end{aligned} \tag{1.6}$$

або в матричної формі:

$$P_n = P \cdot R, \tag{1.7}$$

де $R = \begin{bmatrix} 1+\cos\varphi_x+\cos\varphi_z & \sin\varphi_z & -\sin\varphi_y & 0 \\ -\sin\varphi_z & 1+\cos\varphi_x+\cos\varphi_z & \sin\varphi_x & 0 \\ \sin\varphi_y & -\sin\varphi_x & 1+\cos\varphi_x+\cos\varphi_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ –

матриця повороту.

1.4 Методи проектування тривимірних образів на плоский екран

При програмуванні візуалізації тривимірних комп'ютерних образів в загальному вигляді необхідно задати обсяг видимості в тривимірному просторі, вид проектування на екранну площину і обсяг виведення на екран. При цьому тривимірні об'єкти відсікаються по межі обсягу видимості і після цього проектуються на екран. Механізм перетворення світових координат в екранні можна зобразити схемою, наведеною на рис. 1.4.

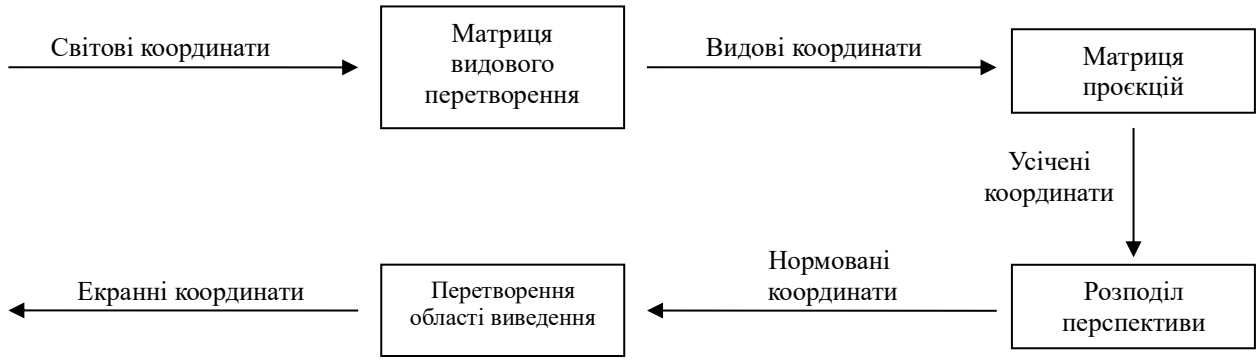


Рисунок 1.4 – Процес перетворення світових в екранні координати

В процесі перетворення координат при перетворенні зі світових в екранні координати три вимірювання відображаються в два, що незмінно призводить до появи спотворень. Від вибору типу проектування залежить не тільки достовірність зображеного образу, але і адекватність його сприйняття.

Проектування виконується за допомогою проектують променів, які називаються проекторами, які йдуть з центру проекції через всі точки об’єкта, що зображується, до екранної площини. По розташуванню центру проекції щодо екранної площини розрізняють центральну і паралельну проекції.

У разі паралельної проекції її центр розташовується на нескінченному відстані від екранної площині. Таким чином, проектори являють собою паралельні пучки прямих. Вид даної проекції залежить від взаємного розташування осей координат, екранної площини і проекторів. У комп’ютерній графіці найчастіше застосовуються ортогональна і аксонометрична проекції.

При ортогональному проектуванні проектори паралельні головної осі спостереження і ортогональні екранної площині.

При аксонометричному проектуванні можливі два варіанти: проектори перпендикулярні екранної площини, яка в свою чергу розташована під деяким кутом до головної осі. Такий тип проектування називається прямокутною аксонометричною проекцією. Якщо ж, навпаки, користуватися проектором буде неможливо ортогональні екранної площині, а головна вісь - ортогональна, то такий тип проектування називається косокутній аксонометричною проекцією.

Зображення, отримане при використанні таких видів проектування, дозволяє за рахунок відсутності перспективи точно відобразити форму і розмір тривимірного об’єкту, однак, воно не досить реалістичне.

При центральному проектуванні відстань до центру проекції звичайно, тому проектори - це пучок прямих, що виходять з центру. Для завдання такого виду проектування необхідно визначити місце розташування центру проекції і картинній площині. Центральне проектування за рахунок появи перспективи призводить до спотворень в формі і розмірах об’єкта, однак, такий об’єкт на екрані виглядає більш наочно (рис. 1.5).

Паралельне проектування можна описати таким матричним співвідношенням:

$$P_n = P \cdot M_p, \tag{1.8}$$

де $M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & z_0 & 1 \end{bmatrix}$ – матриця паралельного проектування;

$z = z_0$ – рівняння, що задає площину проектування.

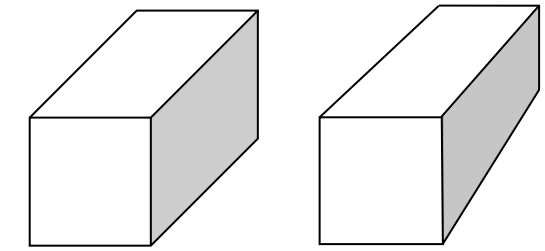


Рисунок 1.5 – Паралельне і центральне проектування

Косокутне проектування в площину $z = 0$ можна описати таким матричним співвідношенням:

$$P_n = P \cdot M_a, \tag{1.9}$$

де $M_a = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \cos\alpha & \sin\alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ – матриця косокутного проектування.

Рівняння для центральної проекції с центром на початку координат можна описати таким співвідношенням:

$$P_n = P \cdot M_c, \tag{1.10}$$

де $M_c = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{d} \\ 0 & 0 & 0 & 0 \end{bmatrix}$ – матриця косокутного проектування.

d – відстань від початку координат до екранної площини, ортогональної осі Z.

1.5 Методи видалення невидимих ліній

Головною проблемою, що виникає при програмуванні тривимірної графіки, є видалення невидимих ліній та поверхонь. Суть її полягає в необхідності визначення і виключення зі списку зображуваних елементів геометрії області ті з них, які в даний момент часу закриваються іншими частинами об’єкта (тобто їх не видно спостерігачеві).

Алгоритми видалення невидимих частин зображуваної сцени прийнято класифікувати наступним чином:

– за вибором частин, що видаляються (точок, ліній, граней, поверхонь, об’ємів);

– по порядку обробки елементів сцени (видалення в заданому або довільному порядку);

– по системі координат (порівняння кожного елемента об’єкта, наприклад, границі або пікселя з іншими).

Найбільш поширеними методами видалення невидимих ліній і поверхонь на практиці є:

– алгоритм Робертса;

– алгоритм Z-буфера;

– порядковий алгоритм Z-буфера;

– алгоритм розбиття області Варнока;

– порядковий алгоритм Уоткінса;

– алгоритм трасування променів.

Більшість з вищенаведених алгоритмів орієнтовані на застосування для різних окремих випадків. Найбільш ефективним, простим і часто використовуваним універсальним методом видалення невидимих ліній є алгоритм Z-буфера.

Ідея даного алгоритму полягає в тому, що на відміну від звичайного буфера кадру зображення, що зберігає кольори всіх утворюють сцену пікселів, використовується додатковий буфер, що містить Z-координату кожної точки образу (звідси і назва алгоритму). Дана координата фактично є глибиною точки. При занесенні кольору чергового пікселя в буфер кадру значення його Z-координати порівнюється з Z-координатою пікселя, який вже знаходиться в буфері. Якщо Z-координата нового пікселя більше, ніж координата старого, тобто він ближче до спостерігача, то атрибути нового пікселя і його Z-координата заносяться в буфер, якщо немає, то ні чого не робиться.

Даний алгоритм є досить простим в реалізації і не залежить від порядку малювання сцени. Головним недоліком алгоритму Z-буфера є необхідність використання значних обсягів оперативної пам’яті комп’ютера, що особливо помітно при візуалізації швидкозмінних динамічних сцен. Також до недоліків даного алгоритму можна віднести появу так званого сходового ефекту і труднощі з малюванням ліній на поверхні об’єктів.

Як вже зазначалося вище, зображення скінчено-елементного тривимірного тіла зводиться до зображення деякої кінцевої сукупності непересічних плоских полігональних областей простої форми, які є поверхневими гранями скінченних елементів, що утворюють тіло. При цьому форма полігонів залежить від типу використовуваного елемента, наприклад, в разі тетраедра - трикутник, призми – чотирикутник.

2 АЛГОРИТМИ ВІЗУАЛІЗАЦІЇ ЧИСЕЛЬНИХ РЕЗУЛЬТАТІВ РОЗРАХУНКУ

2.1 Побудова кольорових зображень

Ідея алгоритму візуалізації розподілу певної числової величини по тривимірному об’єкту, утвореному певною сукупністю граничних елементів полягає в розбитті кожного граничного на певну кількість трикутників, вершинам яких відповідають однакові значення функції, що аналізується. Після чого ці однокольорові трикутники зображуються і утворюють певну кольорову картину розподілу досліджуваної величини по об’єкту (рис. 2.1).

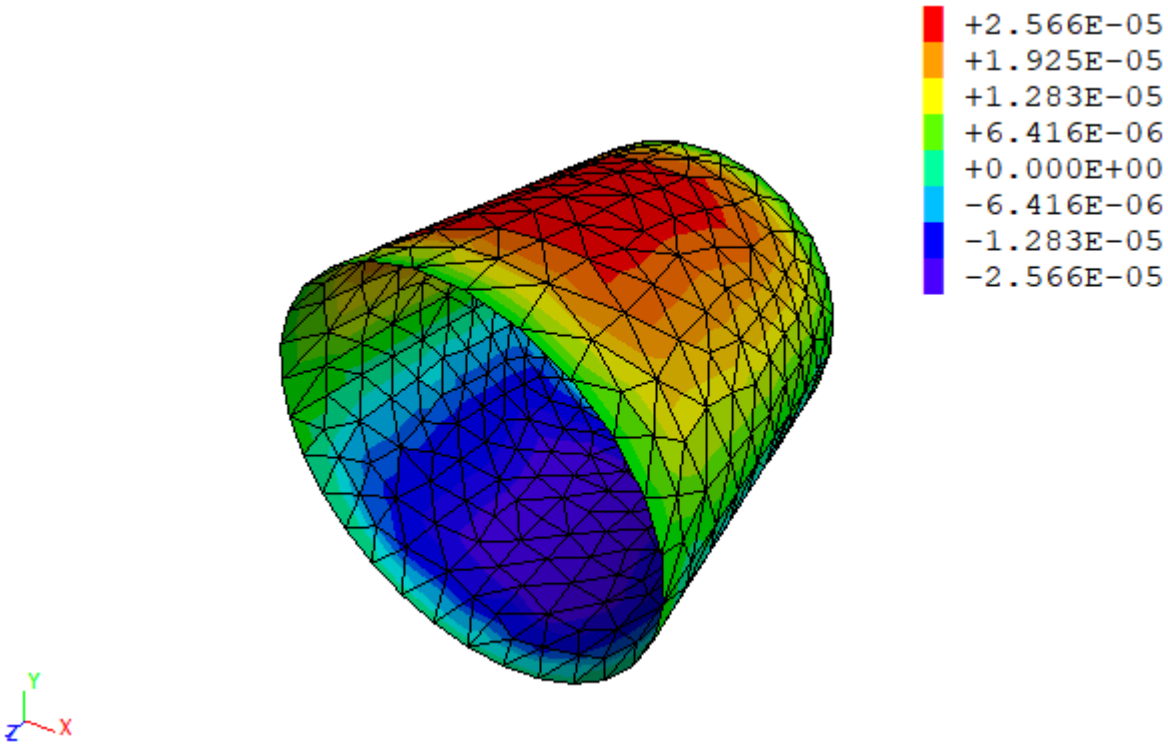


Рисунок 2.1 – Приклад візуалізації розподілу числової функції по тривимірній області

Для програмної реалізації такого алгоритму в першу чергу потрібно розробити процедуру візуалізації трикутника, в вершинах якого задано значення досліджуваної числової величини.

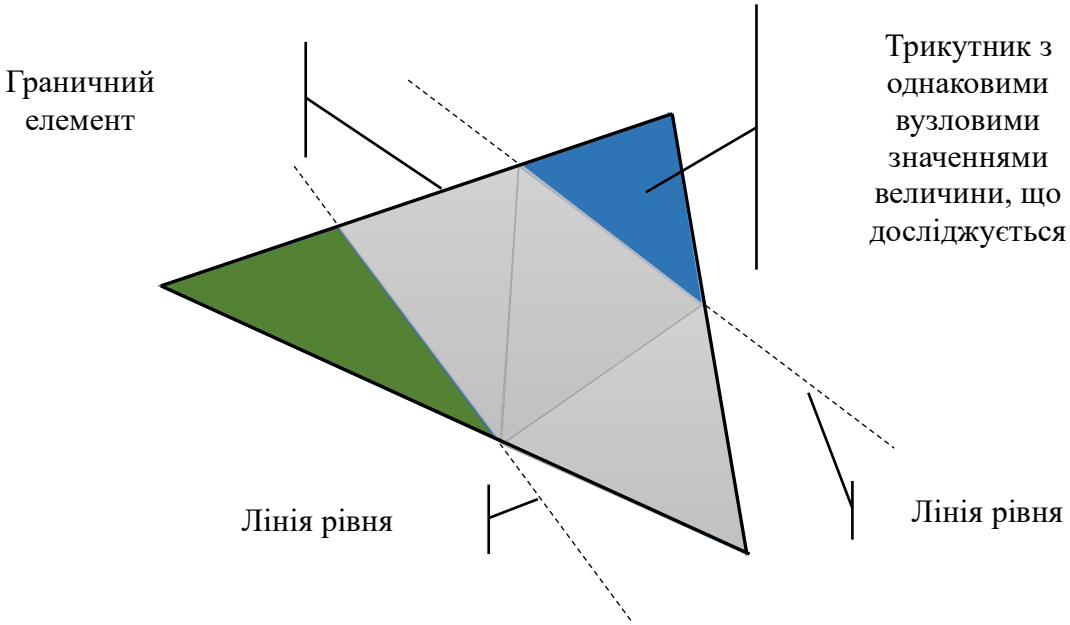


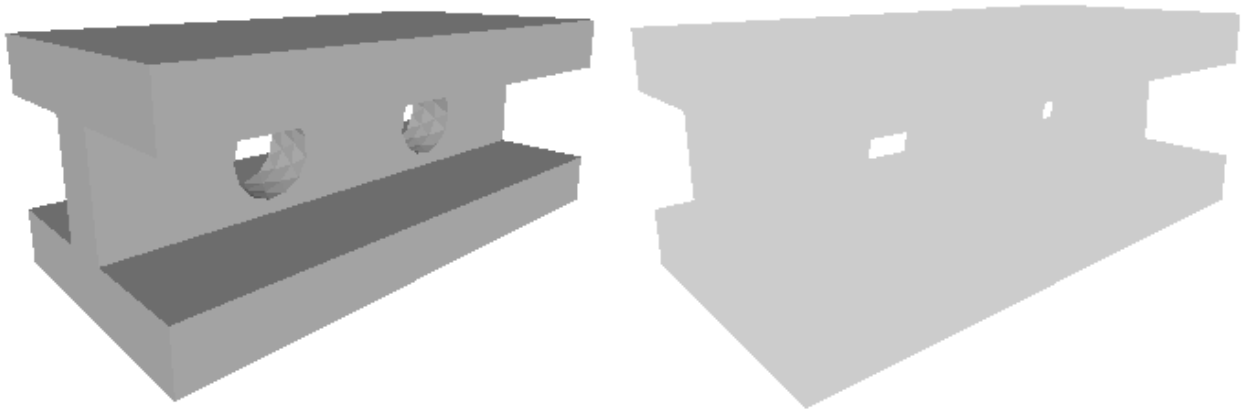
Рисунок 2.2 – Схема розбиття граничного елемента на трикутники одного кольору

Приклад роботи такого алгоритму наведено на рис. 2.1. Він послідовно оброблює всі граничні сегменти, що утворюють поверхню вихідного тривимірного об’єкта, і розбиває кожний поточний граничний сегмент на певну множину трикутників, які мають один колір (тобто, значення досліджуваної величини в їх вершинах однакові або належать до одного кольорового діапазону).

Програмна реалізація вищеописаної процедури на мові C++ з використанням графічного стандарту OpenGL та кросплатформної бібліотеки Qt наведена в Додатку А.

2.2 Використання освітлення тривимірних об’єктів

Для підвищення наочності візуалізації результатів розрахунку є застосування освітлення об’єкту (рис. 2.3).



а) б)

Рисунок 2.3 – Візуалізація тривимірного об’єкта а) із застосуванням освітлення і б) без застосування освітлення

Для зображення освітленій сцени в комп’ютерній графіці найчастіше застосовується моделі дифузного і дзеркального відображення об’єктом світла від заданого одного або декількох точкових джерел освітлення.

Дифузійне відбиття світла точкового джерела від ідеального розсіювача визначається законом Ламберта, згідно з яким падаюче світло розсіюється в усі боки з однаковою інтенсивністю. В цьому випадку освітленість точки пропорційна частці її площі, видимої від джерела і описується наступним співвідношенням:

$$I_r = I_p \cdot P_d \cdot \cos\phi, \tag{2.5}$$

де I_r – інтенсивність відбитого світла;

I_p – інтенсивність точкового джерела;

$P_d \in [0,1]$ – коефіцієнт дифузного відбиття, що залежить від властивостей матеріалу відбиває світло, і кольору джерела освітлення;

$\phi \in [0, {}^{\pi}/_{2}]$ – кут, утворений напрямом світла і нормаллю поверхні.

Для підвищення реалістичності сприйняття малюнка в комп'ютерній графіці також враховується і наявність розсіяного світла, який описується за допомогою коефіцієнта розсіювання:

$$I = I_r \cdot P_r + I_p \cdot P_d \cdot \cos\phi, \tag{2.6}$$

де I_r – інтенсивність розсіяного світла;

$P_r \in [0,1]$ – коефіцієнт дифузного відбиття розсіяного світла.

Моделювання загасання світла з відстанню від джерела описується наступним співвідношенням:

$$I = I_r \cdot P_r + \frac{I_p \cdot P_d \cdot \cos\phi}{d + K}, \tag{2.7}$$

де d – відстань від центру проекції до об'єкта;

K – довільна константа, що задає ступінь загасання світла.

При використанні паралельного проектування, облік відстані забезпечується тим, що найближчий до спостерігача об'єкт висвітлюється з максимальною інтенсивністю, а всі розташовані далі – з меншою. При цьому в якості відстані d використовується відстань до найближчого до точки спостереження об'єкта.

Відбитий від ідеального дзеркала світло видно тільки в тому випадку, якщо кут, утворений напрямками спостереження і відображення, дорівнює нулю. Для неідеальних відбивають використовується модель Фонга:

$$I_s = I_p \cdot W(\lambda, \phi) \cdot \cos^n \alpha, \tag{2.8}$$

де $W(\lambda, \phi)$ – крива відображення, що залежить від довжини хвилі світла джерела і кута падіння ϕ;

$\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ – кут між напрямками спостереження і відображення;

$n \in [1{,}200]$ – показник ступеня, що задає спадання інтенсивності при зміні кута.

Для простоти на практиці зазвичай $W(\lambda, \phi)$ замінюють деякою константою K_s, підібраною таким чином, щоб побудована картина суб'єктивно сприймалася реалістичною.

Таким чином, використовувана в комп'ютерній графіці сумарна модель освітлення має вигляд:

$$I = I_r \cdot P_r + \frac{I_p}{d + K} \cdot (P_d \cdot \cos\phi + W(\lambda, \phi) \cdot \cos^n \alpha), \tag{2.9}$$

або:

$$I = I_r \cdot P_r + \frac{I_p}{d + K} \cdot (P_d \cdot \cos\phi + K_s \cdot \cos^n \alpha), \tag{2.10}$$

При використанні нормованих векторів напрямку падіння **L**, нормалізує обмін речовин **N**, відображення **R** і спостереження **V**, модель освітлення для одного джерела описується наступним чином:

$$I = I_r \cdot P_r + \frac{I_p}{d + K} \cdot (P_d \cdot L \cdot N + K_s \cdot (R \cdot V)^n), \tag{2.11}$$

При візуалізації плоскої полігональної фігури (наприклад, границі скінченного елемента), якщо джерело світла знаходиться на нескінченності, то $\mathbf{L} \cdot \mathbf{N}$ одно константі, а $\mathbf{R} \cdot \mathbf{V}$ змінюється в межах цієї фігури. Таким чином, візуалізація освітленого скінченно-елементного об'єкта вимагає повузлової побудови образу, що призводить до необхідності використання оригінального алгоритму Z-буфера. Однак, можливості сучасних комп'ютерів з використанням сучасних графічних стандартів OpenGL або DirectX дозволяють ефективно візуалізувати освітлені об'єкти.

Для завдання форми поверхні в таких бібліотеках потрібно визначати вектору нормалей. У цьому випадку в якості компонент цих векторів можна використовувати відповідні компоненти вектору переміщень, деформацій або напружень. Якщо до координат вузлів додати відповідним чином нормовані значення досліджуваної функції і зобразити отриману гофровану поверхню об’єкта освітленій боковим джерелом світла, то отримана картина також дозволить судити про розподіл досліджуваної числової величини по області.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Кросплатформна бібліотека Qt

Qt – це кросплатформна бібліотека, яка була написана на мові програмування C++ для розробки висококласного програмного забезпечення графічним інтерфейсом користувача (GUI – Graphical User Interface), яке працює в середовищі операційних систем сімейства Windows, Linux і MacOS.

Бібліотека Qt була розроблена в 1996 році і з тих пір активно розвивається. Її використовують такі гіганти IT-індустрії, як: Autodesk, Google, Microsoft, Nokia, Panasonic, Siemens, Walt Disney Animation Studios та багато інших.

Крім створення графічного GUI Qt дозволяє:

– створювати багатопотокові програми;

– розробляти мережеві програми;

– розробляти програми з 2D- і 3D-графікою (в тому числі і з використанням графічного стандарту OpenGL);

– програмувати бази даних (в тому числі з використанням мови запитів SQL);

– працювати з файлами у форматі XML та Json;

– розробляти веб-додатки;

– працювати з мультимедіа;

– взаємодіяти з ActiveX і COM (для розробників під операційну систему Windows);

– і багато іншого.

Найпростішу програму з GUI із застосуванням можна, наприклад, написати таким чином:

```
#include <QtWidgets>
```

```
int main(int argc, char **argv)
{
    QApplication app(argc, argv);
    QMainWindow mw;
    mw.setWindowTitle("Hello world!");
    mw.show();
    return app.exec();
}
```

Для компіляції цього вихідного тексту потрібно створити спеціальний файл налаштувань компіляція:

```
QT += gui widgets
CONFIG += c++17
SOURCES += \
        main.cpp
```

В результаті компіляції цієї програми буде створено додаток, який складається з одного пустого вікна (рис. 3.1).

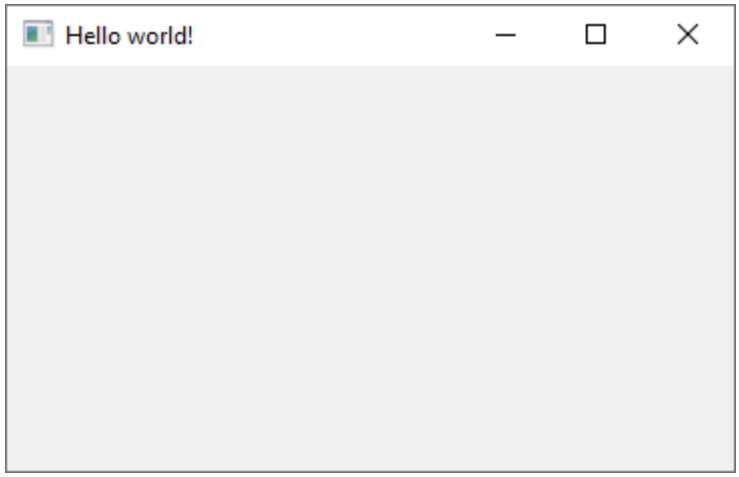


Рисунок 3.1 – Проста програма з GUI

Бібліотека Qt складається з понад ніж 500 класів, що охоплюють більшу частину функціонала сучасних операційних систем. Для зручності програмістів вони об'єднані в модулі. Основні серед них наведені в табл. 3.1.

Таблиця 3.1 – Основні бібліотеки Qt

Модуль	**Обозначение**	**Назначение**
QtCore	core	Набір базових класів, не пов'язаних з GUI
QtGui	gui	Набір базових класів для програмування GUI
QtWidgets	widgets	Модуль, що містить графічні віджети для GUI
QtNetwork	network	Набір класів для програмування мережі
QtOpenGL	opengl	Модуль для програмування графіки OpenGL
QtSql	sql	Модуль для програмування баз даних
QtXml	xml	Набір класів для роботи з XML
QtMultimedia	mulimedia	Класи для роботи з мультимедіа
QtWebKit	webkit	Модуль для створення веб-додатків
QtPrintSupport	printsupport	Модуль для роботи з принтером

Більшість класів бібліотеки Qt унаслідуванні від базового суперкласу QObject, який підтримує такі основні можливості:

– сигнали і слоти (signal / slot);
– таймер;
– механізми об’єднання об’єктів в ієрархії;
– події та механізми їх фільтрації;
– метаоб’ектную інформацію;

– приведення типів;

– властивості.

Сигнали і слоти – це механізм обміну інформацією про виникнення різноманітних подій, що генерується об’єктами, а також способи їх обробки, реалізовані в Qt.

Використання таймера, визначеного в QObject, дозволяє класам-нащадкам використовувати його в своїх цілях, економлячи при цьому час на розробку.

Механізми об’єднання об’єктів в ієрархії дозволяють скоротити часові витрати на розробку програм, не піклуючись при цьому про звільнення пам’яті для створюваних об’єктів, оскільки батьківські об’єкти автоматично очистять пам’ять, займану своїми дочірніми об’єктами.

Механізм фільтрації подій дозволяє здійснювати їх перехоплення, завдяки чому можна змінити реакцію об'єктів на події, що відбуваються без зміни вихідного коду класу.

Метаоб’єктна інформація містить дані про ієрархії класів, а також дозволяє дізнатися ім’я класу.

Приведення типів дозволяє на базі метаоб’єктной інформації здійснювати перетворення між класами-спадкоємцями від QObject.

Властивості - спеціальні поля, для доступу до яких в класі повинні існувати спеціальні методи читання. Властивості широко застосовуються в візуальному середовищі розробки GUI – Qt Designer.

Класичним підходом до програмної реалізації GUI є використання функцій зворотного виклику (callback function), які викликаються для обробки дій користувача з елементами графічного інтерфейсу. Такий підхід за своєю суттю не є об'єктно-орієнтованим і його використання ускладнює аналіз вихідного коду програми. В бібліотеці Qt для обміну повідомленнями між об’єктами і їх обробки використовується повністю об’єктно-орієнтована механізм сигналів та слотів. За його допомоги можна з’єднувати незв’язані спочатку один з одним об’єкти.

Механізм сигналів і слотів – основний концепт програмування з використанням Qt. Кожен похідний від QObject клас може як приймати, так і

відправляти сигнали. Наприклад, таймер генерує і відправляє повідомлення (сигнал) про те, що час оновився. Призначений для користувача віджет, отримавши це повідомлення, викликає відповідний метод (слот) для його обробки (поновлення свого вмісту).

Сигнали в Qt – це спеціальні методи класів, які можуть здійснювати відправку повідомлень. Наприклад:

```
class TDrawMesh : public QGLWidget
{
    Q_OBJECT          // Директива метаоб’єктного компілятору
    // …
public:
    void rotateX(int angle)
    {
        emit xRotate(angle);       // Відправка сигналу
    }
    // …
signals:
    void xRotation(int);        // Сигнал, що генерується при обертанні
                                // навколо осі X
    void yRotation(int);  // … Y
    void zRotation(int);  // … Z
};
```

Слоти в Qt – це спеціальні методи, які автоматично викликаються при отриманні сигналу (слот при цьому повинен бути попередньо з'єднаний з сигналом). Наприклад:

```
class TGLFunction : public TGLMesh
{
```

```
    Q_OBJECT
    // …
public slots:
    void mouseDoubleClickEvent(QMouseEvent*);
    // …
};
```

З’єднання сигналів і слотів відбувається за допомогою спеціального методу connect () класу QObject.

```
QMetaObject::Connection QObject::connect(const QObject *sender,
                                         const char *signal,
                                         const QObject *receiver,
                                         const char *method,
                                         Qt::ConnectionType type = Qt::AutoConnection);
```

де: sender – покажчик на об'єкт, що відправляє сигнал;
signal – сигнал, з яким здійснюється з'єднання;
receiver – покажчик на об'єкт об'єкт, що містить слот;
method – викликаний слот;
type – режим обробки (за замовчуванням - автоматичний).

Наприклад:

```
QLabel *label = new QLabel();
QScrollBar *scrollBar = new QScrollBar();
QObject::connect(scrollBar, SIGNAL(valueChanged(int)),
                 label, SLOT(setNum(int)));
```

Процедура відключення сигналу від слоту відбувається автоматично при видаленні відповідного об'єкта або за допомогою спеціального методу disconnect() класу QObject.

```
bool Qobject::disconnect(const char *signal = nullptr,
                         const QObject *receiver = nullptr,
                         const char *method = nullptr) const
```

Наприклад:

```
// Відключення всіх сигналів
disconnect(myObject, nullptr, nullptr, nullptr);
```

3.2 Графічна бібліотека OpenGL

Графічний стандарт OpenGL (Open Graphics Library) був прийнятий у 1992 р. провідними IT-компаніями як апаратно-незалежний інтерфейс реалізації дво- і тривимірної графіки, який може бути реалізований на різних комп'ютерних та програмних платформах. За основу для OpenGL була взята бібліотека IRIS GL, яка була розроблена компанією Silicon Graphics Inc.

На сьогодні OpenGL де-факто підтримується більшістю сучасних виробників програмного та апаратного забезпечення. Вона підтримується всіма сучасними операційними системами: Windows, Linux та MacOS.

Сучасна реалізація бібліотеки OpenGL містить понад 400 різноманітних команд як для опису різноманітних об'єктів, так і для управління їх поведінкою при візуалізації. Ці функції можна поділити на наступні категорій:

– опис примітивів (точок, ліній, багатокутників тощо);

– опис джерел світла;

– функції, що задають атрибути графічних об'єктів (як вони будуть виглядати на екрані); атрибутами є колір, характеристики матеріалу, текстури, параметри освітлення тощо;

– функції, що дозволяють задати параметри положення спостерігача та графічної сцени;

– функцій, що виконують геометричні перетворення – поворот, зсув та масштабування.

Основним принципом роботи OpenGL є отримання наборів векторних графічних примітивів у вигляді точок, ліній та багатокутників з наступною математичною обробкою отриманих даних та побудовою растрової картинки на екрані і/або в пам’яті. Векторні трансформації та растеризація виконуються графічним конвеєром (graphics pipeline), який власне являє собою дискретний автомат. Абсолютна більшість команд OpenGL потрапляють в одну з двох груп: або вони додають графічні примітиви на вхід в конвеєр, або конфігурують конвеєр на різне виконання трансформацій.

OpenGL є низькорівневим процедурним API, що змушує програміста диктувати точну послідовність кроків, щоб побудувати результуючу растрову графіку (імперативний підхід). Це є основною відмінністю від дескрипторних підходів, коли вся сцена передається у вигляді структури даних (найчастіше дерева), яке обробляється і будується на екрані. З одного боку, імперативний підхід вимагає від програміста глибокого знання законів тривимірної графіки та математичних моделей, з іншого боку – дає свободу впровадження різних інновацій.

Так як OpenGL це лише специфікація стандартів, користувачеві потрібна реалізація цих стандартів на потрібній йому мові програмування. Бібліотека Qt підтримує роботу з OpenGL. Цей потужних фреймворк дозволяє дуже швидко реалізувати інтерфейс користувача із застосуванням OpenGL завдяки модулю QtOpenGL.

3.3 Програма реалізація візуалізації даних

Для реалізації поставленої задачі із застосуванням бібліотеки Qt і графічного стандарту Qt було реалізовано програмний застосунок «mpp», загальний інтерфейс якого наведено на рис. 3.2.

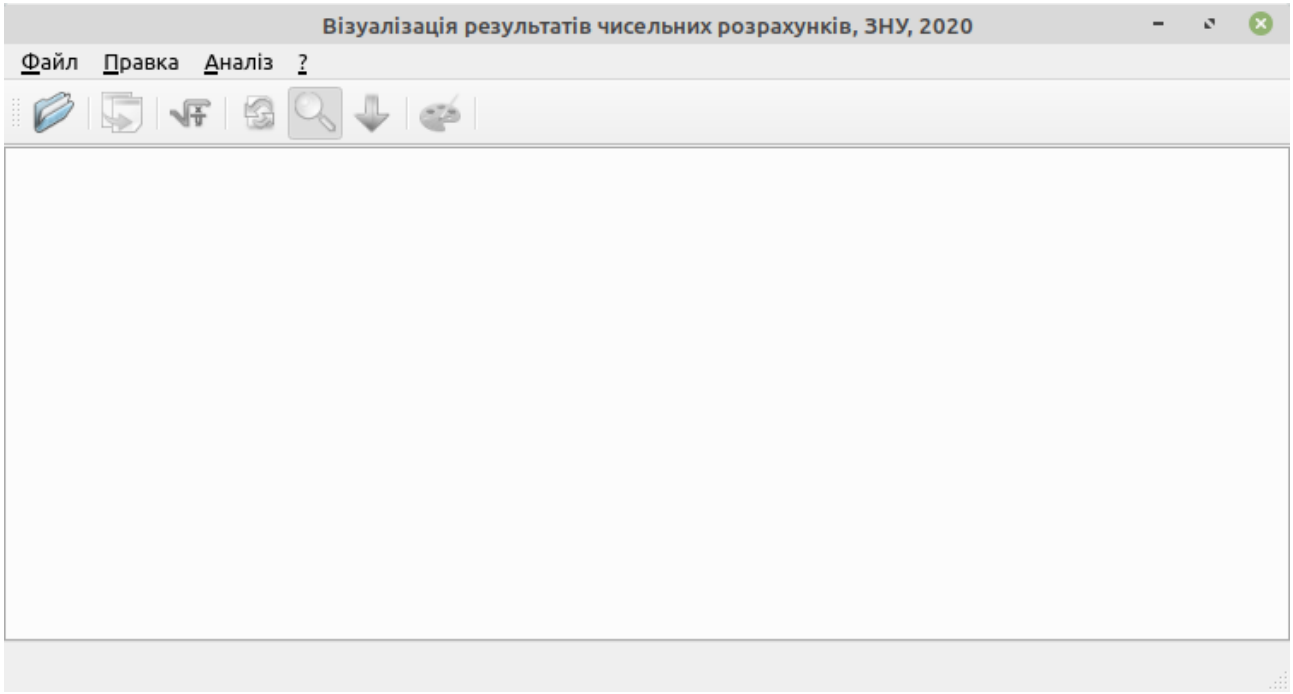


Рисунок 3.1 – Загальний інтерфейс кросплатформного додатку «mpp»

Для завантаження він використовує дані у формати JSON, приклад яких наведено у Додатку Б. візуалізації.

Після завантаження файлу даних застосунок відображає узагальнену інформацію про результати розрахунку (рис. 3.3).

Приклад візуалізації певної функції наведено на рис. 3.4.

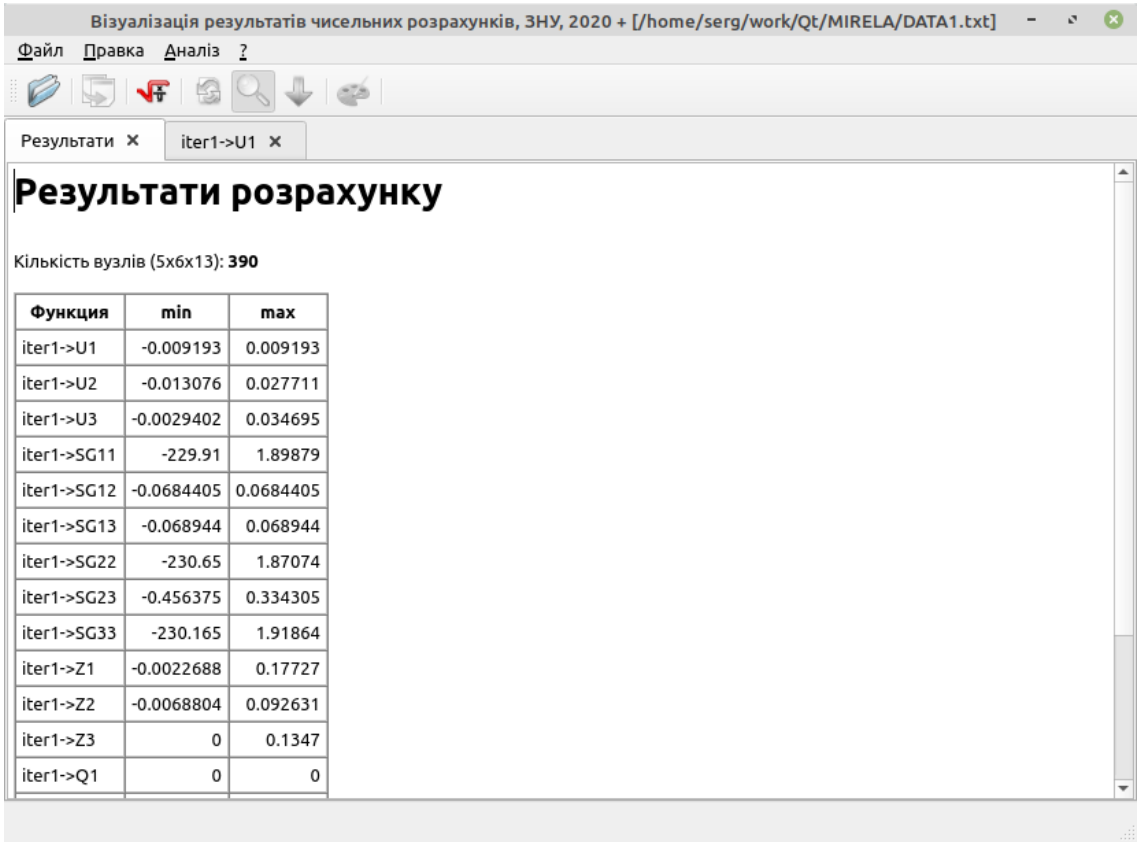


Рисунок 3.3 – Результат завантаження файлу даних

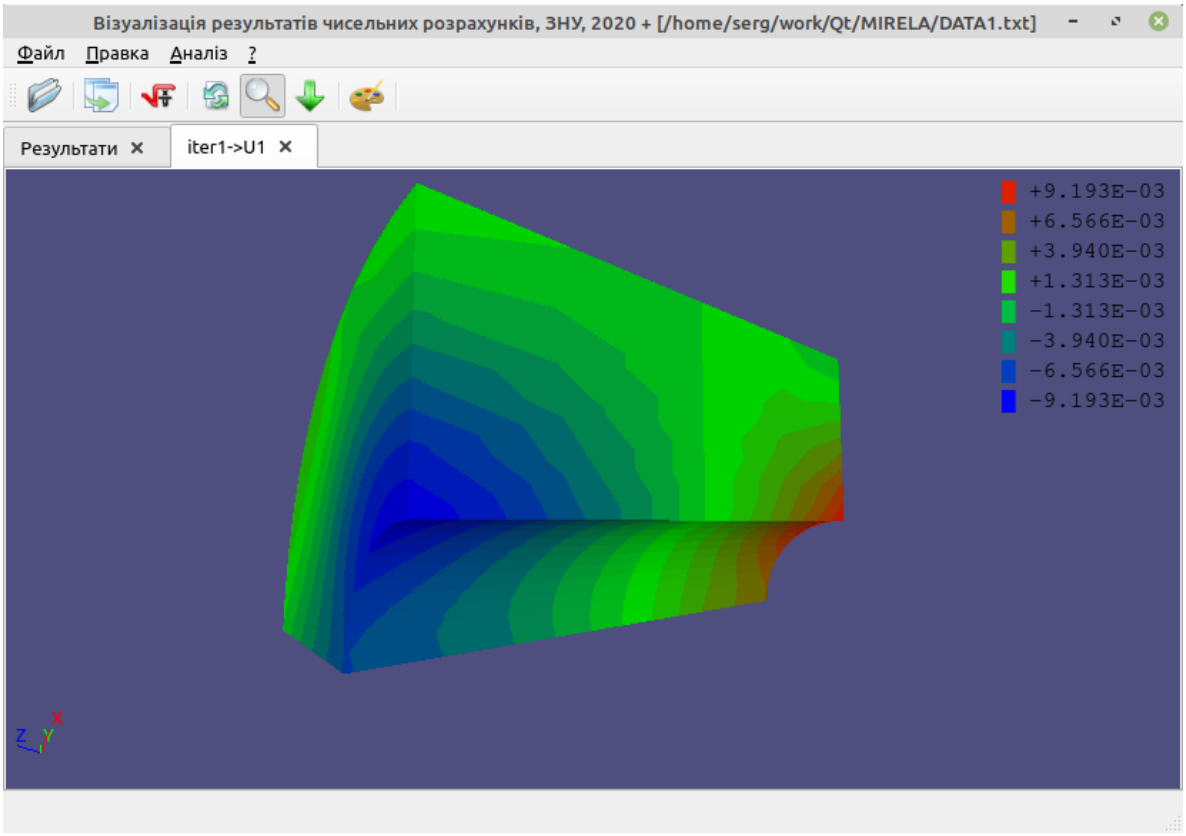


Рисунок 3.4 – Приклад візуалізації функції переміщень

ВИСНОВКИ

У кваліфікаційній роботі отримані результати, які у сукупності є вирішенням науково-технічної проблеми відображення чисельних розрахунків за допомогою програмного забезпечення.

У роботі був виконано аналіз проблема та сучасних методів візуалізації, в якому було розібрані методи візуалізації просторових об’єктів, системи координат, а також методи перетворення цих координат. Важливим аспектом було перетворення тривимірних образів у плоскі екранні об’єкти. Разом з тим було розглянуто алгоритми візуалізації чисельних розрахунків за допомогою будування напівтонових зображень та к використовуванням освітлення.

Під час розбору програмної реалізації була розглянута мова C++ і кросплатформна бібліотека Qt. При описанні третього розділу були описанні готові рішення для візуалізації, такі, наприклад, як графічний стандарт OpenGL.

Для практичної реалізації отриманих результатів розроблено програмний продукт, який дозволяє візуалізовати дво- та тривимірні моделі.

ПЕРЕЛІК ПОСИЛАНЬ

1. OpenGL. URL https://uk.wikipedia.org/wiki/OpenGL (дата звернення: 20.02.19)

2. DirectX. URL https://uk.wikipedia.org/wiki/DirectX

3. (дата звернення: 20.02.19)

4. Python. URL https://uk.wikipedia.org/wiki/Python (дата звернення: 20.02.19)

5. Qt. URL https://uk.wikipedia.org/wiki/Qt (дата звернення: 20.02.19)

6. PyQt. URL https://uk.wikipedia.org/wiki/PyQt (дата звернення: 20.02.19)

7. PyOpenGL. URL https://uk.wikipedia.org/wiki/PyQt (дата звернення: 20.02.19)

8. QtDocumentation. URL https://doc.qt.io/ (дата звернення: 20.02.19)

9. Python 3.7.3 Documentation. URL https://docs.python.org/3/ (дата звернення: 20.02.19)

10. PyQt5 Reference Guide. URL https://www.riverbankcomputing.com/static/Docs/PyQt5/ (дата звернення: 20.02.19)

11. Learning Python / Лутц М. : O'Reilly Media, 2013. – 1200 с.

12. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3 / Шрайней Д. [та ін.]. Addison-Wesley, 2013. 986 с.

ДОДАТОК А. КОД КОМПОНЕНТУ ВІЗУАЛІЗАЦІЇ ГРАНИЧНОГО СЕГМЕНТА

```
void TGLFunction::drawTriangle3D(QVector<QVector4D>& tri)
{
    QVector<QVector4D> p02, p012;
    QVector4D x, h;
    float step,
          a,
          b,
          c;
    int color_index,
        color_ind[3],
        inv = sort(tri); // Сортируем треугольник по возрастанию узловых значений

    glFrontFace((inv == 1) ? GL_CCW : GL_CW);
    for (int i = 0; i < 3; i++)
        color_ind[i] = getColorIndex(tri[i].w());

    if (params.isLight)
    {
        a  = (tri[1].y() - tri[0].y())*(tri[2].z() - tri[0].z()) - (tri[2].y() -
tri[0].y())*(tri[1].z() - tri[0].z());
        b  = (tri[2].x() - tri[0].x())*(tri[1].z() - tri[0].z()) - (tri[1].x() -
tri[0].x())*(tri[2].z() - tri[0].z());
        c  = (tri[1].x() - tri[0].x())*(tri[2].y() - tri[0].y()) - (tri[2].x() -
tri[0].x())*(tri[1].y() - tri[0].y());
        glNormal3f(float(inv)*a, float(inv)*b, float(inv)*c);
    }

    if (color_ind[0] == color_ind[1] and color_ind[1] == color_ind[2])
    {
        // Треугольник одного цвета
        glBegin(GL_TRIANGLES);
            setColor(int(color_ind[0]));
            glVertex3f(tri[0].x(), tri[0].y(), tri[0].z());
            glVertex3f(tri[1].x(), tri[1].y(), tri[1].z());
            glVertex3f(tri[2].x(), tri[2].y(), tri[2].z());
        glEnd();
    }
    else
    {
        // Изолинии проходят по треугольнику
        step = color_ind[2] - color_ind[0] + 1;
        x = QVector4D(tri[0].x(), tri[0].y(), tri[0].z(), color_ind[0]);
        h = QVector4D((tri[2].x() - tri[0].x())/step, (tri[2].y() - tri[0].y())/step, (tri[2].z() -
tri[0].z())/step, (color_ind[2] - color_ind[0])/step);

        for (int i = 0; i < step; i++)
            p02.push_back(QVector4D(x.x() + i*h.x(), x.y() + i*h.y(), x.z() + i*h.z(), color_ind[0]
+ i*h.w()));
        p02.push_back(QVector4D(tri[2].x(), tri[2].y(), tri[2].z(), color_ind[2]));

        step = color_ind[1] - color_ind[0] + 1;
        x = QVector4D(tri[0].x(), tri[0].y(), tri[0].z(), color_ind[0]);
        h = QVector4D((tri[1].x() - tri[0].x())/step, (tri[1].y() - tri[0].y())/step, (tri[1].z() -
tri[0].z())/step, (color_ind[1] - color_ind[0])/step);
        for (int i = 1; i < step; i++)
            p012.push_back(QVector4D(x.x() + i*h.x(), x.y() + i*h.y(), x.z() + i*h.z(), color_ind[0]
+ i*h.w()));
        p012.push_back(QVector4D(tri[1].x(), tri[1].y(), tri[1].z(), color_ind[1]));

        step = color_ind[2] - color_ind[1] + 1;
        x = QVector4D(tri[1].x(), tri[1].y(), tri[1].z(), color_ind[1]);
        h = QVector4D((tri[2].x() - tri[1].x())/step, (tri[2].y() - tri[1].y())/step, (tri[2].z() -
tri[1].z())/step, (color_ind[2] - color_ind[1])/step);
        for (int i = 1; i < step; i++)
            p012.push_back(QVector4D(x[0] + i*h[0], x[1] + i*h[1], x[2] + i*h[2], color_ind[1] +
i*h[3]));

        for (int i = 0; i < p02.length() - 1; i++)
            if (i < p012.length())
```

```
            {
                color_index = int(round(min(min(p02[i].w(), p02[i + 1].w()), p012[i].w())));
                setColor(color_index);
                glBegin(GL_TRIANGLES);
                    glVertex3f(p02[i + 1].x(), p02[i + 1].y(), p02[i + 1].z());
                    glVertex3f(p02[i].x(), p02[i].y(), p02[i].z());
                    glVertex3f(p012[i].x(), p012[i].y(), p012[i].z());
                glEnd();
                if (i + 1 < p012.length())
                {
                    color_index = int(round(min(min(p02[i + 1].w(), p012[i].w()), p012[i +
1].w())));
                    setColor(color_index);
                    glBegin(GL_TRIANGLES);
                        glVertex3f(p02[i + 1].x(), p02[i + 1].y(), p02[i + 1].z());
                        glVertex3f(p012[i].x(), p012[i].y(), p012[i].z());
                        glVertex3f(p012[i + 1].x(), p012[i + 1].y(), p012[i + 1].z());
                    glEnd();
                }
            }
    }
}
```

ДОДАТОК Б. ПРИКЛАД ДАНИХ ДЛЯ ВІЗУАЛІЗАЦІЇ


```
{
    "Header": {
        "DateTime": "вс март 29 12:06:35 2020",
        "Object": "console4",
        "Title": "QFEM results file"
    },
    "Mesh": {
        "BE": [
            [
                0,
                2
            ],
            [
                2,
                4
            ],
            [
                4,
                6
            ], ...
        ],
        "Coordinates": [
            [
                0,
                0
            ],
            [
                0,
                1
            ],
            [
                1,
                0
            ] …

        ],
        "FE": [
            [
                0,
                2,
                3,
                1
            ],
            [
                2,
                4,
                5,
                3
            ],
            [
                4,
                6,
                7,
                5
            ]…
        ],
        "FEType": 3
    },
    "Results": [
        {
            "Function": "U",
            "Time": 0,
            "Values": [
                0,
                0,
                -0.00028155555476530646,
                0.00028155555476530434,
                -0.0005071111096878014,
                -0.0009099999974459717,
                …
        }…
    ]
}
```

ДОДАТОК В. КОД КОМПОНЕНТУ ГОЛОВНОГО ВІКНА

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>586</width>
    <height>331</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Maximum" vsizetype="Maximum">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="windowTitle">
   <string>РµРsСЃС,РїСЂРsС†РµСЃСЃРsСЂ СЃРёСЃС,РµРjС‹ РњР†Р Р•Р›Р‡+, Р—РќРJ, 2014</string>
  </property>
  <widget class="QWidget" name="centralWidget"/>
  <widget class="QMenuBar" name="menuBar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>586</width>
     <height>21</height>
    </rect>
   </property>
   <widget class="QMenu" name="menuFile">
    <property name="title">
     <string>&amp;amp;Р¤Р°РѕР»</string>
    </property>
    <addaction name="actionOpen"/>
    <addaction name="actionClose"/>
    <addaction name="separator"/>
    <addaction name="actionExit"/>
   </widget>
   <widget class="QMenu" name="menuAnalyse">
    <property name="title">
     <string>&amp;amp;РђРЅР°Р»РёР·</string>
    </property>
    <addaction name="actionAnalyse"/>
    <addaction name="separator"/>
    <addaction name="actionRotate"/>
    <addaction name="actionScale"/>
    <addaction name="actionTranslate"/>
    <addaction name="separator"/>
    <addaction name="actionSetup"/>
   </widget>
   <widget class="QMenu" name="menuHelp">
    <property name="title">
     <string>&amp;amp;?</string>
    </property>
    <addaction name="actionAbout"/>
   </widget>
   <widget class="QMenu" name="menu">
    <property name="title">
     <string>&amp;amp;РџСЂР°РІРєР°</string>
    </property>
    <addaction name="actionCopy"/>
   </widget>
   <addaction name="menuFile"/>
   <addaction name="menu"/>
   <addaction name="menuAnalyse"/>
   <addaction name="menuHelp"/>
  </widget>
  <widget class="QToolBar" name="mainToolBar">
   <attribute name="toolBarArea">
    <enum>TopToolBarArea</enum>
   </attribute>
```

```
   <attribute name="toolBarBreak">
    <bool>false</bool>
   </attribute>
   <addaction name="actionOpen"/>
   <addaction name="separator"/>
   <addaction name="actionCopy"/>
   <addaction name="separator"/>
   <addaction name="actionAnalyse"/>
   <addaction name="separator"/>
   <addaction name="actionRotate"/>
   <addaction name="actionScale"/>
   <addaction name="actionTranslate"/>
   <addaction name="separator"/>
   <addaction name="actionSetup"/>
   <addaction name="separator"/>
   <addaction name="actionStop"/>
  </widget>
  <widget class="QStatusBar" name="statusBar"/>
  <action name="actionExit">
   <property name="text">
    <string>P’&amp;amp;C‹C…PsPr</string>
   </property>
   <property name="statusTip">
    <string>P—P°PIPµCЂC€C,CЊ CЂP°P±PsC,Cґ PïCЂPsPiCЂP°PjPjC‹</string>
   </property>
  </action>
  <action name="actionAbout">
   <property name="text">
    <string>&amp;amp;Pћ PïCЂPsPiCЂP°PjPjPµ...</string>
   </property>
   <property name="statusTip">
    <string>Pћ PïCЂPsPiCЂP°PjPjPµ...</string>
   </property>
  </action>
  <action name="actionOpen">
   <property name="icon">
    <iconset resource="mpp.qrc">
     <normaloff>:/image/open.png</normaloff>:/image/open.png</iconset>
   </property>
   <property name="text">
    <string>&amp;amp;PћC,PєCЂC‹C,CЊ...</string>
   </property>
   <property name="toolTip">
    <string>PћC,PєCЂC‹C,CЊ CЃCґC‰PµCЃC,PICґCЂC‰PëP№ C„P°P№P» CЂPµP·CґP»CЊC,P°C,PsPI
PњP†P P•P›PЋ+</string>
   </property>
   <property name="statusTip">
    <string>PћC,PєCЂC‹C,CЊ CЃCґC‰PµCЃC,PICґCЂC‰PëP№ C„P°P№P» CЂPµP·CґP»CЊC,P°C,PsPI
PњP†P P•P›PЋ+</string>
   </property>
   <property name="shortcut">
    <string>F3</string>
   </property>
  </action>
  <action name="actionClose">
   <property name="text">
    <string>&amp;amp;P—P°PєCЂC‹C,CЊ</string>
   </property>
   <property name="toolTip">
    <string>P—P°PєCЂC‹C,CЊ C,PµPєCґC‰PëP№ C„P°P№P» PrP°PSPSC‹C…</string>
   </property>
   <property name="statusTip">
    <string>P—P°PєCЂC‹C,CЊ C,PµPєCґC‰PëP№ C„P°P№P» PrP°PSPSC‹C…</string>
   </property>
  </action>
  <action name="actionAnalyse">
   <property name="icon">
    <iconset resource="mpp.qrc">
     <normaloff>:/image/kformula.png</normaloff>:/image/kformula.png</iconset>
   </property>
   <property name="text">
    <string>&amp;amp;P¤CґPSPєC†PëCЏ...</string>
   </property>
   <property name="toolTip">
    <string>P’C‹P±CЂP°C,CЊ C„CґPSPєC†PëCЋ PrP»CЏ P°PSP°P»PëP·P°</string>
   </property>
   <property name="statusTip">
    <string>P’C‹P±CЂP°C,CЊ C„CґPSPєC†PëCЋ PrP»CЏ P°PSP°P»PëP·P°</string>
   </property>
```

```
 <property name="shortcut">
  <string>F4</string>
 </property>
</action>
<action name="actionRotate">
 <property name="checkable">
  <bool>true</bool>
 </property>
 <property name="icon">
  <iconset resource="mpp.qrc">
   <normaloff>:/image/rotate.png</normaloff>:/image/rotate.png</iconset>
 </property>
 <property name="text">
  <string>&amp;РџPsPIPsСЂPsC,</string>
 </property>
 <property name="toolTip">
  <string>P’РєP»СЋС‡РёС,СЊ СЂРµР¶РёРj PICЂP°C‰РµPSPёСЏ PsP±СЉРµРєC,P°</string>
 </property>
 <property name="statusTip">
  <string>P’РєP»СЋС‡РёС,СЊ СЂРµР¶РёРj PICЂP°C‰РµPSPёСЏ PsP±СЉРµРєC,P°</string>
 </property>
</action>
<action name="actionScale">
 <property name="checkable">
  <bool>true</bool>
 </property>
 <property name="icon">
  <iconset resource="mpp.qrc">
   <normaloff>:/image/scale.png</normaloff>:/image/scale.png</iconset>
 </property>
 <property name="text">
  <string>&amp;РњР°СЃС€С,Р°Р±РёСЂPsPIP°PSPёРµ</string>
 </property>
 <property name="toolTip">
  <string>P’РєP»СЋС‡РёС,СЊ СЂРµР¶РёРj PjP°CЃС€С,Р°Р±РёСЂPsPIP°PSPёСЏ PsP±СЉРµРєC,P°</string>
 </property>
 <property name="statusTip">
  <string>P’РєP»СЋС‡РёС,СЊ СЂРµР¶РёРj PjP°CЃС€С,Р°Р±РёСЂPsPIP°PSPёСЏ PsP±СЉРµРєC,P°</string>
 </property>
</action>
<action name="actionTranslate">
 <property name="checkable">
  <bool>true</bool>
 </property>
 <property name="icon">
  <iconset resource="mpp.qrc">
   <normaloff>:/image/build.png</normaloff>:/image/build.png</iconset>
 </property>
 <property name="text">
  <string>&amp;РЎРґPIPёPi</string>
 </property>
 <property name="toolTip">
  <string>P’РєP»СЋС‡РёС,СЊ СЂРµР¶РёРj СЃPjРµС‰РµPSPёСЏ PsP±СЉРµРєC,P°</string>
 </property>
 <property name="statusTip">
  <string>P’РєP»СЋС‡РёС,СЊ СЂРµР¶РёРj СЃPjРµС‰РµPSPёСЏ PsP±СЉРµРєC,P°</string>
 </property>
</action>
<action name="actionSetup">
 <property name="icon">
  <iconset resource="mpp.qrc">
   <normaloff>:/image/funct.png</normaloff>:/image/funct.png</iconset>
 </property>
 <property name="text">
  <string>&amp;РµP°СЂP°PjРµC,СЂС‹...</string>
 </property>
 <property name="toolTip">
  <string>Рќ°СЃС,СЂPsPёС,СЊ РїP°СЂP°PjРµC,СЂС‹ PIPёP·СѓP°P»РёP·P°С†РёРё PsP±СЉРµРєC,P°</string>
 </property>
 <property name="statusTip">
  <string>Рќ°СЃС,СЂPsPёС,СЊ РїP°СЂP°PjРµC,СЂС‹ PIPёP·СѓP°P»РёP·P°С†РёРё PsP±СЉРµРєC,P°</string>
 </property>
</action>
<action name="actionStop">
 <property name="icon">
  <iconset resource="mpp.qrc">
   <normaloff>:/image/cancel.png</normaloff>:/image/cancel.png</iconset>
 </property>
 <property name="text">
```

```
    <string> PЎC,PsPï</string>
   </property>
   <property name="toolTip">
    <string>PћCЃC,P°PSPsPIPëC,CЊ P·P°PiCЂCґP·PeCґ PrP°PSPSC‹C…</string>
   </property>
   <property name="statusTip">
    <string>PћCЃC,P°PSPsPIPëC,CЊ P·P°PiCЂCґP·PeCґ PrP°PSPSC‹C…</string>
   </property>
  </action>
  <action name="actionCopy">
   <property name="icon">
    <iconset resource="mpp.qrc">
     <normaloff>:/image/copy.png</normaloff>:/image/copy.png</iconset>
   </property>
   <property name="text">
    <string>&amp;amp;PљPsPïPëCЂPsPIP°C,CЊ</string>
   </property>
   <property name="toolTip">
    <string>PљPsPïPëCЂPsPIP°C,CЊ C,PµPeCґC‰PµPµ PëP·PsP±CЂP°P¶PµPSPëPµ PI P±CґC„PµCЂ
PsP±PjPµPSP°</string>
   </property>
   <property name="statusTip">
    <string>PљPsPïPëCЂPsPIP°C,CЊ C,PµPeCґC‰PµPµ PëP·PsP±CЂP°P¶PµPSPëPµ PI P±CґC„PµCЂ
PsP±PjPµPSP°</string>
   </property>
  </action>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources>
  <include location="mpp.qrc"/>
 </resources>
 <connections>
  <connection>
   <sender>actionExit</sender>
   <signal>triggered()</signal>
   <receiver>MainWindow</receiver>
   <slot>close()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>-1</x>
     <y>-1</y>
    </hint>
    <hint type="destinationlabel">
     <x>292</x>
     <y>165</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>actionOpen</sender>
   <signal>triggered()</signal>
   <receiver>MainWindow</receiver>
   <slot>openDocument()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>-1</x>
     <y>-1</y>
    </hint>
    <hint type="destinationlabel">
     <x>292</x>
     <y>165</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>actionClose</sender>
   <signal>triggered()</signal>
   <receiver>MainWindow</receiver>
   <slot>closeDocument()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>-1</x>
     <y>-1</y>
    </hint>
    <hint type="destinationlabel">
     <x>292</x>
     <y>165</y>
    </hint>
   </hints>
```

```
</connection>
<connection>
 <sender>actionAnalyse</sender>
 <signal>triggered()</signal>
 <receiver>MainWindow</receiver>
 <slot>analyseFunction()</slot>
 <hints>
  <hint type="sourcelabel">
   <x>-1</x>
   <y>-1</y>
  </hint>
  <hint type="destinationlabel">
   <x>292</x>
   <y>165</y>
  </hint>
 </hints>
</connection>
<connection>
 <sender>actionRotate</sender>
 <signal>triggered()</signal>
 <receiver>MainWindow</receiver>
 <slot>setRotate()</slot>
 <hints>
  <hint type="sourcelabel">
   <x>-1</x>
   <y>-1</y>
  </hint>
  <hint type="destinationlabel">
   <x>292</x>
   <y>165</y>
  </hint>
 </hints>
</connection>
<connection>
 <sender>actionScale</sender>
 <signal>triggered()</signal>
 <receiver>MainWindow</receiver>
 <slot>setScale()</slot>
 <hints>
  <hint type="sourcelabel">
   <x>-1</x>
   <y>-1</y>
  </hint>
  <hint type="destinationlabel">
   <x>292</x>
   <y>165</y>
  </hint>
 </hints>
</connection>
<connection>
 <sender>actionTranslate</sender>
 <signal>triggered()</signal>
 <receiver>MainWindow</receiver>
 <slot>setTranslate()</slot>
 <hints>
  <hint type="sourcelabel">
   <x>-1</x>
   <y>-1</y>
  </hint>
  <hint type="destinationlabel">
   <x>292</x>
   <y>165</y>
  </hint>
 </hints>
</connection>
<connection>
 <sender>actionSetup</sender>
 <signal>triggered()</signal>
 <receiver>MainWindow</receiver>
 <slot>setupImageParams()</slot>
 <hints>
  <hint type="sourcelabel">
   <x>-1</x>
   <y>-1</y>
  </hint>
  <hint type="destinationlabel">
   <x>292</x>
   <y>165</y>
  </hint>
```

```
   </hints>
  </connection>
  <connection>
   <sender>actionStop</sender>
   <signal>triggered()</signal>
   <receiver>MainWindow</receiver>
   <slot>stop()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>-1</x>
     <y>-1</y>
    </hint>
    <hint type="destinationlabel">
     <x>292</x>
     <y>165</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>actionCopy</sender>
   <signal>triggered()</signal>
   <receiver>MainWindow</receiver>
   <slot>copyImage()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>-1</x>
     <y>-1</y>
    </hint>
    <hint type="destinationlabel">
     <x>292</x>
     <y>165</y>
    </hint>
   </hints>
  </connection>
 </connections>
 <slots>
  <slot>openDocument()</slot>
  <slot>closeDocument()</slot>
  <slot>analyseFunction()</slot>
  <slot>setRotate()</slot>
  <slot>setScale()</slot>
  <slot>setTranslate()</slot>
  <slot>setupImageParams()</slot>
  <slot>stop()</slot>
  <slot>copyImage()</slot>
 </slots>
</ui>
```