**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №1

**на тему:** *“Візуалізація статичних двовимірних зображень за допомогою бібліотеки OpenGL”*

**з дисципліни** *“Засоби програмування комп'ютерної графіки”*

**Лектор:**

асис. каф. ПЗ

Журавчак Л.М.

**Виконав:**

ст. гр. ПЗ-33

Юшкевич. А.І.

**Прийняв:**

ст. викл. каф. ПЗ

Журавчак Л.М.

«\_\_» \_\_\_\_\_\_\_\_\_ 2024р.

∑=\_\_\_\_\_

Львів – 2024

**Тема:** Візуалізація статичних двовимірних зображень за допомогою бібліотеки OpenGL.

**Мета:** Вивчити методи роботи з найпростішими функціями бібліотеки OpenGL.

**Теоретичні відомості**

OpenGL (англ. Open Graphics Library – відкрита графічна бібліотека) – це потужний програмний інтерфейс, який використовують для одержання високоякісних, програмно генерованих зображень та інтерактивних застосунків, які описують дво- і тривимірні об’єкти, а також растрові зображення.

### **Складові OpenGL**

Імена функцій **основної бібліотеки**OpenGL починаються з префіксу gl, кожне слово, яке входить в ім’я функції, починається з великої букви, а самі функції зберігаються в бібліотеці, котру позначають GL. Крім основної, існує і декілька **додаткових бібліотек**.

**Бібліотека графічних утиліт** (GLU) включає стандартні функції, які дають змогу настроювати матриці проекції і візуалізації, описувати складні об’єкти наближено прямими і многокутниками, зображати квадратичні й бі-сплайни з допомогою лінійного наближення, зафарбовувати поверхні, тобто в її склад входять функції формування складних об’єктів типу сферичних поверхонь. Функції цієї бібліотеки звертаються тільки до функцій GL.

**OpenGL Utility Toolkit**(GLUT) містить функції для роботи з будь-якою системою вікон на екрані та методи, які дають змогу зафарбовувати криві і поверхні другого порядку. Вона забезпечує користувача основними можливостями, характерними для більшості сучасних багатовіконних систем.

### **Примітиви OpenGL**

За допомогою функцій примітивів, які є в основній бібліотеці OpenGL, можна будувати точки, прямолінійні відрізки, зафарбовані опуклі многокутники, а також піксельні чи растрові матричні структури. В бібліотеці GLUT можна знайти стандартні процедури для зображення рядків символів. Інші види примітивів, такі як кола, еліпси чи зафарбовані увігнуті багатокутні області, можна будувати за допомогою функцій основної бібліотеки або стандартними процедурами з бібліотек GLU і GLUT.

Базові примітиви OpenGL специфікують набором точок у просторі (вершин). Формат визначення об’єктів такий:

**glBegin (тип);**

**glVertex\* (…);**

**…**

**glEnd ();**

Значення «тип» визначає вигляд об’єкту і несе інформацію про те, яким чином треба інтерпретувати подальший список вершин. Функція **glVertex** повинна бути розміщена в програмі між функціями **glBegin** і **glEnd**. Аргумент функції **glBegin** визначає **тип графічного примітиву**:

1. **GL\_POINTS** – точка;
2. **GL\_LINES** – відрізок за 2-ма вершинами;
3. **GL\_LINE\_STRIP** – ламана лінія;
4. **GL\_POLYGON** – многокутник і т.д.

Коренева бібліотека OpenGL надає тільки випуклі многокутники, проте бібліотека GLU пропонує функції, що дають змогу працювати з увігнутими многокутниками та іншими невипуклими об’єктами з лінійними межами.

Під час візуалізації набору заданих у програмі об’єктів вони послідовно накладаються один на одний. Щоб видалити котрийсь із них, потрібно перемалювати весь набір примітивів, окрім зайвого.

### **Використання тексту**

У складі основної бібліотеки OpenGL нема спеціального примітиву для формування тексту, однак GLUT містить декілька визначених наборів символів та стандартні процедури для зображення шрифтів. **Символи растрового шрифту** визначаються на прямокутній області і є блоками бітів. Символи можна отримати за допомогою функції **glutBitmapCharacter(font, character)**; де **font** – значення символьної константи GLUT, яка вказує на визначений набір накреслень (шрифт з постійною шириною символів (моноширинний) – **GLUT\_BITMAP\_8\_BY\_13** тощо; шрифт з пропорційними проміжками (пропорційний) – **GLUT\_BITMAP\_TIMES\_ROMAN\_10**, **GLUT\_BITMAP\_ HELVETICA\_10** тощо); **character** – код ASCII або окремий символ, який потрібно зобразити (наприклад, код 65 або 'A'). Щоб задати координати поточного растрового положення, застосовують процедуру **glRasterPos\*(…)**; параметри й індекси аналогічні тим, що використовують у функції **glVertex**.

### **Налаштування кольорів**

Більшість налаштувань кольорів примітивів OpenGL виконується в режимі RGB. Поточні колірні компонент вибирають за допомогою функції **glColor\*(colorComponents)**; індекси аналогічні описаним для функції **glVertex**, причому перший може мати значення лише 3 або 4; колірні компоненти послідовно для червоного, зеленого та синього задають як числа від 0 до 1. Можливий четвертий параметр – коефіцієнт альфа – використовують для моделювання ефектів прозорості: коли він 0 – об’єкт повністю прозорий, 1 – непрозорий.

### **Загальна структура програми**

Мінімальний набір операцій, необхідний для створення зображення з використанням бібліотеки OpenGL, має такий вигляд.

1. Ініціалізація бібліотеки GLUT для створення сеансу зв’язку між підсистемою вікон і OpenGL командою **glutInit(&argc, argv)**.
2. Створення вікна зображення з відповідною назвою в рядку заголовка – **glutCreateWindow("<Ім’я вікна>")**. Для зміни параметрів вікна, встановлених системою за замовчуванням (розмір, розташування на екрані, колірний режим тощо), треба перед викликом **glutCreateWindow** звернутись до відповідних функцій бібліотеки GLUT (**glutInitDisplayMode**, **glutInitWindowPosition**, **glutInitWindowSize** і т.д.).
3. Передавання у вікно зображення графічних об’єктів, створених відповідною процедурою, за допомогою функції **glutDisplayFunc(<Ім’я процедури>)**.
4. Активізація всіх створених вікон зображень разом зі своїм графічним вмістом – **glutMainLoop()**. Ця функція, що має бути останнім оператором у програмі, відображає графічні об’єкти на екрані та запускає нескінченний цикл, в якому система приймає дані, що надходять від клавіатури чи миші. Це дає змогу організувати реакцію програми на дії користувача.

Процедури для створення рисунків, імена яких є аргументами для **glutDisplayFunc**, називають **функціями зворотного виклику зображень**. В кінці їхнього коду має стояти **glFlush()** – стандартна функція для прискорення виконання функцій OpenGL, записаних у буферах, що розташовані в різних місцях обчислювальної системи. В завантаженій мережі можуть виникати затримки обробки даних з деяких буферів, але виклик **glFlush** призводить до звільнення буферів та обробки функцій OpenGL.

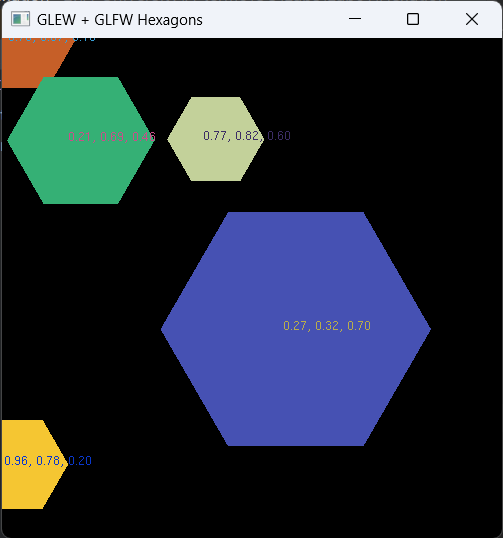
**Завдання**

15. Скласти програму побудови **шестикутників** різних розмірів та кольорів у різних частинах вікна з використанням випадкового вибору. Підписати кожну фігуру назвою чи кодом кольору. Послідовно витерти всі фігури та підписи.

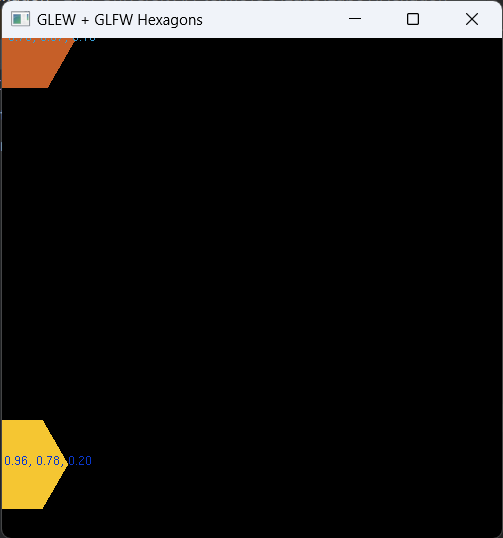
**Код програми**

#include <GL/glew.h>  
#include <GL/freeglut.h>  
#include <GLFW/glfw3.h>  
#include <iostream>  
#include <cstdlib>  
#include <ctime>  
#include <cmath>  
#include <sstream>  
#include <iomanip>  
  
const int NUM\_HEXAGONS = 5;  
  
struct Hexagon {  
 float x, y;  
 float size;  
 float r, g, b;  
};  
  
Hexagon hexagons[NUM\_HEXAGONS];  
  
void generateHexagons() {  
 srand(time(0));  
 for (int i = 0; i < NUM\_HEXAGONS; i++) {  
 hexagons[i].x = (rand() % 400) / 200.0f - 1.0f;  
 hexagons[i].y = (rand() % 400) / 200.0f - 1.0f;  
 hexagons[i].size = 0.1f + static\_cast<float>(rand()) / RAND\_MAX \* 0.5f;  
 hexagons[i].r = static\_cast<float>(rand()) / RAND\_MAX;  
 hexagons[i].g = static\_cast<float>(rand()) / RAND\_MAX;  
 hexagons[i].b = static\_cast<float>(rand()) / RAND\_MAX;  
 }  
}  
  
void drawHexagon(float x, float y, float size, float r, float g, float b) {  
 glColor3f(r, g, b);  
 glBegin(GL\_POLYGON);  
 for (int i = 0; i < 6; i++) {  
 float angle = 2.0f \* 3.1415926f \* float(i) / 6.0f;  
 float dx = size \* cos(angle);  
 float dy = size \* sin(angle);  
 glVertex2f(x + dx, y + dy);  
 }  
 glEnd();  
}  
  
// Відображення тексту (залежить від GLUT)  
void renderText(float x, float y, const std::string& text) {  
 glRasterPos2f(x, y);  
 for (char c : text) {  
 glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12, c);  
 }  
}  
  
void render(int recursionRate = NUM\_HEXAGONS) {  
 glClear(GL\_COLOR\_BUFFER\_BIT);  
  
 // Спочатку малюємо шестикутники  
 for (int i = 0; i < recursionRate; i++) {  
 drawHexagon(hexagons[i].x, hexagons[i].y, hexagons[i].size, hexagons[i].r, hexagons[i].g, hexagons[i].b);  
  
 std::ostringstream colorText;  
 colorText << std::fixed << std::setprecision(2)  
 << hexagons[i].r << ", " << hexagons[i].g << ", " << hexagons[i].b;  
  
 // Протилежний колір  
 float textR = 1.0f - hexagons[i].r;  
 float textG = 1.0f - hexagons[i].g;  
 float textB = 1.0f - hexagons[i].b;  
 glColor3f(textR, textG, textB);  
  
 renderText(hexagons[i].x - 0.05f, hexagons[i].y, colorText.str());  
 }  
  
 glFlush();  
}  
  
int main() {  
 if (!glfwInit()) {  
 std::cerr << "Failed to initialize GLFW!" << std::endl;  
 return -1;  
 }  
  
 GLFWwindow\* window = glfwCreateWindow(500, 500, "GLEW + GLFW Hexagons", nullptr, nullptr);  
 if (!window) {  
 std::cerr << "Failed to create window!" << std::endl;  
 glfwTerminate();  
 return -1;  
 }  
  
 glfwMakeContextCurrent(window);  
  
 if (glewInit() != GLEW\_OK) {  
 std::cerr << "Failed to initialize GLEW!" << std::endl;  
 return -1;  
 }  
  
 int argc = 0;  
 char\*\* argv = nullptr;  
 glutInit(&argc, argv);  
  
 glClearColor(0, 0, 0, 1);  
 generateHexagons();  
  
 int renderRate = NUM\_HEXAGONS;  
 double lastTime = glfwGetTime();  
 while (!glfwWindowShouldClose(window)) {  
  
 render(renderRate);  
 glfwSwapBuffers(window);  
 glfwPollEvents();  
  
 if (glfwGetKey(window, GLFW\_KEY\_R) == GLFW\_PRESS) {  
 generateHexagons();  
 render();  
 renderRate = NUM\_HEXAGONS;  
 }  
  
 if (glfwGetKey(window, GLFW\_KEY\_C) == GLFW\_PRESS) {  
  
 while (renderRate > 0){  
 double currentTime = glfwGetTime();  
 double deltaTime = currentTime - lastTime;  
  
 if (deltaTime > 1.f) {  
 render(--renderRate);  
 glfwSwapBuffers(window);  
 glfwPollEvents();  
 lastTime = currentTime;  
 }  
 }  
 }  
 }  
  
 glfwDestroyWindow(window);  
 glfwTerminate();  
 return 0;  
}

**Результати виконання роботи**

****

*Рис. 1. Рендер шестикутників з випадковими параметрами та відповідними підписами з кодом використаного кольору*

******

*Рис. 2. Стан програми після послідовно видалених трьох шестикутників*

**Висновки:** протягом виконання цієї лабораторної роботи я вивчив методи роботи з найпростішими функціями бібліотеки OpenGL. Оскільки на моїй системі примітиви не підтримуються я використав складнішу систему відмальовування графічних обʼєктів з використанням обʼєкта масиву вершинб обʼєкта буферу вершин, різних кастомних шейдерів та ортогональної матриці для рендеру тексту.