МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Лабораторна робота № 2

з дисципліни «Програмування віртуальної реальності»
(назва дисципліни)
на тему: «Фотореалістичність об'єктів віртуальної реальності (Перспективна проекція та управління точкою зору)»

Виконав: студент 4 курсу групи № 545В
напряму підготовки (спеціальності)
123 - Комп’ютерна інженерія
(шифр і назва напряму підготовки /спеціальності)
Ткаченко І. Д.
(прізвище й ініціали студента)
Прийняв: доцент Лучшев П. О.
(посада, науковий ступінь, прізвище й ініціали)
Національна шкала: _____
Кількість балів: _____
Оцінка ECTS: _____

Харків – 2023

1. Постановка задачі

1.1. Загальне завдання

Створення програмного продукту, спрямованого на підвищення рівня фотореалізму об'єктів у віртуальній реальності (VR). У цьому проекті основний фокус приділяється вдосконаленню технік перспективної проекції та реалістичного управління точкою зору з метою забезпечення максимально іммерсивного враження користувача. В якості базової платформи для розробки програми розглядається бібліотека OpenGL, а реалізація можлива за допомогою обраної розробником мови програмування високого рівня (наприклад, C++).

2. Теоретичні відомості

1. **Перспективна проекція:**
   - **Визначення:** Перспективна проекція у графіці представляє собою метод відображення тривимірних об'єктів на площині екрану з урахуванням ефектів віддаленості та просторової перспективи.
   - **Принципи:**
     - **Зменшення з відстанню:** Зменшення з відстанню: Об'єкти, які знаходяться далеко, відображаються меншими, відтворюючи ефект віддаленості.
     - **Лінії збігаються:** Паралельні лінії, що простягаються углиб, здаються збігатися у віддаленому плані.
   - **Застосування:** Застосовується у комп'ютерних іграх, віртуальній реальності та архітектурному дизайні для створення реалістичних зображень.

2. **Управління точкою зору:**
   - **Визначення:** Управління точкою зору означає зміну положення та напрямку перегляду віртуальної камери чи спостерігача.

3. Лістинг програми

3.1. Файл Camera.cpp

```
#pragma once
#ifndef __CAMERA
#define __CAMERA
#include <Windows.h>
#include <gl/GL.h>
#include <gl/glu.h>
#include "glmath.h"
#include <math.h>

static vec3 YAW_AXIS = { 0, 1, 0 };
static vec3 PITCH_AXIS = { 1, 0, 0 };
```

```
static vec3 ROLL_AXIS = { 0, 0, 1 };

static vec3 FORWARD = { 0, 0, -1 };
static vec3 UP = { 0, 1, 0 };

enum camera_flags_e
{
	CF_LOCKANGLES = (1 << 0),
};

class CCamera {
public:
	CCamera() : m_bLockAngles(true) {};
	CCamera(float fov, vec3 vec, int flags = 0) {
		m_fSensitivity = 0.1f;
		m_FOV = fov;
		m_vecOrigin.x = vec.x;
		m_vecOrigin.y = vec.y;
		m_vecOrigin.z = vec.z;
		m_Yaw = -25;
		m_Pitch = 25;
		//m_Roll = 0;
		m_Quat = quat(0.f, 0.f, 0.f, 0.f);
		updateEulerOrientation();
		m_vecDirection = getForward();
		m_vecUp = getUp();
	};
	~CCamera() {};

	// Function to update mouse input and correct angles
	void UpdateMouseInput(HWND hWnd) {
		POINT pp;
		GetCursorPos(&pp);

		RECT rect;
		GetClientRect(hWnd, &rect);
		ClientToScreen(hWnd, (POINT*)&rect.left);
		ClientToScreen(hWnd, (POINT*)&rect.right);
		int centerX = rect.right >> 1;
		int centerY = rect.bottom >> 1;
		if (pp.x == centerX && pp.y == centerY)
			return;

		m_Yaw -= (centerX - pp.x) * m_fSensitivity;
		m_Pitch -= (centerY - pp.y) * m_fSensitivity;

		CorrectAngles();

		SetCursorPos(centerX, centerY);
	}

	// Function to update camera state based on user input
	void UpdateCameraState(HWND hWnd) {
		if (m_bActive)
			UpdateMouseInput(hWnd);

		updateEulerOrientation();
		m_vecDirection = getForward();
		m_vecUp = getUp();

		m_vecOrigin = add(m_vecOrigin, m_vecMovement.z * m_vecDirection); //
updateForwardMovement
```

```
			m_vecOrigin = add(m_vecOrigin, m_vecMovement.y * m_vecUp); //
updateUpMovement
			vec3 sideVector = normalize(cross(m_vecUp, m_vecDirection)); //
updateSideMovement
			m_vecOrigin = add(m_vecOrigin, m_vecMovement.x * sideVector);
		}

		// Function to update Euler orientation based on yaw, pitch, and roll
		void updateEulerOrientation() {
			quat qYaw = quat_from_angle_axis(m_Yaw, YAW_AXIS);
			quat qPitch = quat_from_angle_axis(m_Pitch, PITCH_AXIS);
			quat qRoll = quat_from_angle_axis(m_Roll, ROLL_AXIS);
			m_Quat = mul(qYaw, mul(qPitch, qRoll));
		}

		// Function to set the camera view
		void Look() {
#ifdef MY_IMPLEMENT
			vec3 forward, side, up;
			forward = m_vecOrigin - m_vecDirection;
			forward = normalize(forward);

			side = cross(forward, m_vecUp);
			normalize(side);

			up = cross(side, forward);

			mat4x4 matrix = Mat4x4_identity;
			matrix.m[0][0] = side[0];
			matrix.m[1][0] = side[1];
			matrix.m[2][0] = side[2];

			matrix.m[0][1] = up[0];
			matrix.m[1][1] = up[1];
			matrix.m[2][1] = up[2];

			matrix.m[0][2] = -forward[0];
			matrix.m[1][2] = -forward[1];
			matrix.m[2][2] = -forward[2];

			glMultMatrixf(matrix.M);
			glTranslated(-m_vecOrigin.x, -m_vecOrigin.y, -m_vecOrigin.z);
#else
			vec3 center = m_vecOrigin + m_vecDirection;
			gluLookAt(m_vecOrigin.x, m_vecOrigin.y, m_vecOrigin.z, center.x, center.y,
center.z, 0.f, 1.f, 0.f);
#endif
		}

		inline vec3 getForward() { return rotate_with_quat(FORWARD, m_Quat); }
		inline vec3 getUp() { return rotate_with_quat(UP, m_Quat); }

		inline void UpdateViewport(int screenWidth, int screenHeight) {
			m_iScreenWidth = screenWidth;
			m_iScreenHeight = screenHeight;
		}

		inline void SetMovement(vec3 vecMove) { m_vecMovement = vecMove; }
		inline vec3 GetMovement() { return m_vecMovement; }
		inline void MoveForward() { m_vecMovement.z = 1.f; }
		inline void MoveBack() { m_vecMovement.z = -1.f; }
		inline void MoveLeft() { m_vecMovement.x = 1.f; }
```

```
inline void MoveRight() { m_vecMovement.x = -1.f; }

inline void SetPitch(float pitch) { m_Pitch = pitch; }
inline float GetPitch() { return m_Pitch; }
inline void SetYaw(float yaw) { m_Yaw = yaw; }
inline float GetYaw() { return m_Yaw; }
inline void SetRoll(float roll) { m_Roll = roll; }
inline float GetRoll() { return m_Roll; }
inline void SetRotation(vec3 rot) {
	m_Pitch = rot.x;
	m_Yaw = rot.y;
	m_Roll = rot.z;
}
inline vec3 GetRotation() { return vec3(m_Pitch, m_Yaw, m_Roll); }

inline void SetActive(bool active) {
	m_bActive = active;
}

void Move(bool* p_keys, float scale) {
	if (!this->IsActive())
		return;

	m_vecMovement = vec3(0.f, 0.f, 0.f);
	if (p_keys['W'])
		m_vecMovement.z = scale;

	if (p_keys['A'])
		m_vecMovement.x = scale;

	if (p_keys['S'])
		m_vecMovement.z = -scale;

	if (p_keys['D'])
		m_vecMovement.x = -scale;

	if (p_keys[' '])
		m_vecMovement.y = scale;

	if (p_keys[VK_DOWN])
		m_vecMovement.y = -scale;

	if (p_keys[VK_UP])
		m_vecMovement.y = scale;
}

inline bool IsActive() { return m_bActive; }

inline bool IsLockedAngles() { return m_bLockAngles; }
void SetLockedAngles(bool status) { m_bLockAngles = status; }

void CorrectAngles() {
	if (m_Pitch > 89.9f)
		m_Pitch = 89.9f;
	if (m_Pitch < -89.9f)
		m_Pitch = -89.9f;

	if (m_Yaw > 359.9f)
		m_Yaw = 0.f;
	if (m_Yaw < 0.f)
		m_Yaw = 359.9f;
}
```

```
        vec3  m_vecOrigin;
        vec3  m_vecDirection;
        vec3  m_vecUp;
        vec3  m_vecMovement;
        float m_fSensitivity;
        float m_FOV;

        float m_Pitch;
        float m_Yaw;
        float m_Roll;
        quat  m_Quat;

        int  m_nFlags;
        int  m_iScreenWidth;
        int  m_iScreenHeight;
        bool m_bActive;
        bool m_bLockAngles;
};
#endif
```

4. Тестове виконання програми

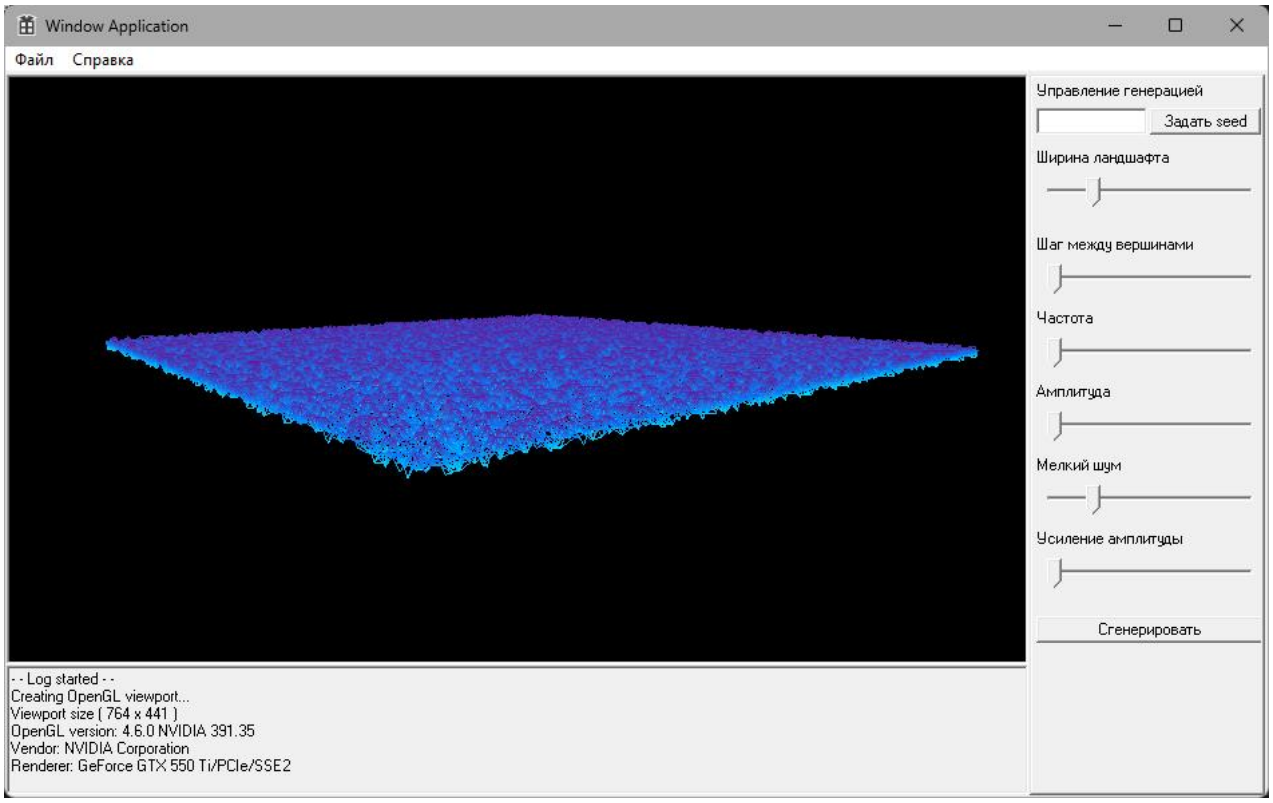

Рис. 1 – Тест 1

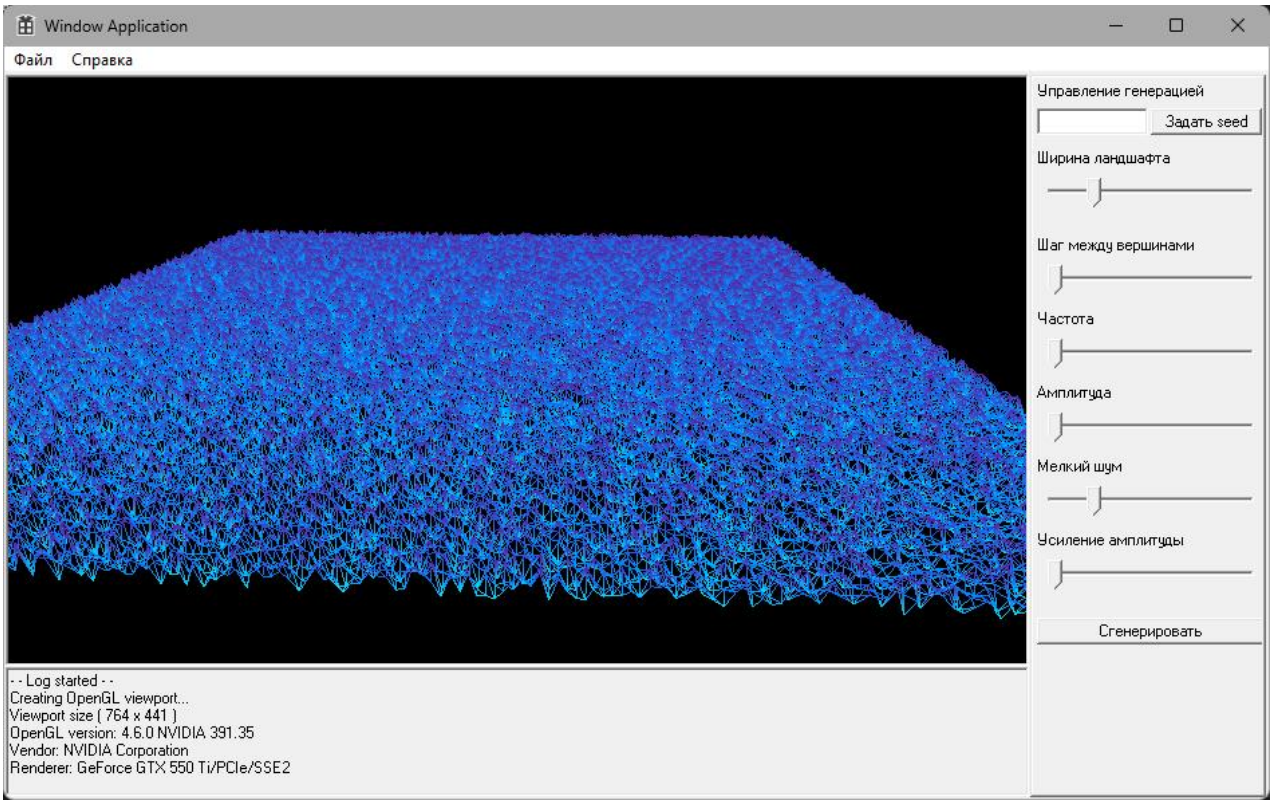

Рис. 2 – Тест 2

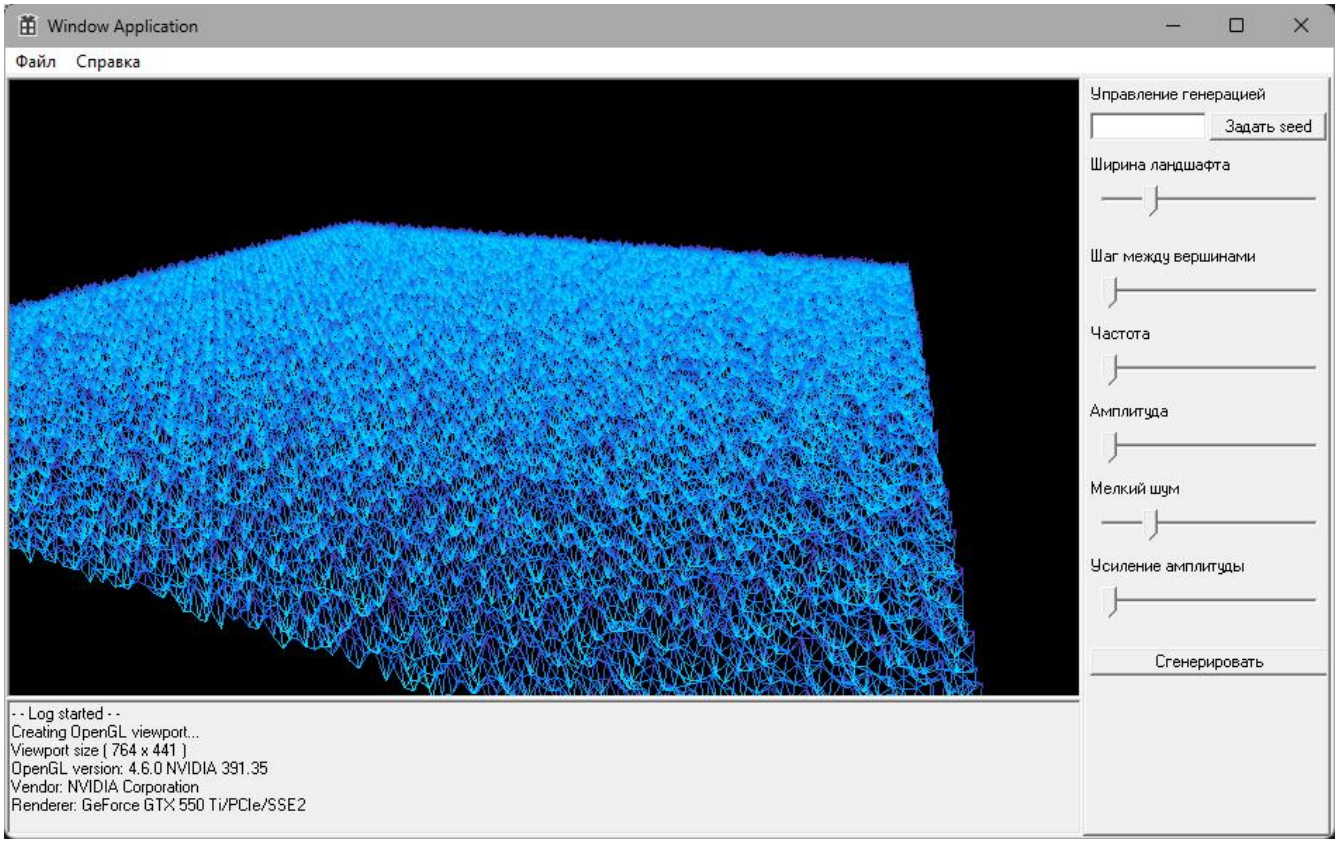

Рис. 3 – Тест 3

Висновки

У ході виконання даної лабораторної роботи я розробив клас "Camera" та успішно інтегрував його у основний клас. Цей крок призвів до підвищення рівня фотореалістичності мого попереднього проекту, який був створений під час лабораторної роботи №1.