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 № 1*

з дисципліни «Програмування віртуальної реальності»
(назва дисципліни)
на тему: «Формування простору віртуальної реальності(Моделювання та генерація ландшафту)»

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*

Розробити програму для формування та відображення на екрані тривимірної поверхні, яка за допомогою обраних методів моделює ландшафт. Згенерована поверхня повинна виводитися на екран у двох режимах: у вигляді каркасної моделі та з градієнтним розфарбуванням, що залежить від висоти.

Як базова платформа для розробки програми розглядається бібліотека OpenGL і підтримує її мову програмування високого рівня на вибір розробника(C++).

# 2. Теоретичні відомості

Для того, щоб згенерувати ландшафт є деякі методи, які надають можливість формування віртуальних ландшафтів у просторі віртуальної реальності з різним рівнем деталізації та природнім виглядом. Застосування цих методів в зображеннях ігор та симуляціях може забезпечити реалістичний візуальний досвід користувачів.

1. **Карти висот:**
   Карти висот є ключовим елементом для представлення ландшафту у тривимірній графіці. Вони представлені як двовимірні масиви, де кожен піксель визначає висоту точки на ландшафті. Яскравість пікселя вказує на висоту, і монохромний режим забезпечує 256 градацій висот.

2. **Простий Генератор Висот:**
   Цей метод використовується для створення початкового масиву з випадковими значеннями висот. Далі застосовується фільтр для згладжування, заснований на простому усередненні, що дозволяє отримати більш природний вигляд. Додатково, застосовується заповнення випадковими значеннями для створення деталізованішого рельєфу.

3. **Зміщення Середньої Точки:**
   Цей метод використовує фрактальний підхід для створення ландшафту. Рекурсивне зміщення середніх точок трикутників дозволяє поділити кожен трикутник на чотири нові, забезпечуючи різні рівні деталізації.

4. **Генератор Пагорбів:**
   - Цей метод використовується для формування більш складного ландшафту. Масив ініціалізується нульовим рівнем, і горби створюються у випадкових точках за допомогою обертання параболи. Ітеративне застосування цього процесу дозволяє створити рельєф з різними рівнями

висот. Після цього виконується нормалізація та долінізація для отримання більш природного вигляду.

# 3. Лістинг програми

## *3.1. Файл Main.cpp – основний файл в якому проходить генерація 3D сітки*

```
//
// Test program for the study of terrain generation
//
#include "main.h"
#include "gl_viewport.h"
#include "controls.h"
#include "camera.h"
#include "SimplexNoise.h"
#include "mesh_generator.h"

HINSTANCE g_instance;
INT panel_fixed_width;
INT log_fixed_height;
HWND h_wnd;
HWND h_panel;
HWND h_log;
HFONT h_deffont;
RECT panel_region;

GLVIEWPORT viewport;
bool keys[1024];
CCamera camera(45.f, vec3(0.f, 10.f, 0.f));

int terrain_width = 10;
int distance_between_vertices = 1;

terrain_generation_properties genprops(5.f, 1.f, 1.f, 1.f, 0.f, 1.f);

std::vector<vec3> vertices;
std::vector<vec3> colors;
```

```
std::vector<unsigned int> indices;

bool active_color = 0;

//PFNGLGENBUFFERSPROC glGenBuffers;
//PFNGLBINDBUFFERPROC glBindBuffer;
//PFNGLBUFFERDATAPROC glBufferData;
//PFNGLDELETEBUFFERSPROC glDeleteBuffers;

void edit_append_text(HWND hEdit, LPCSTR newText)
{
	int TextLen = SendMessageA(hEdit, WM_GETTEXTLENGTH, 0, 0);
	SendMessageA(hEdit, EM_SETSEL, (WPARAM)TextLen, (LPARAM)TextLen);
	SendMessageA(hEdit, EM_REPLACESEL, FALSE, (LPARAM)newText);
}

void edit_insertline(HWND hEdit, LPCSTR p_text, ...)
{
	CHAR buf[1024];
	va_list argptr;
	va_start(argptr, p_text);
	vsprintf_s(buf, sizeof(buf), p_text, argptr);
	va_end(argptr);
	strcat_s(buf, sizeof(buf), "\r\n");
	edit_append_text(hEdit, buf);
}

void resize_userinterface(HWND hWnd)
{
	RECT rect;
	GetClientRect(hWnd, &rect);
	panel_fixed_width = PERCENTOF(GetSystemMetrics(SM_CXSCREEN), 15);
	log_fixed_height = PERCENTOF(GetSystemMetrics(SM_CYSCREEN), 10);

	RECT vrect;
```

```
	vrect.left = 1;
	vrect.top = 1;
	vrect.right = rect.right - panel_fixed_width;
	vrect.bottom = rect.bottom - log_fixed_height;
	MoveWindow(viewport.h_viewport, vrect.left, vrect.top, vrect.right, vrect.bottom,
FALSE);

	RECT prect;
	prect.left = vrect.right + PADDING_PX;
	prect.top = 1;
	prect.right = rect.right - vrect.right - PADDING_PX - PADDING_PX;
	prect.bottom = rect.bottom - PADDING_PX;
	MoveWindow(h_panel, prect.left, prect.top, prect.right, prect.bottom, TRUE);

	panel_region.left = prect.left + PADDING_PX;
	panel_region.top = prect.top + PADDING_PX;
	panel_region.right = prect.right - PADDING_PX;
	panel_region.bottom = prect.bottom - PADDING_PX;

	RECT erect;
	erect.left = 1;
	erect.top = vrect.bottom + PADDING_PX;
	erect.right = prect.left - PADDING_PX;
	erect.bottom = rect.bottom - vrect.bottom - PADDING_PX - PADDING_PX;
	MoveWindow(h_log, erect.left, erect.top, erect.right, erect.bottom, TRUE);
}

void compute_pair_rects_centred(const RECT *p_src, RECT *p_rect_a, RECT *p_rect_b, LONG
*p_y, LONG height)
{
	LONG half_width = p_src->right >> 1;
	p_rect_a->left = p_src->left;
	p_rect_a->top = *p_y;
	p_rect_a->right = half_width - PADDING_PX - PADDING_PX;
	p_rect_a->bottom = height;
```

```
	p_rect_b->left = half_width;
	p_rect_b->top = *p_y;
	p_rect_b->right = half_width - PADDING_PX - PADDING_PX;
	p_rect_b->bottom = height;
	*p_y += height;
}

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

BOOL init_userinterface(HWND h_parent)
{
	return TRUE;
}

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
nCmdShow)
{
	InitCommonControls();
	h_deffont = (HFONT)GetStockObject(ANSI_VAR_FONT);

	/* main controls window class */
	WNDCLASSEXA wcex;
	memset(&wcex, NULL, sizeof(wcex));
	wcex.cbSize = sizeof(WNDCLASSEX);
	wcex.style = CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc = WndProc;
	wcex.hInstance = hInstance;
	wcex.hIcon = LoadIconA(hInstance, MAKEINTRESOURCEA(IDI_WINDOWSPROJECT1));
	wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
	wcex.hbrBackground = (HBRUSH)(COLOR_BTNFACE + 1);
	wcex.lpszMenuName = MAKEINTRESOURCEA(IDC_WINDOWSPROJECT1);
	wcex.lpszClassName = CONTROLS_WINDOW_CLASS;
	wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
	if (!RegisterClassExA(&wcex)) {
```

```
            MessageBox(HWND_DESKTOP, _T("Couldn't create main window class"),
_T("Critical error"), MB_OK | MB_ICONERROR);
            return 1;
      }

      g_instance = hInstance;
      INT sw = GetSystemMetrics(SM_CXSCREEN);
      INT sh = GetSystemMetrics(SM_CYSCREEN);
      INT width = PERCENTOF(sw, 80);
      INT height = PERCENTOF(sw, 50);
      INT xpos = (sw / 2) - (width / 2);
      INT ypos = (sh / 2) - (height / 2);
      h_wnd = CreateWindowExA(0, wcex.lpszClassName, "Window Application",
WS_OVERLAPPEDWINDOW, xpos, ypos, width, height, NULL, NULL, hInstance, NULL);
      if (!h_wnd) {
            MessageBox(HWND_DESKTOP, _T("Couldn't create window"), _T("Critical error"),
MB_OK | MB_ICONERROR);
            return 2;
      }

      panel_fixed_width = PERCENTOF(sw, 15);
      log_fixed_height = PERCENTOF(sh, 10);

      RECT rect;
      RECT vrect;
      GetClientRect(h_wnd, &rect);
      vrect.left = 1;
      vrect.top = 1;
      vrect.right = rect.right - panel_fixed_width;
      vrect.bottom = rect.bottom - log_fixed_height;

      //
      // create panel
      //
      RECT prect;
      prect.left = vrect.right + PADDING_PX;
```

```
	prect.top = 1;
	prect.right = rect.right - vrect.right - PADDING_PX - PADDING_PX;
	prect.bottom = rect.bottom - PADDING_PX;
	panel_region.left = PADDING_PX;
	panel_region.top = PADDING_PX;
	panel_region.right = prect.right - PADDING_PX;
	panel_region.bottom = prect.bottom - PADDING_PX;
	h_panel = CreateWindowExA(WS_EX_DLGMODALFRAME, wcex.lpszClassName, "", WS_VISIBLE |
WS_CHILD,
		prect.left, prect.top, prect.right, prect.bottom, h_wnd, (HMENU)0,
hInstance, NULL);


	//
	// create log
	//
	RECT erect;
	erect.left = 1;
	erect.top = vrect.bottom + PADDING_PX;
	erect.right = prect.left - PADDING_PX;
	erect.bottom = rect.bottom - vrect.bottom - PADDING_PX - PADDING_PX;
	h_log = CreateWindowExA(WS_EX_CLIENTEDGE, WC_EDIT, "- - Log started - -\r\n",
		WS_VISIBLE|WS_CHILD|ES_MULTILINE|ES_AUTOHSCROLL|ES_READONLY,
		erect.left, erect.top, erect.right, erect.bottom, h_wnd, (HMENU)0,
hInstance, NULL);
	SendMessageA(h_log, WM_SETFONT, (WPARAM)h_deffont, (LPARAM)TRUE);


	set_controls_default_font(h_deffont);
	edit_insertline(h_log, "Creating OpenGL viewport...");
	edit_insertline(h_log, "Viewport size ( %d x %d )", vrect.right, vrect.bottom);
	if (gl_viewport_init(&viewport, WS_VISIBLE | WS_CHILD, vrect.left, vrect.top,
vrect.right, vrect.bottom, h_wnd, wcex.lpszClassName, 24, 32) != VRES_OK) {
		MessageBoxA(h_wnd, "Failed to create OpenGL viewport!", "Viewport error",
MB_OK | MB_ICONERROR);
		DestroyWindow(h_wnd);
		UnregisterClassA(wcex.lpszClassName, hInstance);
		return 3;
	}
```

```
	edit_insertline(h_log, "OpenGL version: %s", glGetString(GL_VERSION)); //print
OpenGL version

	edit_insertline(h_log, "Vendor: %s", glGetString(GL_VENDOR)); //print vendor

	edit_insertline(h_log, "Renderer: %s", glGetString(GL_RENDERER)); //print renderer


	// load GL extensions

	//if (!LOAD_GL_EXTENSION(&glGenBuffers, "glGenBuffers") ||
!LOAD_GL_EXTENSION(&glBindBuffer, "glBindBuffer") || !LOAD_GL_EXTENSION(&glBufferData,
"glBufferData") || !LOAD_GL_EXTENSION(&glDeleteBuffers, "glDeleteBuffers")) {

	//    //remove all user interface

	//    return 1;

	//}


	genprops.seed = 0.f;


	//

	// add controls to main panel

	//


	LONG x = PADDING_PX, y = PADDING_PX;

	RECT left_rect, right_rect;

	add_label(h_panel, panel_region.left, panel_region.top, panel_region.right,
panel_region.top + 20, "Управление генерацией");

	y += 20;

	compute_pair_rects_centred(&panel_region, &left_rect, &right_rect, &y, 20);

	add_editbox(h_panel, left_rect.left, left_rect.top, left_rect.right,
left_rect.bottom, IDC_SEEDEDIT, "");

	add_button(h_panel, right_rect.left, right_rect.top, right_rect.right,
right_rect.bottom, "Задать seed", IDC_SETSEED);

	y += UP_PADDING_PX;


	LONG control_width = panel_region.right - PADDING_PX - PADDING_PX;

	HWND h_terrain_width_trackbar = add_trackbar_with_description_autoheight(h_panel,
panel_region.left, &y, control_width, 25, UP_PADDING_PX, IDC_WIDTHTRACK, "Ширина
ландшафта");

	SendMessageA(h_terrain_width_trackbar, TBM_SETRANGE, (WPARAM)TRUE,
(LPARAM)MAKELONG(5, 1000));

	SendMessageA(h_terrain_width_trackbar, TBM_SETPOS, (WPARAM)TRUE,
(LPARAM)(int)genprops.world_width);
```

```
	y += UP_PADDING_PX;


	HWND h_dist_between_verts = add_trackbar_with_description_autoheight(h_panel,
panel_region.left, &y, control_width, 25, UP_PADDING_PX, IDC_VERTSDIST, "Шаг между
вершинами");

	SendMessageA(h_dist_between_verts, TBM_SETRANGE, (WPARAM)TRUE, (LPARAM)MAKELONG(1,
10));

	SendMessageA(h_dist_between_verts, TBM_SETPOS, (WPARAM)TRUE,
(LPARAM)(int)genprops.verts_step);


	HWND h_frequency = add_trackbar_with_description_autoheight(h_panel,
panel_region.left, &y, control_width, 25, UP_PADDING_PX, IDC_FREQTRACK, "Частота");

	genprops.frequency = 0.1f;

	SendMessageA(h_frequency, TBM_SETRANGE, (WPARAM)TRUE, (LPARAM)MAKELONG(1, 1000));

	SendMessageA(h_frequency, TBM_SETPOS, (WPARAM)TRUE, (LPARAM)1);


	HWND h_amplitude = add_trackbar_with_description_autoheight(h_panel,
panel_region.left, &y, control_width, 25, UP_PADDING_PX, IDC_AMPLITUDE, "Амплитуда");

	genprops.amplitude = 0.1f;

	SendMessageA(h_frequency, TBM_SETRANGE, (WPARAM)TRUE, (LPARAM)MAKELONG(1, 1000));

	SendMessageA(h_frequency, TBM_SETPOS, (WPARAM)TRUE, (LPARAM)1);


	HWND h_smallnoise = add_trackbar_with_description_autoheight(h_panel,
panel_region.left, &y, control_width, 25, UP_PADDING_PX, IDC_SMALLNOISE, "Мелкий шум");

	genprops.small_noiseK = 0.f;

	SendMessageA(h_smallnoise, TBM_SETRANGE, (WPARAM)TRUE, (LPARAM)MAKELONG(0, 1000));

	SendMessageA(h_smallnoise, TBM_SETPOS, (WPARAM)TRUE, (LPARAM)0);


	HWND h_multiplier = add_trackbar_with_description_autoheight(h_panel,
panel_region.left, &y, control_width, 25, UP_PADDING_PX, IDC_NOISEMULTIPLIER, "Усиление
амплитуды");

	genprops.multiplier = 1.f;

	SendMessageA(h_multiplier, TBM_SETRANGE, (WPARAM)TRUE, (LPARAM)MAKELONG(1, 10));

	SendMessageA(h_multiplier, TBM_SETPOS, (WPARAM)TRUE, (LPARAM)0);

	y += UP_PADDING_PX;


	add_button(h_panel, panel_region.left, y, control_width, 20, "Сгенерировать",
IDC_STARTGENERATION);
```

```
	ShowWindow(h_wnd, SW_SHOW);
	UpdateWindow(h_wnd);

	generate_terrain_mesh(&genprops, colors, vertices, indices);

	//
	// setup scene parameters
	//
	GetClientRect(viewport.h_viewport, &rect);
	glViewport(0, 0, rect.right, rect.bottom);
	glMatrixMode(GL_PROJECTION); //select projection matrix
	glLoadIdentity(); //reset projection matrix

	// prevent division by zero
	if (!rect.bottom)
		rect.bottom = 1;

	gluPerspective(45.0, rect.right / (double)rect.bottom, 0.1, 10000.0); //set
perspective projection
	glMatrixMode(GL_MODELVIEW); //select modelview matrix
	glLoadIdentity(); //reset modelview matrix
	glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
	glEnableClientState(GL_VERTEX_ARRAY);

    MSG msg;
    while (GetMessage(&msg, nullptr, 0, 0)) {
		TranslateMessage(&msg);
		DispatchMessage(&msg);

		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
		glLoadIdentity();

		camera.UpdateCameraState(viewport.h_viewport);
		camera.Look();
		camera.Move(keys, 1.f);
```

```
            // Enable client-side arrays
            glEnableClientState(GL_VERTEX_ARRAY);
            if (active_color)
                  glEnableClientState(GL_COLOR_ARRAY);



            // Set up vertex and color pointers
            glVertexPointer(3, GL_FLOAT, 0, vertices.data());

            if (active_color)
                  glColorPointer(3, GL_FLOAT, 0, colors.data());



            // Rendering with smooth color gradient based on height using glDrawElements
            glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT,
indices.data());

            // Disable client-side arrays
            glDisableClientState(GL_VERTEX_ARRAY);
            if (active_color)
                  glDisableClientState(GL_COLOR_ARRAY);

            SwapBuffers(viewport.h_device_context);
    }
      glDisableClientState(GL_VERTEX_ARRAY);
      gl_viewport_shutdown(&viewport);
    return msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
      RECT rect;
    switch (message)
    {
```

```
    case WM_COMMAND:
        {
                    DWORD wmId = LOWORD(wParam);
                    DWORD param = HIWORD(wParam);
            switch (wmId)
            {
            case IDM_ABOUT:
                DialogBox(g_instance, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                break;
            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;

                    case IDC_STARTGENERATION:
                           srand((unsigned int)__rdtsc());
                           genprops.seed = (float)rand();
                           generate_terrain_mesh(&genprops, colors, vertices, indices);
                           break;

            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
        }
        break;

      case WM_HSCROLL:
             genprops.world_width = (float)SendMessageA(GetDlgItem(hWnd, IDC_WIDTHTRACK),
TBM_GETPOS, 0, 0);
             genprops.verts_step = (float)SendMessageA(GetDlgItem(hWnd, IDC_VERTSDIST),
TBM_GETPOS, 0, 0);
             genprops.frequency = (float)SendMessageA(GetDlgItem(hWnd, IDC_FREQTRACK),
TBM_GETPOS, 0, 0) * 0.1f;
             genprops.amplitude = (float)SendMessageA(GetDlgItem(hWnd, IDC_AMPLITUDE),
TBM_GETPOS, 0, 0) * 0.1f;
             genprops.small_noiseK = (float)SendMessageA(GetDlgItem(hWnd,
IDC_SMALLNOISE), TBM_GETPOS, 0, 0) * 0.01f;
```

```
            genprops.multiplier = (float)SendMessageA(GetDlgItem(hWnd,
IDC_NOISEMULTIPLIER), TBM_GETPOS, 0, 0);
            generate_terrain_mesh(&genprops, colors, vertices, indices);
            break;

    case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            EndPaint(hWnd, &ps);
        }
        break;

    case WM_DESTROY:
        PostQuitMessage(0);
        break;

      //case WM_ERASEBKGND:
      //    if (hWnd == viewport.h_viewport)
      //          return 1;

      //    break;

      case WM_KEYDOWN:
            if (wParam == VK_F1) {
                  static bool active_camera = true;
                  camera.SetActive(active_camera);
                  active_camera = !active_camera;
            }
            if (wParam == VK_F2) {
                  active_color = !active_color;
            }

            keys[wParam] = true;
            break;
```

```
        case WM_KEYUP:
                keys[wParam] = false;
                break;

        case WM_SIZE: {
                if (hWnd == h_wnd) {
                        resize_userinterface(hWnd);
                }
                else if (hWnd == viewport.h_viewport) {
                        GetClientRect(viewport.h_viewport, &rect);
                        glMatrixMode(GL_PROJECTION);
                        glLoadIdentity();
                        glViewport(0, 0, rect.right, rect.bottom);

                        // prevent division by zero
                        if (!rect.bottom)
                                rect.bottom = 1;

                        gluPerspective(45.0, rect.right / (double)rect.bottom, 0.1, 10000.0);
                        glMatrixMode(GL_MODELVIEW);
                        glLoadIdentity();
                }
                break;
        }

    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
```

```
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}

bool load_gl_extension(LPVOID *ppfuncptr, LPCSTR procname)
{
       char msgbuf[512];
       *ppfuncptr = wglGetProcAddress(procname);
       if (!(*ppfuncptr)) {
             sprintf_s(msgbuf, sizeof(msgbuf), "Failed to load extension proc: %s",
procname);
             MessageBoxA(HWND_DESKTOP, msgbuf, "GL extension fail", MB_ICONWARNING);
             return false;
       }
       return true;
}
```

# 4. Тестове виконання програми

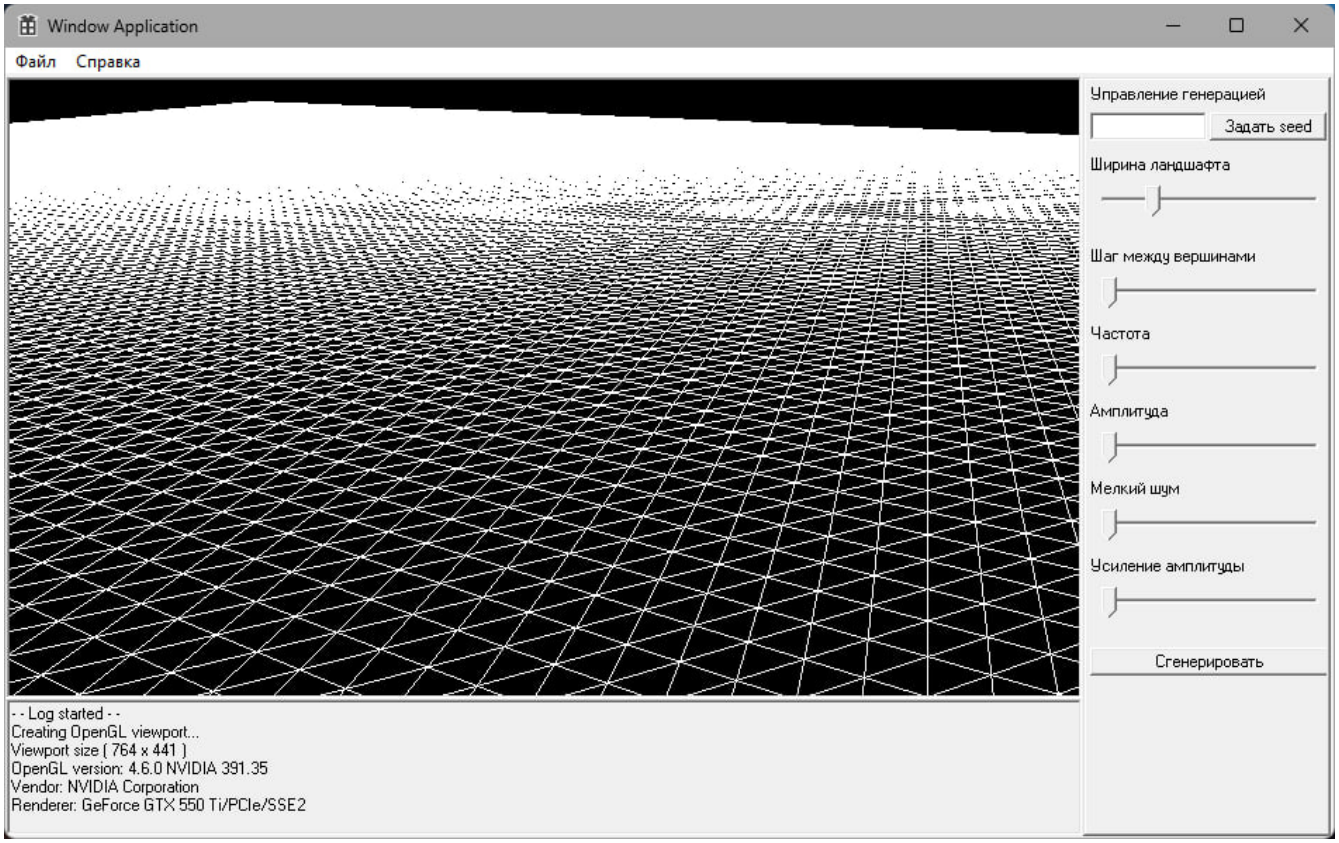


Рис. 4.1 – Тест 1(Запуск програми)

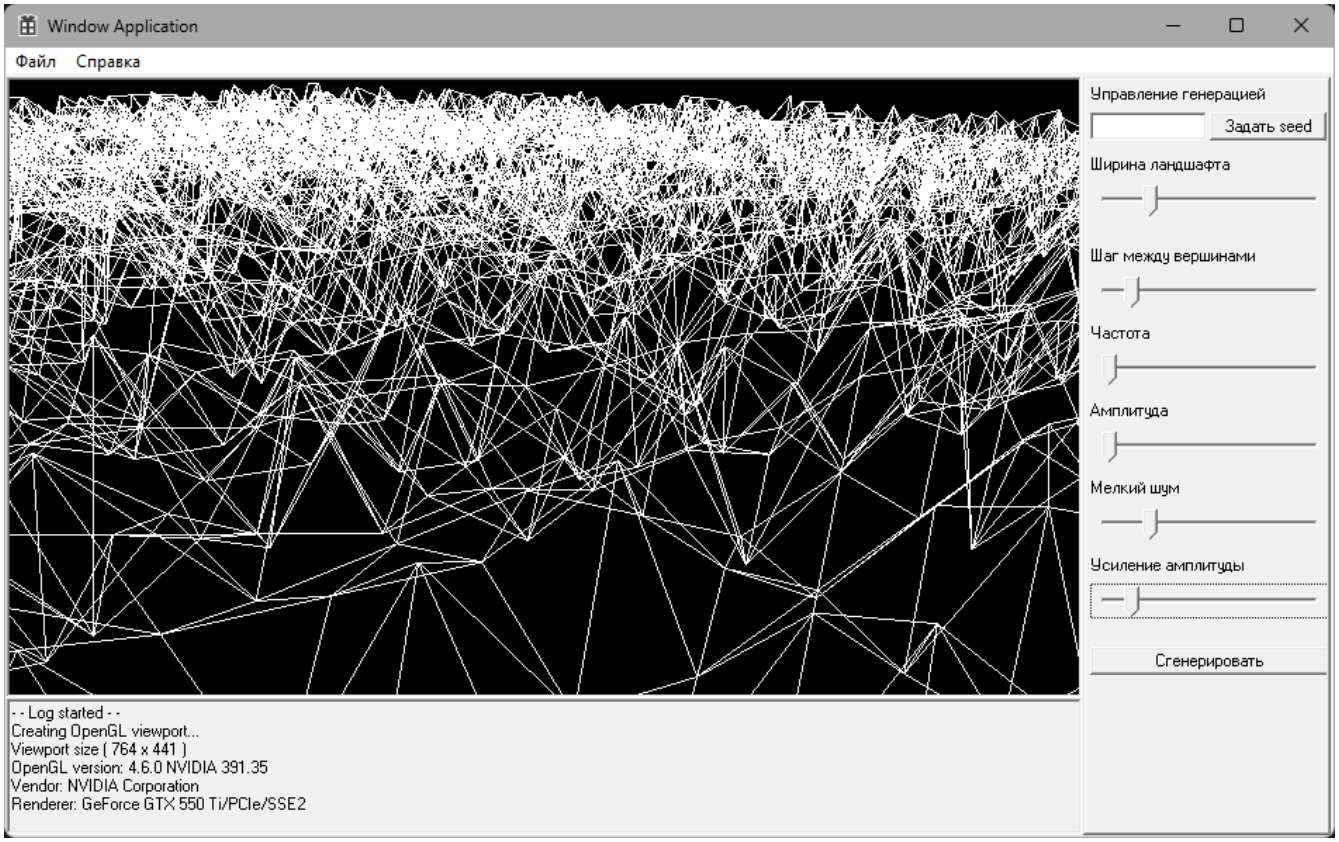


Рис. 4.2 – Тест 2(За допомогою меню можливо змінювати рельєф)

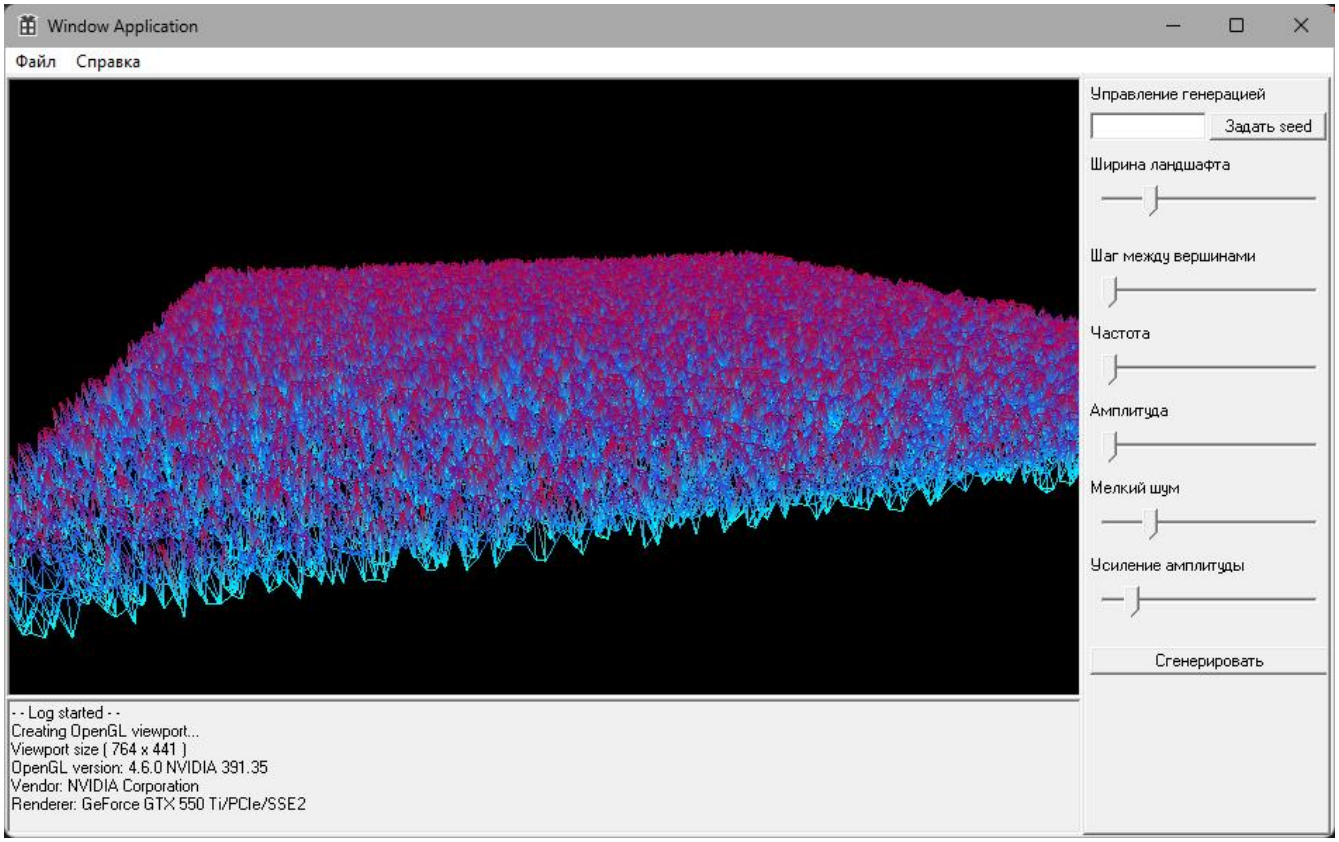


Рис. 4.4 – Тест 3(За допомогою клавіш F2 можливо включити та виключити колір висот)

## 5. Контроль виконання завдання

| № п/п | Складність | Вимоги | Результат |
|---|---|---|---|
| 1 | Базовий рівень | Формування ландшафту у вигляді каркасної моделі | + |
| 2 | | Градієнтне забарвлення поверхні в залежно від висоти ландшафту | + |
| 3 | | ООП. Реалізація модельованого об'єкта у вигляді одного або кількох класів власної розробки | + |
| 4 | Підвищений рівень | Інтерактивне керування параметрами модельованого ландшафту (порожнина, рівень деталізації, розмальовка тощо) | + |

## Висновки

Опанував процес генерації ландшафту за допомогою алгоритму Шуму Перлина. Розробив програму, що не лише формує ландшафт, але й дозволяє користувачеві управляти режимом відображення, налаштовувати рівень деталізації висот, а також вносити зміни у висоти даного ландшафту.