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 № 3

з дисципліни «Програмування віртуальної реальності»
(назва дисципліни)
на тему: «Створення та моделювання об'єктів віртуальної реальності (Експорт та імпорт 3D моделей)»

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

Створити програмний продукт, який дозволить користувачам ефективно працювати з 3D моделями у віртуальному середовищі. Використовуючи OpenGL на мові програмування C++, програма повинна забезпечувати можливість експорту та імпорту 3D об'єктів для зручного обміну та редагування даних між різними програмами та проєктами.

2. Теоретичні відомості

3D моделі мають різні формати, такі як Wavefront OBJ, кожен з яких використовується для конкретних завдань. Експорт і імпорт 3D моделей полягає у збереженні та завантаженні їх у файли для подальшого використання в інших програмах чи проектах.

У віртуальній реальності якісні та оптимізовані 3D моделі грають важливу роль у створенні реалістичних віртуальних середовищ. Ці моделі можуть відображати об'єкти, персонажів або поверхні, надаючи іммерсивність виртуальному світу.

При інтеграції процесу експорту та імпорту важливо враховувати сумісність з обраними форматами файлів. Параметри якості, розміру файлу та зручності роботи грають рішальну роль у виборі оптимального формату для задач проекту.

3. Лістинг програми

3.1. Файл Source.cpp

```
#include "OBJFile.h"
#include <GL/freeglut.h>

OBJFile model;


//чтение названия модели из файла., для тестов
string getFileName() {

	string filename;
	string buffer;
	ifstream getFile("model_name.txt");
	if (!getFile) {
		cout << "cannot find file - > model_name.txt." << endl;
		exit(-1);
	}
	getline(getFile, buffer);

	filename = buffer;
	getFile.close();
	return filename;
```

```
}


void init() {

	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	GLfloat light_pos[] = { -1.0f, 10.0f, 100.0f, 1.0f };
	glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
	glClearColor(0.2f, 0.2f, 0.2f, 1.0f);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(20.0, 1.0, 1.0, 2000.0);
	glMatrixMode(GL_MODELVIEW);
	glEnable(GL_BLEND);
	glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
	glEnable(GL_LINE_SMOOTH);
	glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	glEnable(GL_TEXTURE_2D);
	glEnable(GL_DEPTH_TEST);
	model.loadOBJFile(getFileName());
}

float cameraDistance = 20;
float cameraAngleX = 30.0f;
float cameraAngleY = 0.0f;
int mouseX, mouseY;
bool isMouseDragging = false;

// Mouse callback for camera rotation
void mouse(int button, int state, int x, int y) {
	if (button == GLUT_LEFT_BUTTON) {
		if (state == GLUT_DOWN) {
			isMouseDragging = true;
			mouseX = x;
			mouseY = y;
		}
		else if (state == GLUT_UP) {
			isMouseDragging = false;
		}
	}
}

// Mouse motion callback for camera rotation
void motion(int x, int y) {
	if (isMouseDragging) {
		int deltaX = x - mouseX;
		int deltaY = y - mouseY;

		cameraAngleY += deltaX * 0.2f;
		cameraAngleX += deltaY * 0.2f;

		mouseX = x;
		mouseY = y;

		glutPostRedisplay();
	}
}
```

```
void updateCamera() {
	glLoadIdentity();
	glTranslatef(0.0f, 0.0f, -cameraDistance);
	glRotatef(cameraAngleX, 1.0f, 0.0f, 0.0f);
	glRotatef(cameraAngleY, 0.0f, 1.0f, 0.0f);
}


void display() {

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	updateCamera();

	model.draw();
	glutSwapBuffers();
}

void mouseWheel(int button, int dir, int x, int y) {
	if (dir > 0) {
		cameraDistance -= 5.0f;
	}
	else {
		cameraDistance += 5.0f;
	}

	glutPostRedisplay();
}

int main(int argc, char** argv) {

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH | GLUT_MULTISAMPLE);
	glEnable(GL_MULTISAMPLE);
	glHint(GL_MULTISAMPLE_FILTER_HINT_NV, GL_NICEST);
	glutSetOption(GLUT_MULTISAMPLE, 8);
	int POS_X = (glutGet(GLUT_SCREEN_WIDTH) - WIDTH) >> 1;
	int POS_Y = (glutGet(GLUT_SCREEN_HEIGHT) - HEIGTH) >> 1;
	glutInitWindowPosition(POS_X, POS_Y);
	glutInitWindowSize(WIDTH, HEIGTH);
	glutCreateWindow("Obj Viewer");
	init();
	//glutKeyboardFunc(keyboard);
	glutMouseFunc(mouse);
	glutMouseWheelFunc(mouseWheel);
	glutMotionFunc(motion);
	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```

3.2. Файл OBJFile.cpp

```
#include "OBJFile.h"

void OBJFile::apply_scaling(int* indices, int size) {
	for (int i = 0; i < size; i++) {
		indices[i] = indices[i] * scalingFactor;
	}
}
```

```
int OBJFile::count_char(string& str, char ch) {
	int c = 0;
	int length = str.length() - 1;
	for (int i = 0; i < length; i++) {
		if (str[i] == ch)
			c++;
	}
	return c;
}

bool OBJFile::has_double_slash(string& str) {
	int length = str.length() - 2;
	for (int i = 0; i < length; i++) {
		if (str[i] == '/' && str[i + 1] == '/')
			return true;
	}
	return false;
}

void OBJFile::face_3v(std::string& line) {
	int v0, v1, v2;
	sscanf_s(line.c_str(), "f %d %d %d", &v0, &v1, &v2);
	int* v = new int[3]{ v0 - 1, v1 - 1, v2 - 1 };
	apply_scaling(v, 3);
	faces.push_back(Faces(3, v, NULL));
}

void OBJFile::face_3vt(std::string& line) {
	int v0, v1, v2, t0, t1, t2;
	sscanf_s(line.c_str(), "f %d/%d %d/%d %d/%d", &v0, &t0, &v1, &t1, &v2, &t2);
	int* v = new int[3]{ v0 - 1, v1 - 1, v2 - 1 };
	int* t = new int[3]{ t0 - 1, t1 - 1, t2 - 1 };
	apply_scaling(v, 3);
	apply_scaling(t, 3);
	faces.push_back(Faces(3, v, NULL));
}

void OBJFile::face_3vn(std::string& line) {
	int v0, v1, v2, n;
	sscanf_s(line.c_str(), "f %d//%d %d//%d %d//%d", &v0, &n, &v1, &n, &v2, &n);
	int* v = new int[3]{ v0 - 1, v1 - 1, v2 - 1 };
	apply_scaling(v, 3);
	faces.push_back(Faces(3, v, n - 1));
}

void OBJFile::face_3vtn(std::string& line) {
	int v0, v1, v2, t0, t1, t2, n;
	sscanf_s(line.c_str(), "f %d/%d/%d %d/%d/%d %d/%d/%d", &v0, &t0, &n, &v1, &t1, &n,
&v2, &t2, &n);
	int* v = new int[3]{ v0 - 1, v1 - 1, v2 - 1 };
	int* t = new int[3]{ t0 - 1, t1 - 1, t2 - 1 };
	apply_scaling(v, 3);
	apply_scaling(t, 3);
	faces.push_back(Faces(3, v, n - 1));
}

void OBJFile::face_4v(std::string& line) {
	int v0, v1, v2, v3;
	sscanf_s(line.c_str(), "f %d %d %d %d", &v0, &v1, &v2, &v3);
	int* v = new int[4]{ v0 - 1, v1 - 1, v2 - 1, v3 - 1 };
	apply_scaling(v, 4);
	faces.push_back(Faces(4, v, NULL));
```

```
}

void OBJFile::face_4vt(std::string& line) {
	int v0, v1, v2, v3, t0, t1, t2, t3;
	sscanf_s(line.c_str(), "f %d/%d %d/%d %d/%d %d/%d", &v0, &t0, &v1, &t1, &v2, &t2,
&v3, &t3);
	int* v = new int[4]{ v0 - 1, v1 - 1, v2 - 1, v3 - 1 };
	int* t = new int[4]{ t0 - 1, t1 - 1, t2 - 1, t3 - 1 };
	apply_scaling(v, 4);
	apply_scaling(t, 4);
	faces.push_back(Faces(4, v, NULL));
}

void OBJFile::face_4vn(std::string& line) {
	int v0, v1, v2, v3, n;
	sscanf_s(line.c_str(), "f %d//%d %d//%d %d//%d %d//%d", &v0, &n, &v1, &n, &v2, &n,
&v3, &n);
	int* v = new int[4]{ v0 - 1, v1 - 1, v2 - 1, v3 - 1 };
	apply_scaling(v, 4);
	faces.push_back(Faces(4, v, n - 1));
}

void OBJFile::face_4vtn(std::string& line) {
	int v0, v1, v2, v3, t0, t1, t2, t3, n;
	sscanf_s(line.c_str(), "f %d/%d/%d %d/%d/%d %d/%d/%d %d/%d/%d", &v0, &t0, &n, &v1,
&t1, &n, &v2, &t2, &n, &v3,
		&t3, &n);
	int* v = new int[4]{ v0 - 1, v1 - 1, v2 - 1, v3 - 1 };
	int* t = new int[4]{ t0 - 1, t1 - 1, t2 - 1, t3 - 1 };
	apply_scaling(v, 4);
	apply_scaling(t, 4);
	faces.push_back(Faces(4, v, n - 1));
}
```

3.3. Файл loader.cpp

```
#include "OBJFile.h"

void OBJFile::loadOBJFile(string filename) {

	vector<string> lines;
	string line;
	ifstream iFile(filename);

	if (!iFile) {
		cout << "error load file - > " << filename << endl;
		exit(-1);
	}

	while (!iFile.eof()) {
		getline(iFile, line);
		lines.push_back(line);
	}
	iFile.close();

	float x, y, z;

	for (auto& line : lines) {
		if (line[0] == 'v') {
			if (line[1] == ' ') {
```

```
                        sscanf_s(line.c_str(), "v %f %f %f", &x, &y, &z);
                        verticies.push_back(new float[3]{ x, y, z });
                }
                else if (line[1] == 'n') {
                        sscanf_s(line.c_str(), "vn %f %f %f", &x, &y, &z);
                        normals.push_back(new float[3]{ x, y, z });
                }
        }
        else if (line[0] == 'f') {
                if (line[1] == ' ') {
                        int edge = count_char(line, ' ');
                        int count_slash = count_char(line, '/');
                        if (count_slash == 0) {
                                if (edge == 3)
                                        face_3v(line);
                                else
                                        face_4v(line);
                        }
                        else if (count_slash == edge) {
                                if (edge == 3)
                                        face_3vt(line);
                                else
                                        face_4vt(line);
                        }
                        else if (count_slash == edge * 2) {
                                if (has_double_slash(line)) {
                                        if (edge == 3)
                                                face_3vn(line);
                                        else
                                                face_4vn(line);
                                }
                                else {
                                        if (edge == 3)
                                                face_3vtn(line);
                                        else
                                                face_4vtn(line);
                                }
                        }
                }
        }
}

list = glGenLists(1);
glNewList(list, GL_COMPILE);
for (Faces& face : faces) {
        if (face.normals != -1)
                glNormal3fv(normals[face.normals]);
        else
                glDisable(GL_LIGHTING);
        glBegin(GL_POLYGON);
        for (int i = 0; i < face.edge; i++) {
                glVertex3fv(verticies[face.verticies[i]]);
        }
        glEnd();

        if (face.normals == -1)
                glEnable(GL_LIGHTING);
}

glEndList();
cout << "Model : " << filename << endl;
cout << "Verticies : " << verticies.size() << endl;
```

```
cout << "Normals : " << normals.size() << endl;
cout << "Faces : " << faces.size() << endl;

for (float* f : verticies)
	delete f;
verticies.clear();

for (float* f : normals)
	delete f;
normals.clear();
faces.clear();
}
```

4. Тестове виконання програми

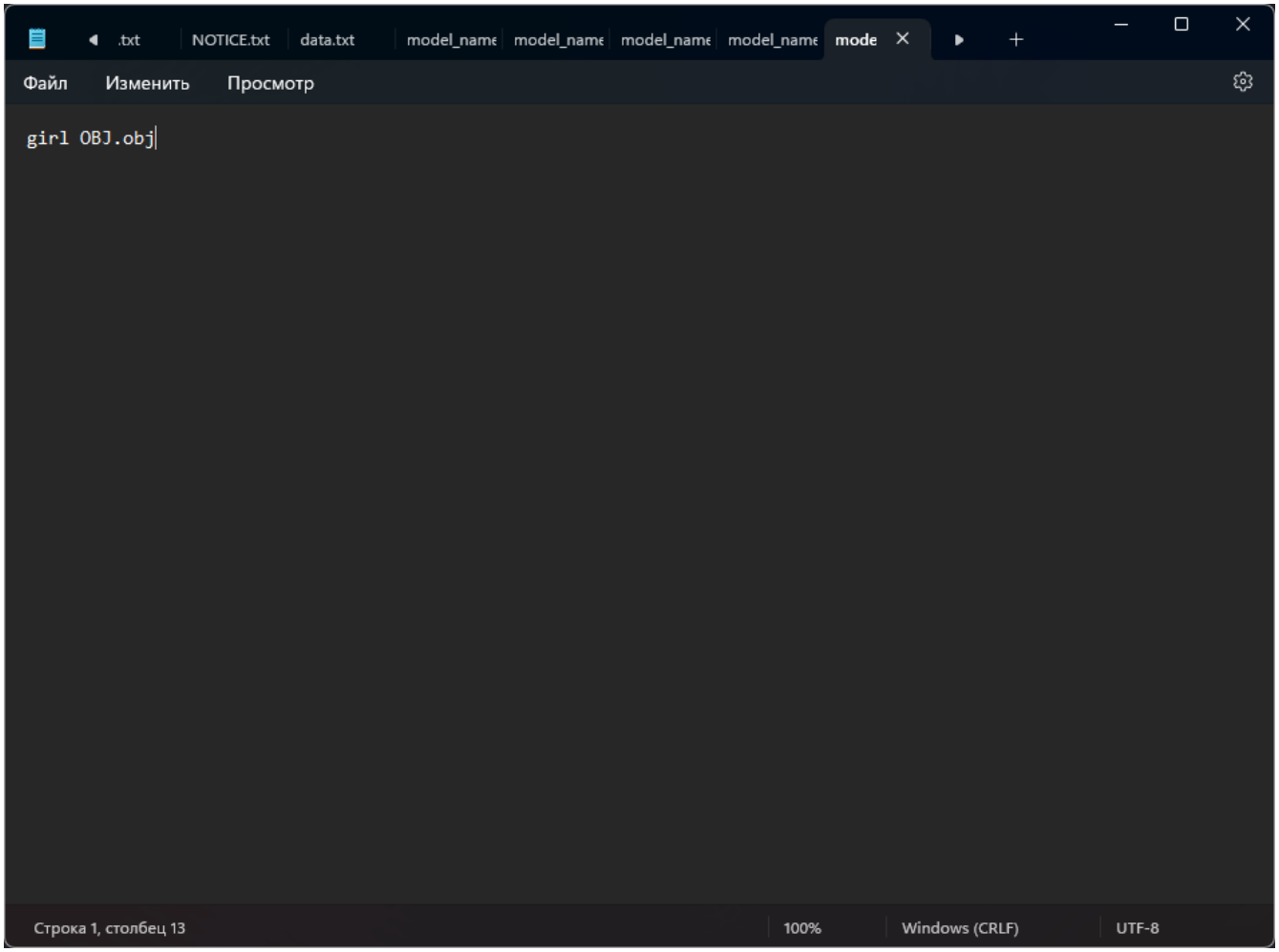

Рис. 1 – Пишемо назву моделі в txt файл.

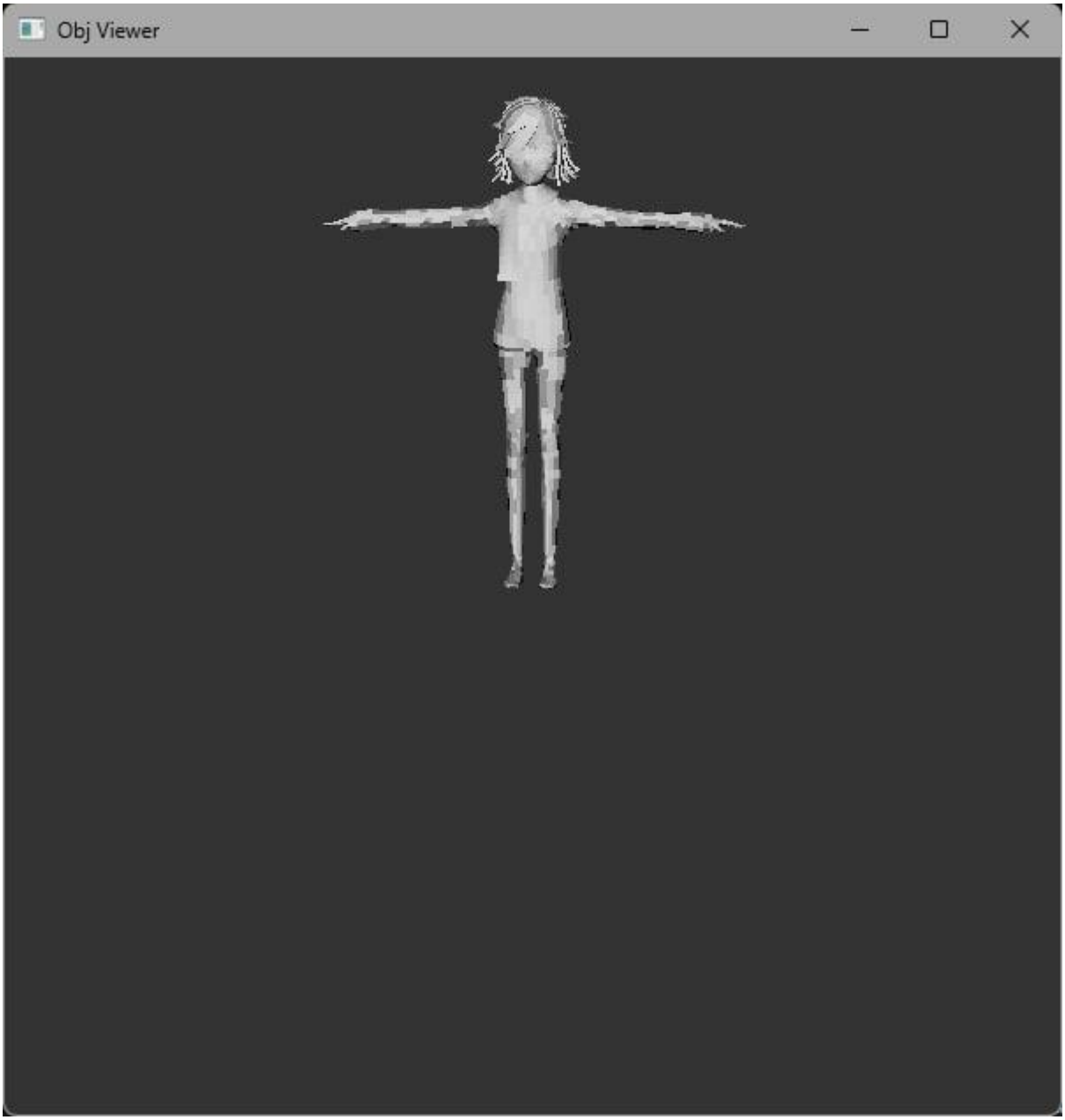

Рис. 2 – Показ моделі дівчинки

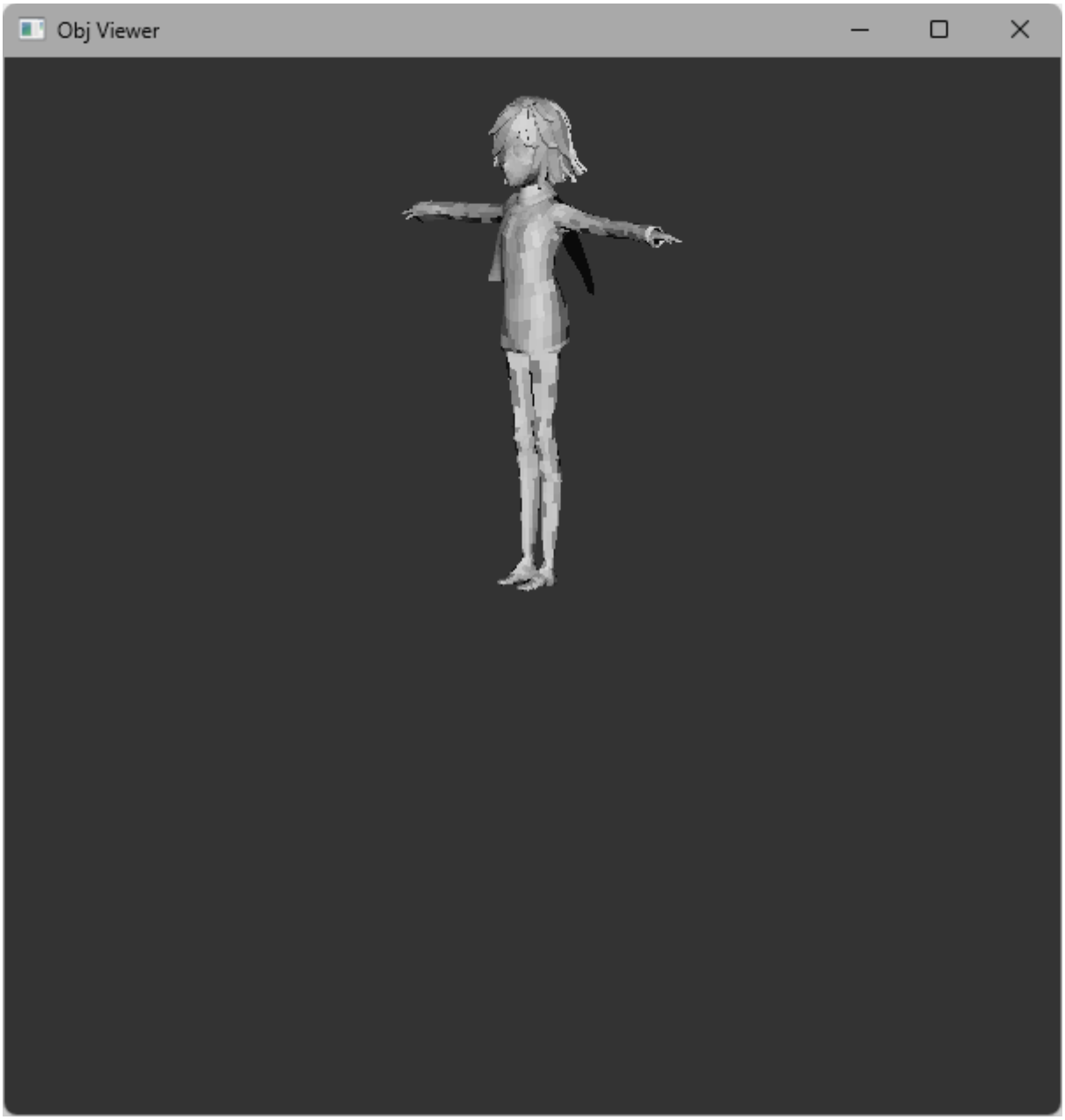

Рис. 3 – Показ моделі дівчинки (інший ракурс)

Висновки

Під Під час виконання цієї лабораторної роботи я навчився імпортувати готові 3D моделі до мого проекту та виводити їх на екран. Додатково, я впровадив функціонал для роботи з освітленням та можливість переміщення світла.