**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

**ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи №5

# на тему: “Проектування динамічних тривимірних об’єктів на екран монітора з урахуванням освітлення та текстурних шаблонів”

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

**Лектор:**

проф. кафедри ПЗ

Журавчак Л. М.

**Виконав:**

студент групи ПЗ-33

Шлом’як Д. В.

**Прийняв:**

проф. кафедри ПЗ

Журавчак Л. М.

Львів – 2025

# Тема: Проектування динамічних тривимірних об’єктів на екран монітора з урахуванням освітлення та текстурних шаблонів.

**Мета:** Навчитися будувати тривимірні об’єкти, покриті текстурою, з подальшим їхнім обертанням та масштабуванням на основі застосування базових геометричних перетворень OpenGL.

**Теоретичні відомості**

Для задання різних перетворень об’єктів сцени в OpenGL використовують операції над матрицями, при цьому розрізняють кілька типів матриць, зокрема вигляду і проекції. Перша матриця визначає перетворення об’єкта у світових координатах, такі як паралельний перенос, зміна масштабу і поворот. Матриця проекцій задає, як будуть проектуватися тривимірні об’єкти на площину екрана (у віконні координати). Для того, щоб вибрати матрицю для опрацювання, використовують команду **glMatrixMode(mode),** виклик якої зі значенням параметра **GL\_MODELVIEW** чи **GL\_PROJECTION**вмикає режими роботи з матрицею виду чи проекцій відповідно.

**Робота з матрицями проекції та вигляду**

Отож, щоб працювати з матрицею проекції, треба активувати відповідний режим командою **glMatrixMode(GL\_PROJECTION)**. Параметри ортогональної проекції вибирають за допомогою функції **glOrtho(…)**, для отримання перспективної проекції сцени використовують функцію **gluPerspective(…)** або **glFrustum(…)**. Якщо команда проектування не задана явно, OpenGL виконує ортогональну проекцію сцени за замовчуванням. Параметри спостереження задають за допомогою функції **gluLookAt(…)**.

Перед роботою з об’єктами необхідно задати режим роботи з матрицею вигляду, виконавши команду **glMatrixMode(GL\_MODELVIEW)**. Прямокутну область виведення результатів визначають функцією **glViewport(…)**. Для правильного відображення сцени у випадку зміни розмірів вікна здебільшого створюють спеціальну функцію, що має такий загальний вигляд:

**void resize(int width,int height)**

**{**

**glViewport(0,0,width,height);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**glOrtho(…);**

**gluLookAt(…);**

**glMatrixMode(GL\_MODELVIEW);**

**}**

Тривимірні примітиви

Для створення тривимірних примітивів використовують відповідні функції додаткових бібліотек OpenGL – **gluSphere()**, **gluCylinder()**, **gluDisk()** із GLU, **glutSolidSphere()**, **glutSolidCube()**, **glutSolidCone()** тощо із GLUT, **auxSolidCube()**, **auxSolidBox()**, **auxSolidTorus()**, **auxSolidCylinder()**, **auxSolidCone()** тощо із GLAUX і т.д. Для коректної побудови цих об’єктів необхідно видаляти невидимі лінії та поверхні, увімкнувши відповідний режим командою**glEnable(GL\_DEPTH\_TEST)**.

Накладання текстури

Щоб створити повноцінну тривимірну сцену, часто на тривимірні об’єкти потрібно накладати зображення – для цього використовують текстури. У загальному випадку для накладання текстури на тривимірний об’єкт потрібно:

* увімкнути режим використання текстури (**glEnable(GL\_Texture\_2d)**);
* завантажити графічний файл (наприклад, функцією **auxDIBImageLoad** із GLAUX);
* створити ім’я-ідентифікатор текстури (**glGenTextures**);
* зробити його активним (**glBindTexture**);
* створити саму текстуру в пам’яті (**glTexImage2D**);
* установити параметри текстури (**glTexParameter**);
* установити параметри взаємодії текстури з об’єктом (**glTexEnv**);
* зв’язати координати текстури з об’єктом (**glTexCoord2d** – вручну, **glTexGen** – автоматично).

Освітлення

Важливою деталлю у тривимірних сценах є освітлення: без нього об’єкти здаватимуться плоскими. За замовчуванням освітлення вимкнуто, його потрібно увімкнути командою **glEnable(GL\_LIGHTING)**. Для створення реалістичних зображень необхідно визначити параметри матеріалу, з якого зроблено об’єкт, та кількість і властивості джерел світла. Властивості матеріалу задають за допомогою функції **glMaterial(…)**. Джерело освітлення вмикають командою **glEnable(GL\_LIGHTn)**, де n може набувати значень від 0 до 7, та задають його властивості за допомогою функції **glLight(…)**. Джерел можна використовувати кілька. Для визначення глобальних параметрів освітлення використовують команду **glLightModel(…)**.

**Завдання**

Завдання повинні бути реалізовані у вікні з наявними елементами керування для запуску виконання окремих дій або етапів роботи. Для оформлення зовнішнього вигляду тривимірних об’єктів (окрім однотонного зафарбування) необхідно застосувати накладання текстури. Для формування текстур матеріалів (граніту, мармуру тощо) можна скористатися текстурним заповненням фігур у програмі Word.

Для освітлення сцени почергово передбачити рух чотирьох основних типів джерел світла: фонове освітлення, точкові, прожектори та віддалені. Реалістичність зображення забезпечити за допомогою функції радіального загасання.

Додати ефект туману, врахувавши лінійну та експоненційну функції загасання з глибиною.

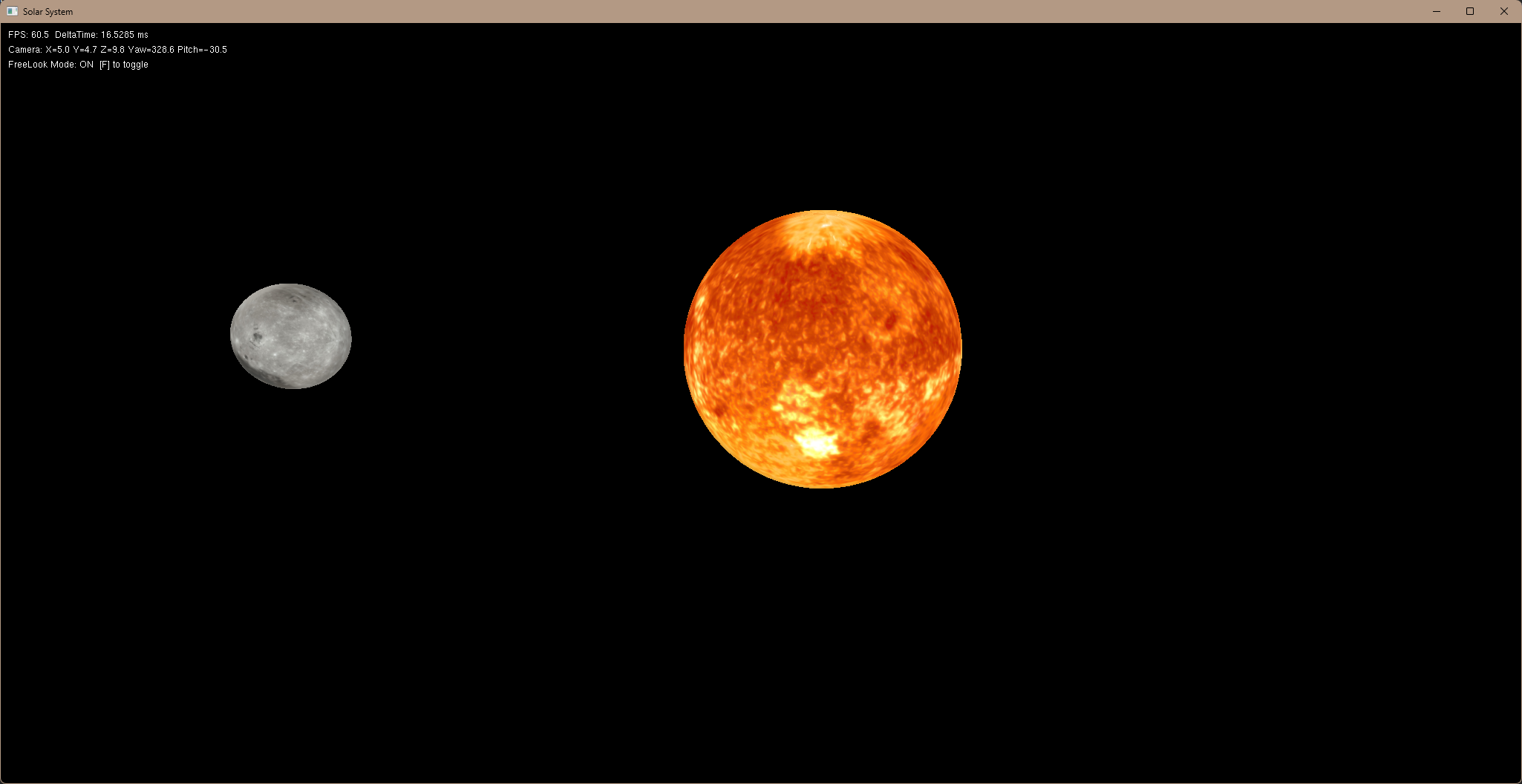
У всіх важливих місцях програми необхідно давати коментарі з описом призначення поданого далі фрагмента коду.

10. Намалювати систему **зоря – планета**. Зобразити на планеті **поверхню місячного типу**. Реалізувати обертання планети навколо зорі та своєї вертикальної осі. Визначити джерелом освітлення жовтого кольору зорю.

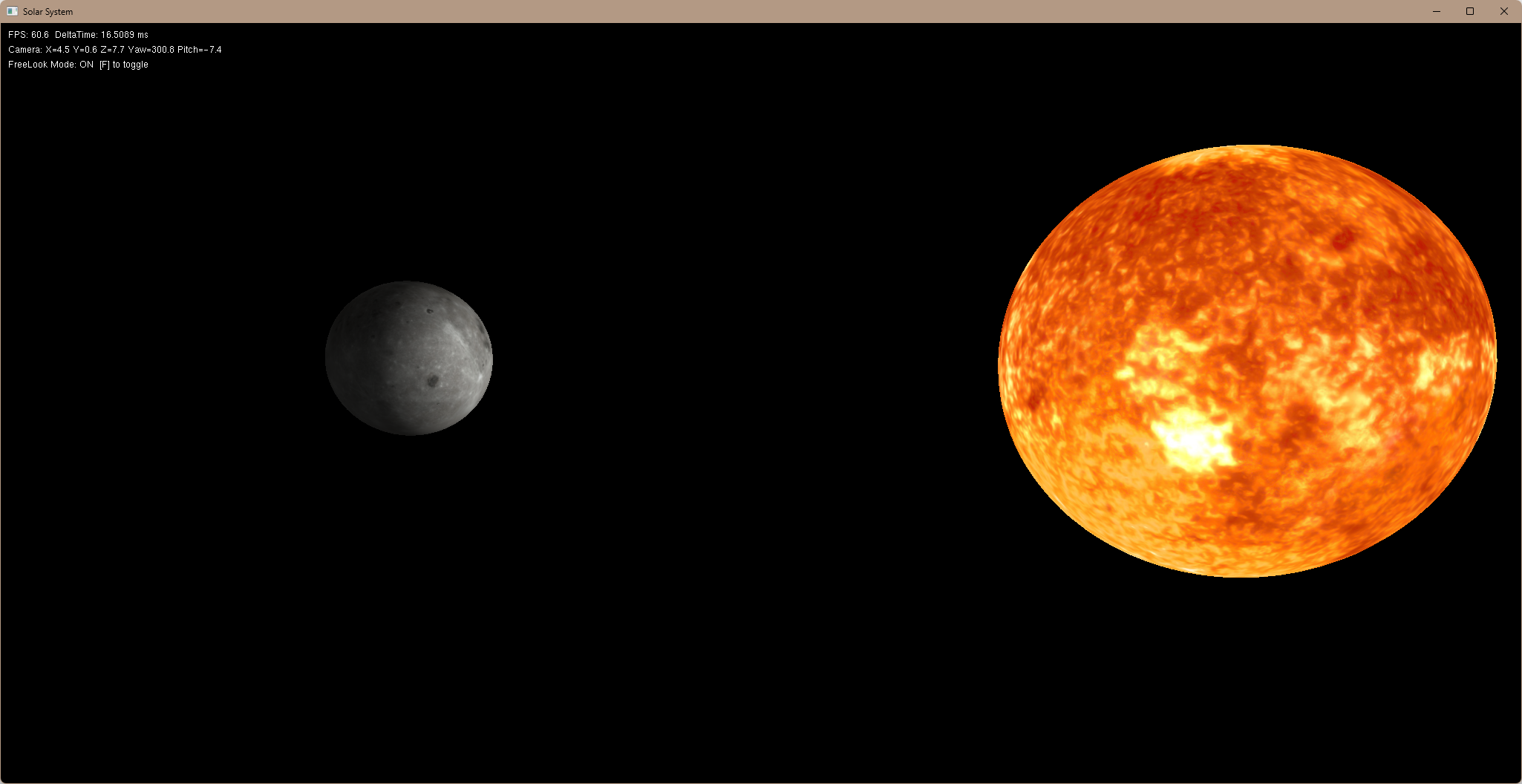
**Код програми**

#include <GL/glut.h>  
#include <GL/gl.h>  
#include <GL/glu.h>  
#include <stdlib.h>  
#include <stdio.h>  
#define \_USE\_MATH\_DEFINES  
#include <cmath>  
#include <chrono>  
  
#define STB\_IMAGE\_IMPLEMENTATION  
#include "ELightSources.h"  
#include "stb\_image.h"  
  
// Light IDs (OpenGL has GL\_LIGHT0 to GL\_LIGHT7)  
#define AMBIENT\_LIGHT GL\_LIGHT0  
#define POINT\_LIGHT GL\_LIGHT1  
#define DIRECTIONAL\_LIGHT GL\_LIGHT2  
#define SPOTLIGHT GL\_LIGHT3  
  
// Змінні для контролю обертання та масштабування  
GLfloat planetRotation = 0.0f; // Обертання планети навколо своєї осі  
GLfloat planetRevolution = 0.0f; // Обертання планети навколо зорі  
GLfloat zoomFactor = -20.0f; // Масштаб сцени  
GLfloat viewAngle = 50.0f; // Кут огляду  
  
// Змінні для текстур  
GLuint starTexture;  
GLuint moonTexture;  
  
// Параметри для туману  
GLfloat fogColor[4] = {0.0f, 0.0f, 0.0f, 1.0f};  
bool enableFog = false;  
  
// Змінна для контролю анімації  
bool animationActive = true;  
  
// Змінні для контролю камери  
float cameraX = 0.0f;  
float cameraY = 0.0f;  
float cameraZ = 20.0f;  
float cameraYaw = 0.0f; // Поворот вліво-вправо  
float cameraPitch = 0.0f; // Поворот вгору-вниз  
float cameraSpeed = 0.2f; // Швидкість руху камери  
  
// Змінні для контролю миші  
int mouseX = 0, mouseY = 0;  
bool mouseLeftDown = false;  
bool mouseRightDown = false;  
bool mouseMiddleDown = false;  
float mouseRotateSpeed = 7.0f; // Швидкість повороту камери  
float mouseMoveSpeed = 0.05f; // Швидкість руху камери  
  
// Змінні для вільного огляду камерою  
bool freeLookMode = true;  
  
// Змінні для вимірювання deltaTime  
double currentTime = 0.0;  
double previousTime = 0.0;  
double deltaTime = 0.0; // Час між кадрами у секундах  
double fps = 0.0; // Частота кадрів за секунду  
  
// Константи швидкості руху та обертання  
const float BASE\_ROTATE\_SPEED = 60.0f; // градусів за секунду  
const float BASE\_MOVE\_SPEED = 500.0f; // одиниць за секунду  
  
// Константи відмальовки кадрів  
const double TARGET\_FPS = 60.0;  
const double FRAME\_TIME = 1.0 / TARGET\_FPS;  
  
// Константи осі планет  
const double AXIS\_ANGLE = 23.5f;  
  
bool keyStates[256] = { false };  
  
void updateMovement();  
GLuint createFallbackTexture(const char\* filename);  
  
#ifdef \_WIN64 || \_WIN32  
#include <windows.h>  
void sleepForSeconds(double seconds) {  
 Sleep(seconds \* 1000);  
}  
#endif  
  
// Отримання поточного часу в секундах  
double getTimeInSeconds() {  
 auto now = std::chrono::high\_resolution\_clock::now();  
 auto duration = now.time\_since\_epoch();  
 return std::chrono::duration<double>(duration).count();  
}  
  
GLuint loadTexture(const char\* filename) {  
 GLuint texture;  
 int width, height, channels;  
 unsigned char\* data;  
  
 // Load image using stb\_image  
 stbi\_set\_flip\_vertically\_on\_load(1); // Flip images vertically (OpenGL expects bottom-left as origin)  
 data = stbi\_load(filename, &width, &height, &channels, STBI\_rgb\_alpha);  
  
 if (!data) {  
 printf("Error loading texture '%s': %s\n", filename, stbi\_failure\_reason());  
  
 // Fall back to a simple procedural texture  
 return createFallbackTexture(filename);  
 }  
  
 // Generate and bind texture  
 glGenTextures(1, &texture);  
 glBindTexture(GL\_TEXTURE\_2D, texture);  
  
 // Set texture parameters  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);  
  
 // Enable mipmapping for older OpenGL versions  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  
  
 // Build mipmaps manually for older OpenGL versions  
 gluBuild2DMipmaps(GL\_TEXTURE\_2D, GL\_RGBA, width, height,  
 GL\_RGBA, GL\_UNSIGNED\_BYTE, data);  
  
 // Free the image data as it's now in GPU memory  
 stbi\_image\_free(data);  
  
 printf("Texture '%s' loaded successfully (%dx%d, %d channels)\n",  
 filename, width, height, channels);  
  
 return texture;  
}  
  
// Fallback function to create a procedural texture when file loading fails  
GLuint createFallbackTexture(const char\* filename) {  
 GLuint texture;  
  
 printf("Creating fallback texture instead of '%s'\n", filename);  
  
 // Create texture object  
 glGenTextures(1, &texture);  
 glBindTexture(GL\_TEXTURE\_2D, texture);  
  
 // Set texture parameters  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  
  
 // Create checkerboard pattern as fallback  
 const int size = 64;  
 unsigned char checkerboard[size][size][4];  
  
 for(int i = 0; i < size; i++) {  
 for(int j = 0; j < size; j++) {  
 int c = ((i & 0x8) == 0 ^ (j & 0x8) == 0) \* 255;  
 checkerboard[i][j][0] = c;  
 checkerboard[i][j][1] = 0;  
 checkerboard[i][j][2] = c;  
 checkerboard[i][j][3] = 255;  
 }  
 }  
  
 // Create mipmapped texture with gluBuild2DMipmaps  
 gluBuild2DMipmaps(GL\_TEXTURE\_2D, GL\_RGBA, size, size,  
 GL\_RGBA, GL\_UNSIGNED\_BYTE, checkerboard);  
  
 return texture;  
}  
  
// Функція ініціалізації  
void init(void) {  
 glEnable(GL\_LIGHTING);  
 glEnable(GL\_NORMALIZE); // For proper lighting calculations  
  
 // Колір фону - чорний (космос)  
 glClearColor(0.0f, 0.0f, 0.0f, 1.0f);  
  
 glutSetCursor(GLUT\_CURSOR\_NONE); // Приховати курсор  
  
 // Увімкнення тесту глибини для правильного відображення 3D об'єктів  
 glEnable(GL\_DEPTH\_TEST);  
  
 // Увімкнення згладжування  
 glEnable(GL\_SMOOTH);  
  
 // Ініціалізація джерела світла (зоря)  
 glEnable(GL\_LIGHTING);  
  
 // Налаштування матеріалу за замовчуванням  
 GLfloat mat\_ambient[] = {0.7f, 0.7f, 0.7f, 1.0f};  
 GLfloat mat\_diffuse[] = {0.8f, 0.8f, 0.8f, 1.0f};  
 GLfloat mat\_specular[] = {0.1f, 0.1f, 0.1f, 1.0f};  
 GLfloat mat\_shininess[] = {15.0f};  
  
 glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);  
 glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);  
 glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);  
 glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);  
  
 // Завантаження текстур  
 moonTexture = loadTexture("..\\Textures\\moon-texture.jpg");  
 starTexture = loadTexture("..\\Textures\\star\_texture.jpg");  
  
 // Налаштування туману для космічного простору  
 glFogi(GL\_FOG\_MODE, GL\_EXP);  
 glFogfv(GL\_FOG\_COLOR, fogColor);  
 glFogf(GL\_FOG\_DENSITY, 0.05f);  
 glHint(GL\_FOG\_HINT, GL\_DONT\_CARE);  
  
 // Ініціалізація часу  
 previousTime = getTimeInSeconds();  
}  
  
// Функція малювання зорі  
void drawStar() {  
  
 glPushAttrib(GL\_LIGHTING\_BIT);  
  
 // Відключаємо освітлення для самої зорі, щоб вона світилася рівномірно  
 if (glIsEnabled(POINT\_LIGHT)) {  
 GLfloat mat\_emission[] = {1.0f, 1.0f, 1.0f, 1.0f}; // Bright white emission  
 glMaterialfv(GL\_FRONT, GL\_EMISSION, mat\_emission);  
 }  
  
 // Накладаємо текстуру зорі  
 glEnable(GL\_TEXTURE\_2D);  
 glBindTexture(GL\_TEXTURE\_2D, starTexture);  
  
 glPushMatrix();  
  
 glRotatef(90.0f, 1.0f, 0.0f, 0.0f);  
  
 // Малюємо сферу для зорі  
 GLUquadricObj \*quadric = gluNewQuadric();  
 gluQuadricTexture(quadric, GL\_TRUE);  
 gluSphere(quadric, 2.0f, 50, 50);  
 gluDeleteQuadric(quadric);  
  
 glPopMatrix();  
  
 glDisable(GL\_TEXTURE\_2D);  
  
 glPopAttrib();  
}  
  
// Функція малювання планети з місячною поверхнею  
void drawPlanet() {  
 // Накладаємо текстуру місячної поверхні  
 glEnable(GL\_TEXTURE\_2D);  
 glBindTexture(GL\_TEXTURE\_2D, moonTexture);  
  
 // Місячно-сірий колір для планети  
 glColor3f(1.0f, 1.0f, 1.0f);  
  
 glPushMatrix();  
  
 glRotatef(90.0f, 1.0f, 0.0f, 0.0f);  
  
  
 // Малюємо сферу для планети  
 GLUquadricObj \*quadric = gluNewQuadric();  
 gluQuadricTexture(quadric, GL\_TRUE);  
 gluSphere(quadric, 0.8f, 30, 30);  
 gluDeleteQuadric(quadric);  
  
 glPopMatrix();  
  
 glDisable(GL\_TEXTURE\_2D);  
}  
  
// Функція для відображення тексту на екрані  
void renderText(float x, float y, const char\* text, void\* font = GLUT\_BITMAP\_HELVETICA\_12) {  
 glDisable(GL\_LIGHTING);  
 glDisable(GL\_TEXTURE\_2D);  
 glColor3f(1.0f, 1.0f, 1.0f);  
  
 // Встановлюємо позицію тексту  
 glMatrixMode(GL\_PROJECTION);  
 glPushMatrix();  
 glLoadIdentity();  
 gluOrtho2D(0, glutGet(GLUT\_WINDOW\_WIDTH), 0, glutGet(GLUT\_WINDOW\_HEIGHT));  
  
 glMatrixMode(GL\_MODELVIEW);  
 glPushMatrix();  
 glLoadIdentity();  
  
 // Виводимо текст  
 glRasterPos2f(x, y);  
 for (const char\* c = text; \*c; c++) {  
 glutBitmapCharacter(font, \*c);  
 }  
  
 // Відновлюємо матриці  
 glPopMatrix();  
 glMatrixMode(GL\_PROJECTION);  
 glPopMatrix();  
 glMatrixMode(GL\_MODELVIEW);  
  
 glEnable(GL\_TEXTURE\_2D);  
 glEnable(GL\_LIGHTING);  
}  
  
void lightUpdate() {  
 // Ambient light  
 GLfloat ambientColor[] = {0.9f, 0.9f, 0.9f, 1.0f};  
 glLightfv(AMBIENT\_LIGHT, GL\_AMBIENT, ambientColor);  
  
 // Point light  
 // Джерело світла розташоване в центрі зорі  
 GLfloat point\_light\_position[] = {0.0f, 0.0f, 0.0f, 1.0f};  
 GLfloat point\_light\_ambient[] = {0.0f, 0.0f, 0.0f, 1.0f};  
 GLfloat point\_light\_diffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};  
 GLfloat point\_light\_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};  
  
 glLightfv(POINT\_LIGHT, GL\_POSITION, point\_light\_position);  
 glLightfv(POINT\_LIGHT, GL\_AMBIENT, point\_light\_ambient);  
 glLightfv(POINT\_LIGHT, GL\_DIFFUSE, point\_light\_diffuse);  
 glLightfv(POINT\_LIGHT, GL\_SPECULAR, point\_light\_specular);  
  
 // Затухання світла з відстанню  
 glLightf(POINT\_LIGHT, GL\_CONSTANT\_ATTENUATION, 1.0f);  
 glLightf(POINT\_LIGHT, GL\_LINEAR\_ATTENUATION, 0.0f);  
 glLightf(POINT\_LIGHT, GL\_QUADRATIC\_ATTENUATION, 0.0001f);  
  
  
 // Directional light  
 GLfloat directional\_light\_position[] = {1.0f, 1.0f, 0.0f, 0.0f};  
 GLfloat directional\_light\_ambient[] = {0.0f, 0.0f, 0.0f, 1.0f};  
 GLfloat directional\_light\_diffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};  
 GLfloat directional\_light\_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};  
  
 glLightfv(DIRECTIONAL\_LIGHT, GL\_POSITION, directional\_light\_position);  
 glLightfv(DIRECTIONAL\_LIGHT, GL\_AMBIENT, directional\_light\_ambient);  
 glLightfv(DIRECTIONAL\_LIGHT, GL\_DIFFUSE, directional\_light\_diffuse);  
 glLightfv(DIRECTIONAL\_LIGHT, GL\_SPECULAR, directional\_light\_specular);  
 // Затухання світла з відстанню  
 glLightf(DIRECTIONAL\_LIGHT, GL\_CONSTANT\_ATTENUATION, 1.0f);  
 glLightf(DIRECTIONAL\_LIGHT, GL\_LINEAR\_ATTENUATION, 0.0f);  
 glLightf(DIRECTIONAL\_LIGHT, GL\_QUADRATIC\_ATTENUATION, 0.0f);  
  
 GLfloat spotlight\_position[] = {0.0f, 100.0f, 0.0f, 1.0f};  
 GLfloat spotlight\_direction[] = {0.0f, -1.0f, 0.0f};  
 GLfloat spotlight\_ambient[] = {0.0f, 0.0f, 0.0f, 1.0f};  
 GLfloat spotlight\_diffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};  
 GLfloat spotlight\_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};  
  
 glLightfv(SPOTLIGHT, GL\_POSITION, spotlight\_position);  
 glLightfv(SPOTLIGHT, GL\_SPOT\_DIRECTION, spotlight\_direction);  
 glLightfv(SPOTLIGHT, GL\_AMBIENT, spotlight\_ambient);  
 glLightfv(SPOTLIGHT, GL\_DIFFUSE, spotlight\_diffuse);  
 glLightfv(SPOTLIGHT, GL\_SPECULAR, spotlight\_specular);  
 // Затухання світла з відстанню  
 glLightf(SPOTLIGHT, GL\_SPOT\_CUTOFF, 2.0f);  
 glLightf(SPOTLIGHT, GL\_SPOT\_EXPONENT, 70.0f);  
 glLightf(SPOTLIGHT, GL\_CONSTANT\_ATTENUATION, 1.0f);  
 glLightf(SPOTLIGHT, GL\_LINEAR\_ATTENUATION, 0.0f);  
 glLightf(SPOTLIGHT, GL\_QUADRATIC\_ATTENUATION, 0.0001f);  
}  
  
// Функція малювання сцени  
void display(void) {  
 // Вимірювання часу для розрахунку deltaTime  
 currentTime = getTimeInSeconds();  
 deltaTime = currentTime - previousTime;  
  
 // Calculate target frame delay in seconds  
 float targetFrameRate = 1.0 / TARGET\_FPS;  
  
 // If we're running too fast, wait until it's time to render the next frame  
 if (deltaTime < targetFrameRate) {  
 // Calculate how much time we need to wait  
 float sleepTime = targetFrameRate - deltaTime;  
  
 // Use a sleep function appropriate for your system  
 // e.g., Sleep(sleepTime \* 1000) on Windows or usleep(sleepTime \* 1000000) on Unix  
 sleepForSeconds(sleepTime);  
  
 // Update currentTime after sleeping  
 currentTime = getTimeInSeconds();  
 deltaTime = currentTime - previousTime;  
 }  
  
 previousTime = currentTime;  
  
 // Розрахунок FPS  
 if (deltaTime > 0) {  
 fps = 1.0 / deltaTime;  
 }  
  
 lightUpdate();  
  
 // Continue with rendering regardless of FPS  
 updateMovement();  
  
 // Очищення буферів кольору та глибини  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
  
 // Ініціалізуємо матрицю моделювання/виду  
 glMatrixMode(GL\_MODELVIEW);  
 glLoadIdentity();  
  
 // Розрахунок напрямку погляду камери на основі кутів  
 float lookX = cameraX + sin(cameraYaw \* M\_PI / 180.0);  
 float lookY = cameraY + sin(cameraPitch \* M\_PI / 180.0);  
 float lookZ = cameraZ - cos(cameraYaw \* M\_PI / 180.0);  
  
 // Розташування камери з урахуванням кутів повороту  
 gluLookAt(cameraX, cameraY, cameraZ, // Позиція камери  
 lookX, lookY, lookZ, // Напрямок погляду  
 0.0, 1.0, 0.0); // Вектор "вгору"  
  
 // Малюємо зорю в центрі  
 glPushMatrix();  
 drawStar();  
 glPopMatrix();  
  
 // Малюємо планету на орбіті  
 glPushMatrix();  
  
 // Обертання навколо зорі  
 glRotatef(planetRevolution, 0.0f, 1.0f, 0.0f);  
  
 // Відстань від зорі  
 glTranslatef(8.0f, 0.0f, 0.0f);  
  
 // Нахил осі планети  
 glRotatef(-AXIS\_ANGLE, 0.0f, 0.0f, 1.0f);  
  
 // Обертання планети навколо своєї осі  
 glRotatef(planetRotation, 0.0f, 1.0f, 0.0f);  
  
 drawPlanet();  
 glPopMatrix();  
  
 // Відображення інформації про стан системи  
 char buffer[128];  
 sprintf(buffer, "FPS: %.1f DeltaTime: %.4f ms", fps, deltaTime \* 1000);  
 renderText(10, glutGet(GLUT\_WINDOW\_HEIGHT) - 20, buffer);  
  
 sprintf(buffer, "Camera: X=%.1f Y=%.1f Z=%.1f Yaw=%.1f Pitch=%.1f",  
 cameraX, cameraY, cameraZ, cameraYaw, cameraPitch);  
 renderText(10, glutGet(GLUT\_WINDOW\_HEIGHT) - 40, buffer);  
  
 sprintf(buffer, "FreeLook Mode: %s [F] to toggle", freeLookMode ? "ON" : "OFF");  
 renderText(10, glutGet(GLUT\_WINDOW\_HEIGHT) - 60, buffer);  
  
 // Відображення результату на екрані  
 glutSwapBuffers();  
}  
  
// Функція для зміни розміру вікна  
void reshape(int width, int height) {  
 // Запобігання ділення на нуль  
 if (height == 0)  
 height = 1;  
  
 // Встановлення розміру вікна перегляду  
 glViewport(0, 0, width, height);  
  
 // Перехід до матриці проекції  
 glMatrixMode(GL\_PROJECTION);  
 glLoadIdentity();  
  
 // Встановлення перспективи  
 gluPerspective(viewAngle, (GLfloat)width / (GLfloat)height, 0.1f, 100.0f);  
  
 // Повернення до матриці моделювання  
 glMatrixMode(GL\_MODELVIEW);  
}  
  
// Функція для анімації  
void animate(int value) {  
 if (animationActive) {  
 // Оновлення обертальних кутів з урахуванням deltaTime для плавності руху  
 planetRotation += BASE\_ROTATE\_SPEED \* deltaTime; // Обертання планети навколо своєї осі  
 if (planetRotation > 360.0f)  
 planetRotation -= 360.0f;  
  
 planetRevolution += BASE\_ROTATE\_SPEED \* 0.1f \* deltaTime; // Обертання планети навколо зорі  
 if (planetRevolution > 360.0f)  
 planetRevolution -= 360.0f;  
 }  
  
 // Перемалювання сцени  
 glutPostRedisplay();  
  
 // Повторний виклик анімації  
 glutTimerFunc(16, animate, 0); // ~60 FPS  
}  
  
// Обробник натискання кнопок миші  
void mouseFunc(int button, int state, int x, int y) {  
 mouseX = x;  
 mouseY = y;  
  
 // Обробка кнопок миші  
 if (button == GLUT\_LEFT\_BUTTON) {  
 if (state == GLUT\_DOWN)  
 mouseLeftDown = true;  
 else  
 mouseLeftDown = false;  
 }  
  
 if (button == GLUT\_RIGHT\_BUTTON) {  
 if (state == GLUT\_DOWN)  
 mouseRightDown = true;  
 else  
 mouseRightDown = false;  
 }  
  
 if (button == GLUT\_MIDDLE\_BUTTON) {  
 if (state == GLUT\_DOWN)  
 mouseMiddleDown = true;  
 else  
 mouseMiddleDown = false;  
 }  
}  
  
// Обробник руху миші з натиснутими кнопками  
void mouseMotionFunc(int x, int y) {  
 // Розрахунок зміщення курсора миші  
 int dx = x - mouseX;  
 int dy = y - mouseY;  
  
 // Оновлення останньої позиції миші  
 mouseX = x;  
 mouseY = y;  
  
 // Швидкість повороту і руху з урахуванням deltaTime  
 float adjustedRotateSpeed = mouseRotateSpeed;  
 float adjustedMoveSpeed = mouseMoveSpeed;  
  
 // Обробка правої кнопки - переміщення камери  
 if (mouseRightDown) {  
 // Переміщення камери вперед-назад та вліво-вправо  
 float deltaForward = -dy \* adjustedMoveSpeed;  
 float deltaStrafe = dx \* adjustedMoveSpeed;  
  
 // Обчислення напрямків руху з урахуванням кутів камери  
 float forwardX = sin(cameraYaw \* M\_PI / 180.0);  
 float forwardZ = -cos(cameraYaw \* M\_PI / 180.0);  
  
 float strafeX = sin((cameraYaw + 90.0) \* M\_PI / 180.0);  
 float strafeZ = -cos((cameraYaw + 90.0) \* M\_PI / 180.0);  
  
 // Оновлення позиції камери  
 cameraX += forwardX \* deltaForward + strafeX \* deltaStrafe;  
 cameraZ += forwardZ \* deltaForward + strafeZ \* deltaStrafe;  
  
 int centerX = glutGet(GLUT\_WINDOW\_WIDTH) / 2;  
 int centerY = glutGet(GLUT\_WINDOW\_HEIGHT) / 2;  
 if (x != centerX || y != centerY) {  
 mouseX = centerX;  
 mouseY = centerY;  
 glutWarpPointer(centerX, centerY);  
 }  
 }  
 // Обертання камери якщо не правою кнопкою і не вільний огляд  
 else if (mouseLeftDown && !freeLookMode) {  
 // Поворот камери вліво-вправо (Yaw)  
 cameraYaw += dx \* adjustedRotateSpeed \* deltaTime;  
 if (cameraYaw > 360.0f)  
 cameraYaw -= 360.0f;  
 if (cameraYaw < 0.0f)  
 cameraYaw += 360.0f;  
  
 // Поворот камери вгору-вниз (Pitch) з обмеженням  
 cameraPitch += dy \* adjustedRotateSpeed \* deltaTime;  
 if (cameraPitch > 89.0f)  
 cameraPitch = 89.0f;  
 if (cameraPitch < -89.0f)  
 cameraPitch = -89.0f;  
 }  
  
 // Обробка середньої кнопки - рух камери вгору-вниз  
 if (mouseMiddleDown) {  
 cameraY += dy \* adjustedMoveSpeed;  
 }  
  
 // Перемалювати сцену  
 glutPostRedisplay();  
}  
  
// Обробник пасивного руху миші (без натиснутих кнопок)  
void mousePassiveMotionFunc(int x, int y) {  
 // Розрахунок зміщення курсора миші  
 int dx = x - mouseX;  
 int dy = y - mouseY;  
  
 // Оновлення останньої позиції миші  
 mouseX = x;  
 mouseY = y;  
  
 // Повороти камери в режимі вільного огляду  
 if (freeLookMode) {  
 // Поворот камери вліво-вправо (Yaw)  
 cameraYaw += dx \* mouseRotateSpeed \* deltaTime;  
 if (cameraYaw > 360.0f)  
 cameraYaw -= 360.0f;  
 if (cameraYaw < 0.0f)  
 cameraYaw += 360.0f;  
  
 // Поворот камери вгору-вниз (Pitch) з обмеженням  
 cameraPitch += -1 \* dy \* mouseRotateSpeed \* deltaTime;  
 if (cameraPitch > 89.0f)  
 cameraPitch = 89.0f;  
 if (cameraPitch < -89.0f)  
 cameraPitch = -89.0f;  
  
 // Опціонально: Центрування курсора миші для неперервного повороту  
 if (freeLookMode) {  
 int centerX = glutGet(GLUT\_WINDOW\_WIDTH) / 2;  
 int centerY = glutGet(GLUT\_WINDOW\_HEIGHT) / 2;  
 if (x != centerX || y != centerY) {  
 mouseX = centerX;  
 mouseY = centerY;  
 glutWarpPointer(centerX, centerY);  
 }  
 }  
  
 // Перемалювати сцену  
 glutPostRedisplay();  
 }  
}  
  
void keyboardDown(unsigned char key, int x, int y) {  
 keyStates[key] = true;  
}  
  
void updateMovement() {  
 // Speed based on deltaTime for smooth movement  
 float moveDistance = BASE\_MOVE\_SPEED \* deltaTime;  
  
 // Check WASD keys and adjust camera  
 if (keyStates['w']) {  
 cameraX += sin(cameraYaw \* M\_PI / 180.0) \* moveDistance \* deltaTime;  
 cameraZ -= cos(cameraYaw \* M\_PI / 180.0) \* moveDistance \* deltaTime;  
 }  
 if (keyStates['s']) {  
 cameraX -= sin(cameraYaw \* M\_PI / 180.0) \* moveDistance \* deltaTime;  
 cameraZ += cos(cameraYaw \* M\_PI / 180.0) \* moveDistance \* deltaTime;  
 }  
 if (keyStates['a']) {  
 cameraX -= cos(cameraYaw \* M\_PI / 180.0) \* moveDistance \* deltaTime;  
 cameraZ -= sin(cameraYaw \* M\_PI / 180.0) \* moveDistance \* deltaTime;  
 }  
 if (keyStates['d']) {  
 cameraX += cos(cameraYaw \* M\_PI / 180.0) \* moveDistance \* deltaTime;  
 cameraZ += sin(cameraYaw \* M\_PI / 180.0) \* moveDistance \* deltaTime;  
 }  
 if (keyStates['q']) {  
 cameraY -= moveDistance \* deltaTime;  
 }  
 if (keyStates['e']) {  
 cameraY += moveDistance \* deltaTime;  
 }  
 if (keyStates['+']) {  
 cameraZ -= moveDistance \* 5.0f;  
 }  
 if (keyStates['-']) {  
 cameraZ += moveDistance \* 5.0f;  
 }  
}  
  
// Обробник клавіатури  
void keyboardUp(unsigned char key, int x, int y) {  
 keyStates[key] = false;  
  
 switch (key) {  
 case 27: // Клавіша Escape - вихід з програми  
 exit(0);  
 break;  
 case ' ': // Пробіл - пауза/продовження анімації  
 animationActive = !animationActive;  
 break;  
 case 'r': // Скидання позиції камери  
 cameraX = 0.0f;  
 cameraY = 0.0f;  
 cameraZ = 20.0f;  
 cameraYaw = 0.0f;  
 cameraPitch = 0.0f;  
 break;  
 case 'f': // Перемикання режиму вільного огляду  
 freeLookMode = !freeLookMode;  
 if (freeLookMode) {  
 glutSetCursor(GLUT\_CURSOR\_NONE); // Приховати курсор  
 } else {  
 glutSetCursor(GLUT\_CURSOR\_INHERIT); // Показати курсор  
 }  
 break;  
 case '1': // Фонове освітлення  
 glEnable(AMBIENT\_LIGHT);  
 glDisable(POINT\_LIGHT);  
 glDisable(DIRECTIONAL\_LIGHT);  
 glDisable(SPOTLIGHT);  
 break;  
 case '2':  
 glDisable(AMBIENT\_LIGHT);  
 glEnable(POINT\_LIGHT);  
 glDisable(DIRECTIONAL\_LIGHT);  
 glDisable(SPOTLIGHT);  
 break;  
 case '3':  
 glDisable(AMBIENT\_LIGHT);  
 glDisable(POINT\_LIGHT);  
 glEnable(DIRECTIONAL\_LIGHT);  
 glDisable(SPOTLIGHT);  
 break;  
 case '4':  
 glDisable(AMBIENT\_LIGHT);  
 glDisable(POINT\_LIGHT);  
 glDisable(DIRECTIONAL\_LIGHT);  
 glEnable(SPOTLIGHT);  
 break;  
 case '5':  
 enableFog = !enableFog;  
 if (enableFog)  
 glEnable(GL\_FOG);  
 else  
 glDisable(GL\_FOG);  
 break;  
 }  
 glutPostRedisplay();  
}  
  
// Функція головного меню  
void mainMenu(int id) {  
 switch(id) {  
 case 1: // Пуск/пауза анімації  
 animationActive = !animationActive;  
 break;  
 case 2: // Скидання позиції камери  
 cameraX = 0.0f;  
 cameraY = 0.0f;  
 cameraZ = 20.0f;  
 cameraYaw = 0.0f;  
 cameraPitch = 0.0f;  
 break;  
 case 3: // Перемикання режиму вільного огляду  
 freeLookMode = !freeLookMode;  
 if (freeLookMode) {  
 glutSetCursor(GLUT\_CURSOR\_NONE); // Приховати курсор  
 } else {  
 glutSetCursor(GLUT\_CURSOR\_INHERIT); // Показати курсор  
 }  
 break;  
 case 4: // Вихід з програми  
 exit(0);  
 break;  
 }  
 glutPostRedisplay();  
}  
  
// Головна функція  
int main(int argc, char\*\* argv) {  
 // Ініціалізація GLUT  
 glutInit(&argc, argv);  
  
 // Режим відображення  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);  
  
 // Розмір вікна  
 glutInitWindowSize(2048, 1024);  
  
 // Позиція вікна  
 glutInitWindowPosition(100, 100);  
  
 // Створення вікна  
 glutCreateWindow("Solar System");  
  
 // Налаштування меню  
 glutCreateMenu(mainMenu);  
 glutAddMenuEntry("Animation Play/Pause", 1);  
 glutAddMenuEntry("Reset Camera Position", 2);  
 glutAddMenuEntry("Toggle Free View", 3);  
 glutAddMenuEntry("Exit", 4);  
 glutAttachMenu(GLUT\_MIDDLE\_BUTTON);  
  
 // Реєстрація функцій-обробників  
 glutDisplayFunc(display);  
 glutReshapeFunc(reshape);  
 glutKeyboardFunc(keyboardDown);  
 glutKeyboardUpFunc(keyboardUp);  
 glutMouseFunc(mouseFunc);  
 glutMotionFunc(mouseMotionFunc);  
 glutPassiveMotionFunc(mousePassiveMotionFunc);  
  
 // Запуск анімації  
 glutTimerFunc(16, animate, 0);  
  
 // Ініціалізація OpenGL  
 init();  
  
 // Запуск головного циклу GLUT  
 glutMainLoop();  
  
 return 0;  
}

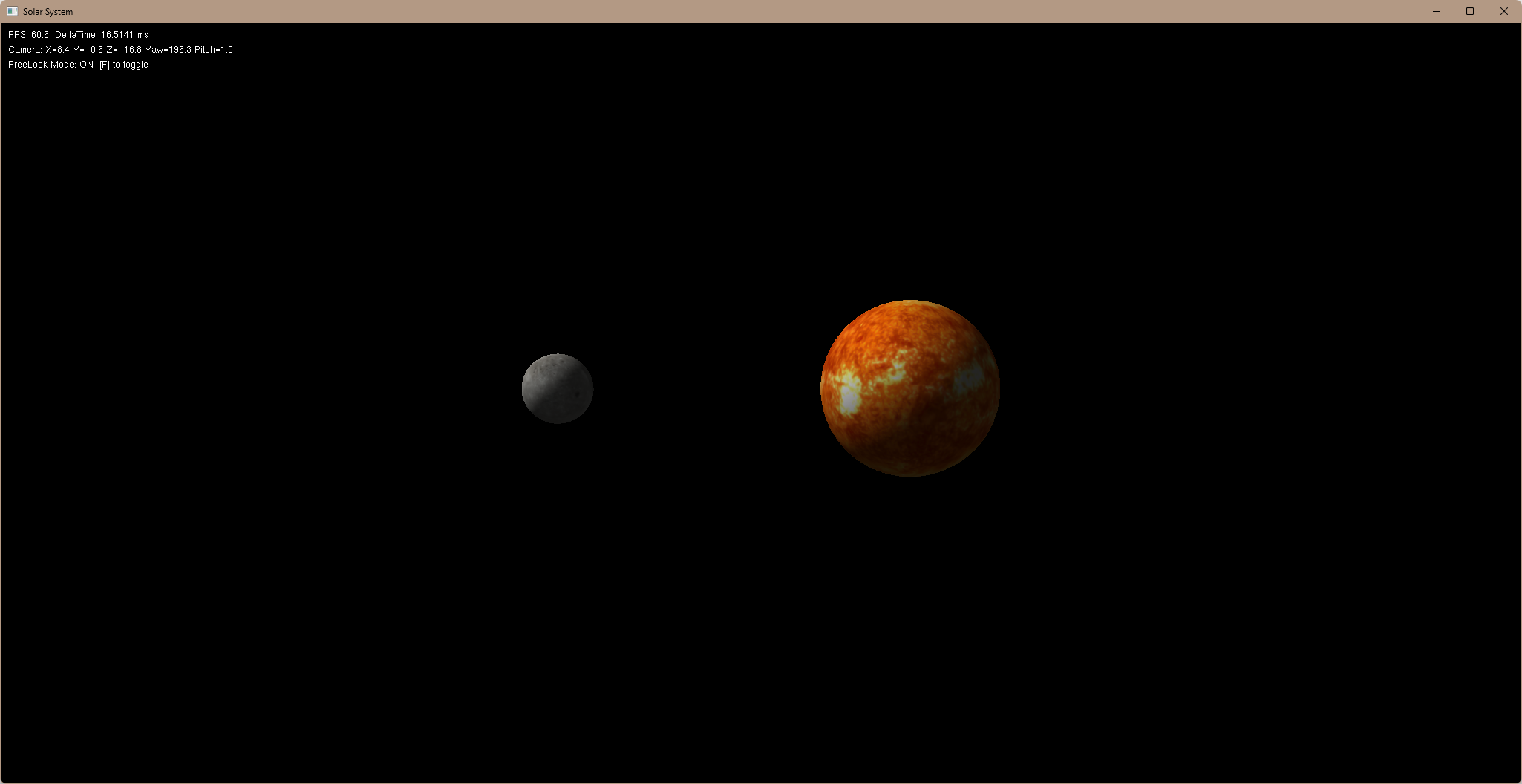
**Результати виконання роботи**

****

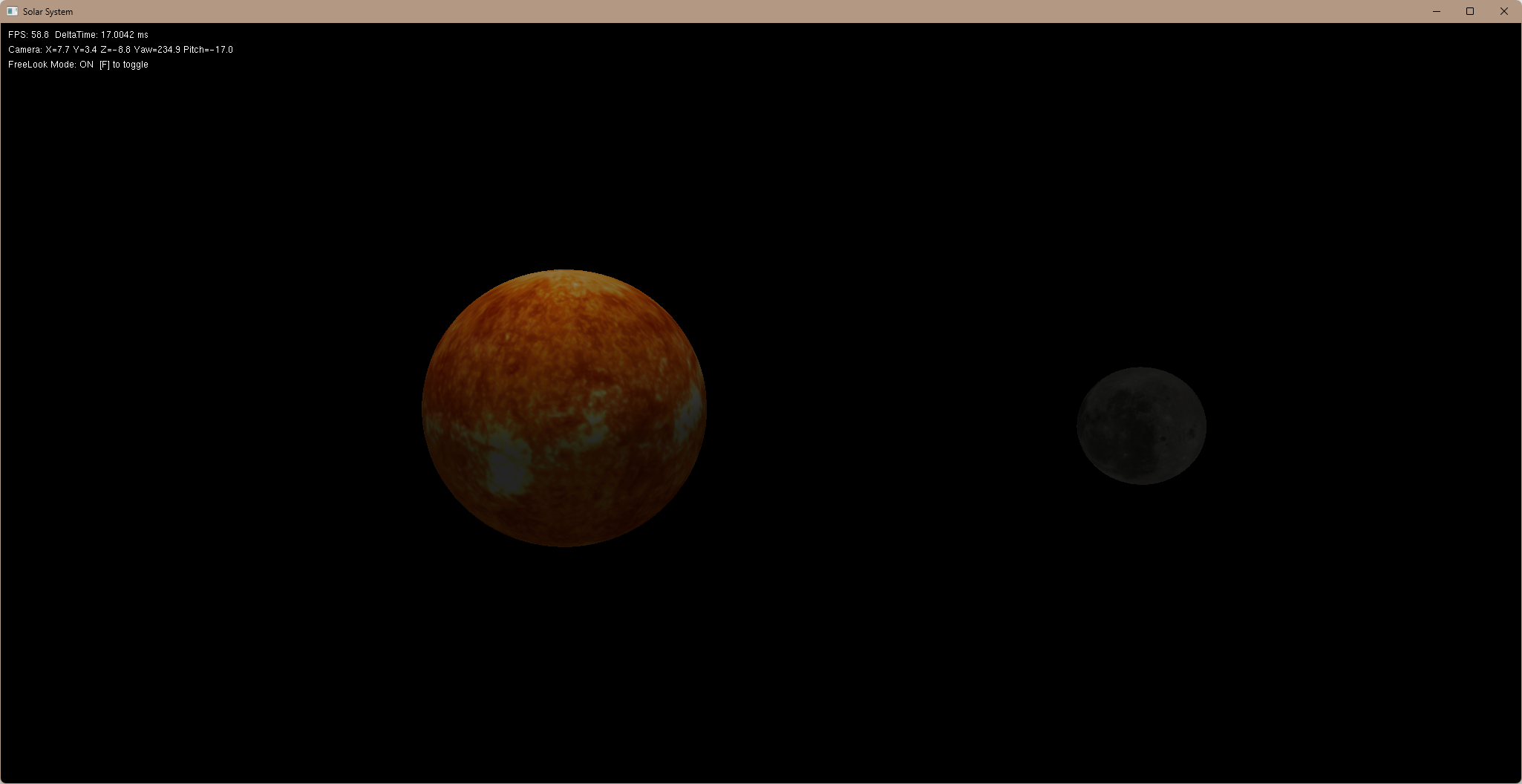
*Рис. 1. Система зоря-планета при фоновому освітленні*

**

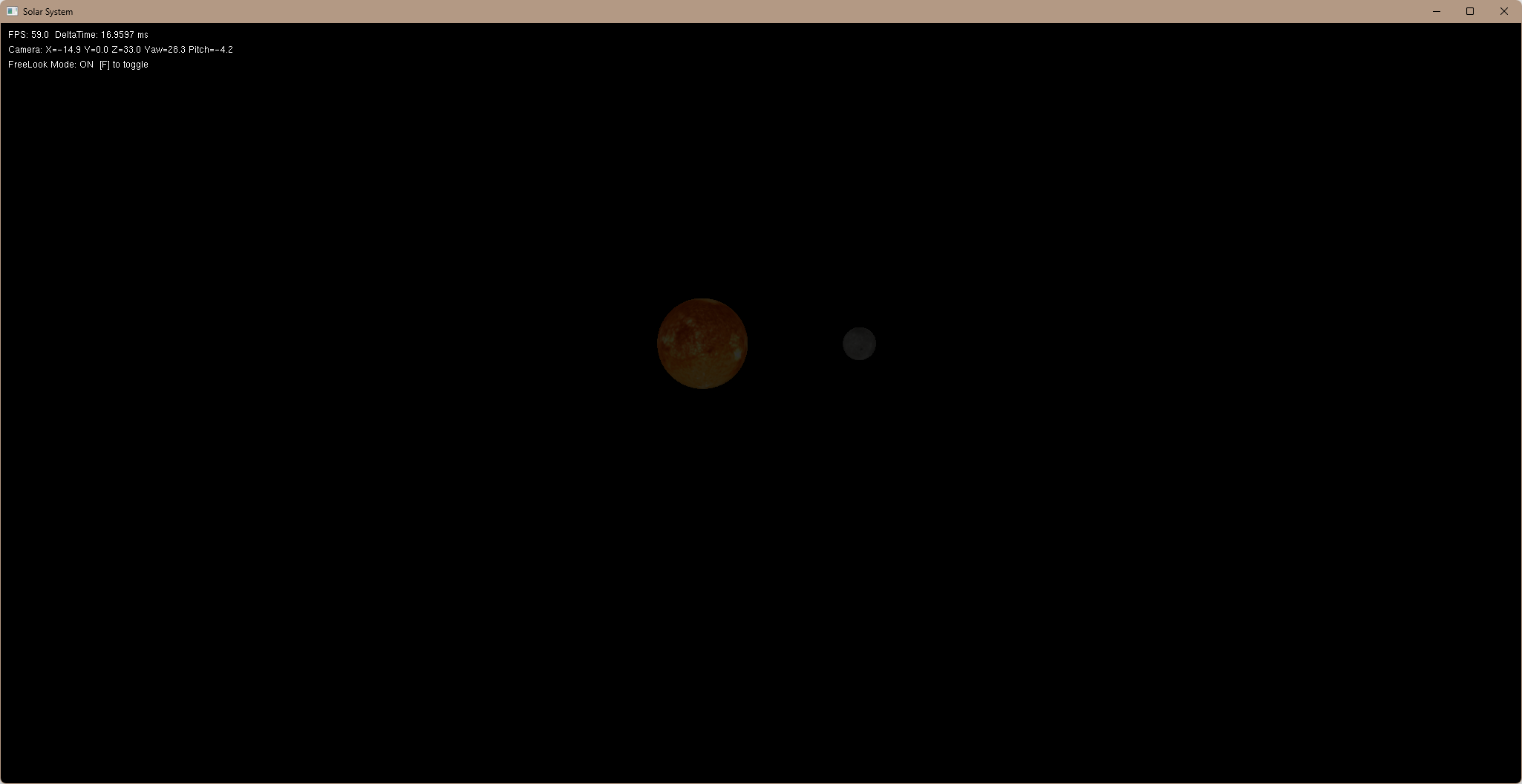
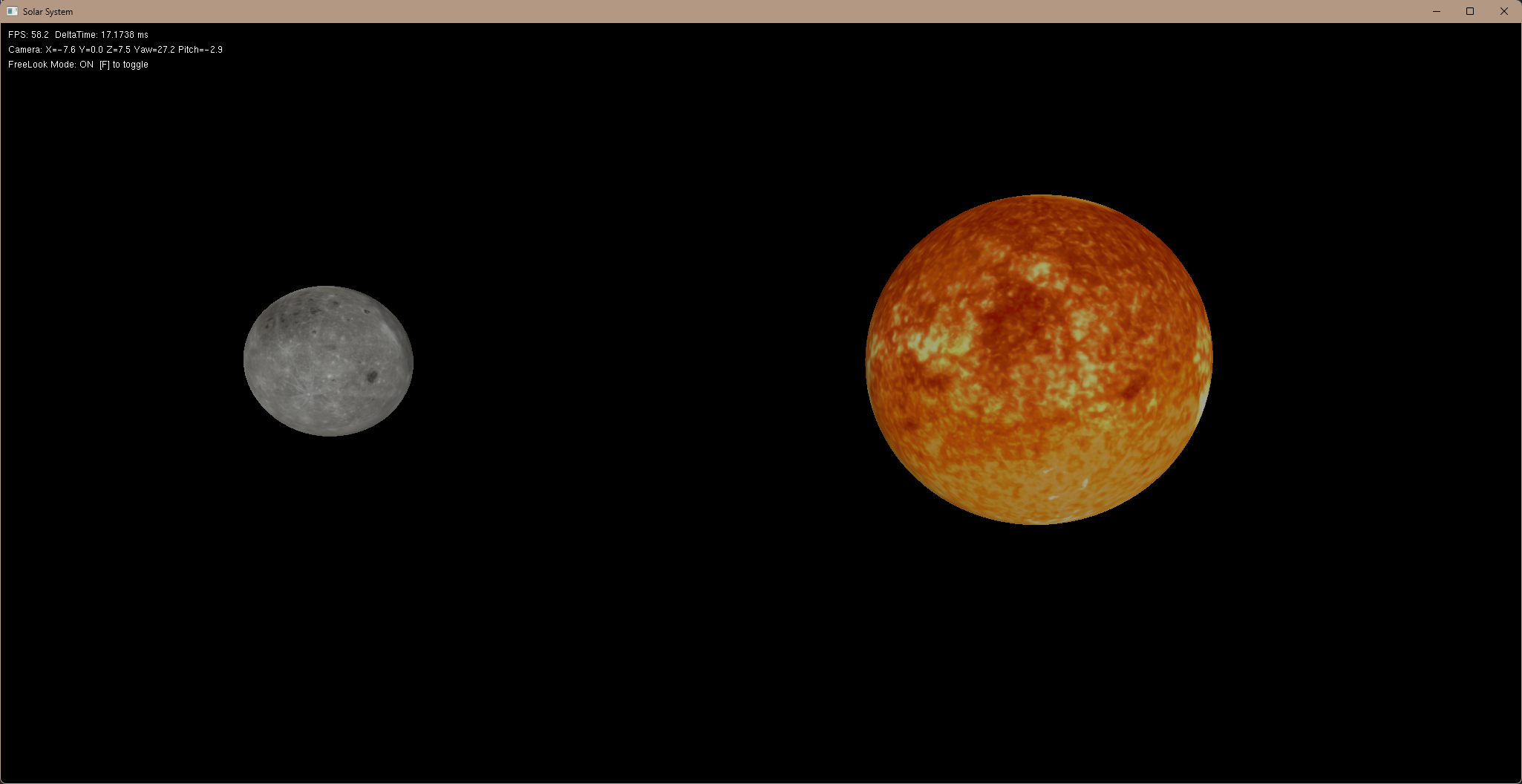
*Рис. 2. Система зоря-планета при точковому освітленні*

**

*Рис. 3. Система зоря-планета при віддаленому освітленні*

**

*Рис. 4. Система зоря-планета при освітленні прожектором*

**

*Рис. 5. Ефект туману з використанням експоненціальних функцій загасання*

**Висновки:** протягом виконання цієї лабораторної роботи я навчився будувати тривимірні об’єкти, покриті текстурою, з подальшим їхнім обертанням та масштабуванням на основі застосування базових геометричних перетворень OpenGL.