Mihai Tuhari
mihai-razvan.tuhari@s.unibuc.ro
https://github.com/mihaituhari/

Grafica pe calculator

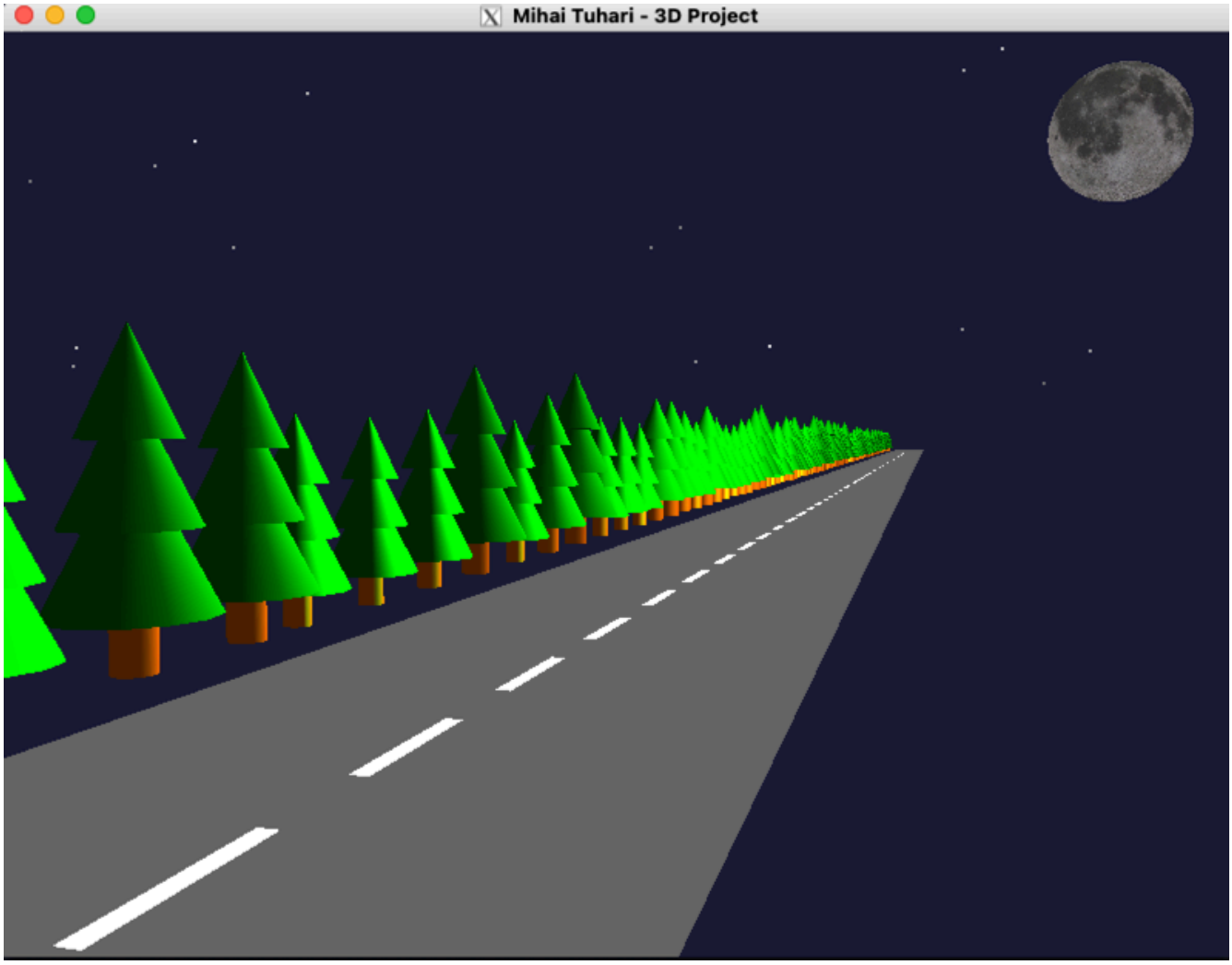
Proiect 2 (3D)

Realizati o scena 3D complexa.

🌲🌲🌲🌲🛣️🌕

Implementare

Pentru o lectura mai coerenta si simpla asupra documentatiei de mai jos, incepem cu o captura de ecran a proiectului



Introducere

Am realizat o simulare a unui drum in miscare. Soseaua are marcaj rutier pe mijloc, iar pe marginea drumului sunt copaci de diverse dimensiuni ce se misca cu aceeasi viteza.

Se mai poate distinge luna din coltul scenei.

Programul principal este in fisierul proiect2.cpp si foloseste libraria STB pentru incarcarea texturii pentru luna.

Specificatii scena

Texturi

Pentru manipularea texturilor am ales STB in loc de SOIL pentru ca pe MacOS SOIL nu este compatibil.

Textura folosita pentru luna se regaseste in folderul textures:

Copacii

Brazii de pe marginea drumului sunt realizati fiecare din catre 3 conuri si un cilindru pentru trunchi. Pentru dinamism si originalitate, fiecare copac are aplicat un factor de scalare intre 0.8 si 1.2.

Luna

Luna este o sfera cu o textura aplicata, impreuna cu setari pentru iluminare puternica.

Marcajul rutier

Acesta este format din dreptunghiuri albe cu o anumita distanta intre ele.

Animatie

Pentru a simula miscarea, am folosit o variabila `roadOffset` care se modifica la fiecare frame cu valoarea constantei `ROAD_SPEED`.

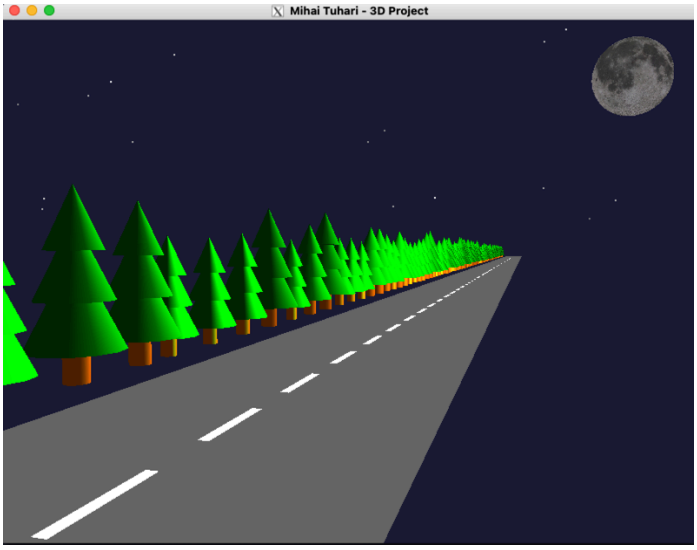
Specificatii control si camera

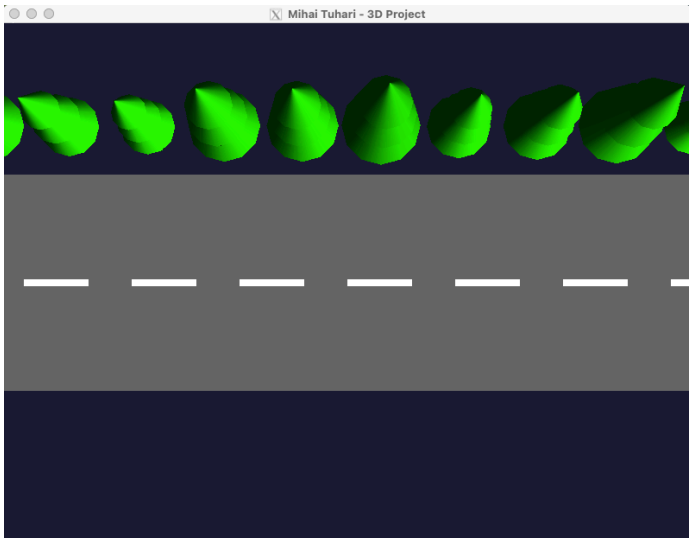
Control camera

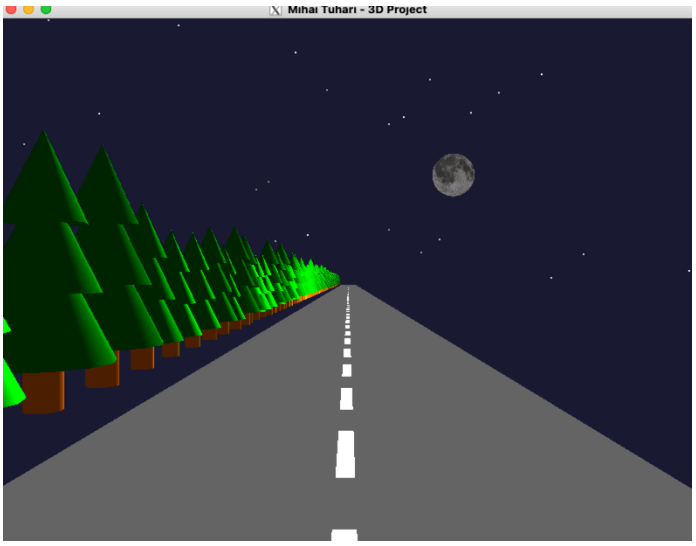
Unghiul implicit este cel afisat in prima imagine.

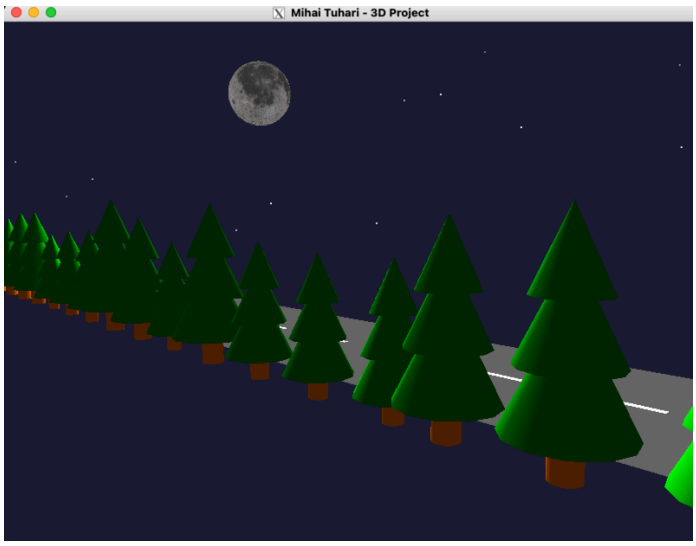
Folositi `sagetile` si tastele `+` si `-` pentru a roti camera si a da zoom in/out.

Apasati tastele 1, 2, 3, 4 pentru a schimba intre camerele presetate.



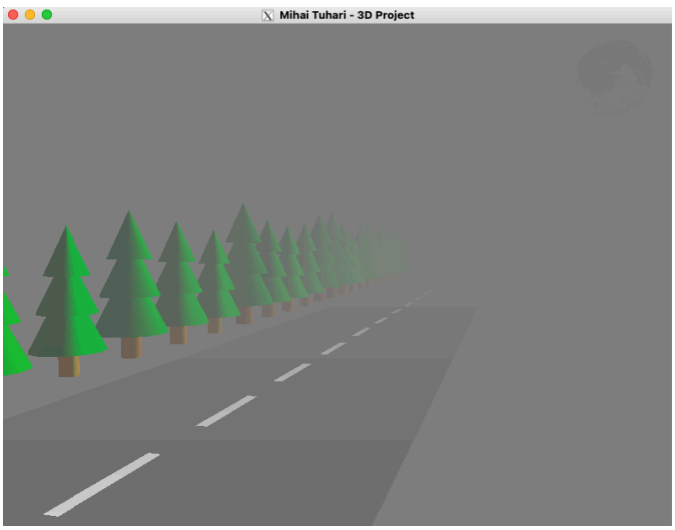


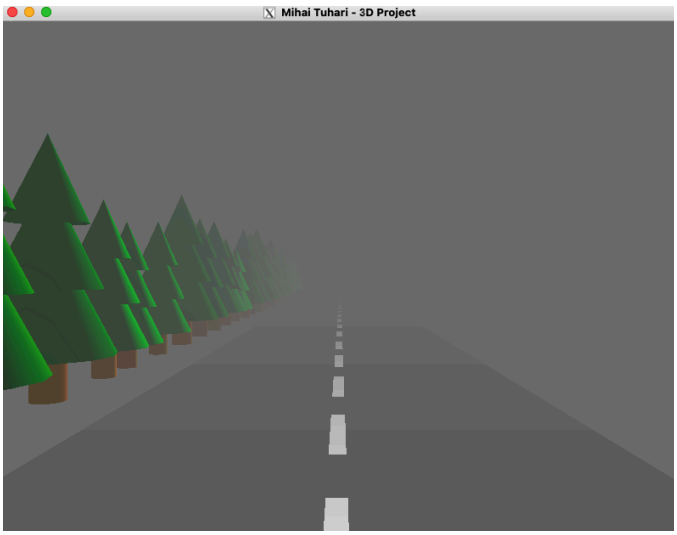




Ceata

Apasati tasta F pentru a activa/dezactiva ceata. Ceata este randata de la inceput dar este dezactivata din flag-ul `fog = false`.





Aspecte tehnice

Design modular

Am abordat proiectul cu un design modular si am incercat sa folosesc cat mai multe functii pentru a separa logica.

Configurabilitate

Animatia este usor configurabila din variabilele globale definite la inceputul fisierului proiect2.cpp.

Acolo regasiti variabile pentru:

- Camera implicita
- Ceata (flag)
- Dimensiune si viteza drum

Limba

Intreg codul (cu tot cu comentarii), este scris in limba Engleza din motive de coerenta si simplitate, pentru a evita combinatia intre termeni in limba Romana si Engleza.

Video

Puteti vedea unvideo cu animatia in actiune la sfarsitul paginii https://github.com/mihaituhari/fmi/tree/main/gc/proiect2 ori direct aici.

Codul sursa

Este inclus pe paginile urmatoare, dar il puteti regasi si pe contul meu de GitHub la adresa:

https://github.com/mihaituhari/fmi/tree/main/gc/proiect2

```cpp
/**
 * Project 2 - 3D Scene
 *
 * @author Mihai Tuhari
 * @date January 2025
*/
#define STB_IMAGE_IMPLEMENTATION

#include <iostream>
#include <GL/freeglut.h>
#include "libs/stb_image.h"

// Camera position
float Refx = 0.0f, Refy = 0.0f, Refz = 0.0f;
float alpha = 0.05f;
float beta = -1.3f;
float dist = 26.0f;
float Obsx, Obsy, Obsz;

// Fog flag
bool fog = false;

// Stars
const int NUM_STARS = 500;
struct Star {
   float x, y, z;
   float brightness;
};
std::vector<Star> stars;

// Road elements
float roadOffset = 0.0f;
const float ROAD_SPEED = 0.1f;
const float ROAD_WIDTH = 10.0f;
const float ROAD_LENGTH = 800.0f;

// Moon texture
GLuint textureMoon;
const std::string texturePath = "/Volumes/mihai/dev/fmi/gc/proiect2/textures/";
int width, height, channels;

// Trees
struct TreePosition {
   float x, y, z;
   float scaleFactor;
};
std::vector<TreePosition> treePositions;

void loadTexture(const std::string &path, GLuint &textureID) {
   stbi_set_flip_vertically_on_load(1);
   unsigned char *image = stbi_load(path.c_str(), &width, &height, &channels,
STBI_rgb_alpha);

   if (!image) {
       std::cerr << "Failed to load texture: " << path << std::endl;
       exit(1);
   }

   glGenTextures(1, &textureID);
   glBindTexture(GL_TEXTURE_2D, textureID);
```

```
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image);
    stbi_image_free(image);
}

void drawMoon() {
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, textureMoon);

    GLfloat moonAmbient[] = {1.0f, 1.0f, 1.0f, 1.0f}; // Bright white
    GLfloat moonDiffuse[] = {1.5f, 1.5f, 1.5f, 1.0f}; // Increased diffuse brightness
    GLfloat moonSpecular[] = {2.0f, 2.0f, 2.0f, 1.0f}; // High specular reflection
    GLfloat moonShininess[] = {128.0f}; // Very shiny surface

    glMaterialfv(GL_FRONT, GL_AMBIENT, moonAmbient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, moonDiffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, moonSpecular);
    glMaterialfv(GL_FRONT, GL_SHININESS, moonShininess);

    glPushMatrix();
    glTranslatef(15.0f, 30.0f, 15.0f);
    GLUquadric *quad = gluNewQuadric();
    gluQuadricTexture(quad, GL_TRUE);
    gluSphere(quad, 3.0f, 30, 30);
    gluDeleteQuadric(quad);
    glPopMatrix();

    glDisable(GL_TEXTURE_2D);
}

void drawRoad() {
    // Road surface
    glDisable(GL_LIGHTING);
    glPushMatrix();

    // Main road surface (gray)
    glColor3f(0.4f, 0.4f, 0.4f);
    glBegin(GL_QUADS);
    glVertex3f(-ROAD_WIDTH / 2, -ROAD_LENGTH / 2, -3.0f);
    glVertex3f(ROAD_WIDTH / 2, -ROAD_LENGTH / 2, -3.0f);
    glVertex3f(ROAD_WIDTH / 2, ROAD_LENGTH / 2, -3.0f);
    glVertex3f(-ROAD_WIDTH / 2, ROAD_LENGTH / 2, -3.0f);
    glEnd();

    // Road divider lines (white)
    glColor3f(1.0f, 1.0f, 1.0f);

    float lineSpacing = 5.0f;
    float offset = 0;

    for (float z = -ROAD_LENGTH / 2; z < ROAD_LENGTH / 2; z += lineSpacing) {
        float adjustedZ = z + roadOffset + offset;

        // Wrap if exceeding road
        if (adjustedZ > ROAD_LENGTH / 2) {
            adjustedZ -= ROAD_LENGTH;
        }
```

```
        if (adjustedZ < -ROAD_LENGTH / 2) {
            adjustedZ += ROAD_LENGTH;
        }

        glBegin(GL_QUADS);
        glVertex3f(-0.15f, adjustedZ, -2.99f);
        glVertex3f(0.15f, adjustedZ, -2.99f);
        glVertex3f(0.15f, adjustedZ + 3.0f, -2.99f);
        glVertex3f(-0.15f, adjustedZ + 3.0f, -2.99f);
        glEnd();
    }

    glPopMatrix();
    glEnable(GL_LIGHTING);
}

void drawTree(float scaleFactor) {
    GLfloat tree_ambient[] = {0.0f, 0.2f, 0.0f, 1.0f};
    GLfloat tree_diffuse[] = {0.0f, 0.6f, 0.0f, 1.0f};
    GLfloat tree_specular[] = {0.0f, 0.1f, 0.0f, 1.0f};
    GLfloat tree_shininess[] = {10.0f};

    glMaterialfv(GL_FRONT, GL_AMBIENT, tree_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, tree_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, tree_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, tree_shininess);

    float baseOffset = -(1.0f - scaleFactor) * 3.0f;

    glPushMatrix();
    glTranslatef(0.0f, 0.0f, baseOffset);
    glScalef(scaleFactor, scaleFactor, scaleFactor);

    // Bottom cone
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, -2.0f);
    glutSolidCone(1.5, 3.0, 12, 20);
    glPopMatrix();

    // Middle cone
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, -0.5f);
    glutSolidCone(1.2, 2.5, 12, 20);
    glPopMatrix();

    // Top cone
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, 1.0f);
    glutSolidCone(0.9, 2.0, 12, 20);
    glPopMatrix();

    // Trunk material
    GLfloat trunk_ambient[] = {0.4f, 0.2f, 0.0f, 1.0f};
    GLfloat trunk_diffuse[] = {0.6f, 0.3f, 0.1f, 1.0f};

    glMaterialfv(GL_FRONT, GL_AMBIENT, trunk_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, trunk_diffuse);

    // Draw trunk
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, -3.0f);
    GLUquadricObj *cylinder = gluNewQuadric();
    gluQuadricDrawStyle(cylinder, GLU_FILL);
```

```
    gluCylinder(cylinder, 0.4f, 0.4f, 1.0f, 20, 20);
    gluDeleteQuadric(cylinder);
    glPopMatrix();

    glPopMatrix();
}

void drawTrees() {
    for (const auto &pos: treePositions) {
        for (int set = -1; set <= 1; set++) {
            float yOffset = pos.y + roadOffset + (set * ROAD_LENGTH);

            if (yOffset >= -ROAD_LENGTH / 2 && yOffset <= ROAD_LENGTH * 0.5) {
                glPushMatrix();
                glTranslatef(pos.x, yOffset, pos.z);
                drawTree(pos.scaleFactor);
                glPopMatrix();
            }
        }
    }
}

void initTreePositions() {
    float spacing = 3.5f;
    int numTrees = (int) (ROAD_LENGTH / spacing) + 1;

    for (int i = 0; i < numTrees; i++) {
        TreePosition tree;

        tree.x = -ROAD_WIDTH / 2 - 2.0f; // Middle of the road
        tree.y = i * spacing; // Even spacing
        tree.z = 0.0f; // Road level

        // Random scale factor (e.g., between 0.8 and 1.2)
        tree.scaleFactor = 0.8f + static_cast<float>(rand()) /
(static_cast<float>(RAND_MAX / 0.4f));
        treePositions.push_back(tree);
    }
}

void initStarPositions() {
    stars.resize(NUM_STARS);

    for (auto &star: stars) {
        // Create a dome of stars
        float theta = static_cast<float>(rand()) / RAND_MAX * 2 * M_PI;
        float phi = static_cast<float>(rand()) / RAND_MAX * M_PI / 2.5f; // Limit to
upper hemisphere
        float radius = 100.0f; // Distance from center

        star.x = radius * cos(phi) * cos(theta);
        star.y = radius * cos(phi) * sin(theta);
        star.z = radius * sin(phi);

        // Random initial brightness
        star.brightness = 0.5f + static_cast<float>(rand()) / RAND_MAX * 0.5f;
    }
}

void drawStars() {
    glDisable(GL_LIGHTING);
    glDisable(GL_TEXTURE_2D);
```

```
    glPointSize(2.0f);
    glBegin(GL_POINTS);

    for (auto &star: stars) {
        glColor3f(star.brightness, star.brightness, star.brightness);
        glVertex3f(star.x, star.y, star.z);
    }

    glEnd();
    glEnable(GL_LIGHTING);
}

void reshapeAndProjection(int w, int h) {
    if (h == 0) h = 1;
    float ratio = (float) w / h;

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, w, h);
    gluPerspective(45.0f, ratio, 0.1f, 500.0f);
    glMatrixMode(GL_MODELVIEW);
}

void setupLighting() {
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT1);

    GLfloat moonLightAmbient[] = {0.5f, 0.5f, 0.5f, 1.0f}; // Stronger ambient light
    GLfloat moonLightDiffuse[] = {1.5f, 1.5f, 1.5f, 1.0f}; // Intense diffuse light
    GLfloat moonLightSpecular[] = {2.0f, 2.0f, 2.0f, 1.0f}; // Bright specular highlights
    GLfloat moonLightPosition[] = {15.0f, 30.0f, 15.0f, 1.0f}; // Position near the moon

    glLightfv(GL_LIGHT1, GL_AMBIENT, moonLightAmbient);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, moonLightDiffuse);
    glLightfv(GL_LIGHT1, GL_SPECULAR, moonLightSpecular);
    glLightfv(GL_LIGHT1, GL_POSITION, moonLightPosition);
}

void setupFog() {
    GLfloat fogColor[] = {0.5, 0.5, 0.5, 1.0}; // Fog color: light gray

    glFogi(GL_FOG_MODE, GL_EXP);
    glFogfv(GL_FOG_COLOR, fogColor);
    glFogf(GL_FOG_DENSITY, 0.05f);
    glHint(GL_FOG_HINT, GL_NICEST);
    glFogf(GL_FOG_START, 10.0f);
    glFogf(GL_FOG_END, 100.0f);
}

void renderAmbient() {
    if (fog) {
        glEnable(GL_FOG);
        glClearColor(0.5, 0.5, 0.5, 1.0); // Fog color
    } else {
        glDisable(GL_FOG);
        glClearColor(0.1f, 0.1f, 0.2f, 1.0f); // Dusk-like background color
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    Obsx = Refx + dist * cos(alpha) * cos(beta);
```

```
    Obsy = Refy + dist * cos(alpha) * sin(beta);
    Obsz = Refz + dist * sin(alpha);

    glLoadIdentity();
    gluLookAt(Obsx, Obsy, Obsz, Refx, Refy, Refz, 0.0f, 0.0f, 1.0f);

    setupLighting();
    setupFog();

    drawRoad();
    drawStars();
    drawMoon();
    drawTrees();

    glutSwapBuffers();
}

void update(int value) {
    roadOffset -= ROAD_SPEED;
    if (roadOffset < -ROAD_LENGTH) {
        roadOffset += ROAD_LENGTH;
    }

    glutPostRedisplay();
    glutTimerFunc(16, update, 0); // ~60 FPS
}

void processSpecialKeys(int key, int xx, int yy) {
    switch (key) {
        case GLUT_KEY_LEFT:
            beta -= 0.05;
            break;
        case GLUT_KEY_RIGHT:
            beta += 0.05;
            break;
        case GLUT_KEY_UP:
            alpha += 0.05;
            break;
        case GLUT_KEY_DOWN:
            alpha -= 0.05;
            break;
    }
    glutPostRedisplay();
}

void processNormalKeys(unsigned char key, int x, int y) {
    switch (key) {
        case '+':
            dist -= 0.5;
            break;
        case '-':
            dist += 0.5;
            break;
        case 'f':
        case 'F':
            fog = !fog;
            renderAmbient();
            break;

        // Default camera
        case '1':
            alpha = 0.05f;
            beta = -1.3f;
```

```
            dist = 26.0f;
            break;

        // Top view camera
        case '2':
            alpha = 1.57f;
            beta = 0.0f;
            dist = 26.0f;
            break;

        // Front view camera
        case '3':
            alpha = 0.0f;
            beta = -1.57f;
            dist = 60.0f;
            break;

        // Behind trees camera
        case '4':
            alpha = 0.1f;
            beta = -2.3f;
            dist = 30.0f;
            break;

        case 27:
            exit(0);
            break;
    }
    glutPostRedisplay();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Mihai Tuhari - 3D Project");

    initStarPositions();
    initTreePositions();

    loadTexture(texturePath + "moon.jpg", textureMoon);

    glutReshapeFunc(reshapeAndProjection);
    glutDisplayFunc(display);
    glutSpecialFunc(processSpecialKeys);
    glutKeyboardFunc(processNormalKeys);
    glutTimerFunc(16, update, 0);

    renderAmbient();
    glEnable(GL_DEPTH_TEST);

    glutMainLoop();
    return 0;
}
```