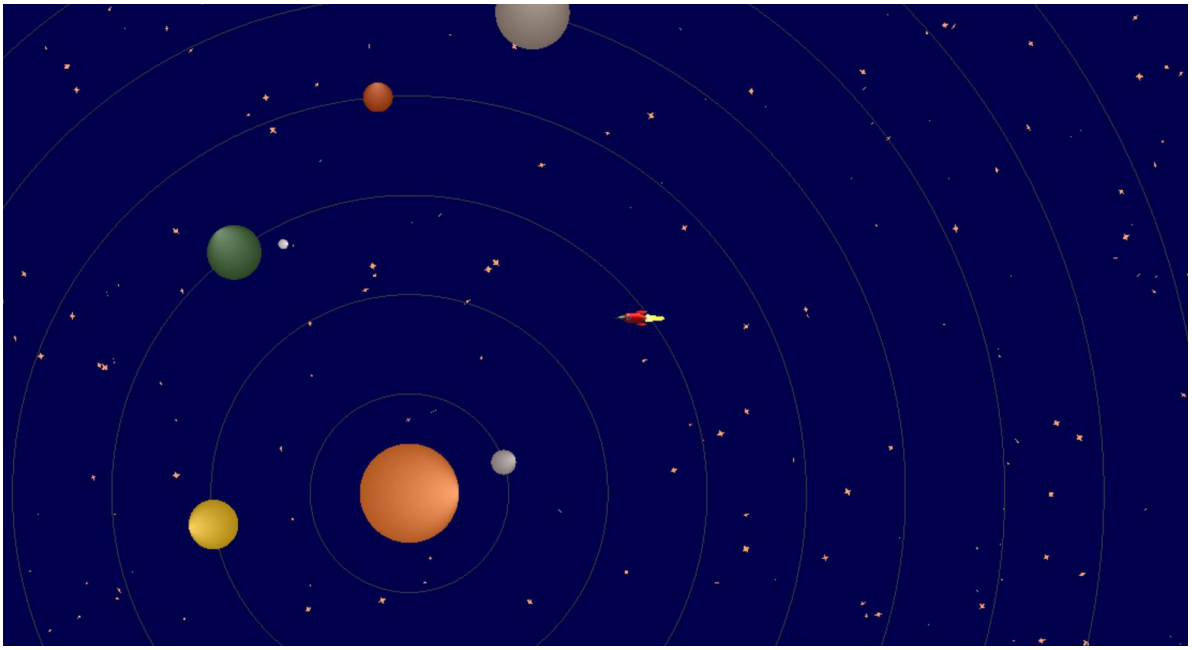
PROIECT 1 - SOLAR SYSTEM

ECHIPA : ADAM ADRIAN CLAUDIU, grupa 342
POINARITA ANDREEA DIANA, grupa 343

1. CONCEPTUL PROIECTULUI

Proiectul reprezinta o animatie 2D a Sistemului Solar, incluzand rotatiile celor 8 planete in jurul Soarelui si a Lunii in jurul Pamantului, simulate intr-un mod cat mai realist (respectand durata si viteza unei rotatii, dimensiunile planetelor etc.). De asemenea, o racheta poate fi lansata de pe Pamant si poate fi deplasata prin intermediul tastelor pentru a explora restul sistemului.



2. TRANSFORMARI INCLUSE

a) Reprezentarea statica a corpurilor ceresti si a orbitelor:

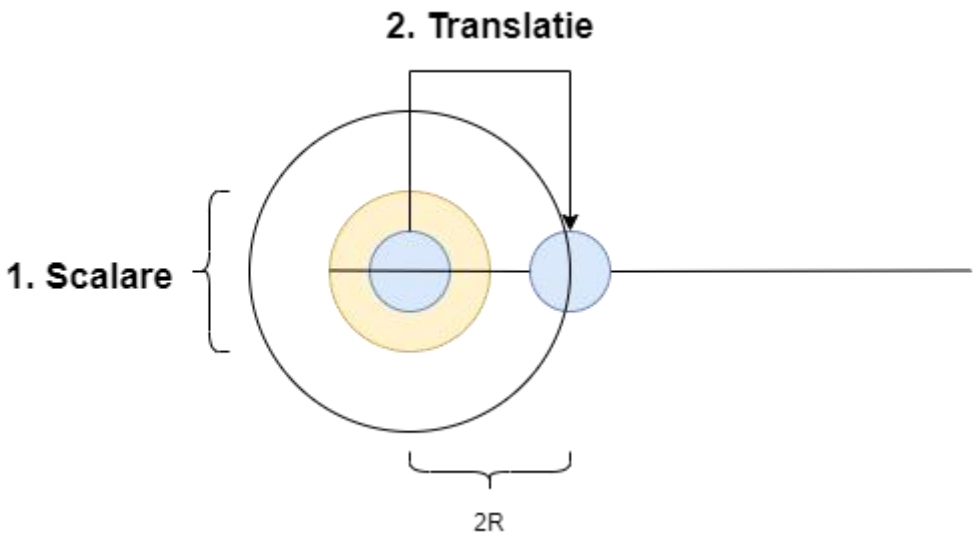
- Corpurile ceresti au fost reprezentate 2D sub forma unui cerc prin generarea a 360 de puncte, coordonatele acestora fiind adaugate o singura data in buffer pentru a nu incarca memoria:

```
vector<GLfloat> getCirclePoints(float r, int numberOfPoints)
{
        vector<GLfloat> circlePoints;
        GLfloat x;
        GLfloat y;
        for (int i = 0; i < numberOfPoints; i++)
        {
                x = (GLfloat)r * cos((2 * i * PI) / numberOfPoints);
                y = (GLfloat)r * sin((2 * i * PI) / numberOfPoints);
                circlePoints.insert(circlePoints.end(), { x, y, 0.0f, 1.0f });
        }
        return circlePoints;
}
```

```
vector<GLfloat> circleVertices = getCirclePoints(R, N);
Vertices.insert(Vertices.end(), circleVertices.begin(), circleVertices.end());
```

- Soarele a fost pozitionat in centrul scenei, iar fiecare planeta a fost obtinuta prin aplicarea asupra acestor puncte a unei scalari pentru a redimensiona si a unei translatii pentru a pozitiona planeta la distanta potrivita fata de Soare si pe orbita corespunzatoare (la inceputul animatiei, fiecare planeta a fost translatata pe axa Ox cu o distanta egala cu diametrul Soarelui = 2R fata de corpul ceresc precedent; variabila p reprezinta numarul planetei incepand de la Soare, iar scaleRaport este o proportie aproximativa intre dimensiunea Soarelui si cea a planetei curente).

```
void drawPlanet(int p, float scaleRaport) {
        scaleMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(scaleRaport, scaleRaport, 1.0));
        translMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(2 * p * R, 0.0, 0.0));
        setMyMatrix(rotateMatrix * translMatrix * scaleMatrix);
        glDrawArrays(GL_TRIANGLE_FAN, PlayerVCount, N);
```



- Orbita fiecarei planete a fost realizata doar prin scalarea atat pe axa Ox, cat si pe axa Oy, a punctelor initiale, fiind desenat doar conturul cercului.

```
void drawOrbit(int p) {
        scaleMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(2.f * p, 2.f * p, 1.0));
        setMyMatrix(scaleMatrix);
        glDrawArrays(GL_LINE_LOOP, PlayerVCount, N);
}
```

b) Rotatia planetelor in jurul Soarelui:

Rotatia planetelor se poate realiza in ambele sensuri, in jurul unui punct fix, al Soarelui, care coincide cu originea axelor de coordonate. Distanta fata de Soare a

fiecarei planete determina viteza cu care aceasta se roteste (cu cat planeta este mai apropiata de Soare, cu atat aceasta se roteste mai repede, de aceea am impartit valoarea unghiului de rotatie la numarul de ordine corespunzator). Initial, valoarea unghiului de rotatie este egala cu 0, iar dupa ce utilizatorul apasa click-dreapta/stanga pentru a porni animatia, valoarea creste la fiecare 50 milisecunde cu 0.1 (am folosit functia *glutTimerFunc* pentru a simula un timer astfel incat rotatia corpurilor ceresti sa se realizeze secvential/treptat). Daca utilizatorul apasa din nou aceeasi comanda a mouse-ului (corespunzatoare directiei de rotatie curente a corpurilor), viteza de rotatie se mareste.

```
for (int p = 1; p <= 8; p++) {
        setColCode(-1);
        drawOrbit(p);
        rotateMatrix = glm::rotate(glm::mat4(1.0f), planetAngle/p, glm::vec3(0.0, 0.0, 1.0));
```

```
void rotatePlanetsAnticlockwise(int n)
{
        moonAngle = moonAngle +0.35;
        planetAngle = planetAngle + 0.1;
        glutPostRedisplay();
        if(checkRotatePlanetsAnticlockwise)
                glutTimerFunc(50, rotatePlanetsAnticlockwise, 0);
}

void rotatePlanetsClockwise(int n)
{
        moonAngle = moonAngle - 0.35;
        planetAngle = planetAngle - 0.1;
        glutPostRedisplay();
        if (!checkRotatePlanetsAnticlockwise)
                glutTimerFunc(50, rotatePlanetsClockwise, 0);
}

void mouse(int button, int state, int x, int y)
{
        switch (button) {
        case GLUT_RIGHT_BUTTON:
                if (state == GLUT_DOWN) {
                        checkRotatePlanetsAnticlockwise = true;
                        glutTimerFunc(100, rotatePlanetsAnticlockwise, 0);
                }
                break;
         case GLUT_LEFT_BUTTON:
                 if (state == GLUT_DOWN) {
                         checkRotatePlanetsAnticlockwise = false;
                         glutTimerFunc(100, rotatePlanetsClockwise, 0);
                 }
                break;
        default:
                break;
        }
}
```

c) Rotatia Lunii in jurul Pamantului:

In acest caz, rotatia Lunii nu se mai realizeaza in jurul unui punct fixat, ci in jurul Pamantului care se roteste simultan in jurul Soarelui. Initial se scaleaza luna si se translateaza pe orbita sa. Se aplica rotatia in jurul Pamantului, apoi se translateaza a.i. centrul rotatiei sa fie chiar Pamantul, nu Soarele. In final se aplica rotatia in jurul Soarelui, aceeasi rotatie aplicata si Pamantului.

```
void drawMoon() {
        rotateMoonMatrix = glm::rotate(glm::mat4(1.0f), moonAngle, glm::vec3(0.0, 0.0, 1.0));
        scaleMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(0.1, 0.1, 1.0));
        translMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(7 * R, 0.0, 0.0));
        translMoonMatrix1 = glm::translate(glm::mat4(1.0f), glm::vec3(-6 * R, 0.0, 0.0));
        translMoonMatrix2 = glm::translate(glm::mat4(1.0f), glm::vec3(6* R, 0.0, 0.0));

        setMyMatrix(rotateMatrix * translMoonMatrix2 * rotateMoonMatrix * translMoonMatrix1 * translMatrix * scaleMatrix);
        glDrawArrays(GL_TRIANGLE_FAN, PlayerVCount, N);
}
```

d) Deplasarea rachetei:

Pentru manipularea camerei am folosit matricea de vizualizare si matricea de proiectie avand urmatoarele coordonate.

```
float Obsx = 0.0f, Obsy = 0.0f, Obsz = 200.f;
float Refx = 0.0f, Refy = 0.0f;
float width = 1200, height = 600, fov = 90, znear = 1, zfar = 1000;
```

Urmariea rachetei de catre camera este posibila, deoarece punctul de referinta este intotdeauna egal cu pozitia observatorului.

```
// se schimba pozitia observatorului
glm::vec3 Obs = glm::vec3(Obsx, Obsy, Obsz);

// pozitia punctului de referinta
Refx = Obsx; Refy = Obsy;
glm::vec3 PctRef = glm::vec3(Refx, Refy, -1.0f);

// verticala din planul de vizualizare
glm::vec3 Vert = glm::vec3(0.0f, 1.0f, 0.0f);


view = glm::lookAt(Obs, PctRef, Vert);
glUniformMatrix4fv(glGetUniformLocation(ProgramId, "view"), 1, GL_FALSE, &view[0][0]);

projection = glm::perspective(fov, GLfloat(width) / GLfloat(height), znear, zfar);
glUniformMatrix4fv(glGetUniformLocation(ProgramId, "projection"), 1, GL_FALSE, &projection[0][0]);
```

Pentru a putea descrie lansarea am folosit 2 variabile care indica momentul lansarii si nevoia de initializare a matricei de lansare.

```
bool PLAYER_LAUNCHED = false;
bool PLAYER_SHOULD_INIT_MATRIX = false;
```

La apasarea tastei SPACE, se initializeaza lansarea.

```
void processNormalKeys(unsigned char key, int x, int y) {

    switch (key) {
    case ' ':
        if (!PLAYER_LAUNCHED) {
            PLAYER_LAUNCHED = true;
            PLAYER_SHOULD_INIT_MATRIX = true;
        }
        break;
    default:
        break;
    }
}
```

Daca urmeaza sa se deseneze planeta Pamant, pentru a evita rotatia ce ar schimba directia rachetei, am rotit sub acelasi unghi in sens invers pentru a-si pastra orientarea verticala la lansare.

```
for (int p = 1; p <= 8; p++) {
    setColCode(-1);
    drawOrbit(p);
    rotateMatrix = glm::rotate(glm::mat4(1.0f), planetAngle/p, glm::vec3(0.0, 0.0, 1.0));
    if (p == 3 && PLAYER_SHOULD_INIT_MATRIX) {
        rotateVerticalPlayerMatrix = glm::rotate(glm::mat4(1.0f), -planetAngle / p, glm::vec3(0.0, 0.0, 1.0));
        rotatePlayerMatrix = rotateMatrix;
    }
    setColCode(p);
    drawPlanet(p, planetScaleRaport[p - 1]);
}
```

Daca s-a initializat lansarea, se calculeaza matricea de lansare o singura data, apoi se deseneaza racheta.

```
if (PLAYER_LAUNCHED)
{
    if (PLAYER_SHOULD_INIT_MATRIX) {
        launchPlayerMatrix = rotatePlayerMatrix * translPlayerMatrix * rotateVerticalPlayerMatrix;
        launchPlayerMatrixTransl = launchPlayerMatrix;
        PLAYER_SHOULD_INIT_MATRIX = false;
    }
    setMyMatrix(launchPlayerMatrix);
    setColCode(0);
    setTexCode(1);
    glDrawArrays(GL_TRIANGLE_FAN, StarsVCount, PlayerVCount);
}
```

Controlul rachetei se efectueaza prin intermediul sagetilor, modificand la fiecare apasare coordonatele observatorului. Pentru a roti racheta corespunzator directiei, am salvat matricea de translatie a player-ului si am aplicat acesteia o rotatie “superficiala” (deoarece la schimbarea directiei nu vrem sa efectuam rotatia de mai multe ori).

```
void processSpecialKeys(int key, int x, int y) {

    switch (key) {
    case GLUT_KEY_LEFT:
        Obsx -= 10;
        launchPlayerMatrixTransl = glm::translate(launchPlayerMatrixTransl, glm::vec3(-10, 0, 0));
        launchPlayerMatrix = launchPlayerMatrixTransl*glm::rotate(PI/2, glm::vec3(0.0f, 0.0f, 0.1f));
        break;
    case GLUT_KEY_RIGHT:
        Obsx += 10;
        launchPlayerMatrixTransl = glm::translate(launchPlayerMatrixTransl, glm::vec3(10, 0, 0));
        launchPlayerMatrix = launchPlayerMatrixTransl *glm::rotate(-PI / 2, glm::vec3(0.0f, 0.0f, 0.1f));
        break;
    case GLUT_KEY_UP:
        Obsy += 10;
        launchPlayerMatrixTransl = glm::translate(launchPlayerMatrixTransl, glm::vec3(0, 10, 0));
        launchPlayerMatrix = launchPlayerMatrixTransl;
        break;
    case GLUT_KEY_DOWN:
        Obsy -= 10;
        launchPlayerMatrixTransl = glm::translate(launchPlayerMatrixTransl, glm::vec3(0, -10, 0));
        launchPlayerMatrix = launchPlayerMatrixTransl * glm::rotate(PI, glm::vec3(0.0f, 0.0f, 0.1f));
        break;
    default:
```

e) Colorarea corpurilor ceresti si texturarea rachetei:

- Pentru realizarea rachetei am folosit texturarea oferita de biblioteca **stb_image.h**. Se poate folosi orice imagine DE REZOLUTIE 1:1. Am aplicat apoi corespunzator fiecarui punct, coordonatele corespunzatoare texturii :

```
static const GLfloat Textures[] =
{
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,

   0.0f, 0.0f,
   1.0f, 0.0f,
   1.0f, 1.0f,
   0.0f, 1.0f,
};
```

Primele coordonate sunt rezervate pentru stele. Am ales aceasta inserare pentru a pastra legatura intre varfuri si coordonatele de textura ale rachetei.

- Pentru a colora corpurile ceresti am folosit un cod corespunzator fiecaruia, transmis prin intermediul unei variabile uniforme in shader-ul de fragment.

```
// Sun
if(col_code == 99) {
        out_Color = mix(ex_Color, vec4(1, 0.5, 0.2, 1.0), 0.7);
}

// Mercury
if(col_code == 1) {
        out_Color = mix(ex_Color, vec4(0.71, 0.655, 0.655, 1.0), 0.7);
}

// Venus
if(col_code == 2) {
        out_Color = mix(ex_Color, vec4(0.976, 0.76, 0.102, 1.0), 0.7);
}
```

De asemenea, pentru a imita ideea de iluminare a corpului ceresc, am atribuit primului punct generat pe cerc culoarea alba.

3. ORIGINALITATE

Miscarea corpurilor ceresti poate fi inteleasa mult mai usor prin vizualizarea ei, proiectul putand fi extins la o varianta 3D.

4. IMBUNATATIRI ULTERIOARE PREZENTARII DIN CADRUL LABORATORULUI

a) Adaugarea stelelor

Pentru a stiliza mai mult animatia, am adaugat stele pe fundalul sistemului solar. Acestea au fost realizate prin scalarea, rotatia si translatia unor coordonate intiale corespunzatoare unei singure stele, folosind valori generate in mod aleator (care au fost stocate intr-un vector la initializare):

```
void generateStars() {
        for (int i = 0; i <= 1000; i++) {
                starXScale.push_back((float)rand() / RAND_MAX);
                starYScale.push_back((float)rand() / RAND_MAX);
                starRotation.push_back((float)rand() / RAND_MAX);
                starXTransl.push_back(rand() % 2000 - 1000);
                starYTransl.push_back(rand() % 2000 - 1000);
        }
}
```

```
void drawStars() {
	for (int i = 0; i <= 1000; i++) {
		scaleStarMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(starXScale[i], starYScale[i], 1.0));
		translStarMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(starXTransl[i], starYTransl[i], 0.0));
		rotateStarMatrix = glm::rotate(glm::mat4(1.0f), starRotation[i], glm::vec3(0.0, 0.0, 1.0));
		setMyMatrix(rotateStarMatrix * translStarMatrix * scaleStarMatrix);
		glDrawArrays(GL_TRIANGLES, 0, StarsVCount);
	}
}
```

b) Rotirea in sens invers a corpurilor ceresti

Initial, corpurile ceresti se roteau in sensul invers al acelor de ceasornic, la apasarea click-dreapta a mouse-ului. Ulterior, am adaugat si rotirea in sensul acelor de ceasornic prin apasarea click-stanga. Utilizatorul poate schimba oricand directia de deplasare, nu doar la inceputul scenei (e.x. daca planetele se rotesc in sensul acelor de ceasornic, utilizatorul poate apasa click-stanga pentru ca acestea sa isi schimbe directia de rotatie).

c) Textura aplicata rachetei

De tinut minte: FOLOSIM IMAGINI DE REZOLUTIE 1:1 PENTRU TEXTURARE.

Pentru a elimina conturul imaginii din jurul rachetei am dat discard in shader la punctele negre.

```
if(tex_code == 1) {
    vec4 texColor =  texture(myTexture, tex_Coord);
    if(texColor.x == 0 && texColor.y == 0 && texColor.z == 0)
        discard;
    out_Color = texColor;
}
```

d) Rotirea rachetei astfel incat sa indice directia de deplasare aleasa

Acest aspect a fost explicat la punctul 2 d). 😊

5. CONTRIBUTII INDIVIDUALE

Amandoi am contribuit in mod egal la realizarea proiectului, Diana axandu-se mai mult pe reprezentarea corpurilor ceresti si pe transformarile aplicate acestora, iar Adrian pe lansarea si deplasarea rachetei.

6. RESURSE UTILIZATE

- Animatiile si scenele 2D prezentate in cadrul laboratorului
- Libraria std_image.h

ANEXE:

Link GITHUB: https://github.com/Seras3/spatial-graphic

1. Cod scena 2D:

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <GL/glew.h>
#include <GL/freeglut.h>

#include "loadShaders.h"

#include "glm/glm/glm.hpp"
#include "glm/glm/gtc/matrix_transform.hpp"
#include "glm/glm/gtx/transform.hpp"
#include "glm/glm/gtc/type_ptr.hpp"
#include "stb_image.h"

#include <vector>
using namespace std;

////////////////////////////////////

GLuint
VaoId,
VboId,
EboId,
ColorBufferId,
TextureId,
ProgramId,
myMatrixLocation,
texture;

const int PlayerVCount = 4;
const int StarsVCount = 18;
const int N = 360;
const int R = 50;
const float PI = 3.141516;
glm::mat4 myMatrix, resizeMatrix, scaleMatrix, translMatrix, rotateMatrix,
    translMoonMatrix1, translMoonMatrix2, rotateMoonMatrix,
    rotateStarMatrix, translStarMatrix, scaleStarMatrix,
    translPlayerMatrix, rotatePlayerMatrix, rotateVerticalPlayerMatrix,
launchPlayerMatrix, launchPlayerMatrixTransl;

glm::mat4 view, projection;

vector<GLfloat> starXScale, starYScale, starRotation, starXTransl, starYTransl;

float Obsx = 0.0f, Obsy = 0.0f, Obsz = 200.f;
float Refx = 0.0f, Refy = 0.0f;
float width = 1200, height = 600, fov = 90, znear = 1, zfar = 1000;

bool PLAYER_LAUNCHED = false;
bool PLAYER_SHOULD_INIT_MATRIX = false;
```

```
float planetAngle = 0.0;
float moonAngle = 0.0;
float planetScaleRaport[] = { 0.25, 0.5, 0.55, 0.3, 0.75, 0.70, 0.60, 0.60 };

bool checkRotatePlanetsAnticlockwise;

vector<GLfloat> getCirclePoints(float r, int numberOfPoints)
{
    vector<GLfloat> circlePoints;
    GLfloat x;
    GLfloat y;
    for (int i = 0; i < numberOfPoints; i++)
    {
        x = (GLfloat)r * cos((2 * i * PI) / numberOfPoints);
        y = (GLfloat)r * sin((2 * i * PI) / numberOfPoints);
        circlePoints.insert(circlePoints.end(), { x, y, 0.0f, 1.0f });
    }
    return circlePoints;
}

void generateStars() {
    for (int i = 0; i <= 1000; i++) {
        starXScale.push_back((float)rand() / RAND_MAX);
        starYScale.push_back((float)rand() / RAND_MAX);
        starRotation.push_back((float)rand() / RAND_MAX);
        starXTransl.push_back(rand() % 2000 - 1000);
        starYTransl.push_back(rand() % 2000 - 1000);
    }
}

void debugMatrix(glm::mat4 matrix)
{
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

void CreateVBO(void)
{
    static vector<GLfloat> Vertices = {
        // Star vertices
        4.5f, 0.0f, 0.0f, 1.0f,
        3.0f, 3.0f, 0.0f, 1.0f,
        6.0f, 3.0f, 0.0f, 1.0f,

        6.0f, 3.0f, 0.0f, 1.0f,
        9.0f, 4.5f, 0.0f, 1.0f,
        6.0f, 6.0f, 0.0f, 1.0f,

        6.0f, 6.0f, 0.0f, 1.0f,
        4.5f, 9.0f, 0.0f, 1.0f,
        3.0f, 6.0f, 0.0f, 1.0f,

        3.0f, 6.0f, 0.0f, 1.0f,
```

```
    6.0f, 6.0f, 0.0f, 1.0f,
    6.0f, 3.0f, 0.0f, 1.0f,

    3.0f, 6.0f, 0.0f, 1.0f,
    6.0f, 3.0f, 0.0f, 1.0f,
    3.0f, 3.0f, 0.0f, 1.0f,

    3.0f, 6.0f, 0.0f, 1.0f,
    3.0f, 3.0f, 0.0f, 1.0f,
    0.0f, 4.5f, 0.0f, 1.0f,

    // Spaceship vertices
    -25.f, -25.0f, 0.0f, 1.0f,
     25.f, -25.0f, 0.0f, 1.0f,
     25.f, 25.0f, 0.0f, 1.0f,
    -25.f, 25.0f, 0.0f, 1.0f,
};

vector<GLfloat> circleVertices = getCirclePoints(R, N);
Vertices.insert(Vertices.end(), circleVertices.begin(), circleVertices.end());


static const GLfloat Colors[] =
{
    // Star
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,

    // Spaceship
    1.0f, 0.0f, 0.0f, 1.0f,
    0.0f, 1.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f,
    1.0f, 0.0f, 0.0f, 1.0f,

    1.0f, 1.0f, 1.0f, 1.0f,
};

static const GLfloat Textures[] =
{
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
    .0f, .0f,
```

```
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,
        .0f, .0f,

        0.0f, 0.0f,
        1.0f, 0.0f,
        1.0f, 1.0f,
        0.0f, 1.0f,
    };

    glGenVertexArrays(1, &VaoId);
    glBindVertexArray(VaoId);

    glGenBuffers(1, &VboId);
    glBindBuffer(GL_ARRAY_BUFFER, VboId);
    glBufferData(GL_ARRAY_BUFFER, Vertices.size() * sizeof(GLfloat),
Vertices.data(), GL_STATIC_DRAW);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(0);

    glGenBuffers(1, &ColorBufferId);
    glBindBuffer(GL_ARRAY_BUFFER, ColorBufferId);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Colors), Colors, GL_STATIC_DRAW);
    glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(1);

    glGenBuffers(1, &TextureId);
    glBindBuffer(GL_ARRAY_BUFFER, TextureId);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Textures), Textures, GL_STATIC_DRAW);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(2);
}

void DestroyVBO(void)
{
    glDisableVertexAttribArray(2);
    glDisableVertexAttribArray(1);
    glDisableVertexAttribArray(0);

    glBindBuffer(GL_ARRAY_BUFFER, TextureId);
    glDeleteBuffers(1, &TextureId);
    glBindBuffer(GL_ARRAY_BUFFER, ColorBufferId);
    glDeleteBuffers(1, &ColorBufferId);
    glBindBuffer(GL_ARRAY_BUFFER, VboId);
    glDeleteBuffers(1, &VboId);

    glBindVertexArray(VaoId);
    glDeleteVertexArrays(1, &VaoId);
```

```
}

void LoadTexture(void)
{
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    stbi_set_flip_vertically_on_load(true);
    int width, height;
    unsigned char* image = stbi_load("resize_rocket.png", &width, &height, 0, 3);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);

    stbi_image_free(image);
    glBindTexture(GL_TEXTURE_2D, texture);
}

void CreateShaders(void)
{
    ProgramId = LoadShaders("SolarSystemShader.vert", "SolarSystemShader.frag");
    glUseProgram(ProgramId);
}

void DestroyShaders(void)
{
    glDeleteProgram(ProgramId);
}


void Cleanup(void)
{
    DestroyShaders();
    DestroyVBO();
}

void Initialize(void)
{
    glClearColor(0.0f, 0.0f, 0.3f, 0.0f);
    CreateVBO();
    CreateShaders();

    LoadTexture();
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);

    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    resizeMatrix = glm::mat4(1.0f);

    generateStars();
}

void setMyMatrix(glm::mat4 matrix)
```

```
{
    myMatrix = matrix;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
}

void setTexCode(int code)
{
    glUniform1i(glGetUniformLocation(ProgramId, "tex_code"), code);
}

void setColCode(int code)
{
    glUniform1i(glGetUniformLocation(ProgramId, "col_code"), code);
}

void rotatePlanetsAnticlockwise(int n)
{
    moonAngle = moonAngle +0.35;
    planetAngle = planetAngle + 0.1;
    glutPostRedisplay();
    if(checkRotatePlanetsAnticlockwise)
        glutTimerFunc(50, rotatePlanetsAnticlockwise, 0);
}

void rotatePlanetsClockwise(int n)
{
    moonAngle = moonAngle - 0.35;
    planetAngle = planetAngle - 0.1;
    glutPostRedisplay();
    if (!checkRotatePlanetsAnticlockwise)
        glutTimerFunc(50, rotatePlanetsClockwise, 0);
}

void mouse(int button, int state, int x, int y)
{
    switch (button) {
    case GLUT_RIGHT_BUTTON:
        if (state == GLUT_DOWN) {
            checkRotatePlanetsAnticlockwise = true;
            glutTimerFunc(100, rotatePlanetsAnticlockwise, 0);
        }
        break;
     case GLUT_LEFT_BUTTON:
         if (state == GLUT_DOWN) {
             checkRotatePlanetsAnticlockwise = false;
             glutTimerFunc(100, rotatePlanetsClockwise, 0);
         }
        break;
    default:
        break;
    }
}

void processNormalKeys(unsigned char key, int x, int y) {

    switch (key) {
    case ' ':
        if (!PLAYER_LAUNCHED) {
            PLAYER_LAUNCHED = true;
```

```
            PLAYER_SHOULD_INIT_MATRIX = true;
        }
        break;
    default:
        break;
    }
}

void processSpecialKeys(int key, int x, int y) {

    switch (key) {
    case GLUT_KEY_LEFT:
        Obsx -= 10;
        launchPlayerMatrixTransl = glm::translate(launchPlayerMatrixTransl,
glm::vec3(-10, 0, 0));
        launchPlayerMatrix = launchPlayerMatrixTransl*glm::rotate(PI/2,
glm::vec3(0.0f, 0.0f, 0.1f));
        break;
    case GLUT_KEY_RIGHT:
        Obsx += 10;
        launchPlayerMatrixTransl = glm::translate(launchPlayerMatrixTransl,
glm::vec3(10, 0, 0));
        launchPlayerMatrix = launchPlayerMatrixTransl *glm::rotate(-PI / 2,
glm::vec3(0.0f, 0.0f, 0.1f));
        break;
    case GLUT_KEY_UP:
        Obsy += 10;
        launchPlayerMatrixTransl = glm::translate(launchPlayerMatrixTransl,
glm::vec3(0, 10, 0));
        launchPlayerMatrix = launchPlayerMatrixTransl;
        break;
    case GLUT_KEY_DOWN:
        Obsy -= 10;
        launchPlayerMatrixTransl = glm::translate(launchPlayerMatrixTransl,
glm::vec3(0, -10, 0));
        launchPlayerMatrix = launchPlayerMatrixTransl * glm::rotate(PI,
glm::vec3(0.0f, 0.0f, 0.1f));
        break;
    default:
        break;
    }
}

void drawOrbit(int p) {
    scaleMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(2.f * p, 2.f * p, 1.0));
    setMyMatrix(scaleMatrix);
    glDrawArrays(GL_LINE_LOOP, StarsVCount + PlayerVCount, N);
}

void drawMoon() {
    rotateMoonMatrix = glm::rotate(glm::mat4(1.0f), moonAngle, glm::vec3(0.0, 0.0,
1.0));
    scaleMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(0.1, 0.1, 1.0));
    translMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(7 * R, 0.0, 0.0));
    translMoonMatrix1 = glm::translate(glm::mat4(1.0f), glm::vec3(-6 * R, 0.0,
0.0));
    translMoonMatrix2 = glm::translate(glm::mat4(1.0f), glm::vec3(6* R, 0.0, 0.0));
```

```
    setMyMatrix(rotateMatrix * translMoonMatrix2 * rotateMoonMatrix *
translMoonMatrix1 * translMatrix * scaleMatrix);
    glDrawArrays(GL_TRIANGLE_FAN, StarsVCount + PlayerVCount, N);
}

void drawPlanet(int p, float scaleRaport) {
    scaleMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(scaleRaport, scaleRaport,
1.0));
    translMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(2 * p * R, 0.0, 0.0));
    setMyMatrix(rotateMatrix * translMatrix * scaleMatrix);
    glDrawArrays(GL_TRIANGLE_FAN, StarsVCount + PlayerVCount, N);
    if (p == 3) {
        if (!PLAYER_LAUNCHED) {
            translPlayerMatrix = translMatrix;
            Obsx = (rotateMatrix * translMatrix)[3][0];
            Obsy = (rotateMatrix * translMatrix)[3][1];
        }
        setColCode(31);
        drawMoon();
    }
}

void drawStars() {
    for (int i = 0; i <= 1000; i++) {
        scaleStarMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(starXScale[i],
starYScale[i], 1.0));
        translStarMatrix = glm::translate(glm::mat4(1.0f),
glm::vec3(starXTransl[i], starYTransl[i], 0.0));
        rotateStarMatrix = glm::rotate(glm::mat4(1.0f), starRotation[i],
glm::vec3(0.0, 0.0, 1.0));
        setMyMatrix(rotateStarMatrix * translStarMatrix * scaleStarMatrix);
        glDrawArrays(GL_TRIANGLES, 0, StarsVCount);
    }
}

void RenderFunction(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    // se schimba pozitia observatorului
    glm::vec3 Obs = glm::vec3(Obsx, Obsy, Obsz);

    // pozitia punctului de referinta
    Refx = Obsx; Refy = Obsy;
    glm::vec3 PctRef = glm::vec3(Refx, Refy, -1.0f);

    // verticala din planul de vizualizare
    glm::vec3 Vert = glm::vec3(0.0f, 1.0f, 0.0f);


    view = glm::lookAt(Obs, PctRef, Vert);
    glUniformMatrix4fv(glGetUniformLocation(ProgramId, "view"), 1, GL_FALSE,
&view[0][0]);

    projection = glm::perspective(fov, GLfloat(width) / GLfloat(height), znear,
zfar);
    glUniformMatrix4fv(glGetUniformLocation(ProgramId, "projection"), 1, GL_FALSE,
&projection[0][0]);
```

```
    setColCode(99);
    setTexCode(0);

    drawStars();

    setMyMatrix(resizeMatrix);
    glDrawArrays(GL_TRIANGLE_FAN, StarsVCount + PlayerVCount, N);

    for (int p = 1; p <= 8; p++) {
        setColCode(-1);
        drawOrbit(p);
        rotateMatrix = glm::rotate(glm::mat4(1.0f), planetAngle/p, glm::vec3(0.0,
0.0, 1.0));
        if (p == 3 && PLAYER_SHOULD_INIT_MATRIX) {
            rotateVerticalPlayerMatrix = glm::rotate(glm::mat4(1.0f), -planetAngle
/ p, glm::vec3(0.0, 0.0, 1.0));
            rotatePlayerMatrix = rotateMatrix;
        }
        setColCode(p);
        drawPlanet(p, planetScaleRaport[p - 1]);
    }

    if (PLAYER_LAUNCHED)
    {
        if (PLAYER_SHOULD_INIT_MATRIX) {
            launchPlayerMatrix = rotatePlayerMatrix * translPlayerMatrix *
rotateVerticalPlayerMatrix;
            launchPlayerMatrixTransl = launchPlayerMatrix;
            PLAYER_SHOULD_INIT_MATRIX = false;
        }
        setMyMatrix(launchPlayerMatrix);
        setColCode(0);
        setTexCode(1);
        glDrawArrays(GL_TRIANGLE_FAN, StarsVCount, PlayerVCount);
    }

    glFlush();
}


int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(width, height);
    glutCreateWindow("Solar System");
    glewInit();
    Initialize();
    glutDisplayFunc(RenderFunction);
    glutMouseFunc(mouse);
    glutKeyboardFunc(processNormalKeys);
    glutSpecialFunc(processSpecialKeys);
    glutCloseFunc(Cleanup);
    glutMainLoop();
}
```

2. Cod shader varfuri:

```
// Shader-ul de varfuri

#version 400

layout(location=0) in vec4 in_Position;
layout(location=1) in vec4 in_Color;
layout(location=2) in vec2 texCoord;

out vec4 gl_Position;
out vec4 ex_Color;
out vec2 tex_Coord;

uniform mat4 myMatrix;
uniform mat4 view;
uniform mat4 projection;


void main(void)
{
   gl_Position = projection * view * myMatrix * in_Position;
   ex_Color = in_Color;
   tex_Coord = texCoord;
}
```

3. Cod shader fragmente:

```
// Shader-ul de fragment / Fragment shader

#version 400

in vec4 ex_Color;
in vec2 tex_Coord;
out vec4 out_Color;

uniform sampler2D myTexture;
uniform int tex_code;
uniform int col_code;

void main(void)
{
    out_Color = ex_Color;

    if(tex_code == 1) {
        vec4 texColor =  texture(myTexture, tex_Coord);
        if(texColor.x == 0 && texColor.y == 0 && texColor.z == 0)
            discard;
        out_Color = texColor;
    }

    if(col_code == -1) {
        out_Color = vec4(0.25, 0.25, 0.25, 1.0);
    }


    // Sun
```

```
    if(col_code == 99) {
        out_Color = mix(ex_Color, vec4(1, 0.5, 0.2, 1.0), 0.7);
    }

    // Mercury
    if(col_code == 1) {
        out_Color = mix(ex_Color, vec4(0.71, 0.655, 0.655, 1.0), 0.7);
    }

    // Venus
    if(col_code == 2) {
        out_Color = mix(ex_Color, vec4(0.976, 0.76, 0.102, 1.0), 0.7);
    }

    // Earth
    if(col_code == 3) {
        out_Color = mix(ex_Color, vec4(0.25, 0.4, 0.208, 1.0), 0.7);
    }

    // Moon
    if(col_code == 31) {
        out_Color = mix(ex_Color, vec4(0.94, 0.906, 0.906, 1.0), 0.7);
    }

    // Mars
    if(col_code == 4) {
        out_Color = mix(ex_Color, vec4(0.757, 0.267, 0.055, 1.0), 0.7);
    }

    // Jupiter
    if(col_code == 5) {
        out_Color = mix(ex_Color, vec4(0.65, 0.57, 0.525, 1.0), 0.7);
    }

    // Saturn
    if(col_code == 6) {
        out_Color = mix(ex_Color, vec4(0.9, 0.878, 0.752, 1.0), 0.7);
    }

    // Uranus
    if(col_code == 7) {
        out_Color = mix(ex_Color, vec4(0.31, 0.815, 0.906, 1.0), 0.7);
    }

    // Neptune
    if(col_code == 8) {
        out_Color = mix(ex_Color, vec4(0.247, 0.329, 0.729, 1.0), 0.7);
    }

}
```