**Documentație SolarSystem - Proiect 2D**

1. **Conceptul proiectului**

În proiectul realizat am reprezentat o versiune simplificată a Sistemului Solar, am adaptat dimensiunile planetelor raportate la valoarea reală, distanța față de Soare și viteza de rotație. Am reprezentat Soarele, Mercur, Venus, Terra, Marte, Jupiter, Saturn, Uranus, Neptun, orbitele acestora, mișcarea de rotație a satelitului natural Luna în jurul Terrei, precum și lansarea unei rachete de pe Terra spre Marte. Când se dă click pe unul dintre butoanele mouse-ului, racheta va fi lansata, iar planetele încep să se rotească.

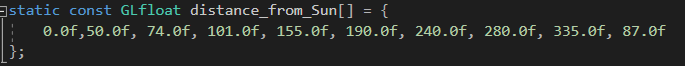
1. **Transformări utilizate**

Am utilizat scalări, translații, rotații și compunerea acestor transformări în diferite moduri pe care le voi exemplifica.

1. **Valori raportate la realitate**

Pentru început am declarat 3 vectori pentru a retine informatiile despre fiecare planeta(raza, distanță față de Soare, unghiul de rotație). Cu ajutorul acestora voi realiza **scalările**( aduc planetele la dimensiunea convenabila), **translațiile(**plasez planetele distinctiv față de Soare), **rotațiile**( fac planetele să se rotească).

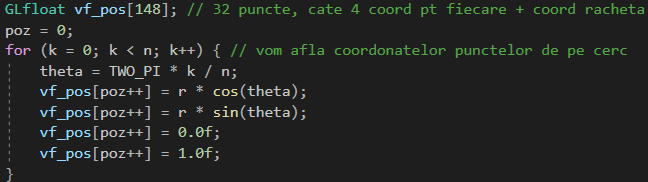






1. **Trasarea cercurilor**

Am desenat atât planetele, cât și orbitele, folosindu-mă de 32 de puncte aflate pe un cerc de raza 1, asupra cărora am aplicat transformări. În vectorul **vf\_pos** sunt stocate aceste coordonate.

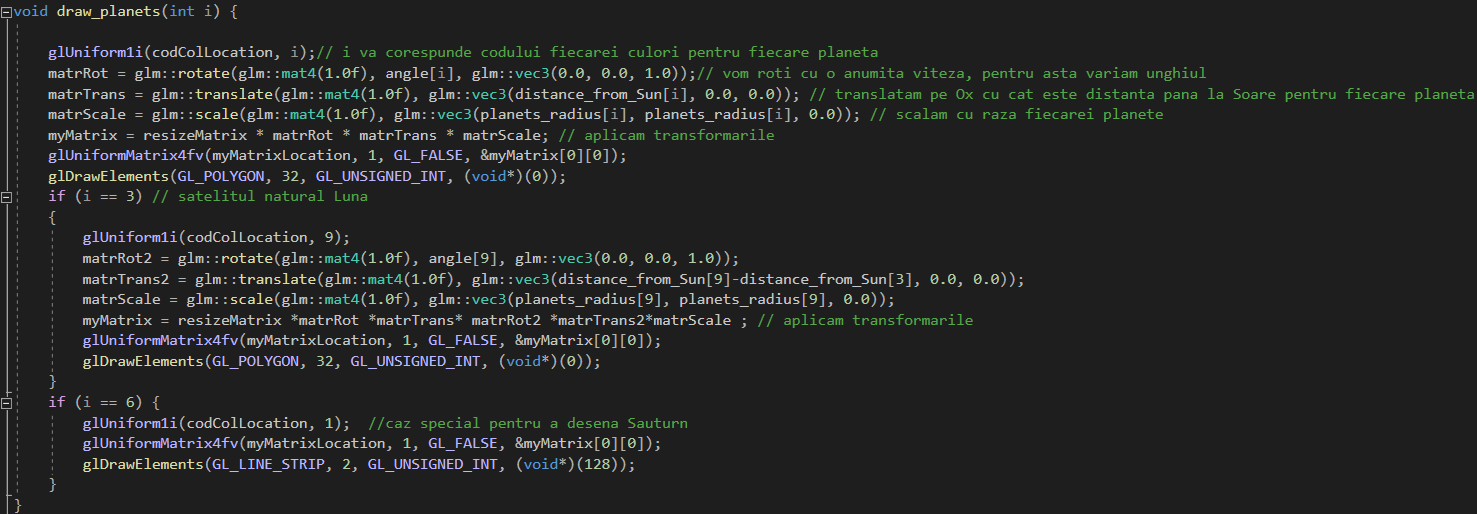


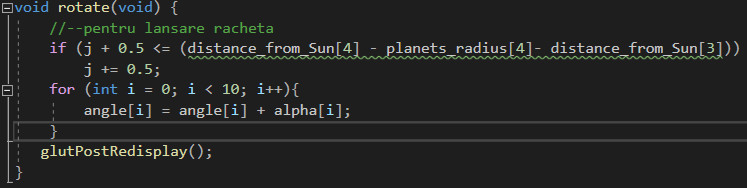
1. **Desenare planete**

Pentru a desena planetele am creat funcția **draw\_planets(int i),** unde **i** reprezintă indicele asociat planetei. Punctelor de pe cercul de **raza 1** le aplic mai întâi o scalare( pe Ox și Oy) cu **planets\_radius[i]** pentru a aduce figura la dimensiunea corespunzătoare planetei, apoi o translație pe Ox cu **distance\_from\_Sun[i**] pentru a simula distanța față de Soare. Ulterior le aplic o rotație în jurul originii cu un unghi variabil în timp( **angle[i]**). Unghiul se modifica în funcția **rotate(void)** pentru fiecare planetă cu o valoare diferită preluată din vectorul **alpha.** La apăsarea unui buton al mouse-ului este apelată funcția glutIdleFunc(rotate), iar planetele încep să se rotească.

1. **Desenare satelit Luna**

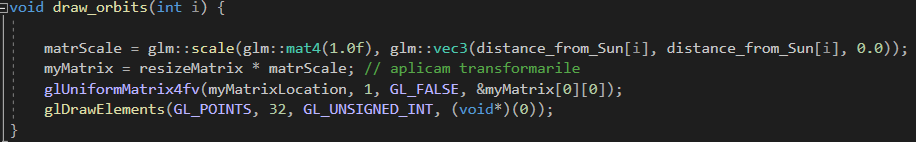
Pentru a reprezenta Luna, am tratat un caz special. În plus față de planete, Luna realizează o mișcare de rotație în jurul Terrei( centrul de rotație nu va mai fi originea sistemului de coordonate), am utilizat matrRot2 și matrTrans2 în compunerea transformărilor.



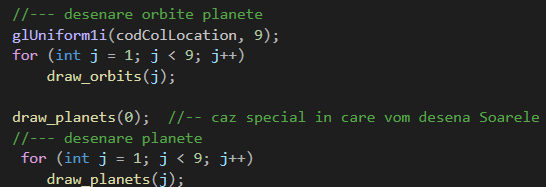


1. **Desenare orbite**

Pentru a desena orbitele pe care se realizează mișcarea de rotație a planetelor, am aplicat o scalare cu **distance\_from\_Sun[i**] corespunzătoare planetei, utilizând funcția **draw\_orbits(int i).**

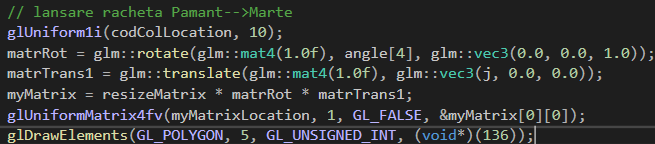


În funcția **RenderFunction(void)** am apelat functiile de desenare pentru fiecare planetă și orbită.



1. **Lansare rachetă de pe Terra pe Marte**

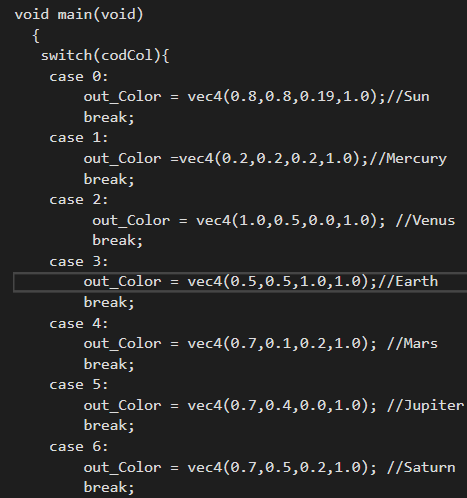
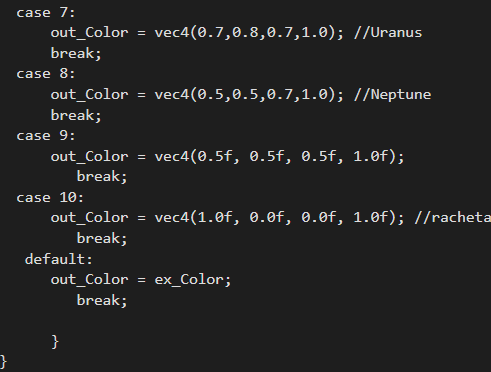
În **vf\_pos[]** am reținut la final coordonatele pentru a desena racheta. La click pe un buton al mouse-ului, racheta de pe deplasa prin translație spre Marte, am folosit o variabilă ajutătoare **j** care se modifică în funcția **rotate(void).**



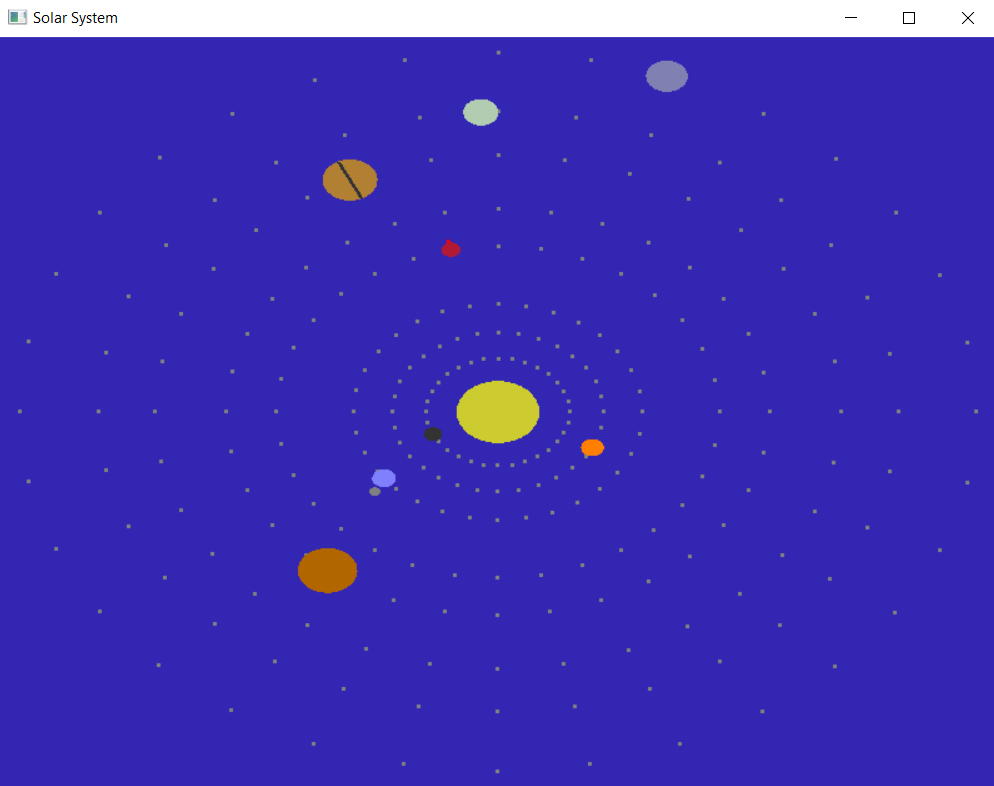
1. **Colorare planete**

Am modificat shader-ul de fragment pentru a colora diferit planetele. Am folosit o variabilă uniformă transmisă din main: 

In funcție de **codCol**, am ales o anumită nuanță:

1. **Rezultatul obținut**

****

1. **Completare etapa de prezentare proiect**

În plus față de etapa de prezentare a proiectului, am adaugat atât lansarea rachetei, cât și satelitul natural al Pământului, Luna, cu mișcarea acestuia de rotație.

**Anexa 1**

**“SolarSystem.cpp”**

#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"

using namespace std;

//--------------

GLuint

VaoId,

VboId,

EboId,

ColorBufferId,

ProgramId,

myMatrixLocation,

codColLocation;

float width = 350.f, height = 350.0f, theta, j=0.0;

const float TWO\_PI = 6.28;

int k, n = 32, poz = 0, r = 1;

glm::mat4 myMatrix, resizeMatrix = glm::ortho(-width, width, -height, height), matrTrans, matrScale, matrRot, matrTrans1, matrTrans2, matrRot2;

//--- retinem raza(cu 2 zecimale prin aproximare) pentru fiecare planeta, in ordine: Mercur, Venus, Terra, Marte, Jupiter, Saturn, Uranus, Neptun

//--- pentru a sti cu cat trebuie sa scalam cercul initial de raza 1 pe care il vom folosi la desenarea tuturor planetelor

//--- pe pozitia 0 vom retine dimensiunea Soarelui

//--- pe ultima pozitie vom retine raza satelitului Luna

static const GLfloat planets\_radius[] = {

29.0f, 6.5f, 8.05f, 8.37f, 6.7f, 20.91f, 19.23f, 12.36f, 14.62f, 4.0f

};

//--- retinem distanta fata de Soare pentru a sti cu cat trebuie sa translatam planetele

static const GLfloat distance\_from\_Sun[] = {

0.0f,50.0f, 74.0f, 101.0f, 155.0f, 190.0f, 240.0f, 280.0f, 335.0f, 87.0f

};

//--- initial unghiul de rotatie va fi 0, in timp se va modifica pentru a descrie miscarea de rotatie

float angle[] = {

0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,0.0f

};

//--- pentru fiecare planeta retinem cu cat se modifica unghiul de rotatie

float alpha[] = {

0.0f, 0.004f, 0.003f, 0.0025f, 0.0020f, 0.001f, 0.0005f, 0.0004f, 0.0003f, 0.004f

};

void rotate(void) {

//--pentru lansare racheta

if (j + 0.5 <= (distance\_from\_Sun[4] - planets\_radius[4]- distance\_from\_Sun[3]))

j += 0.5;

for (int i = 0; i < 10; i++){

angle[i] = angle[i] + alpha[i];

}

glutPostRedisplay();

}

//--- la click pe un buton al mouse ului planetele vor incepe sa se roteasca

void mouse(int button, int state, int x, int y)

{

if( button == GLUT\_LEFT\_BUTTON || button == GLUT\_RIGHT\_BUTTON)

glutIdleFunc(rotate);

}

void CreateVBO(void)

{

//---coordonatele punctelor

GLfloat vf\_pos[148]; // 32 puncte, cate 4 coord pt fiecare + coord racheta

poz = 0;

for (k = 0; k < n; k++) { // vom afla coordonatelor punctelor de pe cerc

theta = TWO\_PI \* k / n;

vf\_pos[poz++] = r \* cos(theta);

vf\_pos[poz++] = r \* sin(theta);

vf\_pos[poz++] = 0.0f;

vf\_pos[poz++] = 1.0f;

}

vf\_pos[poz++] = 108.4f ;

vf\_pos[poz++] =-3.0f ;

vf\_pos[poz++] = 0.0f;

vf\_pos[poz++] = 1.0f;

//---

vf\_pos[poz++] = 113.4f;

vf\_pos[poz++] = -3.0f;

vf\_pos[poz++] = 0.0f;

vf\_pos[poz++] = 1.0f;

//--

vf\_pos[poz++] = 118.4f;

vf\_pos[poz++] = 0.0f;

vf\_pos[poz++] = 0.0f;

vf\_pos[poz++] = 1.0f;

//--

vf\_pos[poz++] = 113.4f;

vf\_pos[poz++] = 3.0f;

vf\_pos[poz++] = 0.0f;

vf\_pos[poz++] = 1.0f;

//--

vf\_pos[poz++] = 108.4f;

vf\_pos[poz++] = 3.0f;

vf\_pos[poz++] = 0.0f;

vf\_pos[poz++] = 1.0f;

//---culorile varfurilor

static const GLfloat vf\_col[] =

{

1.0f, 0.0f, 0.0f, 1.0f,

0.0f, 1.0f, 0.0f, 1.0f,

1.0f, 0.0f, 0.0f, 1.0f,

1.0f, 1.0f, 0.0f, 1.0f,

};

//---indici pentru desenare

static GLuint vf\_ind[] =

{

0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,0,16,32,33,34,35,36

};

//--- buffer pentru varfuri si unul pentru indici

glGenBuffers(1, &VboId);

glGenBuffers(1, &EboId);

// se creeaza, apoi se leaga un VAO (Vertex Array Object)

glGenVertexArrays(1, &VaoId);

glBindVertexArray(VaoId);

//---buffer-ul este setat ca buffer curent

glBindBuffer(GL\_ARRAY\_BUFFER, VboId);

//---buffer-ul ce contine coordonatele si culorile varfurilor

glBufferData(GL\_ARRAY\_BUFFER, sizeof(vf\_col) + sizeof(vf\_pos), NULL, GL\_STATIC\_DRAW);

glBufferSubData(GL\_ARRAY\_BUFFER, 0, sizeof(vf\_pos), vf\_pos);

glBufferSubData(GL\_ARRAY\_BUFFER, sizeof(vf\_pos), sizeof(vf\_col), vf\_col);

//---buffer-ul pentru indici

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, EboId);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(vf\_ind), vf\_ind, GL\_STATIC\_DRAW);

//---atribute

//--- atributul 0 = pozitie, atributul 1 = culoare

glVertexAttribPointer(0, 4, GL\_FLOAT, GL\_FALSE, 0, NULL);

glVertexAttribPointer(1, 4, GL\_FLOAT, GL\_FALSE, 0, (const GLvoid\*)sizeof(vf\_pos));

glEnableVertexAttribArray(0);

glEnableVertexAttribArray(1);

}

void DestroyVBO(void)

{

glDisableVertexAttribArray(1);

glDisableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glDeleteBuffers(1, &EboId);

glDeleteBuffers(1, &ColorBufferId);

glDeleteBuffers(1, &VboId);

glBindVertexArray(0);

glDeleteVertexArrays(1, &VaoId);

}

void CreateShaders(void)

{

ProgramId = LoadShaders("SolarSystem.vert", "SolarSystem.frag");

glUseProgram(ProgramId);

}

void DestroyShaders(void)

{

glDeleteProgram(ProgramId);

}

void Initialize(void)

{

glClearColor(0.2f,0.15f, 0.7f, 0.0f); // culoarea de fond a ecranului

CreateShaders();

}

//--- functie pentru a desena fiecare planeta

void draw\_planets(int i) {

glUniform1i(codColLocation, i);// i va corespunde codului fiecarei culori pentru fiecare planeta

matrRot = glm::rotate(glm::mat4(1.0f), angle[i], glm::vec3(0.0, 0.0, 1.0));// vom roti cu o anumita viteza, pentru asta variam unghiul

matrTrans = glm::translate(glm::mat4(1.0f), glm::vec3(distance\_from\_Sun[i], 0.0, 0.0)); // translatam pe Ox cu cat este distanta pana la Soare pentru fiecare planeta

matrScale = glm::scale(glm::mat4(1.0f), glm::vec3(planets\_radius[i], planets\_radius[i], 0.0)); // scalam cu raza fiecarei planete

myMatrix = resizeMatrix \* matrRot \* matrTrans \* matrScale; // aplicam transformarile

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

glDrawElements(GL\_POLYGON, 32, GL\_UNSIGNED\_INT, (void\*)(0));

if (i == 3) // satelitul natural Luna

{

glUniform1i(codColLocation, 9);

matrRot2 = glm::rotate(glm::mat4(1.0f), angle[9], glm::vec3(0.0, 0.0, 1.0));

matrTrans2 = glm::translate(glm::mat4(1.0f), glm::vec3(distance\_from\_Sun[9]-distance\_from\_Sun[3], 0.0, 0.0));

matrScale = glm::scale(glm::mat4(1.0f), glm::vec3(planets\_radius[9], planets\_radius[9], 0.0));

myMatrix = resizeMatrix \*matrRot \*matrTrans\* matrRot2 \*matrTrans2\*matrScale ; // aplicam transformarile

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

glDrawElements(GL\_POLYGON, 32, GL\_UNSIGNED\_INT, (void\*)(0));

}

if (i == 6) {

glUniform1i(codColLocation, 1); //caz special pentru a desena Sauturn

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

glDrawElements(GL\_LINE\_STRIP, 2, GL\_UNSIGNED\_INT, (void\*)(128));

}

}

//--- functie pentru a desena orbitele planetelor

void draw\_orbits(int i) {

matrScale = glm::scale(glm::mat4(1.0f), glm::vec3(distance\_from\_Sun[i], distance\_from\_Sun[i], 0.0));

myMatrix = resizeMatrix \* matrScale; // aplicam transformarile

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

glDrawElements(GL\_POINTS, 32, GL\_UNSIGNED\_INT, (void\*)(0));

}

void RenderFunction(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

//---Creare VBO

CreateVBO();

//---Transmitere variabile uniforme

myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");

codColLocation = glGetUniformLocation(ProgramId, "codCol");

//---Desenare

glLineWidth(3.0);

glPointSize(3.0);

//--- desenare orbite planete

glUniform1i(codColLocation, 9);

for (int j = 1; j < 9; j++)

draw\_orbits(j);

// lansare racheta Pamant-->Marte

glUniform1i(codColLocation, 10);

matrRot = glm::rotate(glm::mat4(1.0f), angle[4], glm::vec3(0.0, 0.0, 1.0));

matrTrans1 = glm::translate(glm::mat4(1.0f), glm::vec3(j, 0.0, 0.0));

myMatrix = resizeMatrix \* matrRot \* matrTrans1;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

glDrawElements(GL\_POLYGON, 5, GL\_UNSIGNED\_INT, (void\*)(136));

draw\_planets(0); //-- caz special in care vom desena Soarele

//--- desenare planete

for (int j = 1; j < 9; j++)

draw\_planets(j);

glFlush();

}

void Cleanup(void)

{

DestroyShaders();

DestroyVBO();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 100);

glutInitWindowSize(800, 600);

glutCreateWindow("Solar System");

glewInit();

Initialize();

glutDisplayFunc(RenderFunction);

glutMouseFunc(mouse);

glutCloseFunc(Cleanup);

glutMainLoop();

}

**Anexa 2**

**“SolarSystem.frag”**

// Shader-ul de fragment / Fragment shader

#version 400

in vec4 ex\_Color;

out vec4 out\_Color;

uniform int codCol;

void main(void) {

switch(codCol){

case 0:

out\_Color = vec4(0.8,0.8,0.19,1.0);//Sun

break;

case 1:

out\_Color =vec4(0.2,0.2,0.2,1.0);//Mercury

break;

case 2:

out\_Color = vec4(1.0,0.5,0.0,1.0); //Venus

break;

case 3:

out\_Color = vec4(0.5,0.5,1.0,1.0);//Earth

break;

case 4:

out\_Color = vec4(0.7,0.1,0.2,1.0); //Mars

break;

case 5:

out\_Color = vec4(0.7,0.4,0.0,1.0); //Jupiter

break;

case 6:

out\_Color = vec4(0.7,0.5,0.2,1.0); //Saturn

break;

case 7:

out\_Color = vec4(0.7,0.8,0.7,1.0); //Uranus

break;

case 8:

out\_Color = vec4(0.5,0.5,0.7,1.0); //Neptune

break;

case 9:

out\_Color = vec4(0.5f, 0.5f, 0.5f, 1.0f);

break;

case 10:

out\_Color = vec4(1.0f, 0.0f, 0.0f, 1.0f); //racheta

break;

default:

out\_Color = ex\_Color;

break;

} }

**Anexa 3**

**“SolarSystem.vert”**

// Shader-ul de varfuri

#version 400

layout(location=0) in vec4 in\_Position;

layout(location=1) in vec4 in\_Color;

out vec4 gl\_Position;

out vec4 ex\_Color;

uniform mat4 myMatrix;

void main(void)

{

gl\_Position = myMatrix\*in\_Position;

ex\_Color = in\_Color;

}