# Solar System

Autori:

Tudorache Alexandru-Theodor (gr. 342)
Zaharia Robert-Cătălin (gr. 342)

## Concept

Proiectul prezintă un sistem solar simplificat format din patru elemente: Soare, Pământ, Lună, Planeta Mercur. Acestea sunt reprezentate de patru cercuri de culori și mărimi diferite care fac următoarele rotații:

- Soarele se învârte în jurul propriei axe.
- Pământul se învârte în jurul propriei axe și în jurul Soarelui.
- Luna se învârte în jurul propriei axe, în jurul Pământului și în jurul Soarelui.
- Planeta Mercur se învârte în jurul propriei axe și în jurul Soarelui.

Pe lângă aceste patru elemente, mai există și opt stele generate aleatoriu odată pe secundă.

Pentru a începe aceste rotații trebuie făcut click în fereastră.

## Transformări incluse

- `ellipseSize=glm::translate(glm::mat4(1.0f), glm::vec3(semiMinor, 0, 0.0));`
  -> ellipseSize setează distanța dintre punctul de rotație și obiect
- `circleScaling = glm::scale(glm::mat4(1.0f), glm::vec3(0.33, 0.33, 0.0));`
  -> o scalare de 1/3 a tuturor cercurilor.
- `ellipseRotation = glm::rotate(glm::mat4(1.0f), twoPI * rotationMult, glm::vec3(0.0, 0.0, 1.0));`
  -> cu ajutorul variabilei dinamice rotationMult al cărei valoare este crescătoare în [0,1], apoi revine la 0, înmulțindu-se cu 2PI rezultă un cerc unitar.
- `sunScaling = glm::scale(glm::mat4(1.0f), glm::vec3(3, 3, 0.0));`
  -> o scalare care face Soarele de 3 ori mai mare decât Pământul.
- `cycle = glm::rotate(glm::mat4(1.0f), 0.1f * cycleMult * twoPI, glm::vec3(0.0, 0.0, 1.0));`
  -> aceeași rotație ca și ellipseRotation, doar că cycleMult nu face overflow la o anumită valoare (cum se întâmplă în cazul rotationMult).

- `moonTranslate = glm::translate(glm::mat4(1.0f), glm::vec3(moonOffset, 0, 0.0));`
  -> deoarece Luna posedă o perioadă de rotație egală cu perioada să de revoluție, aceasta nu are nevoie de o rotație în jurul propriei axe, aceasta fiind 0 în raport cu rotația în jurul Pământului. În practică centrul de rotație al lunii este Pământul.
- `moonScale = glm::scale(glm::mat4(1.0f), glm::vec3(0.25, 0.25, 0.0));`
  -> o scalare care face Luna de 4 ori mai mică decât Pământul.
- `glm::translate(glm::mat4(1.0f), glm::vec3(randX, randY, 0));`
  -> o translație care aduce o stea într-o poziție generată aleatoriu.

# De ce este original?

Proiectul este original pentru că:

- Cercurile (care reprezintă elementele din Sistemul Solar) nu se învârt unul în jurul celuilalt într-o formă regulată (de exemplu în formă de cerc). Acestea se învârt unul în jurul celuilalt în formă de elipsă care se învârte la rândul ei în jurul propriei axe. Această elipsă are dimensiunile bine definite în cod.
- Cercurile nu sunt desenate folosind o funcție generică/o funcție preluată din surse externe. Ele sunt desenate folosind o funcție de desenare a unui poligon regulat cu n vârfuri pornind de la centrul și raza cercului circumscris poligonului respectiv. Această funcție este aplicată pentru un poligon cu 30 de vârfuri, astfel poligonul având un număr suficient de mare de vârfuri încât să poată fi considerat cerc.
- Am definit propriile rapoarte de mărimi între Soare, Pământ, Luna și planeta Mercur:
- Pământul și planeta Mercur au o mărime(aceeași) bine definită de la început.
  Soarele este de 3 ori mai mare decât Pământul.
  Luna este de 4 ori mai mică decât Pământul.
  Nu este fezabil să folosim rapoartele de mărime din realitate deoarece Luna este de aproximativ 400 de ori mai mică decât Soarele.
- În urma discuției individuale de la laborator, am mai adăugat 8 stele (reprezentate de 8 hexagoane regulate) generate în poziții aleatorii.

# Implementare

Vectorii de poziții, culori și indici nu au mai fost definiți static în funcția *CreateVBO,* ci au fost definiți global și generați dinamic folosind o funcție de desenare a cercurilor.

```
GLfloat vf_pos[1000];
GLfloat vf_col[1000];
GLuint vf_ind[250];
int ind_sun, ind_earth, ind_moon, ind_mercury, ind_star, ind_background;
```

Funcția de desenare a cercurilor este *CreateCircle* având următorii parametri:

```
int CreateCircle(GLfloat Cx = 0.0f, GLfloat Cy = 0.0f, GLfloat radius = 10.0f, int noOfVertex = 30,
int index = 0, GLfloat red = 0.0f, GLfloat green = 0.0f, GLfloat blue = 0.0f, GLfloat offset = 0.01f)
```

unde: *Cx* și *Cy* reprezintă coordonatele centrului cercului, radius reprezintă raza cercului, *noOfVertex* reprezintă numărul de vârfuri ale poligonului care formează cercul, *index* este indicele de la care se începe desenarea în vectorul de indici, *red*, *green*, *blue* reprezintă culoarea cercului, iar *offset* reprezintă cantitatea de culoare adăugată fiecăruia dintre *red*, *green*, *blue* după fiecare vârf desenat.

Cercurile inițiale au fost desenate astfel

```
void CreateVBO(void)
{
    ind_sun = CreateCircle(0.0f, 0.0f, 50.0f, 30, 0, 1.0f, 0.8f, 0.0f, 0.01f);
    ind_earth = CreateCircle(0.0f, 0.0f, 50.0f, 30, 30, 0.0f, 0.2f, 0.5f, -0.015f);
    ind_moon = CreateCircle(0.0f, 0.0f, 50.0f, 30, 61, 1.0f, 1.0f, 0.1f, -0.02f);
```

```
void setMatrix(glm::mat4 myMatrix) {
    CreateShaders();
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
}
```

```
resizeMatrix= glm::scale(glm::mat4(1.0f), glm::vec3(1.f/width, 1.f/height, 1.0)); // scalam, "aducem" scena la "patratul standard" [-1,1]x[-1,1]
```

```
myMatrix = resizeMatrix;// *rotationOwnAxis(1)* sunScaling* circleScaling;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_sun, GL_UNSIGNED_INT, (void*)(0)); //SOARE

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

myMatrix = resizeMatrix;// * cycle * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(3) * circleScali
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_earth, GL_UNSIGNED_INT, (void*)((ind_sun + 1) * sizeof(GLuint))); //PAMANT

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

myMatrix = resizeMatrix;// * cycle * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(3) * moonTransla
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_moon, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1) * sizeof(GLuint))); //LUNA
```

Fiecărui cerc i s-a adăugat o scalare

```
circleScaling = glm::scale(glm::mat4(1.0f), glm::vec3(0.33, 0.33, 0.0));
```

```
myMatrix = resizeMatrix * circleScaling;// * rotationOwnAxis(1) * sunScaling * circleScaling;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_sun, GL_UNSIGNED_INT, (void*)(0)); //SOARE

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

myMatrix = resizeMatrix * circleScaling;// * cycle * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_earth, GL_UNSIGNED_INT, (void*)((ind_sun + 1) * sizeof(GLuint))); //PAMANT

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

myMatrix = resizeMatrix * circleScaling;// * cycle * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_moon, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1) * sizeof(GLuint))); //LUNA
```

Soarelui și Lunii le-am adăugat scalări proprii

```
sunScaling = glm::scale(glm::mat4(1.0f), glm::vec3(3, 3, 0.0));
```

```
moonScale = glm::scale(glm::mat4(1.0f), glm::vec3(0.25, 0.25, 0.0));
```

```
myMatrix = resizeMatrix * sunScaling * circleScaling;// *rotationOwnAxis(1);
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_sun, GL_UNSIGNED_INT, (void*)(0)); //SOARE

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

myMatrix = resizeMatrix * circleScaling;// * cycle * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_earth, GL_UNSIGNED_INT, (void*)((ind_sun + 1) * sizeof(GLuint))); //PAMANT

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

myMatrix = resizeMatrix * moonScale * circleScaling;// * cycle * ellipseMovement * ellipseRotation * ellipseSize * rota
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_moon, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1) * sizeof(GLuint))); //LUNA
```

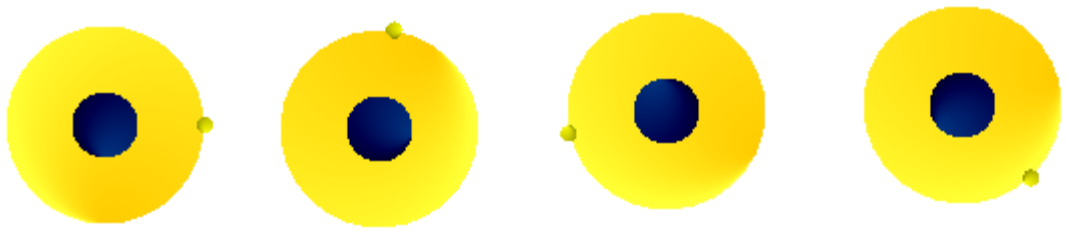
Pământului și Soarelui le-am adăugat rotație în jurul propriilor axe, iar Lunii o translație și o rotație pentru a orbita în jurul Pământului.

```
glm::highp_mat4 rotationOwnAxis(float speed) {
    return glm::rotate(glm::mat4(1.0f), speed * twoPI * rotationMult, glm::vec3(0.0, 0.0, 1.0));
}
```

```
float moonOffset = 50;
```

```
moonTranslate = glm::translate(glm::mat4(1.0f), glm::vec3(moonOffset, 0, 0.0)); // translatia lunii
```

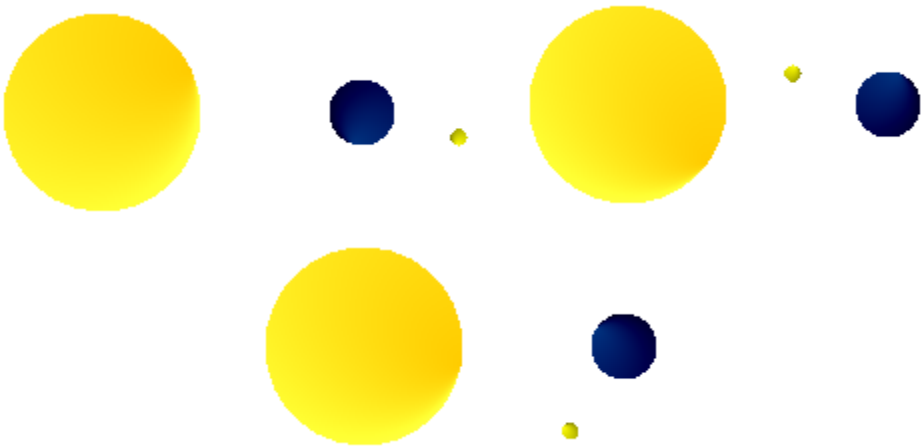
```
myMatrix = resizeMatrix * rotationOwnAxis(1) * sunScaling * circleScaling ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_sun, GL_UNSIGNED_INT, (void*)(0)); //SOARE

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

myMatrix = resizeMatrix * rotationOwnAxis(3) * circleScaling ;// * cycle * ellipseMovement * ellipseRotation * ellipseS
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_earth, GL_UNSIGNED_INT, (void*)((ind_sun + 1) * sizeof(GLuint))); //PAMANT

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

myMatrix = resizeMatrix * rotationOwnAxis(3) * moonTranslate * moonScale * circleScaling ;// * cycle * ellipseMovement
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_moon, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1) * sizeof(GLuint))); //LUNA
```



Pământului și Lunii le-am adăugat o translație pentru a se îndepărta de Soare.

```
ellipseSize=glm::translate(glm::mat4(1.0f), glm::vec3(semiMinor, 0, 0.0)); // seteaza cat de lata va fi elipsa
```

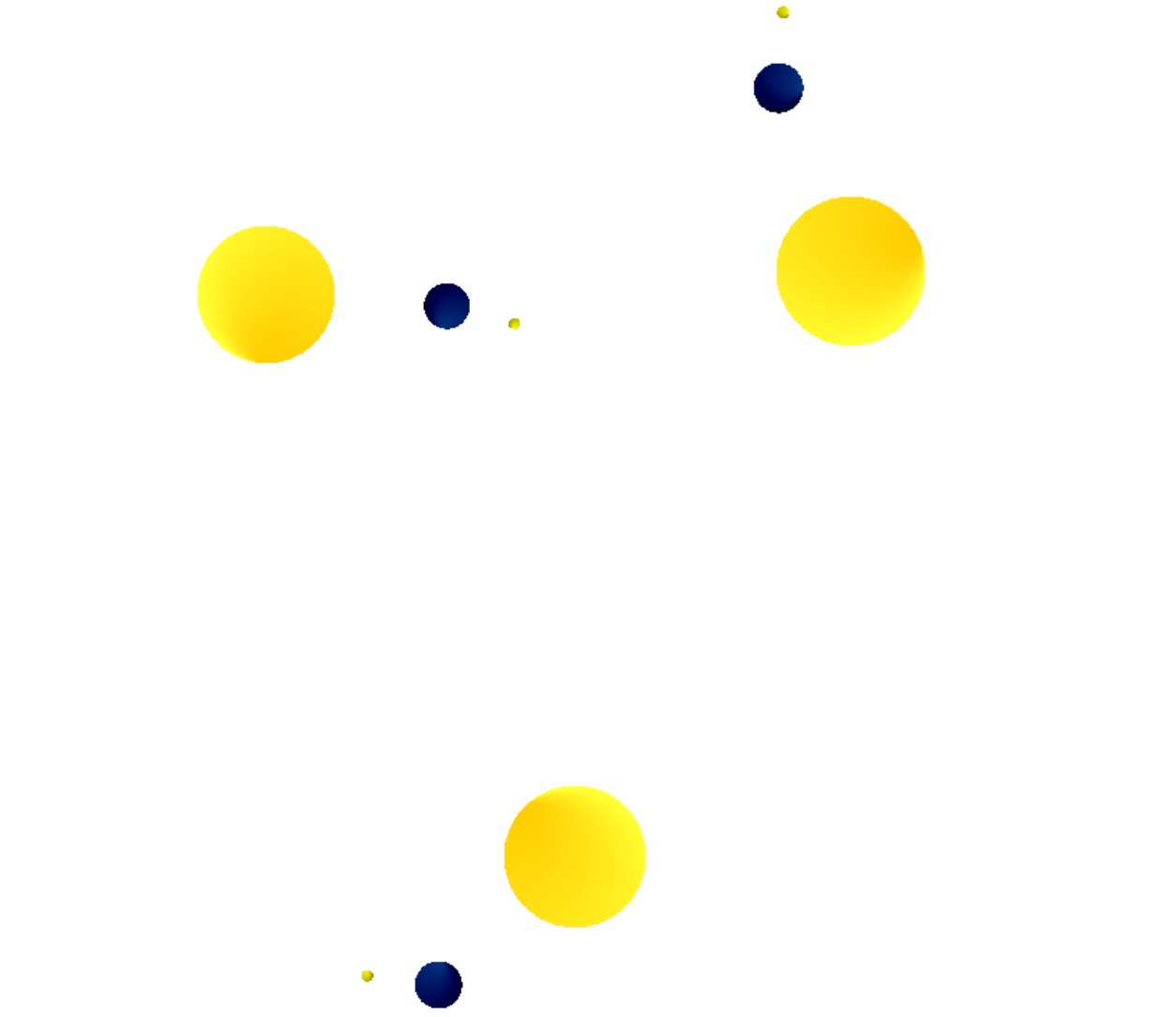
```
myMatrix = resizeMatrix * rotationOwnAxis(1) * sunScaling * circleScaling ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_sun, GL_UNSIGNED_INT, (void*)(0)); //SOARE

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

myMatrix = resizeMatrix * ellipseSize * rotationOwnAxis(3) * circleScaling ;// * cycle * ellipseMovement * ellipseRotat
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_earth, GL_UNSIGNED_INT, (void*)((ind_sun + 1) * sizeof(GLuint))); //PAMANT

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

myMatrix = resizeMatrix * ellipseSize * rotationOwnAxis(3) * moonTranslate * moonScale * circleScaling ;// * cycle * el
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_moon, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1) * sizeof(GLuint))); //LUNA
```



Pământului și Lunii le-am adăugat o rotație pentru a orbita în jurul Soarelui.

```
ellipseRotation = glm::rotate(glm::mat4(1.0f), twoPI*rotationMult, glm::vec3(0.0, 0.0, 1.0)); // rotatia elipsei
```

```
myMatrix = resizeMatrix * rotationOwnAxis(1) * sunScaling * circleScaling ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_sun, GL_UNSIGNED_INT, (void*)(0)); //SOARE

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

myMatrix = resizeMatrix * ellipseRotation * ellipseSize * rotationOwnAxis(3) * circleScaling ;// * cycle * ellipseMovement
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_earth, GL_UNSIGNED_INT, (void*)((ind_sun + 1) * sizeof(GLuint))); //PAMANT

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

myMatrix = resizeMatrix * ellipseRotation * ellipseSize * rotationOwnAxis(3) * moonTranslate * moonScale * circleScaling ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_moon, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1) * sizeof(GLuint))); //LUNA
```

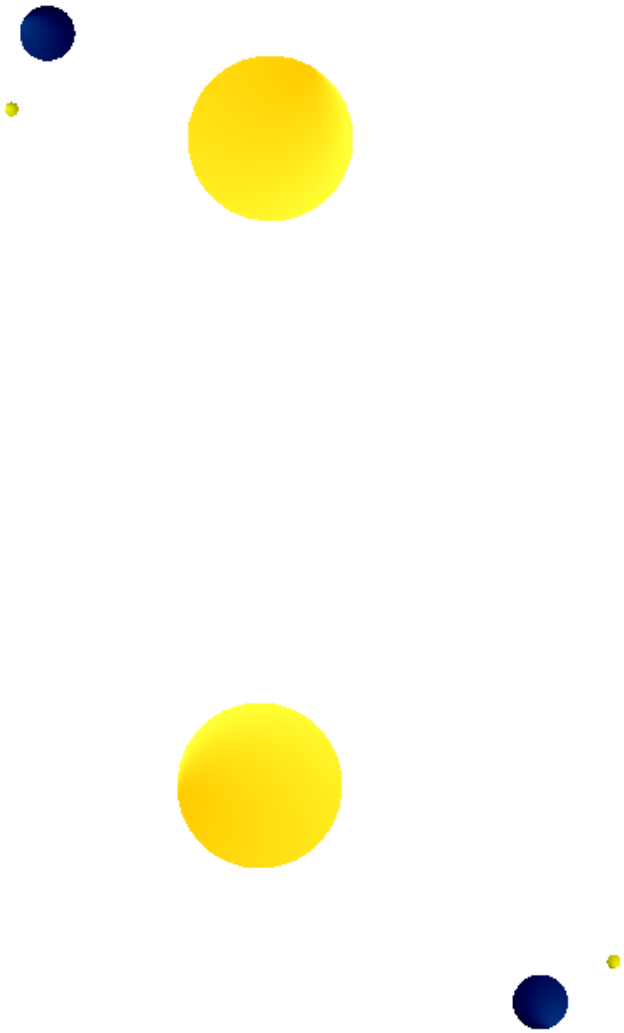


Pământului și Lunii le-am adăugat o translație pentru a orbita în jurul Soarelui în forma unei elipse.

```
float eccentricity = 170; // dimensiunea mare a elipsei
float xOffset = 130;
float semiMinor = 130; // dimensiunea mica a elipsei
```

```
ellipseMovement = glm::translate(glm::mat4(1.0f), glm::vec3(xOffset+ eccentricity* glm::cos(twoPI*rotationMult), 0, 0.0));
```

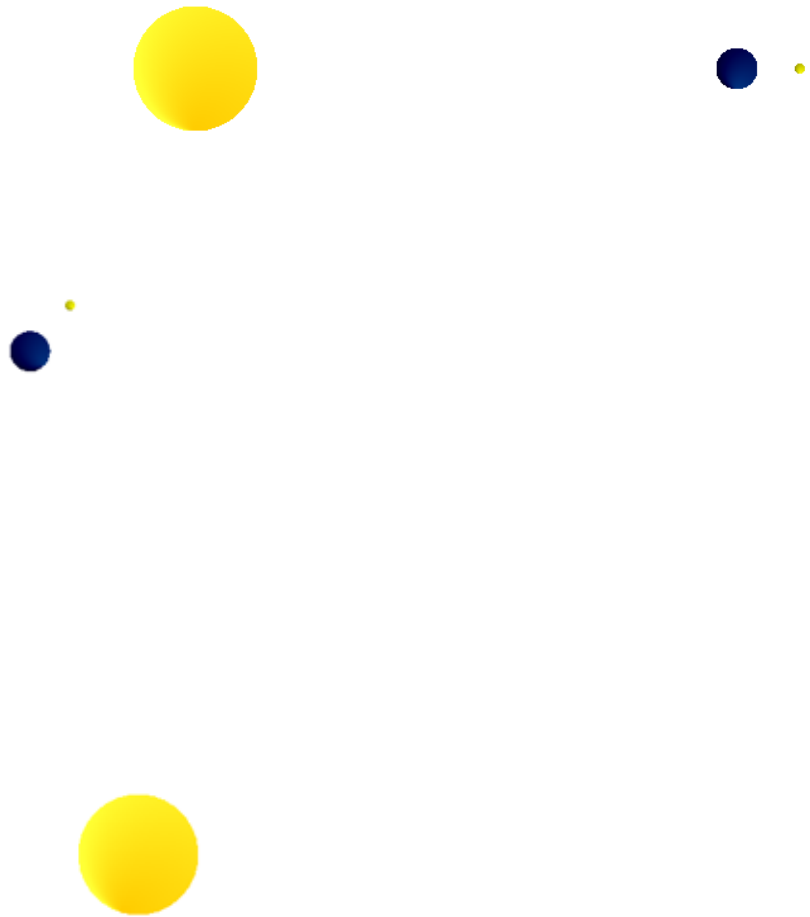
```
myMatrix = resizeMatrix * rotationOwnAxis(1) * sunScaling * circleScaling ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_sun, GL_UNSIGNED_INT, (void*)(0)); //SOARE

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

myMatrix = resizeMatrix * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(3) * circleScaling ;// * cycle ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_earth, GL_UNSIGNED_INT, (void*)((ind_sun + 1) * sizeof(GLuint))); //PAMANT

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

myMatrix = resizeMatrix * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(3) * moonTranslate * moonScale * circleScaling ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_moon, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1) * sizeof(GLuint))); //LUNA
```



Pământului și Lunii le-am adăugat o rotație pentru a orbita în jurul Soarelui în forma unei elipse care se învârte în jurul propriei axe.

```
cycle = glm::rotate(glm::mat4(1.0f), 0.1f * cycleMult * twoPI, glm::vec3(0.0, 0.0, 1.0));
```

```
myMatrix = resizeMatrix * rotationOwnAxis(1) * sunScaling * circleScaling ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_sun, GL_UNSIGNED_INT, (void*)(0)); //SOARE

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

myMatrix = resizeMatrix * cycle * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(3) * circleScaling ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_earth, GL_UNSIGNED_INT, (void*)((ind_sun + 1) * sizeof(GLuint))); //PAMANT

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

myMatrix = resizeMatrix * cycle * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(3) * moonTranslate * moonScale * circleScaling ;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_moon, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1) * sizeof(GLuint))); //LUNA
```

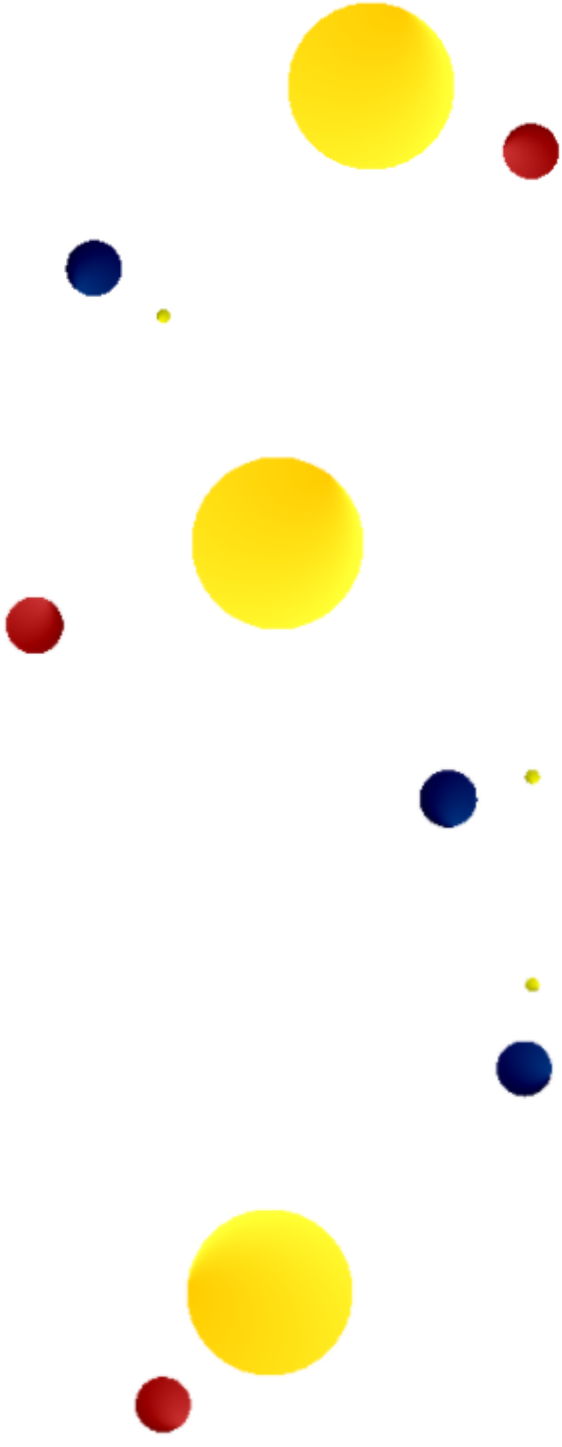


În urma discuției individuale de la laborator am adăugat următoarele elemente:

- Planeta Mercur căreia i-am adăugat scalarile, translatiile și rotațiile Pământului, dar modificate astfel încât să fie mai aproape de Soare.

```
ind_mercury = CreateCircle(0.0f, 0.0f, 50.0f, 30, 92, 0.8f, 0.2f, 0.2f, -0.015f);
```

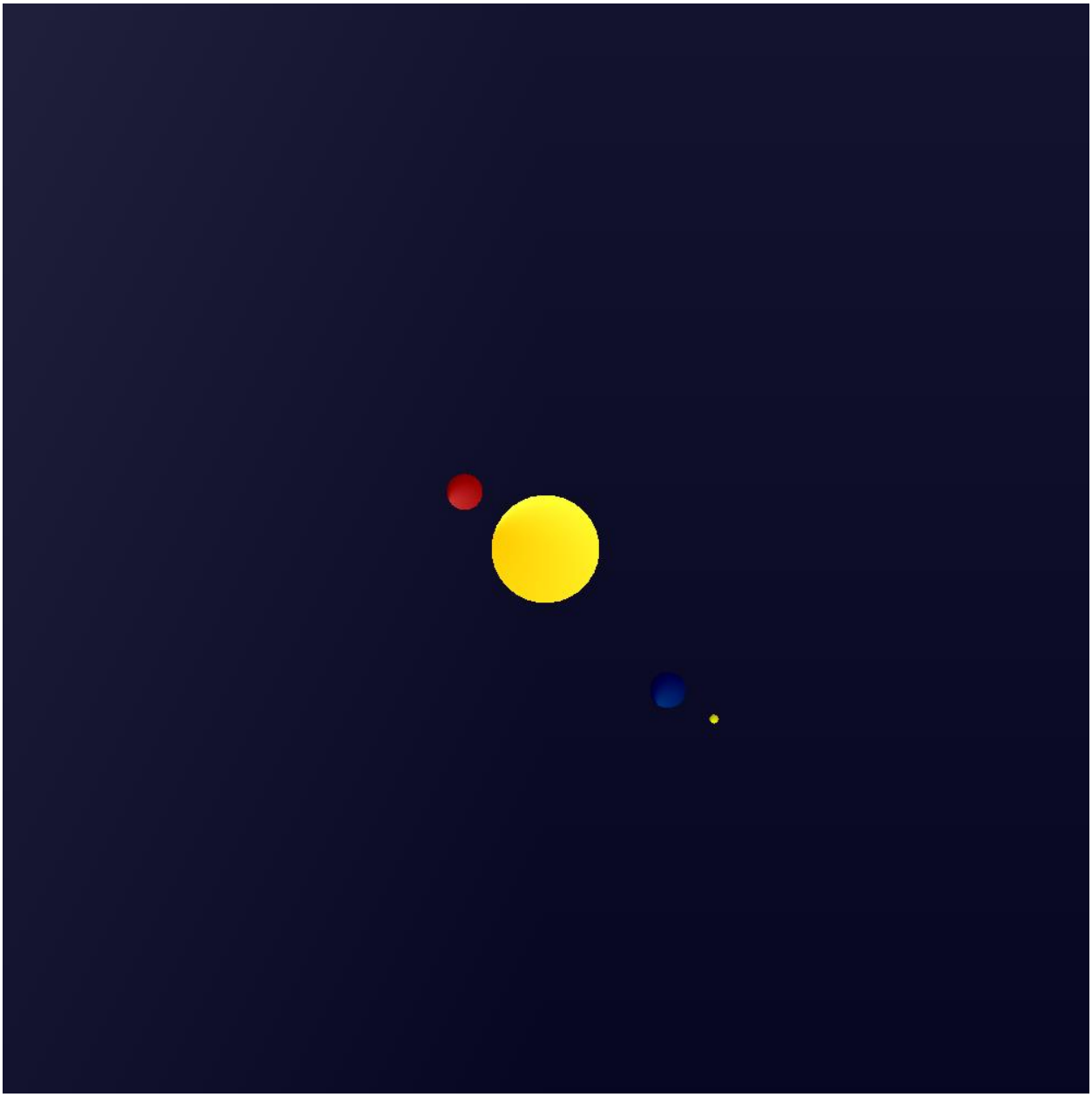
```
ellipseMovement = glm::translate(glm::mat4(1.0f), glm::vec3(60 + 70 * glm::cos(twoPI * mercuryMult), 0, 0.0)); // seteaza distanta elipsei fata de centru
ellipseSize = glm::translate(glm::mat4(1.0f), glm::vec3(100, 0, 0.0)); // seteaza cat de lata va fi elipsa
ellipseRotation = glm::rotate(glm::mat4(1.0f), twoPI * mercuryMult, glm::vec3(0.0, 0.0, 1.0)); // rotatia elipsei


myMatrix = resizeMatrix * cycle * ellipseMovement * ellipseRotation * ellipseSize * rotationOwnAxis(2) * circleScaling;
setMatrix(myMatrix);

glDrawElements(GL_POLYGON, ind_mercury, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1 + ind_moon + 1) * sizeof(GLuint))); //MERCUR
```



- Un fundal de culoare albastru-închis.

```
ind_background = CreateCircle(0.0f, 0.0f, 1000.0f, 4, 130, 0.0f, 0.0f, 0.1f, 0.05f);
```

```
myMatrix = resizeMatrix;
setMatrix(myMatrix);
glDrawElements(GL_POLYGON, ind_background, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1 + ind_moon + 1 + ind_mercury + 1 + ind_star + 1) * sizeof(GLuint))); //FUNDAL
```

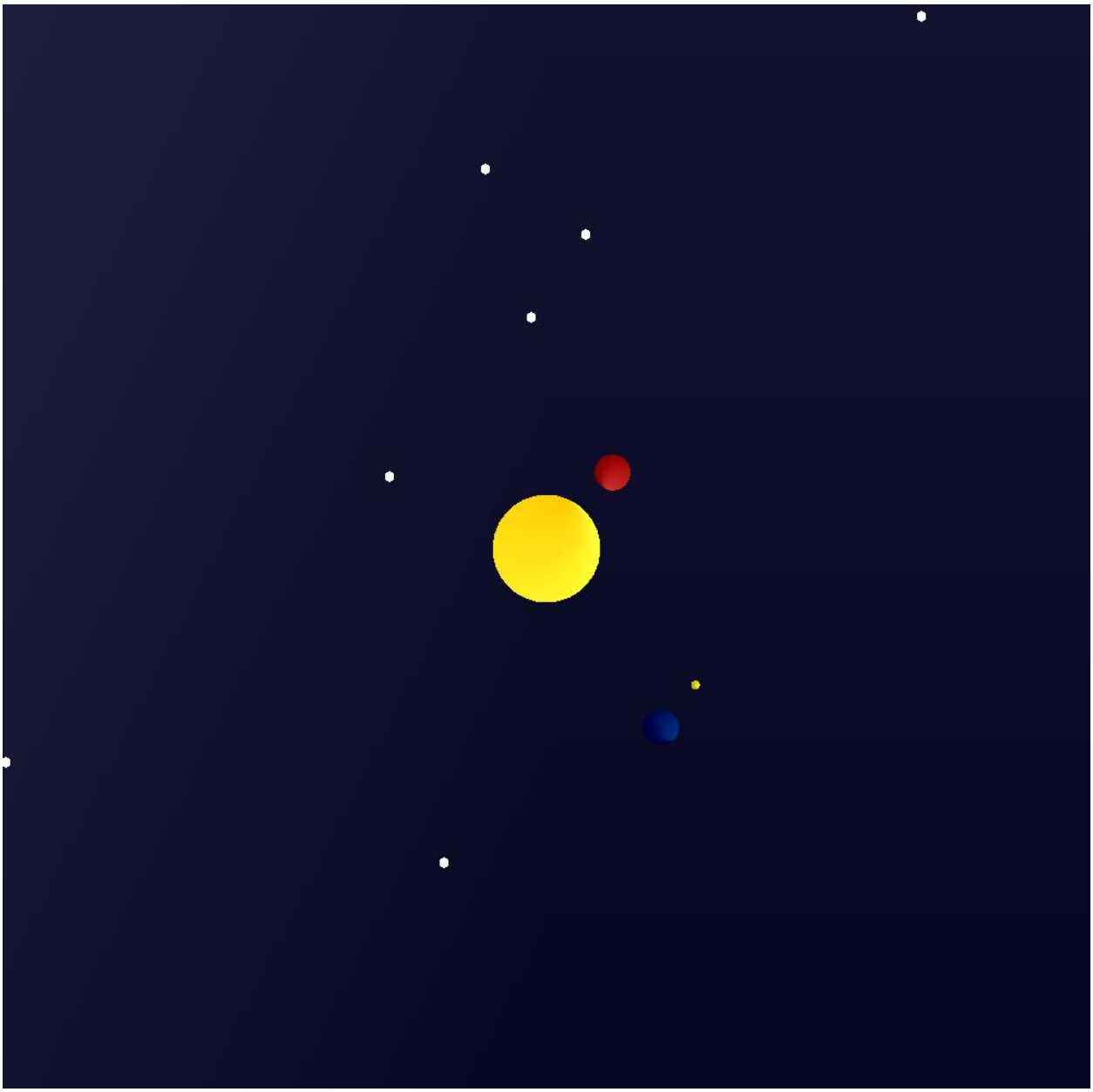


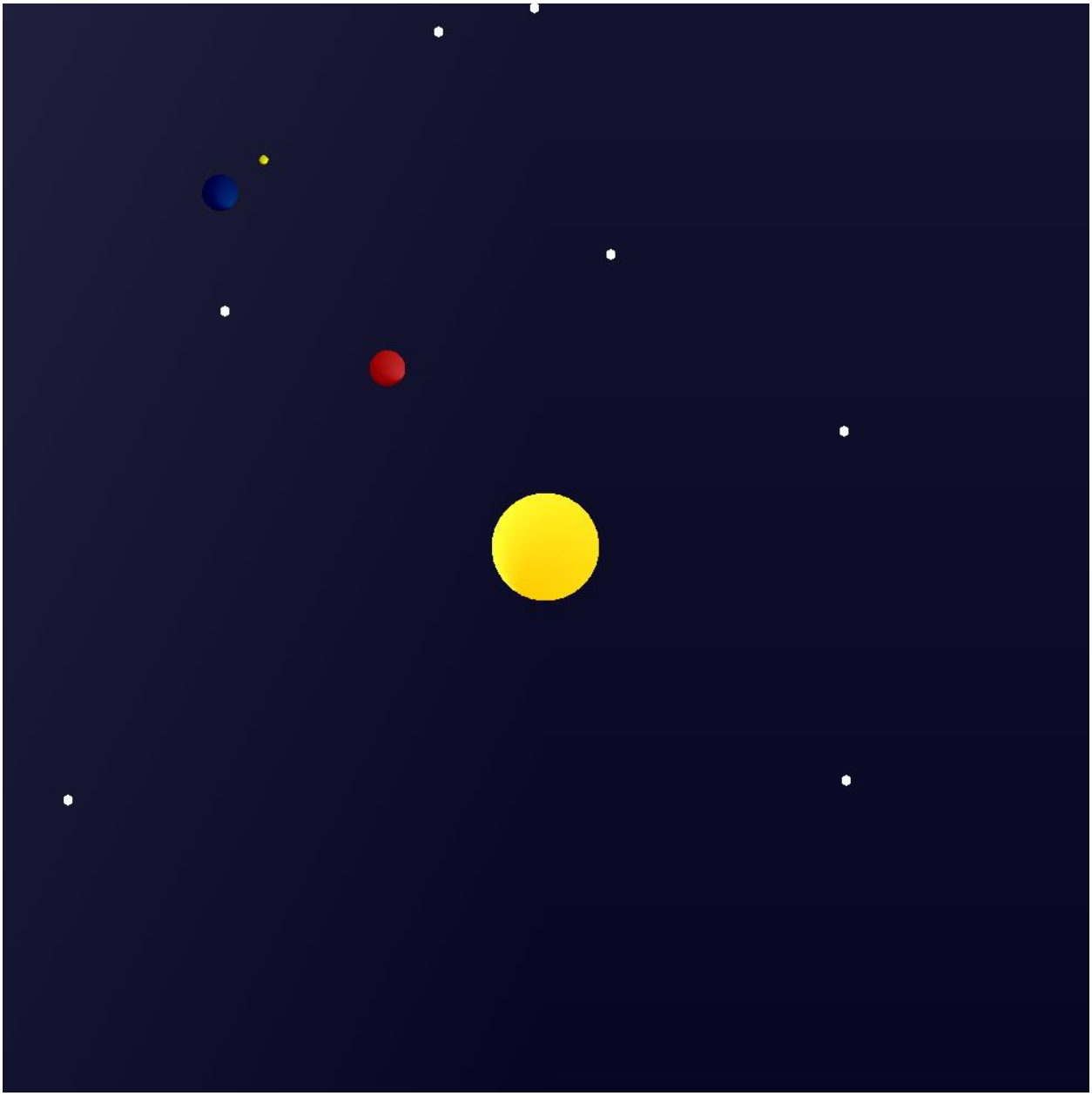
- 8 stele generate aleatoriu odată pe secundă.

```
ind_star = CreateCircle(0.0f, 0.0f, 5.0f, 6, 123, 1, 1, 1, 0);
```

```
glm::highp_mat4 randomStar() {
    float randX = (rand() % width * 2) - width;
    float randY = (rand() % height * 2) - height;
    return glm::translate(glm::mat4(1.0f), glm::vec3(randX, randY, 0));
}
```

```
for (int i = 0; i < 7; i++) {

    myMatrix = resizeMatrix * randomStar();
    setMatrix(myMatrix);
    glDrawElements(GL_POLYGON, ind_star, GL_UNSIGNED_INT, (void*)((ind_sun + 1 + ind_earth + 1 + ind_moon + 1 + ind_mercury + 1) * sizeof(GLuint))); //STELE
}
```





Rezultatul final poate fi urmărit la:

https://www.youtube.com/watch?v=1QPp7chdQ3U

# Contribuții individuale

Fiecare dintre membri echipei a contribuit la proiect astfel:

*Tudorache Alexandru-Theodor:*

- Găsirea unei formule matematice pentru calcularea coordonatelor punctelor unui poligon regulat cu n vârfuri pornind de la raza și centrul cercului circumscris acestui poligon.
- Aplicarea formulei menționată la punctul precedent într-o funcție care desenează folosind OpenGL un poligon regulat cu n vârfuri, plecând de la centrul și raza cercului circumscris poligonului și indexul vectorului de indici ai varfurilor de la care trebuie să înceapă desenarea poligonului.
- Aplicarea funcției menționată la punctul precedent pentru a desena un poligon regulat cu un număr de vârfuri suficient de mare (30) încât să ajungă să fie cerc.
- Colorarea cercurilor astfel încât să nu aibă o culoare uniformă și să se vadă transformările aplicate fiecărui cerc.
- Adăugarea fundalului.
- Realizarea documentației.

*Zaharia Robert-Catalin:*

- Conceperea unei rotații eliptice care să simuleze o orbită reală văzută în două dimensiuni.
- Sincronizarea pendulării centrului de rotație a obiectului cu rotația obiectului în jurul centrului menționat pentru a obține o elipsă perfectă.
- Crearea unor variabile care să își schimbe valoarea în timp pentru a obține rotații dinamice.
- Ordonarea diverselor transformări simple precum rotația și translația pentru a obține transformări complexe precum o elipsă care se rotește.
- Utilizarea valorilor adecvate pentru variabile astfel încât rezultatul să fie satisfăcător.
- Adăugarea celor 8 stele generate aleatoriu în desen.
- Realizarea documentației.

# Anexe (Cod sursă):

## *05_02_Shader.frag*

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

 #version 400

in vec4 ex_Color;
uniform int codCol;

out vec4 out_Color;

void main(void)
  {
       if ( codCol==0 )
              out_Color = ex_Color;
       if ( codCol==1 )
              out_Color=vec4 (0.0, 0.0, 1.0, 0.0);
       if ( codCol==2 )
              out_Color=vec4 (1.0, 0.0, 0.0, 0.0);

  }
```

## *05_02_Shader.vert*

```
// Shader-ul de varfuri

 #version 400


in vec4 in_Position;
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;
   }
```

## *loadShaders.h*

```
GLuint LoadShaders(const char * vertex_file_path,const char * fragment_file_path);
```

## *loadShaders.cpp*

```
//  Preluat si adaptat dupa http://www.opengl-tutorial.org/beginners-tutorials
```

```
#include <vector>
#include <iostream>
#include <string>
#include <fstream>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <minmax.h>

using namespace std;

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


GLuint LoadShaders(const char * vertex_file_path,const char * fragment_file_path){

    // Creaza shadere
    GLuint VertexShaderID = glCreateShader(GL_VERTEX_SHADER);
    GLuint FragmentShaderID = glCreateShader(GL_FRAGMENT_SHADER);

    // Citeste din fisier shader-ul de varf
    std::string VertexShaderCode;
    std::ifstream VertexShaderStream(vertex_file_path, std::ios::in);
    if(VertexShaderStream.is_open())
    {
        std::string Line = "";
        while(getline(VertexShaderStream, Line))
            VertexShaderCode += "\n" + Line;
        VertexShaderStream.close();
    }

    // Citeste din fisier shader-ul de fragment
    std::string FragmentShaderCode;
    std::ifstream FragmentShaderStream(fragment_file_path, std::ios::in);
    if(FragmentShaderStream.is_open()){
        std::string Line = "";
        while(getline(FragmentShaderStream, Line))
            FragmentShaderCode += "\n" + Line;
        FragmentShaderStream.close();
    }

    GLint Result = GL_FALSE;
    int InfoLogLength;

    // Compileaza shader-ul de varf
    printf("Compilare shader : %s\n", vertex_file_path);
    char const * VertexSourcePointer = VertexShaderCode.c_str();
    glShaderSource(VertexShaderID, 1, &VertexSourcePointer , NULL);
    glCompileShader(VertexShaderID);

   /*
    // Verifica
    glGetShaderiv(VertexShaderID, GL_COMPILE_STATUS, &Result);
    glGetShaderiv(VertexShaderID, GL_INFO_LOG_LENGTH, &InfoLogLength);
    std::vector<char> VertexShaderErrorMessage(InfoLogLength);
```

```
    glGetShaderInfoLog(VertexShaderID, InfoLogLength, NULL,
&VertexShaderErrorMessage[0]);
    fprintf(stdout, "%s\n", &VertexShaderErrorMessage[0]);
       */

    // Compileaza shader-ul de fragemnt
    printf("Compilare shader : %s\n", fragment_file_path);
    char const * FragmentSourcePointer = FragmentShaderCode.c_str();
    glShaderSource(FragmentShaderID, 1, &FragmentSourcePointer , NULL);
    glCompileShader(FragmentShaderID);
 /*

    // Verifica
    glGetShaderiv(FragmentShaderID, GL_COMPILE_STATUS, &Result);
    glGetShaderiv(FragmentShaderID, GL_INFO_LOG_LENGTH, &InfoLogLength);
    std::vector<char> FragmentShaderErrorMessage(InfoLogLength);
    glGetShaderInfoLog(FragmentShaderID, InfoLogLength, NULL,
&FragmentShaderErrorMessage[0]);
    fprintf(stdout, "%s\n", &FragmentShaderErrorMessage[0]);
 */
    // Leaga programul
    fprintf(stdout, "Legare program\n");
    GLuint ProgramID = glCreateProgram();
    glAttachShader(ProgramID, VertexShaderID);
    glAttachShader(ProgramID, FragmentShaderID);
    glLinkProgram(ProgramID);
 /*
    // Verifica
    glGetProgramiv(ProgramID, GL_LINK_STATUS, &Result);
    glGetProgramiv(ProgramID, GL_INFO_LOG_LENGTH, &InfoLogLength);
    std::vector<char> ProgramErrorMessage( max(InfoLogLength, int(1)) );
    glGetProgramInfoLog(ProgramID, InfoLogLength, NULL, &ProgramErrorMessage[0]);
    fprintf(stdout, "%s\n", &ProgramErrorMessage[0]);
 */
    glDeleteShader(VertexShaderID);
    glDeleteShader(FragmentShaderID);

       glUseProgram(0);

  glDetachShader(ProgramID, VertexShaderID);
  glDetachShader(ProgramID, FragmentShaderID);

  glDeleteShader(FragmentShaderID);
  glDeleteShader(VertexShaderID);

    return ProgramID;
}
```

## *SolarSystem.cpp*

```
#include <windows.h>  // biblioteci care urmeaza sa fie incluse
#include <stdlib.h> // necesare pentru citirea shader-elor
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <GL/glew.h> // glew apare inainte de freeglut
#include <GL/freeglut.h> // nu trebuie uitat 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 <ctime>


#define PI 3.141592f

using namespace std;


GLuint
VaoId,
VboId,
EboId,
ColorBufferId,
ProgramId,
myMatrixLocation,
codColLocation;

glm::mat4 myMatrix, resizeMatrix, circleScaling, ellipseRotation, ellipseSize,
ellipseMovement, sunScaling, cycle, moonTranslate, moonScale;

int codCol;
float twoPI = 2 * PI;
float tx = 0; float ty = 0;
int width = 500, height = 500;
float beta = 0.01f;


float rotationMult = 0.0f;
float mercuryMult = 0.0f;
float cycleMult = 0.0;


GLfloat vf_pos[1000];
GLfloat vf_col[1000];
GLuint vf_ind[250];
int ind_sun, ind_earth, ind_moon, ind_mercury, ind_star, ind_background;


int CreateCircle(GLfloat Cx = 0.0f, GLfloat Cy = 0.0f, GLfloat radius = 10.0f, int
noOfVertex = 3, int index = 0, GLfloat red = 0.0f, GLfloat green = 0.0f, GLfloat blue =
0.0f, GLfloat offset = 0.01f)
{
    int initial_index = index;
    int index_pos = index * 4;
    int index_ind;
    if (index)
        index_ind = index + 1;
    else
        index_ind = index;
```

```
    float theta = PI * 1.5f;
    GLfloat Px = Cx + radius * GLfloat(glm::cos(theta));
    GLfloat Py = Cy + radius * GLfloat(glm::sin(theta));
    vf_col[index_pos] = red;
    red += offset;
    vf_pos[index_pos++] = Px;
    vf_col[index_pos] = green;
    green += offset;
    vf_pos[index_pos++] = Py;
    vf_col[index_pos] = blue;
    blue += offset;
    vf_pos[index_pos++] = 0.0f;
    vf_col[index_pos] = 1.0f;
    vf_pos[index_pos++] = 1.0f;

    vf_ind[index_ind++] = GLuint(index++);

    for (int i = 0; i < noOfVertex - 1; i++)
    {
        theta += twoPI / noOfVertex;
        Px = Cx + radius * GLfloat(glm::cos(theta));
        Py = Cy + radius * GLfloat(glm::sin(theta));
        vf_col[index_pos] = red;
        red += offset;
        vf_pos[index_pos++] = Px;
        vf_col[index_pos] = green;
        green += offset;
        vf_pos[index_pos++] = Py;
        vf_col[index_pos] = blue;
        blue += offset;
        vf_pos[index_pos++] = 0.0f;
        vf_col[index_pos] = 1.0f;
        vf_pos[index_pos++] = 1.0f;

        vf_ind[index_ind++] = index++;
    }

    vf_ind[index_ind] = 0;
    return noOfVertex;
}

void miscas(void)
{
    rotationMult = (rotationMult + beta);
    if (rotationMult > 1) rotationMult -= 1;

    mercuryMult = (mercuryMult + beta / 2);
    if (mercuryMult > 1) mercuryMult -= 1;

    cycleMult += beta;
    glutPostRedisplay();
}

void mouse(int button, int state, int x, int y)
{
    switch (button) {
    case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN)
```

```
            glutIdleFunc(miscas);
        break;
    default:
        break;
    }
}

void CreateVBO(void)
{
    srand((unsigned)time(0));
    ind_sun = CreateCircle(0.0f, 0.0f, 50.0f, 30, 0, 1.0f, 0.8f, 0.0f, 0.01f);
    ind_earth = CreateCircle(0.0f, 0.0f, 50.0f, 30, 30, 0.0f, 0.2f, 0.5f, -0.015f);
    ind_moon = CreateCircle(0.0f, 0.0f, 50.0f, 30, 61, 1.0f, 1.0f, 0.1f, -0.02f);
    ind_mercury = CreateCircle(0.0f, 0.0f, 50.0f, 30, 92, 0.8f, 0.2f, 0.2f, -0.015f);
    ind_star = CreateCircle(0.0f, 0.0f, 5.0f, 6, 123, 1, 1, 1, 0);
    ind_background = CreateCircle(0.0f, 0.0f, 1000.0f, 4, 130, 0.0f, 0.0f, 0.1f, 0.05f);

    // se creeaza un buffer nou pentru varfuri
    glGenBuffers(1, &VboId);
    // buffer pentru indici
    glGenBuffers(1, &EboId);
    // se creeaza / se leaga un VAO (Vertex Array Object)
    glGenVertexArrays(1, &VaoId);

    // legare VAO
    glBindVertexArray(VaoId);

    // buffer-ul este setat ca buffer curent
    glBindBuffer(GL_ARRAY_BUFFER, VboId);

    // buffer-ul va contine atat coordonatele varfurilor, cat si datele de culoare
    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);

    // se activeaza lucrul cu atribute; atributul 0 = pozitie, atributul 1 = culoare,
acestea sunt indicate corect in VBO
    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("05_02_Shader.vert", "05_02_Shader.frag");
    glUseProgram(ProgramId);
}

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

void Initialize(void)
{
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f); // culoarea de fond a ecranului
}

glm::highp_mat4 rotationOwnAxis(float speed) {
    return glm::rotate(glm::mat4(1.0f), speed * twoPI * rotationMult, glm::vec3(0.0, 0.0,
1.0));
}

glm::highp_mat4 randomStar() {
    float randX = (rand() % width * 2) - width;
    float randY = (rand() % height * 2) - height;
    return glm::translate(glm::mat4(1.0f), glm::vec3(randX, randY, 0));
}

void setMatrix(glm::mat4 myMatrix) {
    CreateShaders();
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
}

void RenderFunction(void)
{
    float eccentricity = 170; // dimensiunea mare a elipsei
    float xOffset = 130;
    float semiMinor = 170; // dimensiunea mica a elipsei
    float moonOffset = 50;

    resizeMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(1.f / width, 1.f / height,
1.0)); // scalam, "aducem" scena la "patratul standard" [-1,1]x[-1,1]

    ellipseMovement = glm::translate(glm::mat4(1.0f), glm::vec3(xOffset + eccentricity *
glm::cos(twoPI * rotationMult), 0, 0.0)); // seteaza distanta elipsei fata de centru

    ellipseSize = glm::translate(glm::mat4(1.0f), glm::vec3(semiMinor, 0, 0.0)); //
seteaza cat de lata va fi elipsa

    circleScaling = glm::scale(glm::mat4(1.0f), glm::vec3(0.33, 0.33, 0.0));

    ellipseRotation = glm::rotate(glm::mat4(1.0f), twoPI * rotationMult, glm::vec3(0.0,
0.0, 1.0)); // rotatia elipsei
```

```
    sunScaling = glm::scale(glm::mat4(1.0f), glm::vec3(3, 3, 0.0));

    cycle = glm::rotate(glm::mat4(1.0f), 0.1f * cycleMult * twoPI, glm::vec3(0.0, 0.0,
1.0)); // cat de repede se roteste elipsa in jurul centrului

    moonTranslate = glm::translate(glm::mat4(1.0f), glm::vec3(moonOffset, 0, 0.0)); //
translatia lunii

    moonScale = glm::scale(glm::mat4(1.0f), glm::vec3(0.25, 0.25, 0.0));

    glClear(GL_COLOR_BUFFER_BIT);
    CreateVBO();

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

    myMatrix = resizeMatrix;
    setMatrix(myMatrix);
    glDrawElements(GL_POLYGON, ind_background, GL_UNSIGNED_INT, (void*)((ind_sun + 1 +
ind_earth + 1 + ind_moon + 1 + ind_mercury + 1 + ind_star + 1) * sizeof(GLuint)));
//FUNDAL

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

    for (int i = 0; i < 7; i++) {

        myMatrix = resizeMatrix * randomStar();
        setMatrix(myMatrix);
        glDrawElements(GL_POLYGON, ind_star, GL_UNSIGNED_INT, (void*)((ind_sun + 1 +
ind_earth + 1 + ind_moon + 1 + ind_mercury + 1) * sizeof(GLuint))); //STELE
    }

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

    myMatrix = resizeMatrix * rotationOwnAxis(1) * sunScaling * circleScaling;
    setMatrix(myMatrix);

    glDrawElements(GL_POLYGON, ind_sun, GL_UNSIGNED_INT, (void*)(0)); //SOARE

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

    myMatrix = resizeMatrix * cycle * ellipseMovement * ellipseRotation * ellipseSize *
rotationOwnAxis(3) * circleScaling;
    setMatrix(myMatrix);

    glDrawElements(GL_POLYGON, ind_earth, GL_UNSIGNED_INT, (void*)((ind_sun + 1) *
sizeof(GLuint))); //PAMANT

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

    myMatrix = resizeMatrix * cycle * ellipseMovement * ellipseRotation * ellipseSize *
rotationOwnAxis(3) * moonTranslate * moonScale * circleScaling;
    setMatrix(myMatrix);

    glDrawElements(GL_POLYGON, ind_moon, GL_UNSIGNED_INT, (void*)((ind_sun + 1 +
ind_earth + 1) * sizeof(GLuint))); //LUNA

    ///////////////
```

```
    ellipseMovement = glm::translate(glm::mat4(1.0f), glm::vec3(60 + 70 * glm::cos(twoPI
* mercuryMult), 0, 0.0)); // seteaza distanta elipsei fata de centru
    ellipseSize = glm::translate(glm::mat4(1.0f), glm::vec3(100, 0, 0.0)); // seteaza cat
de lata va fi elipsa
    ellipseRotation = glm::rotate(glm::mat4(1.0f), twoPI * mercuryMult, glm::vec3(0.0,
0.0, 1.0)); // rotatia elipsei


    myMatrix = resizeMatrix * cycle * ellipseMovement * ellipseRotation * ellipseSize *
rotationOwnAxis(2) * circleScaling;
    setMatrix(myMatrix);

    glDrawElements(GL_POLYGON, ind_mercury, GL_UNSIGNED_INT, (void*)((ind_sun + 1 +
ind_earth + 1 + ind_moon + 1) * sizeof(GLuint))); //MERCUR

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

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

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


    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(1000, 1000);
    glutCreateWindow("Orbite 2D");
    glewInit();
    Initialize();
    glutDisplayFunc(RenderFunction);
    glutMouseFunc(mouse);
    glutCloseFunc(Cleanup);
    glutMainLoop();
}
```