Documentație Proiect – Sistem Solar 2D (OpenGL + GLUT)

1. Conceptul Proiectului

Proiectul reprezintă un sistem solar 2D stil „radar”, realizat în OpenGL și GLUT. În centrul scenei se află Soarele, iar în jurul lui se rotesc cele opt planete principale: Mercur, Venus, Pământ, Marte, Jupiter, Saturn, Uranus și Neptun. Fiecare planetă are o orbită proprie, mărime diferită și viteză de rotație diferită. Planetele și Soarele sunt randate folosind texturi BMP cu fundal transparent (color-key).

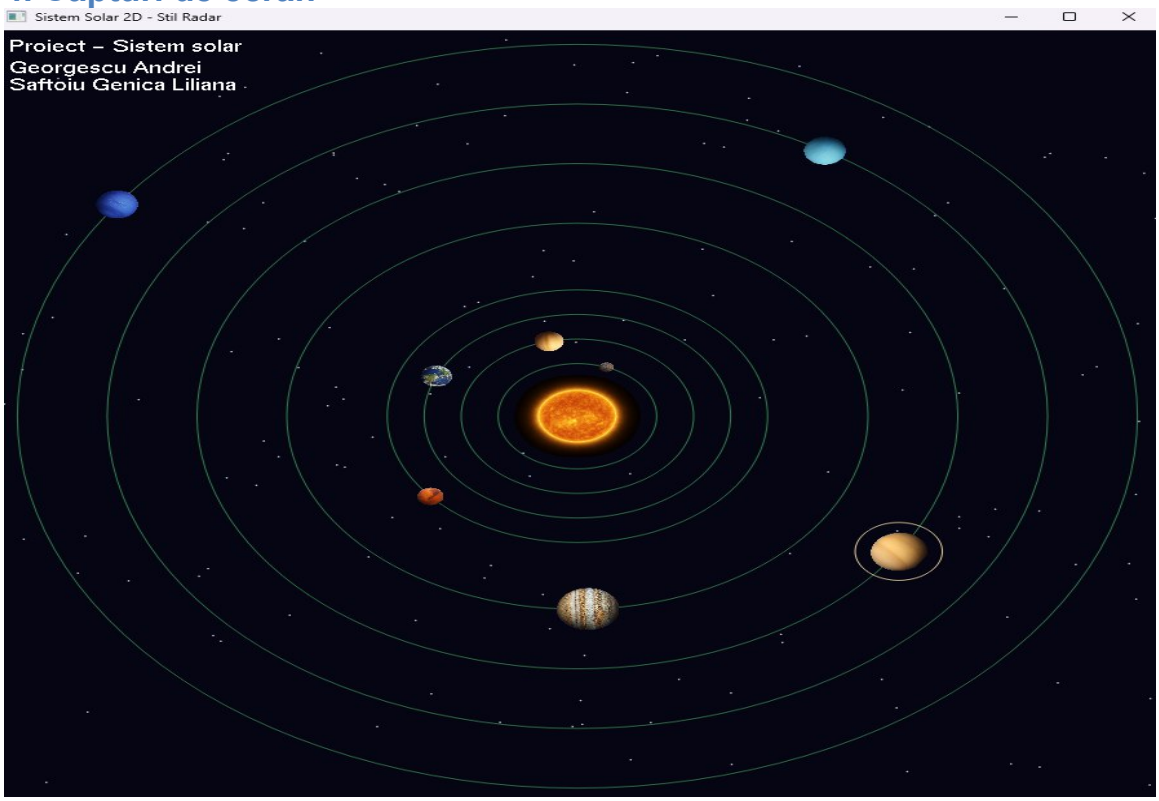
2. Transformări utilizate

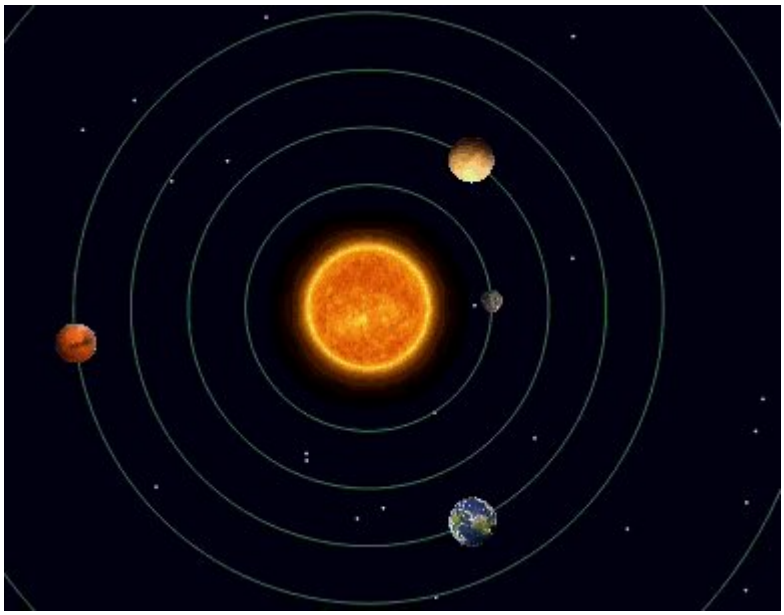
• **Rotire (glRotatef)** – folosită pentru animarea planetelor în jurul Soarelui.
• **Translație (glTranslatef)**– folosită pentru poziționarea planetelor de-a lungul orbitelor.
• **Proiecție ortografică (gluOrtho2D)** – menține sistemul solar centrat și scalat corect.
• **Coordonate 2D pentru HUD** – afișarea textului fix în colțul ferestrei.

3. Elemente de originalitate

• Texturi BMP pentru toate planetele + Soarele, cu transparență aplicată manual.
• Loader BMP personalizat care citește header-ul, reconstruiește imaginea în format RGBA și elimină fundalul negru.
• Fundal procedural cu stele generate aleator.
• Orbite stil „radar”, cu efect vizual coerent.
• Set parametri planetari (raze, viteze, poziții) ușor de modificat.

4. Capturi de ecran





I) Transformări + animație

Mișcarea planetelor se întâmplă din combinația a **două lucruri**:

1. **actualizarea unghiului fiecărei planete în timp - actualizare()**
2. **apelarea continuă a funcției de desenare – afisare()**

```
// actualizeaza unghiurile planetelor (animatie)
void actualizare(int value)
{
    for (int i = 0; i < nrPlanete; i++)
    {
        planete[i].unghiDeg += planete[i].viteza;
        if (planete[i].unghiDeg > 360.0f)
            planete[i].unghiDeg -= 360.0f;
    }

    glutPostRedisplay();
    glutTimerFunc(16, actualizare, 0); // ~60 FPS (16 ms)
}
```

```
// PLANETE
for (int i = 0; i < nrPlanete; i++)
{
    Planeta& p = planete[i];

    glPushMatrix();

    // 1) rotatie in jurul soarelui (origine)
    glRotatef(p.unghiDeg, 0.0f, 0.0f, 1.0f);

    // 2) translatie pe axa X cu raza orbitei
    glTranslatef(p.razaOrbitei, 0.0f, 0.0f);

    if (texturiPlanete[i] != 0)
    {
        // PLANETA TEXTURATA
        glEnable(GL_TEXTURE_2D);
```

Prin urmare mișcarea planetelor este realizată prin actualizarea unghiului fiecărei planete în funcția actualizare() și aplicarea unei transformări de rotație + translație în funcția afisare()

II) <u>Stelele generate aleator + desenate</u>

Numărul de stele este definit în variabila **NR_STELE**,iar pozitiile X/Y ale acestora in **steaX și steaY**

```
// stele fundal
const int NR_STELE = 150;
float steaX[NR_STELE];
float steaY[NR_STELE];
```

Funcția care generează pozițiile aleatoare X/Y ale stelelor este – genereazaStele(xmin, xmax, ymin, ymax), aceasta:

- Primeste ca argumente dimensiunile (xmin/max, ymin/max) in funcție de dimensiunea ferestrei care este calculată în funcția redimensioneaza()
- După care cu ajutorul acestora calculează pozițiile X/Y pentru fiecare stea – între [-1,1]f

```
void genereazaStele(float xmin, float xmax, float ymin, float ymax)
{
    for (int i = 0; i < NR_STELE; i++)
    {
        float rx = (float)rand() / RAND_MAX; // [0,1]
        float ry = (float)rand() / RAND_MAX;

        //
        steaX[i] = xmin + rx * (xmax - xmin);
        steaY[i] = ymin + ry * (ymax - ymin);
    }
}
```

- Stelele sunt desenate apoi cu ajutorul funcției deseneazaSteleFundal()

```
void deseneazaSteleFundal()
{
    glPointSize(1.5f);
    glBegin(GL_POINTS);
    for (int i = 0; i < NR_STELE; i++)
    {
        glColor3f(0.9f, 0.9f, 1.0f); // culoare stele fundal
        glVertex2f(steaX[i], steaY[i]);
    }
    glEnd();
}
```

- Funcția redimensionează

```
void redimensioneaza(int w, int h)
{
    latFereastra = w;
    inaltimeFereastra = h;

    glViewport(0, 0, w, h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    float aspect = (float)w / (float)h;

    float xmin, xmax, ymin, ymax;

    if (aspect >= 1.0f) {
        // fereastra mai lata decat inalta
        xmin = -1.1f * aspect;
        xmax = 1.1f * aspect;
        ymin = -1.1f;
        ymax = 1.1f;
    }
    else {
        // fereastra mai inalta decât lata
        xmin = -1.1f;
        xmax = 1.1f;
        ymin = -1.1f / aspect;
        ymax = 1.1f / aspect;
    }

    gluOrtho2D(xmin, xmax, ymin, ymax);
```

III) <u>Texturarea planetelor + desenare</u>

Pentru texturare am folosit un vector texturiPlanete[] prin care se asociază cate un fișier .bmp fiecărei planete astfel

texturiPlanete[1] = mercury.bmp

texturiPlanete[2] = venus.bmp

…

```
// texturi pentru planete
GLuint texturiPlanete[nrPlanete];
const char* fisiereTexturiPlanete[nrPlanete] = {
    "mercury.bmp",
    "venus.bmp",
    "earth.bmp",
    "mars.bmp",
    "jupiter.bmp",
    "saturn.bmp",
    "uranus.bmp",
    "neptune.bmp"
};
```

După care am încărcat fișierele BMP în memorie cu ajutorul funcției incarcaBMP() care:

- Deschide fișierul BMP
- Citește headerul
- Citește pixelii
- Transformă fundalul negru în transparent
- Creează textura OpenGL (RGBA) = fiecare imagine BMP devine o textură OpenGL

Funcția primește numeFisier care reprezintă numele imaginii și returnează GLuint care este ID-ul texturii OpenGL

```
// ========================================================================
GLuint incarcaBMP(const char* numeFisier)
{
    FILE* f = NULL;
```

Deschiderea fișierului BMP – deschide în modul rb = read binary

```
GLuint incarcaBMP(const char* numeFisier)
{
    FILE* f = NULL;
#ifdef _MSC_VER
    fopen_s(&f, numeFisier, "rb");
#else
    f = fopen(numeFisier, "rb");
#endif
    if (!f)
    {
        printf("Fisierul nu se poate deschide: %s\n", numeFisier);
        return 0;
    }

    unsigned char header[54];
    if (fread(header, 1, 54, f) != 54)
    {
        printf("Fisier BMP invalid (header prea scurt): %s\n", numeFisier);
        fclose(f);
        return 0;
    }
```

Citirea header-ului BMP (primii 54 biți) și verificarea semnăturii BM – standard BMP

```
unsigned char header[54];
if (fread(header, 1, 54, f) != 54)
{
    printf("Fisier BMP invalid (header prea scurt): %s\n", numeFisier);
    fclose(f);
    return 0;
}

// verificăm semnătura "BM"
if (header[0] != 'B' || header[1] != 'M')
{
    printf("Fisierul nu este BMP: %s\n", numeFisier);
    fclose(f);
    return 0;
}
```

Citirea informațiilor esențiale din header și verificarea daca suportă formatul 24 sau 32 bpp

```
int offsetDate = *(int*)&header[10];
int latime = *(int*)&header[18];
int inaltime = *(int*)&header[22];
short bitiPePixel = *(short*)&header[28];

if (bitiPePixel != 24 && bitiPePixel != 32)
{
    printf("BMP cu %d bpp nu este suportat (doar 24 sau 32): %s\n",
        bitiPePixel, numeFisier);
    fclose(f);
    return 0;
}
```

Calculul pentru rânduri

```
int bytesPerPixel = bitiPePixel / 8;
int bytesPeRand = latime * bytesPerPixel;
int randCuPadding = (bytesPeRand + 3) & (~3);
```

Alocarea memoriei pentru un rând și pentru toată imaginea

```
unsigned char* randBMP = (unsigned char*)malloc(randCuPadding);
unsigned char* sursa = (unsigned char*)malloc(latime * inaltime * bytesPerPixel);

if (!randBMP || !sursa)
{
    printf("Nu exista suficienta memorie pentru imagine: %s\n", numeFisier);
    if (randBMP) free(randBMP);
    if (sursa)   free(sursa);
    fclose(f);
    return 0;
}
```

Citirea pixelilor de jos în sus => sursa va conține imaginea completă în format BGR în memorie

```
// BMP stochează liniile de jos în sus
for (int y = 0; y < inaltime; y++)
{
    fread(randBMP, 1, randCuPadding, f);
    int indexDest = (inaltime - 1 - y) * bytesPeRand;
    for (int x = 0; x < bytesPeRand; x++)
    {
        sursa[indexDest + x] = randBMP[x];
    }
}
```

Închidem fișierul și eliberăm bufferul

```
fclose(f);
free(randBMP);
```

Construim bufferul final – RGBA (4 canale)

```
// Construim buffer RGBA și eliminăm fundalul negru (color key)
unsigned char* rgba = (unsigned char*)malloc(latime * inaltime * 4);
if (!rgba)
{
    printf("Nu exista memorie pentru RGBA: %s\n", numeFisier);
    free(sursa);
    return 0;
}
```

Copiem pixelii și aplicăm transparența

```
for (int i = 0; i < latime * inaltime; i++)
{
    unsigned char b = sursa[i * bytesPerPixel + 0];
    unsigned char g = sursa[i * bytesPerPixel + 1];
    unsigned char r = sursa[i * bytesPerPixel + 2];
    unsigned char a = 255;

    // daca exista alpha original (32 bpp) il putem citi
    if (bitiPePixel == 32)
    {
        unsigned char alphaOriginal = sursa[i * bytesPerPixel + 3];
        a = alphaOriginal;
    }

    // daca pixelul este aproape negru => tranparent
    if (r < 5 && g < 5 && b < 5)
    {
        a = 0;
    }

    rgba[i * 4 + 0] = r;
    rgba[i * 4 + 1] = g;
    rgba[i * 4 + 2] = b;
    rgba[i * 4 + 3] = a;
}
```

Crearea texturii OpenGL (RGBA)

```
// Textura RGBA
GLuint idTextura;
glGenTextures(1, &idTextura);
glBindTexture(GL_TEXTURE_2D, idTextura);

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);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);

glTexImage2D(
    GL_TEXTURE_2D, 0,
    GL_RGBA,          // format intern
    latime, inaltime,
    0,
    GL_RGBA,          // format sursa (RGBA)
    GL_UNSIGNED_BYTE,
    rgba
);

free(rgba);

return idTextura;
```

- Se creaza idTextura (index pentru textura)
- Se aplică setări pentru fiecare textură (glBindTexture)
- Atunci când imaginea este micșorată sau mărită - GL_TEXTURE_MIN/MAG_FILTER se aplică GL_LINEAR pentru netezirea imaginii
- Pentru cazul cân coordonatele texturii depășesc 0..1 se aplică GL_CLAMP pentru a folosi pixelii de la marginea texturii (fără repetarea texturii)

Trimiterea imaginii în OpenGL (VRAM în GPU)

```
glTexImage2D(
    GL_TEXTURE_2D, 0,
    GL_RGBA,          // format intern
    latime, inaltime,
    0,
    GL_RGBA,          // format sursa (RGBA)
    GL_UNSIGNED_BYTE,
    rgba
);
```

Încărcarea texturilor în funcția initializare()

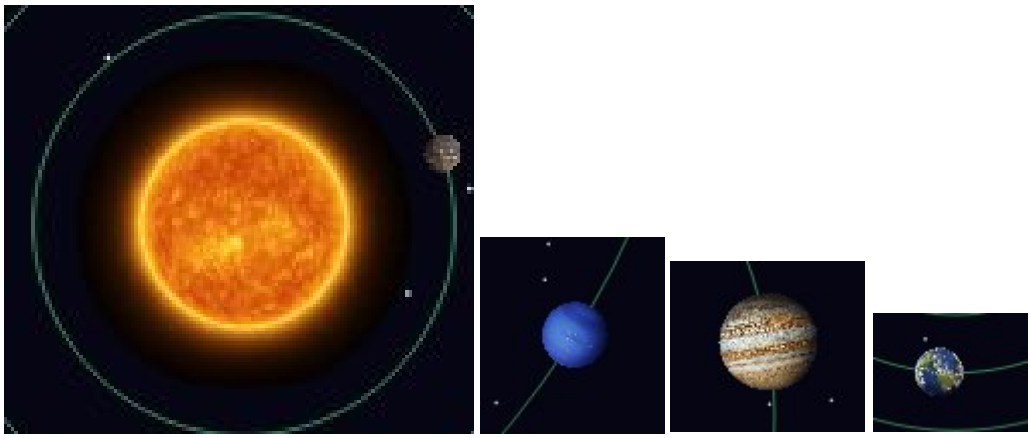
```
// incarcam texturile pentru fiecare planeta
for (int i = 0; i < nrPlanete; i++)
{
    texturiPlanete[i] = incarcaBMP(fisiereTexturiPlanete[i]);
    if (texturiPlanete[i] == 0)
    {
        printf("Atentie: textura pentru %s (%s) nu a fost incarcata!\n",
            planete[i].nume, fisiereTexturiPlanete[i]);
    }
}

// incarcam textura pentru soare
texturaSoare = incarcaBMP(fisierTexturaSoare);
if (texturaSoare == 0)
{
    printf("Atentie: textura pentru Soare (%s) nu a fost incarcata!\n",
        fisierTexturaSoare);
}
}
```

Activăm texturile înainte de desenare în funcția afisare()

```
if (texturiPlanete[i] != 0)
{
    // PLANETA TEXTURATA
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texturiPlanete[i]);
    glColor3f(1.0f, 1.0f, 1.0f);
    deseneazaDiscTexturat(p.razaPlanetei, 40);
    glDisable(GL_TEXTURE_2D);
}
```

- Apelăm glBindTexture pentru a folosi GL_TEXTURE_2D pentru fiecare planetă din lista texturiPlanete încarcate anterior
- Desenarea planetelor texturate cu ajutorul funcției deseneazaDiscTexturat care mapează coordonatele texturii cu cele ale discului și folosește GL_TRIANGLE_FAN pentru a desena planeta

```
void deseneazaDiscTexturat(float raza, int segmente)
{
    glBegin(GL_TRIANGLE_FAN);

    // centru texturii
    glTexCoord2f(0.5f, 0.5f); // centrul imaginii BMP
    glVertex2f(0.0f, 0.0f); // centrul cercului pe ecran

    for (int i = 0; i <= segmente; i++)
    {
        float a = 2.0f * PI * i / segmente;
        float x = cosf(a) * raza;
        float y = sinf(a) * raza;

        // mapare cos/sin [-1,1] -> [0,1]
        float s = (cosf(a) + 1.0f) * 0.5f;
        float t = (sinf(a) + 1.0f) * 0.5f;

        glTexCoord2f(s, t);
        glVertex2f(x, y);
    }

    glEnd();
}
```



5. Contribuții individuale

• Saftoiu Genica Liliana – Soarele, generare stele, orbite(radar), animație, redimensionare, planete

• Georgescu Andrei - Procesare texturi BMP, integrarea loader-ului RGBA, planete

6. Resurse utilizate

• **Documentația oficială OpenGL** - https://www.khronos.org/opengl/
• **Tutoriale GLUT (funcții callback și gestionarea ferestrelor)** - https://learnopengl.com/Getting-started/Textures
• **Articole privind structura fișierelor BMP și transparența color-key**. –

https://www.opengl-tutorial.org/assets/pdf/opengl_tutorial_2017_06_07.pdf
• **Texturi BMP pentru planete** – generate/editate