PROIECT

SCENA 2D

GRAFICA PE CALCULATOR

Conceptul proiectului

Pornind de la una dintre temele propuse, am realizat o scena 2D reprezentand o ambulanta care depaseste alte masini. Ambulanta poate fi controlata folosind tastatura. Scena este gandita sub forma unui joc, in care trebuie sa eviti celelalte masini, viteza crescand odata cu trecerea timpului.

Transformarile incluse

In realizarea proiectului am folosit scalari si translatii. Masinile sunt desenate folosind aceleasi coordonate ale varfurilor, pozitiile lor fiind influentate de translatii. Atat rotile ambulantei, cat si cele ale masinilor adversare sunt desenate pornind de la aceleasi coordonate, prin translatii fiind decisa pozitia fiecareia. Girofarul ambulantei este desenat folosind aceleasi coordonate ca pentru roti, dar este aplicata si o scalare pentru a mari dreptunghiurile folosite. Copacii de pe marginea soselei sunt realizati tot prin translatii. Scena este adusa la patratul standard folosind o scalare.

Originalitate

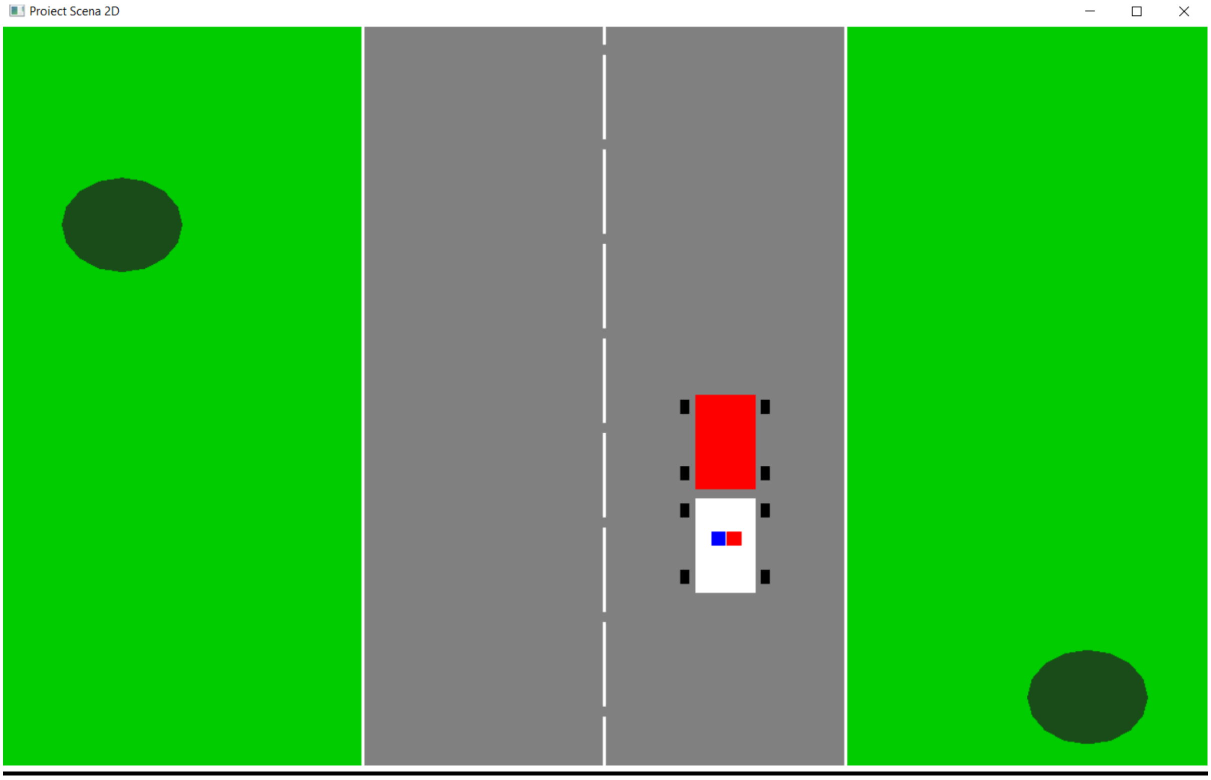
Ce am adus in plus fata de cerinta proiectului este caracterul de joc si controlarea masinii de la tastatura, fapt ce face proiectul mai interactiv si original.

Progres

In urma prezentarii din cadrul laboratorului, am urmat sugestiile primate si am reusit sa implementez urmatoarele imbunatatiri:

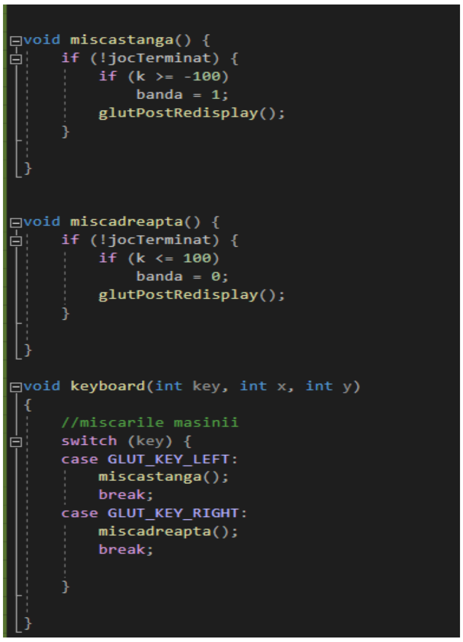
- Ingustarea soselei
- Forma rotunda a copacilor
- Linia intrerupta pe mijlocul soselei
- Schimbarea culorilor girofarului

Rezultat final



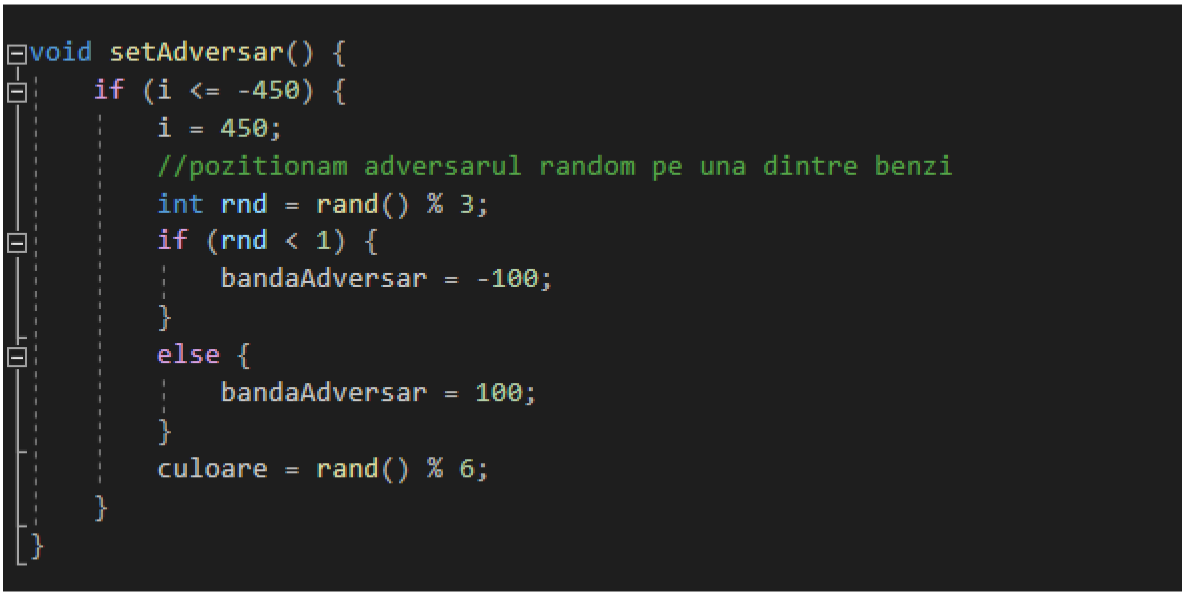
Capturi de ecran

Jocul incepe la un click cu mouse-ul. Viteza creste la fiecare 10 secunde. Apoi masina este controlata prin taste.



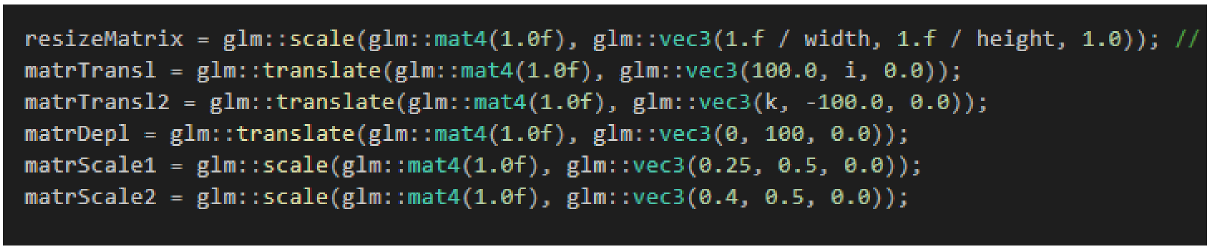
```
void miscastanga() {
    if (!jocTerminat) {
        if (k >= -100)
            banda = 1;
        glutPostRedisplay();
    }

}

void miscadreapta() {
    if (!jocTerminat) {
        if (k <= 100)
            banda = 0;
        glutPostRedisplay();
    }

}

void keyboard(int key, int x, int y)
{
    //miscarile masinii
    switch (key) {
    case GLUT_KEY_LEFT:
        miscastanga();
        break;
    case GLUT_KEY_RIGHT:
        miscadreapta();
        break;

    }

}
```

Atat pozitia, cat si culoarea masinii care trebuie depasita sunt alese cu ajutorul functiei “rand”. Aceeasi masina este folosita in mod repetat pentru crearea “adversarilor”.



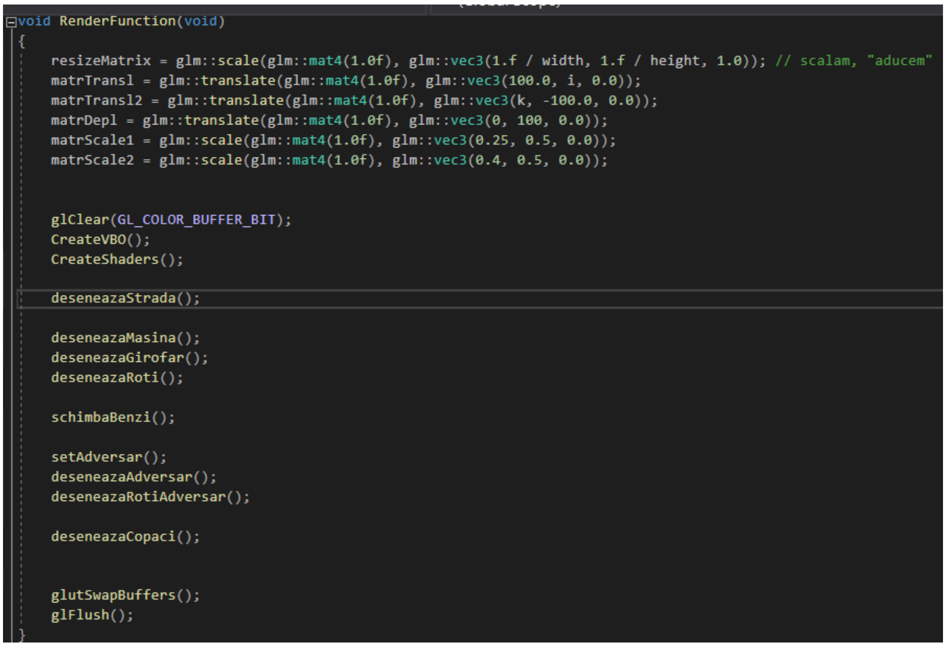
```
void setAdversar() {
    if (i <= -450) {
        i = 450;
        //pozitionam adversarul random pe una dintre benzi
        int rnd = rand() % 3;
        if (rnd < 1) {
            bandaAdversar = -100;
        }
        else {
            bandaAdversar = 100;
        }
        culoare = rand() % 6;
    }
}
```

Matricele de transformari folosite



```
resizeMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(1.f / width, 1.f / height, 1.0)); //
matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(100.0, i, 0.0));
matrTransl2 = glm::translate(glm::mat4(1.0f), glm::vec3(k, -100.0, 0.0));
matrDepl = glm::translate(glm::mat4(1.0f), glm::vec3(0, 100, 0.0));
matrScale1 = glm::scale(glm::mat4(1.0f), glm::vec3(0.25, 0.5, 0.0));
matrScale2 = glm::scale(glm::mat4(1.0f), glm::vec3(0.4, 0.5, 0.0));
```

Fiecare element este desenat in cate o functie pentru lizibilitatea codului



```
void RenderFunction(void)
{
    resizeMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(1.f / width, 1.f / height, 1.0)); // scalam, "aducem"
    matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(100.0, i, 0.0));
    matrTransl2 = glm::translate(glm::mat4(1.0f), glm::vec3(k, -100.0, 0.0));
    matrDepl = glm::translate(glm::mat4(1.0f), glm::vec3(0, 100, 0.0));
    matrScale1 = glm::scale(glm::mat4(1.0f), glm::vec3(0.25, 0.5, 0.0));
    matrScale2 = glm::scale(glm::mat4(1.0f), glm::vec3(0.4, 0.5, 0.0));


    glClear(GL_COLOR_BUFFER_BIT);
    CreateVBO();
    CreateShaders();

    deseneazaStrada();

    deseneazaMasina();
    deseneazaGirofar();
    deseneazaRoti();

    schimbaBenzi();

    setAdversar();
    deseneazaAdversar();
    deseneazaRotiAdversar();

    deseneazaCopaci();


    glutSwapBuffers();
    glFlush();
}
```

Resurse utilizate

Am pornit de la codurile din laboratoare, folosind exemplele de transformari si de utilizare a mouse-ului si a tastaturii.

Cod

```
#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 <chrono>

#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;
using namespace std::chrono;

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

GLuint
VaoId,
VboId,
ColorBufferId,
ProgramId,
myMatrixLocation,
matrScaleLocation,
matrTranslLocation,
matrRotlLocation,
codColLocation;



int codCol;
float PI = 3.141592;
int width = 500, height = 450;
float i = -450.0, k = 100.0, alpha = 0.0, beta = 0.2, j = 150;
int banda = 0, bandaAdversar = 0;
auto start = high_resolution_clock::now();
auto startCul = high_resolution_clock::now();
float viteza = 1;
bool jocTerminat = false;
int x = -450.0;
int culoare = 0;
int n = 16;

glm::mat4 myMatrix, resizeMatrix, matrTransl, matrTransl2, matrTransl3,
matrDepl, matrScale1, matrScale2, matrScale3;
```

```
void startJoc(void)
{
    if (!jocTerminat) {

        i = i + alpha * viteza;
        x = x + alpha * viteza;
        j = j + alpha * viteza;

        //la fiecare 10 secunde, viteza creste
        if (duration_cast<seconds>(high_resolution_clock::now() -
start).count() > 10 && viteza < 4) {
            viteza += 0.5;
            start = high_resolution_clock::now();
        }
        //coliziune
        if (bandaAdversar == k && (i >= -300 && i <= -90)) {
            jocTerminat = true;
        }
        if (x + 500 < -450) {
            x = 450;
        }
        if (j < 50) {
            j = 150;
        }
        glutPostRedisplay();
    }

}

void mouse(int button, int state, int x, int y)
{
    //start joc
    switch (button) {
    case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN)
            alpha = -10.0; glutIdleFunc(startJoc);
        break;
    default:
        break;
    }
}


void miscastanga() {
    if (!jocTerminat) {
        if (k >= -100)
            banda = 1;
        glutPostRedisplay();
    }

}


void miscadreapta() {
    if (!jocTerminat) {
        if (k <= 100)
            banda = 0;
        glutPostRedisplay();
    }

}

void keyboard(int key, int x, int y)
{
    //miscarile masinii
    switch (key) {
    case GLUT_KEY_LEFT:
        miscastanga();
        break;
    case GLUT_KEY_RIGHT:
        miscadreapta();
        break;

    }

}

void setAdversar() {
    if (i <= -450) {
        i = 450;
        //pozitionam adversarul random pe una dintre benzi
        int rnd = rand() % 3;
        if (rnd < 1) {
            bandaAdversar = -100;
        }
        else {
            bandaAdversar = 100;
        }
        culoare = rand() % 6;
    }
}

void schimbaBenzi() {
    if (banda == 0 && (k != 100))
        k = k + 20;
    else if (banda == 1 && (k != -100))
        k = k - 20;
}

void deseneazaMasina() {

    myMatrix = resizeMatrix * matrTransl2 * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    codCol = 1;
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);
    // Apelare DrawArrays
    glDrawArrays(GL_POLYGON, 4, 4);
}


void deseneazaRoti() {
    codCol = 4;
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(-30+k, -70.0,
0.0));
```

```
    myMatrix = resizeMatrix * matrTransl3 * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(37 + k, -70.0,
0.0));
    myMatrix = resizeMatrix * matrTransl3 * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(37 + k, -140.0,
0.0));
    myMatrix = resizeMatrix * matrTransl3 * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(-30 + k, -
140.0, 0.0));
    myMatrix = resizeMatrix * matrTransl3 * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);
}

void deseneazaGirofar() {
    //girofarul isi schimba culorile la fiecare jumatate de secunda

    //partea stanga a girofarului
    if (duration_cast<seconds>(high_resolution_clock::now() -
startCul).count() >= 0.5 &&
        duration_cast<seconds>(high_resolution_clock::now() -
startCul).count() <= 1) {
        codCol = 2;
    }
    else {
        codCol = 5;
    }
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);

    myMatrix = resizeMatrix * matrTransl2 * matrScale2;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);

    //partea dreapta a girofarului
    if (duration_cast<seconds>(high_resolution_clock::now() -
startCul).count() >= 0.5 &&
        duration_cast<seconds>(high_resolution_clock::now() -
startCul).count() <= 1) {
        codCol = 5;
    }
    else {
        codCol = 2;
    }
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);
    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(k + 13, -100,
0.0));
    myMatrix = resizeMatrix * matrTransl3 * matrScale2;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);


    if (duration_cast<seconds>(high_resolution_clock::now() -
startCul).count() > 1) {
        startCul = high_resolution_clock::now();
    }
}

void deseneazaAdversar() {
    codCol = culoare;
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);

    matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(bandaAdversar,
i, 0.0));

    myMatrix = resizeMatrix * matrTransl * matrDepl * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");

    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 4, 4);
}

void deseneazaRotiAdversar() {
    codCol = 4;
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(-30 +
bandaAdversar, i+30, 0.0));
    myMatrix = resizeMatrix * matrTransl3  * matrDepl * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(37 +
bandaAdversar, i+30, 0.0));
    myMatrix = resizeMatrix * matrTransl3  * matrDepl * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(37 +
bandaAdversar, i-40, 0.0));
    myMatrix = resizeMatrix * matrTransl3 *  matrDepl * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(-30 +
bandaAdversar, i-40, 0.0));
    myMatrix = resizeMatrix * matrTransl3 * matrDepl * matrScale1;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 8, 4);
}




void deseneazaCopaci() {

    codCol = 6;
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);

    matrScale3 = glm::scale(glm::mat4(1.0f), glm::vec3(2.5, 2.5, 0.0));
    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(400, x, 0.0));
    myMatrix = resizeMatrix * matrTransl3 * matrDepl * matrScale3;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 26, n);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(-400, x + 500,
0.0));
    myMatrix = resizeMatrix * matrTransl3 * matrDepl * matrScale3;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 26, n);

}

void deseneazaStrada() {
    myMatrix = resizeMatrix;

    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawArrays(GL_POLYGON, 0, 4);
    glLineWidth(3);
    codCol = 3;
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);
    glDrawArrays(GL_LINES, 12, 2);
    glDrawArrays(GL_LINES, 14, 2);
    glDrawArrays(GL_LINES, 16, 2);

    matrTransl3 = glm::translate(glm::mat4(1.0f), glm::vec3(0, j, 0.0));
    myMatrix = resizeMatrix * matrTransl3;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    codCol = 7;
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
    glUniform1i(codColLocation, codCol);
    glPointSize(10);
    glDrawArrays(GL_POINTS, 18, 8);
    glPointSize(1);
}



void CreateVBO(void)
{
    // varfurile
    GLfloat Vertices[300] = {
```

```
    // varfuri pentru strada
     -200.0f, 450.0f, 0.0f, 1.0f,
     -200.0f, -450.0f, 0.0f, 1.0f,
     200.0f,  -450.0f, 0.0f, 1.0f,
     200.0f,  450.0f, 0.0f, 1.0f,


     // varfuri pentru masina
    -100.0f,  -100.0f, 0.0f, 1.0f,
    100.0f, -100.0f, 0.0f, 1.0f,
    100.0f,  100.0f, 0.0f, 1.0f,
    -100.0f,  100.0f, 0.0f, 1.0f,

    //varfuri pentru roti si girofar
    0.0f, 30.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 0.0f, 1.0f,
    -30.0f, 0.0f, 0.0f, 1.0f,
    -30.0f, 30.0f, 0.0f, 1.0f,


    // varfuri pentru delimitari
    0.0f, 450.0f, 0.0f, 1.0f,
    0.0f, -450.0f, 0.0f, 1.0f,
    -200.0f, 450.0f, 0.0f, 1.0f,
    -200.0f, -450.0f, 0.0f, 1.0f,
     200.0f, 450.0f, 0.0f, 1.0f,
    200.0f, -450.0f, 0.0f, 1.0f,

    0.0f, 350.0f, 0.0f, 1.0f,
    0.0f, 250.0f, 0.0f, 1.0f,
    0.0f, 150.0f, 0.0f, 1.0f,
    0.0f, 50.0f, 0.0f, 1.0f,
    0.0f, -50.0f, 0.0f, 1.0f,
    0.0f, -150.0f, 0.0f, 1.0f,
    0.0f, -250.0f, 0.0f, 1.0f,
    0.0f, -350.0f, 0.0f, 1.0f,

};
int vertices_length = 104;
for (int i = 0; i < n; i++) {
    Vertices[vertices_length] =20 * cos(2 * PI * i / n);
    Vertices[vertices_length + 1] = 20 * sin(2 * PI * i / n);
    Vertices[vertices_length + 2] = 0.0f;
    Vertices[vertices_length + 3] = 1.0f;
    vertices_length += 4;
}



GLfloat Colors[] = {
  0.5f, 0.5f, 0.5f, 1.0f,
  0.5f, 0.5f, 0.5f, 1.0f,
  0.5f, 0.5f, 0.5f, 1.0f,
  0.5f, 0.5f, 0.5f, 1.0f,
};


// se creeaza un buffer nou
glGenBuffers(1, &VboId);
// este setat ca buffer curent
glBindBuffer(GL_ARRAY_BUFFER, VboId);
```

```
    // punctele sunt "copiate" in bufferul curent
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, 
GL_STATIC_DRAW);

    // se creeaza / se leaga un VAO (Vertex Array Object) - util cand se 
utilizeaza mai multe VBO
    glGenVertexArrays(1, &VaoId);
    glBindVertexArray(VaoId);
    // se activeaza lucrul cu atribute; atributul 0 = pozitie
    glEnableVertexAttribArray(0);
    // 
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);

    // un nou buffer, pentru culoare
    glGenBuffers(1, &ColorBufferId);
    glBindBuffer(GL_ARRAY_BUFFER, ColorBufferId);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Colors), Colors, GL_STATIC_DRAW);
    // atributul 1 =  culoare
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, 0);


}
void DestroyVBO(void)
{


    glDisableVertexAttribArray(1);
    glDisableVertexAttribArray(0);

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDeleteBuffers(1, &ColorBufferId);
    glDeleteBuffers(1, &VboId);

    glBindVertexArray(0);
    glDeleteVertexArrays(1, &VaoId);


}

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

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

void Initialize(void)
{
    glClearColor(0.0f, 0.8f, 0.0f, 0.0f); // culoarea de fond a ecranului
}
void RenderFunction(void)
{
    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]
    matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(100.0, i, 0.0));
    matrTransl2 = glm::translate(glm::mat4(1.0f), glm::vec3(k, -100.0,
0.0));
    matrDepl = glm::translate(glm::mat4(1.0f), glm::vec3(0, 100, 0.0));
    matrScale1 = glm::scale(glm::mat4(1.0f), glm::vec3(0.25, 0.5, 0.0));
    matrScale2 = glm::scale(glm::mat4(1.0f), glm::vec3(0.4, 0.5, 0.0));


    glClear(GL_COLOR_BUFFER_BIT);
    CreateVBO();
    CreateShaders();

    deseneazaStrada();

    deseneazaMasina();
    deseneazaGirofar();
    deseneazaRoti();

    schimbaBenzi();

    setAdversar();
    deseneazaAdversar();
    deseneazaRotiAdversar();

    deseneazaCopaci();


    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(1200, 900);
    glutCreateWindow("Proiect Scena 2D");
    glewInit();
    Initialize();
    glutDisplayFunc(RenderFunction);
    glutMouseFunc(mouse);
    glutSpecialFunc(keyboard);
    glutCloseFunc(Cleanup);

    glutMainLoop();


}
```

Shadere

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 (1.0, 1.0, 1.0, 0.0);
       if ( codCol==2 )
              out_Color=vec4 (1.0, 0.0, 0.0, 0.0);
       if(codCol == 3)
              out_Color = vec4 (1.0, 1.0, 1.0, 0.0);
       if(codCol == 4)
              out_Color = vec4 (0.0, 0.0, 0.0, 0.0);
       if(codCol == 5)
              out_Color = vec4 (0.0, 0.0, 1.0, 0.0);
       if(codCol == 6)
              out_Color = vec4 (0.1, 0.3, 0.1, 0.0);
       if(codCol == 7)
              out_Color = vec4 (0.5, 0.5, 0.5, 0.0);


  }
```

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;
   }
```