# Grafică Pe Calculator

## Proiect 2D: Depășire

Echipa:
Barbu Cosmina Anamaria
Oprea Anca Ioana
Spataru Mara Andreea

## 1. Conceptul proiectului

Proiectul prezintă perspectiva privind de sus a unei mașini care depășește un alt autovehicul care se deplasează uniform pe un drum cu trafic pe ambele sensuri. Pentru reprezentarea fundalului, a mașinilor și a palmierilor am folosit texturare, iar pentru a reprezenta marcajele drumului am adăugat elemente liniare.

## 2. Transformări incluse

Pentru deplasarea mașinilor am folosit transformarea de translație, iar pentru depășire am folosit transformarea de rotație.

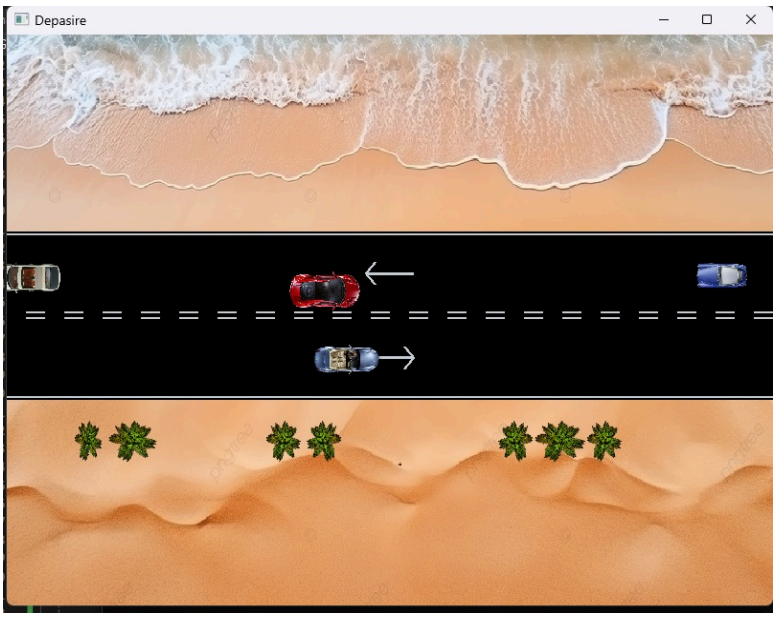
## 3. Elemente de originalitate

Elemente de originalitate ale proiectului sunt reprezentate de dinamicitatea traficului pe un segment de drum, adaugarea unei texturi pentru reprezentarea mașinilor, a palmierilor și a fundalului, drumul fiind pe o plajă.

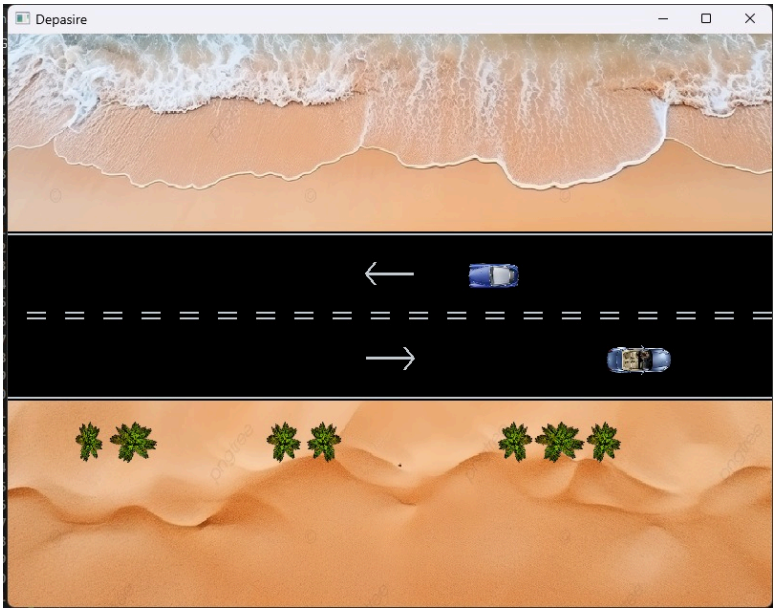
De asemenea, am adăugat și marcaje discontinue ale drumului folosind funcția glLineStipple.

Dinamicitatea proiectului este oferită de cadrul aglomerat în care mașina roșie reușește să efectueze cu succes o depășire.

## 4. Rezultate


Depasire


Depasire

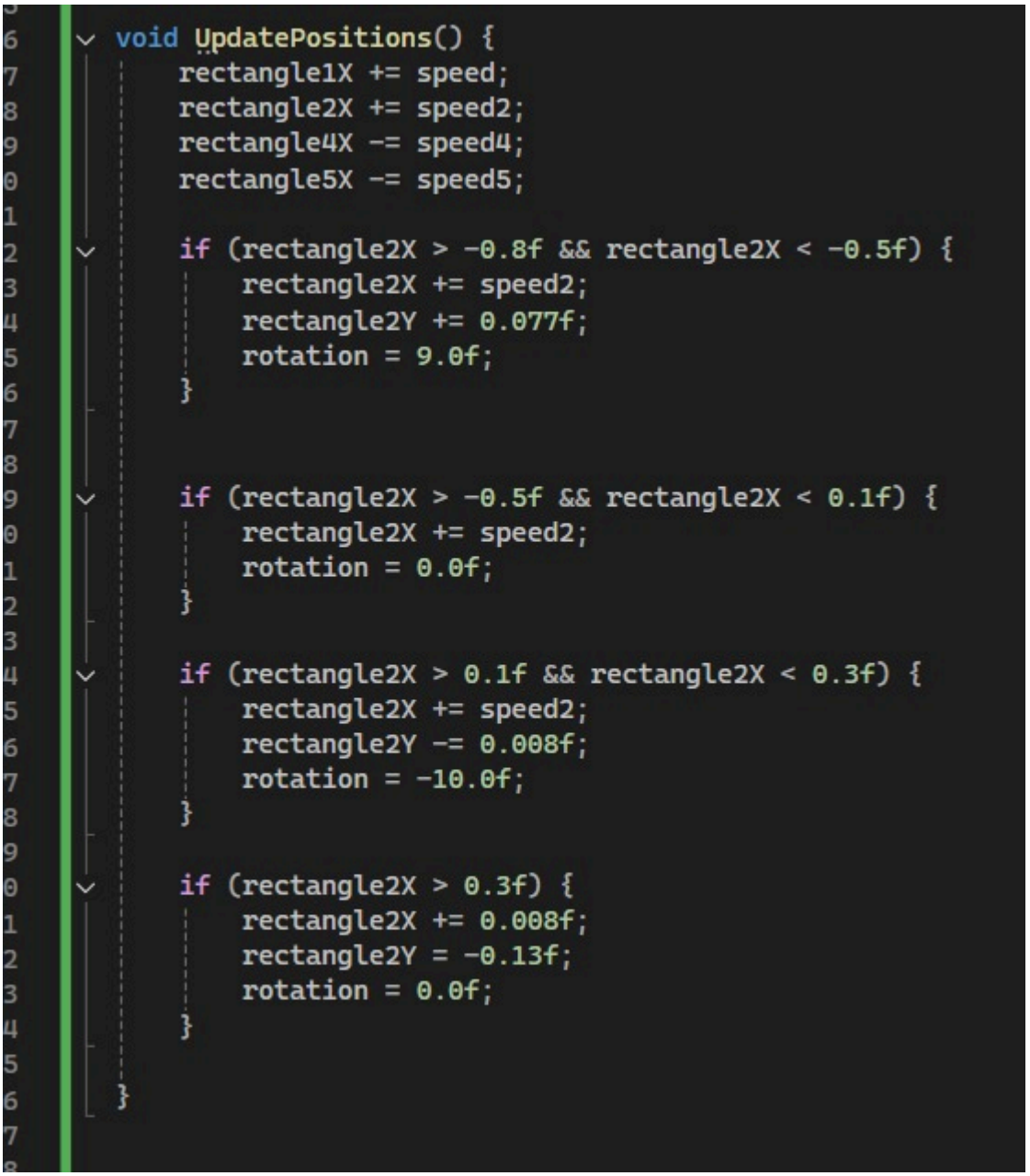

Depasire

## 5. Contribuții individuale ale membrilor echipei

Barbu Cosmina Anamaria - texturare și documentație
Oprea Anca Ioana - fundal, elemente de design și documentație
Spataru Mara Andreea - translație și rotație

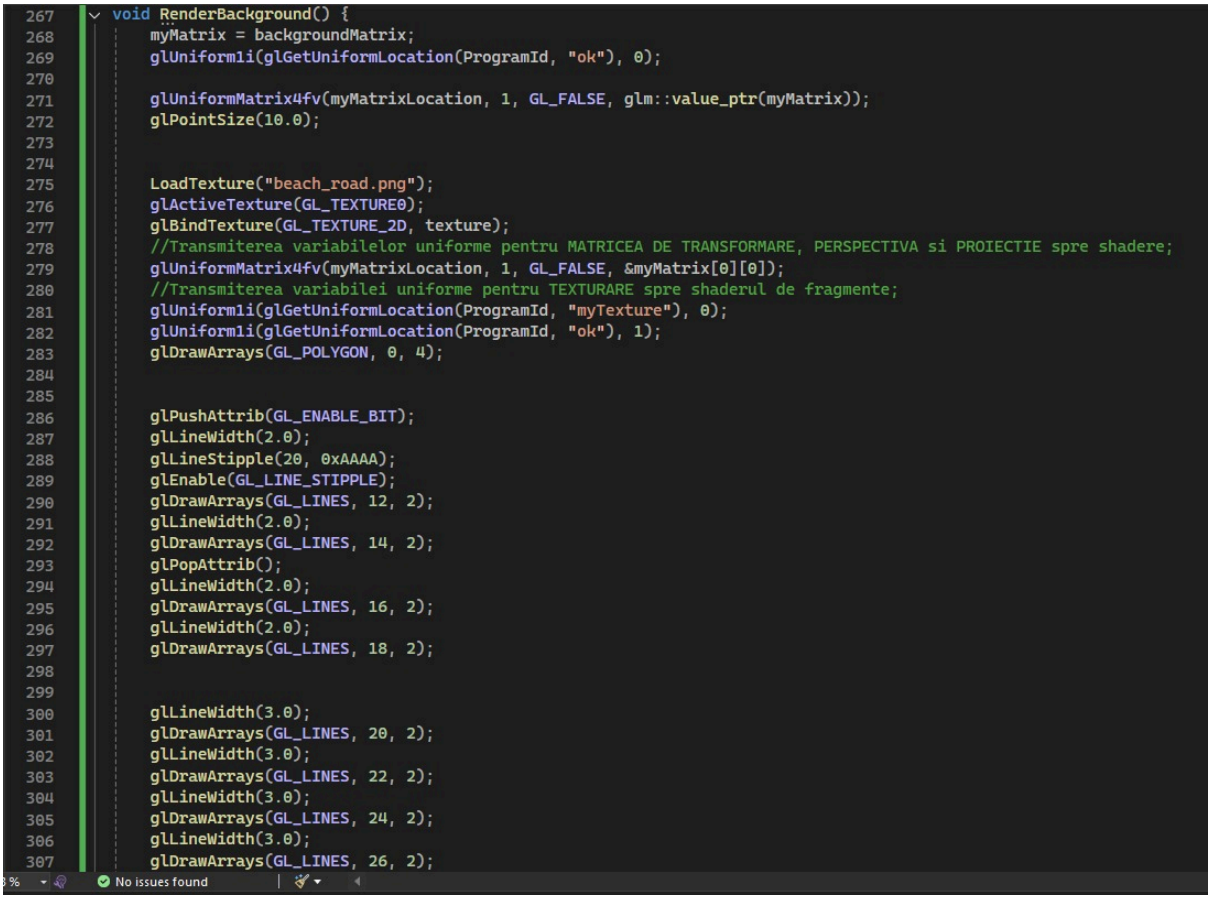
## 6. Cod relevant

a. funcția UpdatePositions actualizează poziția mașinilor în funcție de viteza acestora și realizeaz``ă depășirea



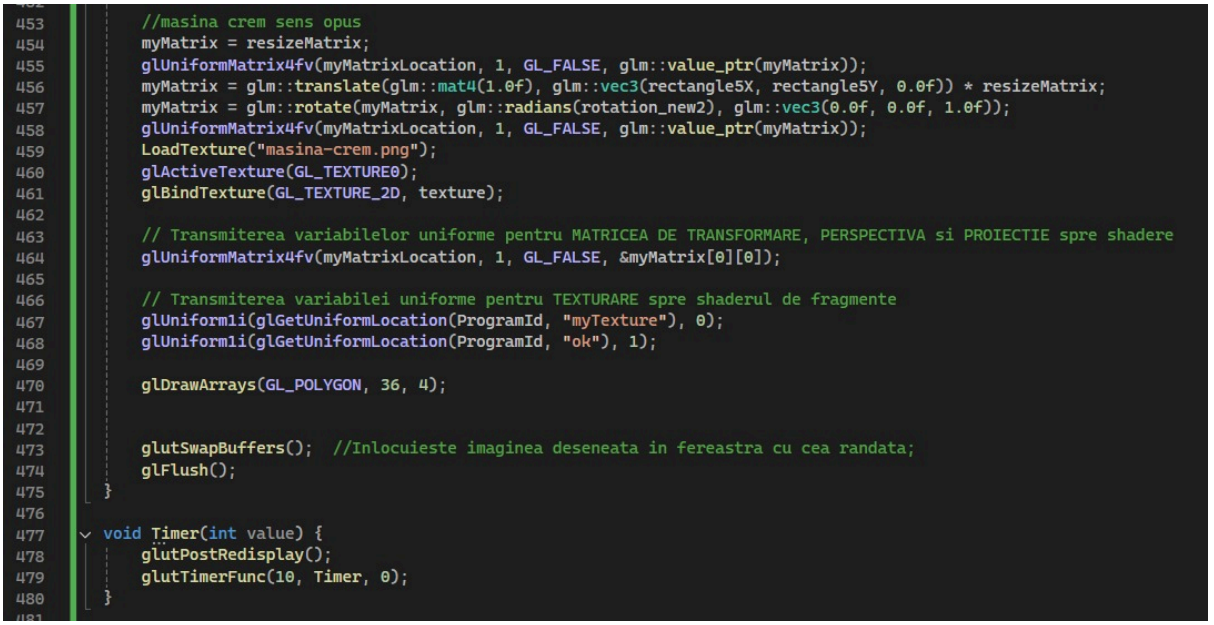
```
void UpdatePositions() {
    rectangle1X += speed;
    rectangle2X += speed2;
    rectangle4X -= speed4;
    rectangle5X -= speed5;

    if (rectangle2X > -0.8f && rectangle2X < -0.5f) {
        rectangle2X += speed2;
        rectangle2Y += 0.077f;
        rotation = 9.0f;
    }


    if (rectangle2X > -0.5f && rectangle2X < 0.1f) {
        rectangle2X += speed2;
        rotation = 0.0f;
    }

    if (rectangle2X > 0.1f && rectangle2X < 0.3f) {
        rectangle2X += speed2;
        rectangle2Y -= 0.008f;
        rotation = -10.0f;
    }

    if (rectangle2X > 0.3f) {
        rectangle2X += 0.008f;
        rectangle2Y = -0.13f;
        rotation = 0.0f;
    }

}
```

b. funcția RenderBackground este folosită pentru a afișa elementele care fac parte din fundal, inclusiv texturarea acestuia



```
void RenderBackground() {
    myMatrix = backgroundMatrix;
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 0);

    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, glm::value_ptr(myMatrix));
    glPointSize(10.0);


    LoadTexture("beach_road.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 0, 4);


    glPushAttrib(GL_ENABLE_BIT);
    glLineWidth(2.0);
    glLineStipple(20, 0xAAAA);
    glEnable(GL_LINE_STIPPLE);
    glDrawArrays(GL_LINES, 12, 2);
    glLineWidth(2.0);
    glDrawArrays(GL_LINES, 14, 2);
    glPopAttrib();
    glLineWidth(2.0);
    glDrawArrays(GL_LINES, 16, 2);
    glLineWidth(2.0);
    glDrawArrays(GL_LINES, 18, 2);


    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 20, 2);
    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 22, 2);
    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 24, 2);
    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 26, 2);
```

c. translația și rotația - am realizat translația în mod asemanator pentru toate mașinile, iar pentru a realiza depășirea, unghiul de rotație a fost manipulat în funcția UpdatePositions; funcția Timer e folosită pentru a fluidiza deplasarea mașinilor



```
    //masina crem sens opus
    myMatrix = resizeMatrix;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, glm::value_ptr(myMatrix));
    myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(rectangle5X, rectangle5Y, 0.0f)) * resizeMatrix;
    myMatrix = glm::rotate(myMatrix, glm::radians(rotation_new2), glm::vec3(0.0f, 0.0f, 1.0f));
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, glm::value_ptr(myMatrix));
    LoadTexture("masina-crem.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);

    // Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

    // Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de fragmente
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);

    glDrawArrays(GL_POLYGON, 36, 4);


    glutSwapBuffers();  //Inlocuieste imaginea deseneata in fereastra cu cea randata;
    glFlush();
}

void Timer(int value) {
    glutPostRedisplay();
    glutTimerFunc(10, Timer, 0);
}
```

MAIN:

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <stdio.h>
#include "loadShaders.h"
#include "glm/glm.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtx/transform.hpp"
#include "glm/gtc/type_ptr.hpp"
#include "SOIL.h"
#include <random>
#include <iostream>
#include <ctime>
using namespace std;


GLuint VaoId, VboId, ProgramId, myMatrixLocation, texture;
glm::mat4 myMatrix, resizeMatrix, backgroundMatrix;

float xMin = -160.0f, xMax = 160.0f, yMin = -100.0f, yMax = 100.0f;

float rectangle1X = -1.0f;  // poz pe axa x dreptunghi1
float rectangle2X = -1.6f;   //  poz pe axa x dreptunghi2
float rectangle2Y = -0.13f;

float rectangle4X = 1.5f;    //partea dr a ecranului
float rectangle4Y = 0.15f;  //inaltime diferita de 1 si 2
//float rectangle5X = 0.0f;    //partea stg a ecranului


float generatePosition(void) {

    static bool seeded = false;
    if (!seeded) {
        srand(static_cast<unsigned int>(time(0)));  //crt time
        seeded = true;
    }

    int randomPos = rand() % 5;  //intre 0 si 4
    float position;

    switch (randomPos) {
    case 0:
        position = -1.0f;
        break;
```

```
    case 1:
        position = -0.5f;
        break;
    case 2:
        position = 0.0f;
        break;
    case 3:
        position = 0.5f;
        break;
    case 4:
        position = 1.0f;
        break;
    default:
        position = -0.5f;
        break;
    }
    cout << "Position: " << position << endl;
    return position;
}
float rectangle5X = generatePosition();



float rectangle5Y = 0.15f; //masina crem

float speed = 0.045f;  // viteza pt prima masina
float speed2 = 0.056f;  // viteza pt a doua masina
float speed4 = 0.035f; //viteza pt masina de pe sens opus
float speed5 = 0.15f;  //masina crem
float angle = 0;          //        Unghiul de rotire al patratului;
float tx = 0; float ty = 0; float auxtx; float auxangle;
float rotation = 0.0f;
float rotation_new = 0.0f;
float rotation_new2 = 0.0f;


void LoadTexture(const char* photoPath)
{
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    //     Desfasurarea imaginii pe orizonatala/verticala in functie de
parametrii de texturare;
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_REPEAT);
```

```
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);

    int width, height;
    unsigned char* image = SOIL_load_image(photoPath, &width, &height,
0, SOIL_LOAD_RGBA);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0,
GL_RGBA, GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);

    SOIL_free_image_data(image);
    glBindTexture(GL_TEXTURE_2D, 0);
}

void UpdatePositions() {
    rectangle1X += speed;
    rectangle2X += speed2;
    rectangle4X -= speed4;
    rectangle5X -= speed5;

    if (rectangle2X > -0.8f && rectangle2X < -0.5f) {
        rectangle2X += speed2;
        rectangle2Y += 0.077f;
        rotation = 9.0f;
    }


    if (rectangle2X > -0.5f && rectangle2X < 0.1f) {
        rectangle2X += speed2;
        rotation = 0.0f;
    }

    if (rectangle2X > 0.1f && rectangle2X < 0.3f) {
        rectangle2X += speed2;
        rectangle2Y -= 0.008f;
        rotation = -10.0f;
    }

    if (rectangle2X > 0.3f) {
        rectangle2X += 0.008f;
        rectangle2Y = -0.13f;
        rotation = 0.0f;
    }

}
```

```
void CreateShaders() {
   ProgramId = LoadShaders("03_05_Shader.vert", "03_05_Shader.frag");
   glUseProgram(ProgramId);
}

void CreateVBO(void)
{
   // Coordonatele varfurilor;
   static const GLfloat Vertices[] =
   {
      //  Cele 4 varfuri din colturi;
      xMin, yMin, 0.0f, 1.0f,     0.0f, 0.8f, 0.0f,          0.0f, 0.0f,
      xMax, yMin, 0.0f, 1.0f,       0.0f, 0.8f, 0.0f,       1.0f, 0.0f,
      xMax, yMax, 0.0f, 1.0f,       0.0f, 0.4f, 0.0f,       1.0f, 1.0f,
      xMin, yMax, 0.0f, 1.0f,       0.0f, 0.4f, 0.0f,       0.0f, 1.0f,

      //masina albastra
      -10.0f, -7.0f,  0.0f,  1.0f,  0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
      20.0f, -7.0f,  0.0f,  1.0f,             1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
      20.0f,  7.0f,  0.0f,  1.0f,             1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
      -10.0f, 7.0f,  0.0f,  1.0f,             1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

      //masina rosie
      -10.0f, -7.0f,  0.0f,  1.0f,  0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
      20.0f, -7.0f,  0.0f,  1.0f,             1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
      20.0f,  7.0f,  0.0f,  1.0f,             1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
      -10.0f, 7.0f,  0.0f,  1.0f,             1.0f, 0.0f, 0.0f, 0.0f, 1.0f,


      // linia din mijloc-stanga a strazii
      xMin, 2.5f, 0.0f, 1.0f,             1.0f, 1.0f, 1.0f,    0.0f, 0.0f,
      xMax, 2.5f, 0.0f, 1.0f,             1.0f, 1.0f, 1.0f,    0.0f, 0.0f,

      // linia din mijloc-dreapta a strazii
      xMin, 0.5f, 0.0f, 1.0f,          1.0f, 1.0f, 1.0f,       0.0f, 0.0f,
      xMax,  0.5f, 0.0f, 1.0f,         1.0f, 1.0f, 1.0f,       0.0f, 0.0f,

      // linia din dreapta a strazii
      xMin, -27.0f, 0.0f, 1.0f,        1.0f, 1.0f, 1.0f,       0.0f, 0.0f,
      xMax,  -27.0f, 0.0f, 1.0f,       1.0f, 1.0f, 1.0f,       0.0f, 0.0f,

      // linia din stanga a strazii
      xMin, 30.0f, 0.0f, 1.0f,         1.0f, 1.0f, 1.0f,       0.0f, 0.0f,
      xMax,  30.0f, 0.0f, 1.0f,        1.0f, 1.0f, 1.0f,       0.0f, 0.0f,
```

```
// sageti
-10.0f, 16.25f, 0.0f, 1.0f,     1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
10.0f,  16.25f, 0.0f, 1.0f,     1.0f, 1.0f, 1.0f,        0.0f, 0.0f,

 -10.0f, -13.25f, 0.0f, 1.0f,          1.0f, 1.0f, 1.0f, 0.0f, 0.0f,
 10.0f,  -13.25f, 0.0f, 1.0f,    1.0f, 1.0f, 1.0f,       0.0f, 0.0f,

 10.0f, -13.25f, 0.0f, 1.0f,             1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
 6.0f, -9.25f, 0.0f, 1.0f,       1.0f, 1.0f, 1.0f,        0.0f, 0.0f,

 10.0f, -13.25f, 0.0f, 1.0f,             1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
 6.0f, -17.25f, 0.0f, 1.0f,      1.0f, 1.0f, 1.0f,        0.0f, 0.0f,

 -10.0f, 16.25f, 0.0f, 1.0f,             1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
 -6.0f, 12.25f, 0.0f, 1.0f,      1.0f, 1.0f, 1.0f,        0.0f, 0.0f,

 -10.0f, 16.25f, 0.0f, 1.0f,             1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
 -6.0f, 20.25f, 0.0f, 1.0f,      1.0f, 1.0f, 1.0f,        0.0f, 0.0f,



//masina de pe sens opus - albastra
-10.0f, -7.0f,  0.0f,  1.0f,  0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
20.0f, -7.0f,  0.0f,  1.0f,             1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
20.0f,  7.0f,  0.0f,  1.0f,             1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
-10.0f, 7.0f,  0.0f,  1.0f,             1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

//masina de pe sens opus - crem
-15.0f, -12.0f,  0.0f,  1.0f,           0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
25.0f, -12.0f,  0.0f,  1.0f,            1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
25.0f,  12.0f,  0.0f,  1.0f,            1.0f, 0.0f, 0.0f, 1.0f, 1.0f,
-15.0f, 12.0f,  0.0f,  1.0f,            1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

//dimensiunile paralelogramelor pentru copaci
60.0f , -35.0f,  0.0f,  1.0f,             1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
45.0f, -35.0f,  0.0f,  1.0f,            1.0f, 1.0f, 1.0f, 1.0f, 0.0f,
45.0f, -50.0f,  0.0f,  1.0f,            1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
60.0f, -50.0f,  0.0f,  1.0f,            1.0f, 1.0f, 1.0f, 0.0f, 1.0f,

60.0f , -35.0f,  0.0f,  1.0f,             1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
82.0f, -35.0f,  0.0f,  1.0f,            1.0f, 1.0f, 1.0f, 1.0f, 0.0f,
82.0f,  -50.0f,  0.0f,  1.0f,             1.0f, 1.0f, 1.0f,        1.0f, 1.0f,
60.0f, -50.0f,  0.0f,  1.0f,            1.0f, 1.0f, 1.0f, 0.0f, 1.0f,

82.0f, -35.0f,  0.0f,  1.0f,    1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
97.0f, -35.0f,  0.0f,  1.0f,            1.0f, 1.0f, 1.0f, 1.0f, 0.0f,
```

```
        97.0f, -50.0f, 0.0f, 1.0f,          1.0f, 1.0f, 1.0f,   1.0f, 1.0f,
        82.0f, -50.0f, 0.0f, 1.0f,          1.0f, 1.0f, 1.0f,   0.0f, 1.0f,

        -20.0f, -35.0f,  0.0f,  1.0f,              1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
        -35.0f, -35.0f,  0.0f,  1.0f,                    1.0f, 1.0f, 1.0f, 1.0f, 0.0f,
        -35.0f, -50.0f, 0.0f, 1.0f,          1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
        -20.0f, -50.0f, 0.0f, 1.0f,          1.0f, 1.0f, 1.0f, 0.0f, 1.0f,

        -52.0f, -35.0f,  0.0f,  1.0f,              1.0f, 1.0f, 1.0f,        0.0f, 0.0f,
        -37.0f, -35.0f,  0.0f,  1.0f,                    1.0f, 1.0f, 1.0f, 1.0f, 0.0f,
        -37.0f, -50.0f,  0.0f,  1.0f,                    1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
        -52.0f, -50.0f,  0.0f,  1.0f,              1.0f, 1.0f, 1.0f,        0.0f, 1.0f,


        -118.0f, -35.0f, 0.0f, 1.0f,        1.0f, 1.0f, 1.0f,   0.0f, 0.0f,
        -97.0f, -35.0f, 0.0f, 1.0f,         1.0f, 1.0f, 1.0f,   1.0f, 0.0f,
        -97.0f, -50.0f, 0.0f, 1.0f,         1.0f, 1.0f, 1.0f,   1.0f, 1.0f,
        -118.0f, -50.0f, 0.0f, 1.0f,        1.0f, 1.0f, 1.0f,   0.0f, 1.0f,

        -132.0f, -35.0f, 0.0f, 1.0f,        1.0f, 1.0f, 1.0f,   0.0f, 0.0f,
        -120.0f, -35.0f, 0.0f, 1.0f,        1.0f, 1.0f, 1.0f,   1.0f, 0.0f,
        -120.0f, -50.0f, 0.0f, 1.0f,        1.0f, 1.0f, 1.0f,   1.0f, 1.0f,
        -132.0f, -50.0f, 0.0f, 1.0f,        1.0f, 1.0f, 1.0f,   0.0f, 1.0f,
    };



    // Transmiterea datelor prin buffere;

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

    glGenBuffers(1, &VboId);                                                    //
Generarea bufferului si indexarea acestuia catre variabila VboId;
    glBindBuffer(GL_ARRAY_BUFFER, VboId);                                       //
Setarea tipului de buffer - atributele varfurilor;
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices,
GL_STATIC_DRAW);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 9 * sizeof(GLfloat),
(GLvoid*)0);
    //  Se asociaza atributul (1 =  culoare) pentru shader;
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 9 * sizeof(GLfloat),
(GLvoid*)(4 * sizeof(GLfloat)));
    //  Se asociaza atributul (2 =  texturare) pentru shader;
```

```
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 9 * sizeof(GLfloat),
(GLvoid*)(7 * sizeof(GLfloat)));


}



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

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

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

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

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

void Initialize() {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);          //  Culoarea de fond a
ecranului;
    CreateVBO();                                   //  Trecerea datelor de randare
spre bufferul folosit de shadere;
    CreateShaders();                               //  Initilizarea shaderelor;
    //      Instantierea variabilelor uniforme pentru a "comunica" cu shaderele;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 0);

    //      Dreptunghiul "decupat";
    resizeMatrix = glm::ortho(xMin, xMax, yMin, yMax);
    backgroundMatrix = glm::ortho(xMin, xMax, yMin, yMax); // Matrice
pentru fundal și linia albă

    glEnable(GL_BLEND);
```

```
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
}

void RenderBackground() {
    myMatrix = backgroundMatrix;
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 0);

    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE,
glm::value_ptr(myMatrix));
    glPointSize(10.0);


    LoadTexture("beach_road.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de
fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 0, 4);


    glPushAttrib(GL_ENABLE_BIT);
    glLineWidth(2.0);
    glLineStipple(20, 0xAAAA);
    glEnable(GL_LINE_STIPPLE);
    glDrawArrays(GL_LINES, 12, 2);
    glLineWidth(2.0);
    glDrawArrays(GL_LINES, 14, 2);
    glPopAttrib();
    glLineWidth(2.0);
    glDrawArrays(GL_LINES, 16, 2);
    glLineWidth(2.0);
    glDrawArrays(GL_LINES, 18, 2);


    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 20, 2);
    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 22, 2);
    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 24, 2);
    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 26, 2);
```

```
    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 28, 2);

    glLineWidth(3.0);
    glDrawArrays(GL_LINES, 30, 2);


    glDisable(GL_POINT_SMOOTH);

}

void RenderFunction() {

    glClear(GL_COLOR_BUFFER_BIT);                    //se curata
ecranul OpenGL pentru a fi desenat noul continut;


    RenderBackground();  //Deseneaza fundalul

    UpdatePositions();


    //copaci
    LoadTexture("palm_tree.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //      Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //      Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul
de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 40, 4);

    LoadTexture("palm_tree.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //      Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //      Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul
de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 44, 4);
```

```
    LoadTexture("palm_tree.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //      Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //      Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul
de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 48, 4);

    LoadTexture("palm_tree.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //      Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //      Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul
de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 52, 4);

    LoadTexture("palm_tree.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //      Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //      Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul
de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 56, 4);

    LoadTexture("palm_tree.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //      Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //      Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul
de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 60, 4);
```

```
    LoadTexture("palm_tree.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //      Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //      Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul
de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 64, 4);

    myMatrix = resizeMatrix;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE,
glm::value_ptr(myMatrix));
    myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(rectangle1X, -0.13f,
0.0f)) * resizeMatrix;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE,
glm::value_ptr(myMatrix));
    LoadTexture("decapotabila.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    //Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de
fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 4, 4);



    myMatrix = resizeMatrix;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE,
glm::value_ptr(myMatrix));
    myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(rectangle2X,
rectangle2Y, 0.0f)) * resizeMatrix;
    myMatrix = glm::rotate(myMatrix, glm::radians(rotation), glm::vec3(0.0f,
0.0f, 1.0f));
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE,
glm::value_ptr(myMatrix));
    LoadTexture("red_car.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
```

```
    //      Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    //      Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul
de fragmente;
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);
    glDrawArrays(GL_POLYGON, 8, 4);



    //masina albastra sens opus
    myMatrix = resizeMatrix;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE,
glm::value_ptr(myMatrix));
    myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(rectangle4X,
rectangle4Y, 0.0f)) * resizeMatrix;
    myMatrix = glm::rotate(myMatrix, glm::radians(rotation_new),
glm::vec3(0.0f, 0.0f, 1.0f));
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE,
glm::value_ptr(myMatrix));
    LoadTexture("blue_car.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);

    // Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

    // Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de
fragmente
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);

    glDrawArrays(GL_POLYGON, 32, 4);



    //masina crem sens opus
    myMatrix = resizeMatrix;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE,
glm::value_ptr(myMatrix));
    myMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(rectangle5X,
rectangle5Y, 0.0f)) * resizeMatrix;
    myMatrix = glm::rotate(myMatrix, glm::radians(rotation_new2),
glm::vec3(0.0f, 0.0f, 1.0f));
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE,
glm::value_ptr(myMatrix));
```

```
    LoadTexture("masina-crem.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);

    // Transmiterea variabilelor uniforme pentru MATRICEA DE
TRANSFORMARE, PERSPECTIVA si PROIECTIE spre shadere
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

    // Transmiterea variabilei uniforme pentru TEXTURARE spre shaderul de
fragmente
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    glUniform1i(glGetUniformLocation(ProgramId, "ok"), 1);

    glDrawArrays(GL_POLYGON, 36, 4);


    glutSwapBuffers();  //Inlocuieste imaginea deseneata in fereastra cu cea
randata;
    glFlush();
}

void Timer(int value) {
    glutPostRedisplay();
    glutTimerFunc(10, Timer, 0);
}

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Depasire");

    glewInit();

    Initialize();
    glutDisplayFunc(RenderFunction);
    glutCloseFunc(Cleanup);
    glutTimerFunc(0, Timer, 0);
    glutMainLoop();

    return 0;
}
```

Shader .FRAG:

```
//
// ================================================
// | Grafica pe calculator                        |
// ================================================
// | Laboratorul III - 03_05_Shader.frag |
// ======================================
//
//  Shaderul de fragment / Fragment shader - afecteaza culoarea pixelilor;
//

#version 330 //  Versiunea GLSL;

//        Variabile de intrare (dinspre Shader.vert);
in vec4 ex_Color;
in vec2 tex_Coord;            //        Coordonata de texturare;

//        Variabile de iesire    (spre programul principal);
out vec4 out_Color;           //        Culoarea actualizata;

//  Variabile uniforme;
uniform sampler2D myTexture;
uniform int ok;

//        Variabile pentru culori;
vec4 red = vec4(1.0,0.0,0.0,1.0);
vec4 green= vec4(0.0,1.0,0.0,1.0);

void main(void)
  {
  //  out_Color=ex_Color;
  //  out_Color=mix(red,green,0.9);
  if(ok == 1)
          out_Color = mix(texture(myTexture, tex_Coord), ex_Color, 0.0);       //
          Amestecarea texturii si a culorii;
  else out_Color=ex_Color;
  }
```

Shader .VERT:

```
//
// ============================================
// | Grafica pe calculator                    |
// ============================================
// | Laboratorul III - 03_05_Shader.vert |
// ======================================
//
//  Shaderul de varfuri / Vertex shader - afecteaza geometria scenei;
//

#version 330    //  Versiunea GLSL;


//  Variabile de intrare (dinspre programul principal);
layout (location = 0) in vec4 in_Position;     //  Se preia din buffer de pe
prima pozitie (0) atributul care contine coordonatele;
layout (location = 1) in vec4 in_Color;        //  Se preia din buffer de pe a
doua pozitie (1) atributul care contine culoarea;
layout (location=2) in vec2 texCoord;          //  Se preia din buffer de pe a
treia pozitie (2) atributul care contine textura;

//  Variabile de iesire;
out vec4 gl_Position;   //  Transmite pozitia actualizata spre programul
principal;
out vec4 ex_Color;      //  Transmite culoarea (de modificat in Shader.frag);
out vec2 tex_Coord;     //  Transmite textura (de modificat in Shader.frag);

//  Variabile uniforme;
uniform mat4 myMatrix;
uniform mat4 view;
uniform mat4 projection;

void main(void)
  {
    //gl_Position = projection*view*myMatrix*in_Position;
    gl_Position = myMatrix*in_Position;
          ex_Color=in_Color;
    tex_Coord = vec2(texCoord.x, 1-texCoord.y);
   }
```