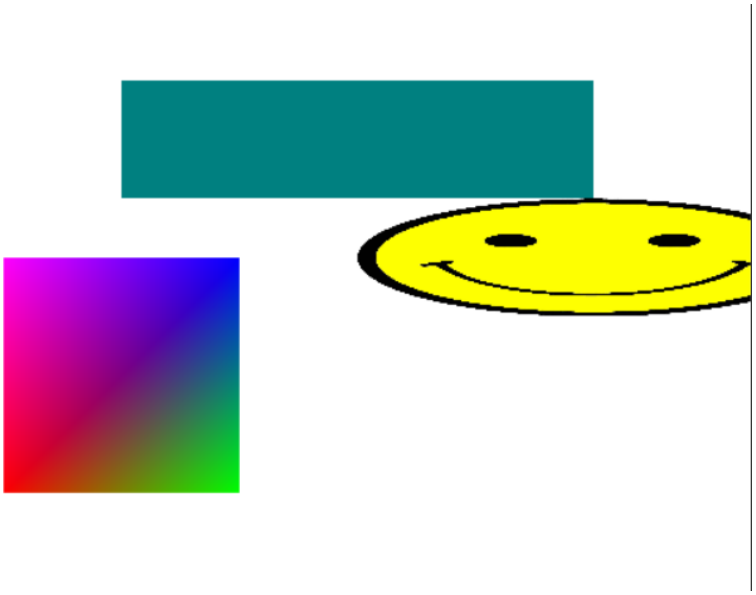
Băicoianu Bianca
Grupa 351

Exercițiul 2

Rezultat:



Cod:

→ în 04_02_Shader.frag, adaugam codul preluat din 04_03_Shader.frag;
definind cele 3 cazuri cu ajutorul switch-ului unde codului 1 îi corespunde texturarea, iar codului 2 - amestecul dintre albastru și verde

04_02_Shader.frag | 04_02_indexare_ex2.cpp

```
7   //
8   //  Shaderul de fragment / Fragment shader - afecteaza culoarea pixelilor;
9   //
10
11  #version 330 core
12
13  //  Variabile de intrare (dinspre Shader.vert);
14  in vec4 ex_Color;
15  in vec2 tex_Coord;      //  Coordonata de texturare;
16
17  //  Variabile de iesire (spre programul principal);
18  out vec4 out_Color;     //  Culoarea actualizata;
19
20  //  Variabile uniforme;
21  uniform sampler2D myTexture;
22  uniform int codFundalShader;
23
24  //  Variabile pentru culori;
25  vec4 blue = vec4(0.0,0.0,1.0,1.0);
26  vec4 green= vec4(0.0,1.0,0.0,1.0);
27
28
29    void main(void)
30    {
31    //  out_Color=ex_Color;
32    //  out_Color=mix(red,green,0.9);
33    // out_Color = mix(texture(myTexture, tex_Coord), ex_Color, 0.2); //  Amestecarea texturii si a culorii;
34      switch(codFundalShader){
35          case 0: out_Color = ex_Color; break;
36          case 1: out_Color = texture(myTexture, tex_Coord); break;
37          case 2: out_Color = mix(blue, green, 0.5); break;
38      }
39    }
```

→ în 04_02_Shader.vert, preluăm codul din 04_03_Shader.vert:

04_02_Shader.vert | 04_02_Shader.frag | 04_02_indexare_ex2.cpp

```
//
// ============================================
// | Grafica pe calculator                    |
// ============================================
// | Laboratorul IV - 04_02_Shader.vert |
// ====================================
//
//  Shaderul de varfuri / Vertex shader - afecteaza geometria scenei;
//

#version 330 core

//  Variabile de intrare (dinspre programul principal);
layout (location = 0) in vec4 in_Position;     //  Se preia din buffer de pe prima pozi
layout (location = 1) in vec4 in_Color;        //  Se preia din buffer de pe a doua poz
layout (location = 2) in vec2 texCoord;          //  Se preia din buffer de pe a treia

//  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 = myMatrix*in_Position;
    ex_Color = in_Color;
    tex_Coord = vec2(texCoord.x, 1-texCoord.y);
   }

```

→ în 04_02_indexare.cpp

→ declarăm *matrTransl* și *matrScal*

→ adăugăm funcția de texturare:

```
//  Functia de incarcare a texturilor in program;
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_RGB);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);

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

→ definim Vertices (sub forma *coordonate* | *culori* | *coord. texturare*)

```
void CreateVBO(void)
{
    //  Coordonatele varfurilor;
    static const GLfloat Vertices[] =
    {
        //  Coordonate;                 Culori;             Coordonate de texturare;
        -15.0f, -15.0f,  0.0f,  1.0f,   1.0f, 0.0f, 0.0f,   0.0f, 0.0f,
         15.0f, -15.0f,  0.0f,  1.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,
         15.0f,  15.0f,  0.0f,  1.0f,   0.0f, 0.0f, 1.0f,   1.0f, 1.0f,
        -15.0f,  15.0f,  0.0f,  1.0f,   1.0f, 0.0f, 1.0f,   0.0f, 1.0f
    };

    //  Indicii care determina ordinea de parcurgere a varfurilor;
    static const GLuint Indices[] =
    {
        0, 1, 2, 3, 0, 2
    };
```

și prelucrăm atributele pentru shadere

```
//  Se activeaza lucrul cu atribute;
//  Se asociaza atributul (0 = coordonate) pentru shader;
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)));
```

→ Initialize(): definim translația și scalarea

```
//  Setarea parametrilor necesari pentru fereastra de vizualizare;
void Initialize(void)
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);       //  Culoarea de fond a ecranului;
    CreateVBO();                                //  Trecerea datelor de randare spre bufferul folosit
    LoadTexture("text_smiley_face.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    CreateShaders();                            //  Initilizarea shaderelor;
    //  Instantierea variabilelor uniforme pentru a "comunica" cu shaderele;
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
    matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(30.f, 30.f, 0.0));
    matrScal = glm::scale(glm::mat4(1.0f), glm::vec3(2.0f, 0.5f, 0.0));
    //  Dreptunghiul "decupat";
    resizeMatrix = glm::ortho(xMin, xMax, yMin, yMax);
}
```

→ RenderFunction():

→ pătratul inițial (corespunzând codului 0) din shader

→ dreptunghiul obținut prin translație și apoi scalare (codul 1 ⇔ texturare)

→ dreptunghi obținut prin scalare și translație (codul 2 ⇔ mix culori)

```
//  Functia de desenarea a graficii pe ecran;
void RenderFunction(void)
{
    glClear(GL_COLOR_BUFFER_BIT);           //  Se curata ecranul OpenGL pentru a fi desenat noul continut;
    codFundalLocation = glGetUniformLocation(ProgramId, "codFundalShader");

    codFundal = 0;
    glUniform1i(codFundalLocation, codFundal);
    glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
    //  Transmiterea variabilei uniforme pentru MATRICEA DE TRANSFORMARE spre shadere;
    myMatrix = resizeMatrix;
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (void*)(0));

    myMatrix = resizeMatrix * matrScal * matrTransl;
    codFundal = 1;
    glUniform1i(codFundalLocation, codFundal);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (void*)(0));

    myMatrix = resizeMatrix * matrTransl * matrScal;
    codFundal = 2;
    glUniform1i(codFundalLocation, codFundal);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (void*)(0));

    glFlush();                              //  Asigura rularea tuturor comenzilor OpenGL apelate anterior;
}
```