Ex2

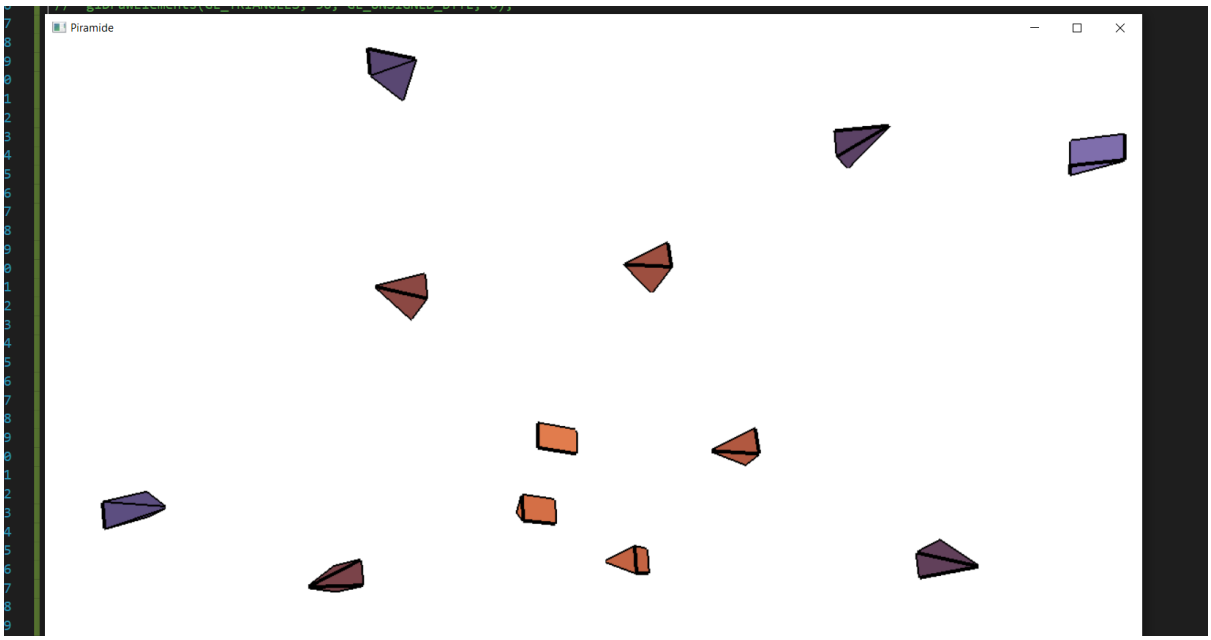
```
// variabile pentru matricea de vizualizare
float Obsx = 0.0, Obsy = 0.0, Obsz = -800.f;
float Refx = 0.0f, Refy = 0.0f;
float Vx = 0.0;

// variabile pentru matricea de proiectie
float width = 800, height = 600, znear = 100, fov = 90;

// vectori
glm::vec3 Obs, PctRef, Vert;

// matrice utilizate
glm::mat4 view, projection;

```

```
// Culorile instantelor
glm::vec4 Colors[INSTANCE_COUNT];
for (int n = 0; n < INSTANCE_COUNT; n++)
{
    float a = float(n) / 4.0f;
    float b = float(n) / 5.0f;
    float c = float(n) / 6.0f;
    Colors[n][0] = 0.35f + 0.30f * (sinf(a + 2.0f) + 1.0f);
    Colors[n][1] = 0.25f + 0.25f * (sinf(b + 3.0f) + 1.0f);
    Colors[n][2] = 0.25f + 0.35f * (sinf(c + 4.0f) + 1.0f);
    Colors[n][3] = 1.0f;
}

// Matricele instantelor
glm::mat4 MatModel[INSTANCE_COUNT];
for (int n = 0; n < INSTANCE_COUNT; n++)
{
    MatModel[n] = glm::translate(glm::mat4(1.0f), glm::vec3(80 * n * sin(10.f * n * 180 / PI), 80 * n * cos(10.f * n * 180 / PI), 0.0)) * glm::rotate(glm::mat4(1.0f), n * PI
}
```

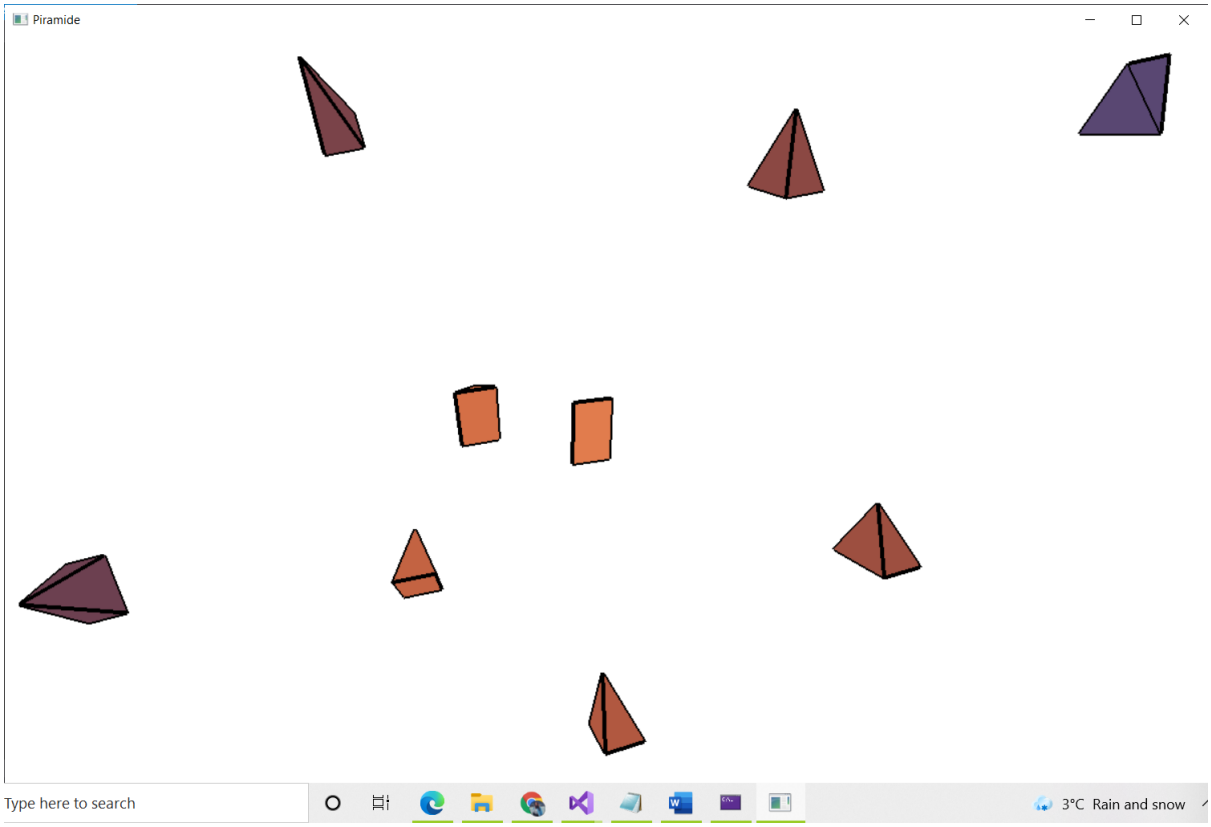
```
// generare buffere
glGenVertexArrays(1, &VaoId);
glGenBuffers(1, &VBPos);
glGenBuffers(1, &VBCol);
glGenBuffers(1, &VBModelMat);
glGenBuffers(1, &EboId);

// legarea VAO
glBindVertexArray(VaoId);

// 0: Pozitie
glBindBuffer(GL_ARRAY_BUFFER, VBPos);
glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 4 * sizeof(GLfloat), (GLvoid*)0);

// 1: Culoare
glBindBuffer(GL_ARRAY_BUFFER, VBCol); // legare buffer
glBufferData(GL_ARRAY_BUFFER, sizeof(Colors), Colors, GL_STATIC_DRAW);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, sizeof(glm::vec4), (GLvoid*)0);
glVertexAttribDivisor(1, 1);  // rata cu care are loc distribuirea culorilor per instanta

```

```
    // 2..5 (2+i): Matrice de pozitie
    glBindBuffer(GL_ARRAY_BUFFER, VBModelMat);
    glBufferData(GL_ARRAY_BUFFER, sizeof(MatModel), MatModel, GL_STATIC_DRAW);
    for (int i = 0; i < 4; i++) // Pentru fiecare coloana
    {
        glEnableVertexAttribArray(2 + i);
        glVertexAttribPointer(2 + i,              // Location
            4, GL_FLOAT, GL_FALSE,                // vec4
            sizeof(glm::mat4),                    // Stride
            (void*)(sizeof(glm::vec4) * i));      // Start offset
        glVertexAttribDivisor(2 + i, 1);
    }

    // Indicii
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices, GL_STATIC_DRAW);
```

```
void RenderFunction(void)

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST); // daca se comenteaza se vor vedea varfurile; functia analizeaza si pozitita lor in spatiu

    // CreateVBO();
    glBindVertexArray(VaoId);
    glBindBuffer(GL_ARRAY_BUFFER, VBPos);
    glBindBuffer(GL_ARRAY_BUFFER, VBCol);
    glBindBuffer(GL_ARRAY_BUFFER, VBModelMat);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId);

    // matricea de vizualizare
    Obs = glm::vec3(Obsx, Obsy, Obsz); // pozitia observatorului
    Refx = Obsx; Refy = Obsy;
    PctRef = glm::vec3(Refx, Refy, 800.0f); // pozitia punctului de referinta
    Vert = glm::vec3(Vx, 1.0f, 0.0f); // verticala din planul de vizualizare
    view = glm::lookAt(Obs, PctRef, Vert);
    glUniformMatrix4fv(viewLocation, 1, GL_FALSE, &view[0][0]);

    // matricea de proiectie, pot fi testate si alte variante
    //projection = glm::perspective(fov, GLfloat(width) / GLfloat(height), znear, zfar);
    // projection = glm::ortho(xwmin, xwmax, ywmin, ywmax, znear, zfar);
    // projection = glm::frustum(xwmin, xwmax, ywmin, ywmax, 80.f, -80.f);
    projection = glm::infinitePerspective(fov * PI / 180, GLfloat(width) / GLfloat(height), znear);
    glUniformMatrix4fv(projLocation, 1, GL_FALSE, &projection[0][0]);

    // Fetele
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    glDrawElementsInstanced(GL_TRIANGLES, 18, GL_UNSIGNED_BYTE, 0, INSTANCE_COUNT);
    // Muchiile
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    glLineWidth(3.5);
    glDrawElementsInstanced(GL_LINE_LOOP, 4, GL_UNSIGNED_BYTE, (void*)(18), INSTANCE_COUNT);
    glDrawElementsInstanced(GL_LINES, 8, GL_UNSIGNED_BYTE, (void*)(22), INSTANCE_COUNT);
```



Rotite spre dreapta



Spre deosebire de subpunctul 2, am schimbat plasarea piramidelor si a observatorului, precum si functiile `void processSpecialKeys(int key, int xx, int yy);` si `processNormalKeys(unsigned char key, int x, int y);`

```
51
52  void processNormalKeys(unsigned char key, int x, int y)
53  {
54      switch (key) {
55      case '-':
56          dist -= 5.0;
57          break;
58      case '+':
59          dist += 5.0;
60          break;
61      }
62      if (key == 27)
63          exit(0);
64  }
65  void processSpecialKeys(int key, int xx, int yy)
66  {
67      switch (key)
68      {
69      case GLUT_KEY_LEFT:
70          beta -= 0.01;
71          break;
72      case GLUT_KEY_RIGHT:
73          beta += 0.01;
74          break;
75      case GLUT_KEY_UP:
76          alpha += incr_alpha1;
77          if (abs(alpha - PI / 2) < 0.05)
78          {
79              incr_alpha1 = 0.f;
80          }
81          else
82          {
83              incr_alpha1 = 0.01f;
```

```cpp
// Matricele instantelor
glm::mat4 MatModel[INSTANCE_COUNT];
for (int n = 0; n < INSTANCE_COUNT; n++)
{
    MatModel[n] = glm::translate(glm::mat4(1.0f), glm::vec3( 20  *n* cos(10.f * n * 180 / PI), 20 * n * sin(10.f * n * 180 / PI), n*20));

}
```

```cpp
//pozitia observatorului ---- ex3
Obsx = Refx + dist * cos(alpha) * cos(beta);
Obsy = Refy + dist * cos(alpha) * sin(beta);
Obsz = Refz + dist * sin(alpha);

// reperul de vizualizare -
glm::vec3 Obs = glm::vec3(Obsx, Obsy, Obsz);   // se schimba pozitia observatorului
glm::vec3 PctRef = glm::vec3(Refx, Refy, Refz); // pozitia punctului de referinta
glm::vec3 Vert = glm::vec3(Vx, Vy, Vz); // verticala din planul de vizualizare
view = glm::lookAt(Obs, PctRef, Vert);
glUniformMatrix4fv(viewLocation, 1, GL_FALSE, &view[0][0]);
```