

Hazırlayan: Tayfun GÜRLEVİK

Öğrenci No: N19139647

ÖDEV NO: 2

Soru1:

Size verilen dino.dat ve polyline.cpp dosyalarından yararlanarak aşağıda gösterilen ekran çıktısını elde ediniz.



Yanıt:

a) Çözüm

Sorunun çözümü için öncelikle pencerenin genişlik ve yüksekliğini tutan iki adet sabit tanımladım(WIDTH ve HEIGHT).

X ve Y yönünde pencerede oluşacak bölme sayısını xNumber ve yNumber adındaki iki değişkene atadım.

Render() yordamını iç içe iki for döngüsü oluşturarak soruda istenen viewport sayısını elde ettim. Viewportların left,bottom,width,height parametrelerini döngü sayaçlarının yardımı ile her bir viewport için tekrar hesapladım ve drawPolylineFile() metodunu çağırdım.

b) Kod

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

#include <fstream>
const int WIDTH = 640;
const int HEIGHT = 480;
int xNumber = 5;
int yNumber = 5;
void drawPolylineFile(const char * fileName) {

    std::ifstream inStream;
    inStream.open(fileName);    // open the file
```

```

    if(inStream.fail())
        return;
    //glClear(GL_COLOR_BUFFER_BIT);        // clear the screen
    GLint numpolys, numLines, x ,y;
    inStream >> numpolys;                // read the number of polylines
    for(int j = 0; j < numpolys; j++)    // read each polyline
    {
        inStream >> numLines;
        glBegin(GL_LINE_STRIP);        // draw the next polyline
        for (int i = 0; i < numLines; i++)
        {
            inStream >> x >> y;        // read the next x, y pair
            glVertex2i(x, y);
        }
        glEnd();
    }
    glFlush();
    inStream.close();
}

//----- setWindow -----
void setWindow(float left, float right, int bottom, int top)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(left, right, bottom, top);
}

//----- setViewport -----
void setViewport(int left, int bottom, int width, int height)
{
    glViewport(left, bottom, width, height);
}

void render() {

    glClear(GL_COLOR_BUFFER_BIT);

    setWindow(0, WIDTH, 0, HEIGHT);        // set a fixed window
    for (size_t i = 0; i < xNumber; i++)
    {
        for (size_t j = 0; j < yNumber; j++)
        {
            setViewport(i * WIDTH / xNumber, j*HEIGHT/yNumber, WIDTH / xNumber,
HEIGHT / yNumber);
            drawPolyLineFile("../dino.dat");        // draw it again
        }
    }

    glFlush();
}

int main(int argc, char** argv)
{
    glutInit( &argc, argv );

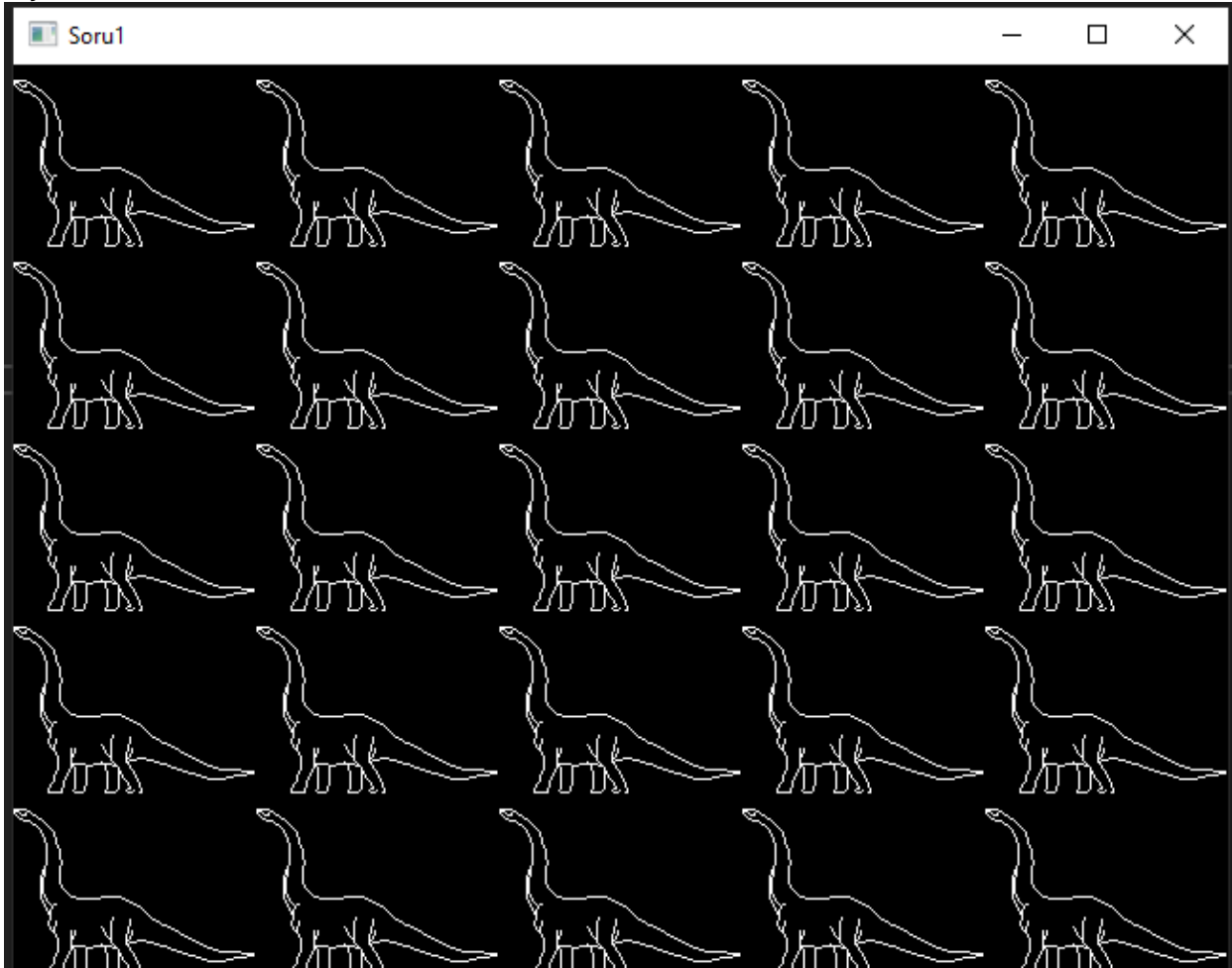
    glutInitDisplayMode( GLUT_SINGLE | GLUT_RGB );
    glutInitWindowSize( 640,480 );
    glutInitWindowPosition( 0, 0 );
    glutCreateWindow( "Soru1" );

    glutDisplayFunc( render );
    glutMainLoop();
}

```

```
return( 0 );
```

```
}
```



c)Ekler

Soru1.cpp, Soru1.exe,dino.dat

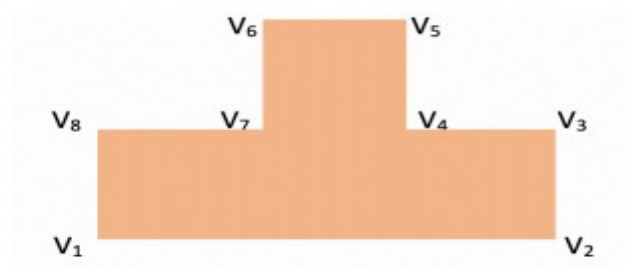
Soru2:

Köşeler dizisi

$v_1 = (20,20); v_2 = (80,20); v_3 = (80,40); v_4 = (60,40); v_5 = (60,60); v_6 = (40,60);$

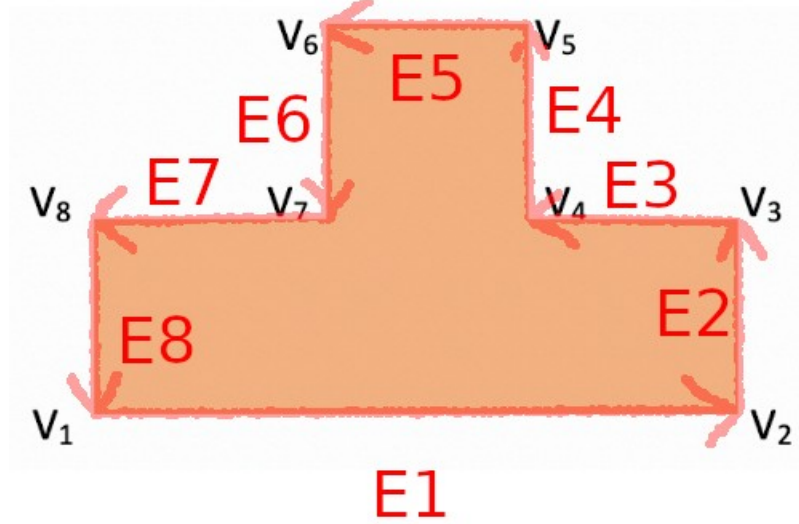
$v_7 = (40,40); v_8 = (20,40);$

olmak üzere şekilde görülen poligon verilmektedir. Bu poligon içbükey olup olmadığını vektörel çarpım (cross product) yöntemiyle döndüren bir C++ programı yazınız.



Yanıt:

a) Çözüm



Şekilde görüldüğü gibi kenar vektörleri tanımlanmıştır. GLM kütüphanesinin içind bulunan cross() metodu vasıtasıyla ardışık vektörlerin vektörel çarpımlarını hesaplayıp sonucun z bileşenini bir vector<float> dizisinin içine attım. Bu değerleri bir döngü içerisinde iterasyonla kendisinden sonra gelen değerin matematiksel çarpımının 0'dan büyük veya küçük olmasına bakarak (pozitif ve negatif işaretli sayıların çarpımı negatiftir, aynı işaretli sayıların çarpımı pozitifdir prensibiyle)poligonun içbükey veya dışbükey olup olmadığını döndürdüm.

b) Kod

```
#include <iostream>
#include <glm.hpp>
#include <vector>
using namespace glm;
using namespace std;
int main()
{
    //Vertexler
    vec3 V1, V2, V3, V4, V5, V6, V7, V8;
    vector<vec3> vertexler;
    //Vertexleri birlestiren kenarlar
    vec3 E1, E2, E3, E4, E5, E6, E7, E8;
    vector<vec3> kenarVektorler;
    //Vertexlerin degerleri
    V1.x = 20.f; V1.y = 20.f;
    vertexler.push_back(V1);
    V2.x = 80.f; V2.y = 20.f;
    vertexler.push_back(V2);
    V3.x = 80.f; V3.y = 40.f;
```

```

vertexler.push_back(V3);
V4.x = 60.f; V4.y = 40.f;
vertexler.push_back(V4);
V5.x = 60.f; V5.y = 60.f;
vertexler.push_back(V5);
V6.x = 40.f; V6.y = 60.f;
vertexler.push_back(V6);
V7.x = 40.f; V7.y = 40.f;
vertexler.push_back(V7);
V8.x = 20.f; V8.y = 40.f;
vertexler.push_back(V8);
cout << "Vertex listesi" << endl;
for (size_t i = 0; i < vertexler.size(); i++)
{
    cout << "Vertex" << i + 1 << ": " << vertexler[i].x << ", " << vertexler[i].y << endl;
}
for (size_t i = 0; i < vertexler.size() - 1; i++)
{
    kenarVektorler.push_back(vertexler[i + 1] - vertexler[i]);
}
kenarVektorler.push_back(vertexler[0] - vertexler[vertexler.size() - 1]);

vector<float> zCrossProducts;
for (size_t i = 0; i < kenarVektorler.size()-1; i++)
{
    zCrossProducts.push_back(cross(kenarVektorler[i], kenarVektorler[i + 1]).z);
}
zCrossProducts.push_back(cross(kenarVektorler[kenarVektorler.size() -
1], kenarVektorler[0] ).z);
cout << "Vektorel carpimlarin z bileşenleri" << endl;
for (size_t i = 0; i < zCrossProducts.size(); i++)
{
    cout << zCrossProducts[i] << endl;
}
bool disbukey = false;
for (size_t i = 0; i < zCrossProducts.size()-1; i++)
{
    if ((zCrossProducts[i + 1] * zCrossProducts[i]) > 0)
        disbukey = true;
    else
    {
        disbukey = false;
        break;
    }
}
if (disbukey)
{
    cout << "Polygon disbukey" << endl;
}
else
{
    cout << "Polygon icbukey" << endl;
}
return 0;
}

```

```
Microsoft Visual Studio Debug Console
Vertex listesi
Vertex1: 20,20
Vertex2: 80,20
Vertex3: 80,40
Vertex4: 60,40
Vertex5: 60,60
Vertex6: 40,60
Vertex7: 40,40
Vertex8: 20,40
Vektorel carpimlarin z bileşenleri
1200
400
-400
400
400
-400
400
1200
Polygon icbukey

C:\Users\Tayfun\Documents\GitHub\BCA-611-0dev-ve-Projeler\0dev2\Soru2\Debug\Soru2.exe (process 3672) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

c)Ekler

Soru2.cpp, Soru2.exe

Soru3)

Aşağıda Gram-Schmidt Ortogonalleştirme algoritması verilmiştir. Bu algoritmayı kullanarak

$$\begin{aligned}\mathbf{e}_1 &= \left\langle \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0 \right\rangle \\ \mathbf{e}_2 &= \langle -1, 1, -1 \rangle \\ \mathbf{e}_3 &= \langle 0, -2, -2 \rangle\end{aligned}$$

vektörlerini ortogonalleştiriniz. (*orthogonalleştirme* verilen vektörler kümesinden her biri birbirine dik olan yeni bir vektörler kümesi elde etmek demektir.)

Algorithm 2.16. Gram-Schmidt Orthogonalization. Given a set of n linearly independent vectors $\mathcal{B} = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_n\}$, this algorithm produces a set $\mathcal{B}' = \{\mathbf{e}'_1, \mathbf{e}'_2, \dots, \mathbf{e}'_n\}$ such that $\mathbf{e}'_i \cdot \mathbf{e}'_j = 0$ whenever $i \neq j$.

- A. Set $\mathbf{e}'_1 = \mathbf{e}_1$.
- B. Begin with the index $i = 2$.
- C. Subtract the projection of \mathbf{e}_i onto the vectors $\mathbf{e}'_1, \mathbf{e}'_2, \dots, \mathbf{e}'_{i-1}$ from \mathbf{e}_i and store the result in \mathbf{e}'_i . That is,

$$\mathbf{e}'_i = \mathbf{e}_i - \sum_{k=1}^{i-1} \frac{\mathbf{e}_i \cdot \mathbf{e}'_k}{\mathbf{e}'_k \cdot \mathbf{e}'_k} \mathbf{e}'_k. \quad (2.43)$$

- D. If $i < n$, increment i and loop to step C.

Yanıt)

a)Çözüm

Adım C’de belirtilen projeksiyonların toplamını bulmak için ProjectionSum adında bir metod oluşturdum. Bu metod denklem 2.43 de belirtilen çıkartılacak olan kısmı hesaplmaktadır.

İkinci vektörden döngüyü başlatacak şekilde bir do-while döngüsü ile Adım D’de belirtilen koşul sağlana kadar iterasyon yaptım. Hesaplanan ortogonal vektörleri e_prime adındaki bir dizide tuttum ve ekrana yazdırdım.

Bunun sonucunda ortogonalleştirilmiş olan vektörler dizisi aşağıdaki gibi çıkmaktadır:

$e_1' = (0.707107, 0.707107, 0)$

$e_2' = (-1, 1, -1)$

$e_3' = (1, -1, -2)$

b) Kod

```
#include <iostream>
#include <glm.hpp>
#include <vector>

using namespace std;
using namespace glm;

vec3 ProjectionSum(glm::vec3* e, glm::vec3* e_prime, int& i)
{
    int k = 0;
    vec3 result;
    result.x = 0;
    result.y = 0;
    result.z = 0;
    while (k < i - 1)
    {
        float e_prime_k_squared = glm::dot(e_prime[k], e_prime[k]);
        result += ((glm::dot(e[i], e_prime[k]) / e_prime_k_squared) * e_prime[k]);
        k++;
    }

    return result;
}

int main()
{
    const int n = 3; // Vektor sayisi
    vec3 e[] = {
        vec3(sqrt(2) / 2, sqrt(2) / 2, 0),
        vec3(-1, 1, -1),
        vec3(0, -2, -2)
    };

    vec3 e_prime[n];
    e_prime[0] = e[0]; // Adim A

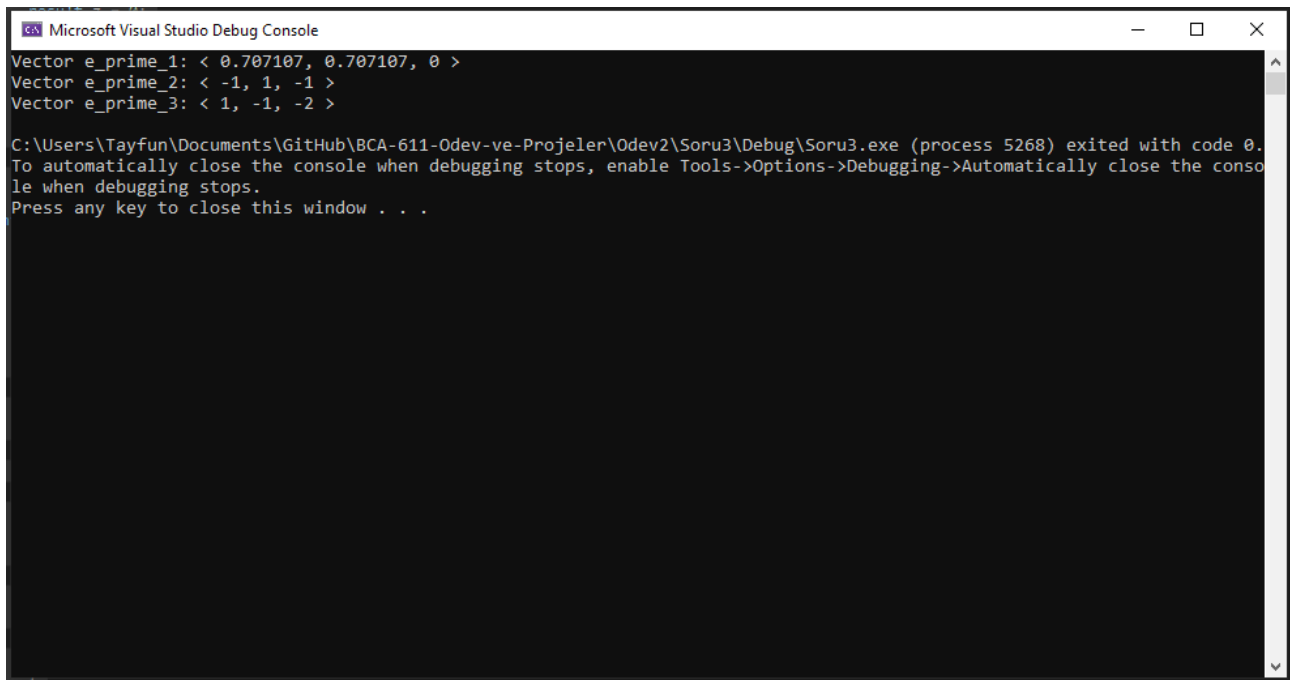
    int i = 0; // Adim B

    do // Adim C
    {
        e_prime[i] = e[i] - ProjectionSum(e, e_prime, i);

        i++; // Adim D
    } while (i < n);

    for (int i = 0; i < n; i++)
    {
        std::cout << "Vector e_prime_" << i + 1 << ": < "
            << e_prime[i].x << ", "
            << e_prime[i].y << ", "
            << e_prime[i].z << " >" << std::endl;
    }
}
```

```
    return 0;  
}
```



The image shows a screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the text "Microsoft Visual Studio Debug Console" and standard window controls (minimize, maximize, close). The console output is as follows:

```
Vector e_prime_1: < 0.707107, 0.707107, 0 >  
Vector e_prime_2: < -1, 1, -1 >  
Vector e_prime_3: < 1, -1, -2 >  
  
C:\Users\Tayfun\Documents\GitHub\BCA-611-Odev-ve-Projeler\Odev2\Soru3\Debug\Soru3.exe (process 5268) exited with code 0.  
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.  
Press any key to close this window . . .
```

c)Ekler
Soru3.cpp, Soru3.exe