20.10.2019

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 );

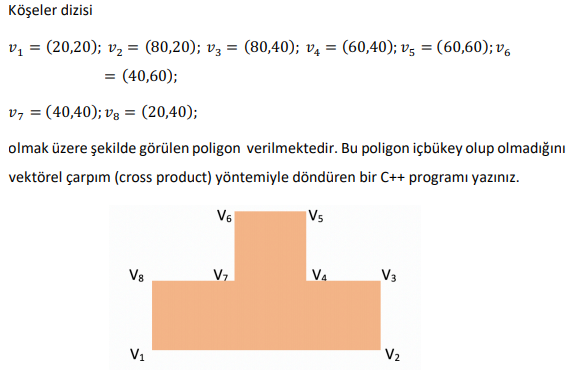
}



c)Ekler

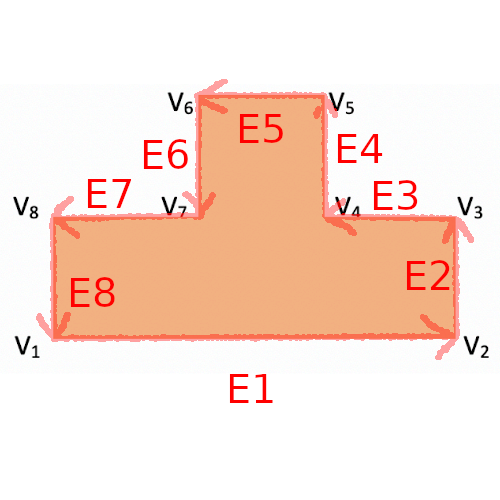
Soru1.cpp, Soru1.exe,dino.dat

Soru2:



Yanıt:

a) Çözüm



Şekilde görüldüğü gibi kenar vektörleri tanımlanmıştır. GLM kütüphanesinin içind bulunan cross() metodu vasıtaysıyla 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 pozitiftir 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;

//Vertextleri 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 bilesenleri" << 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

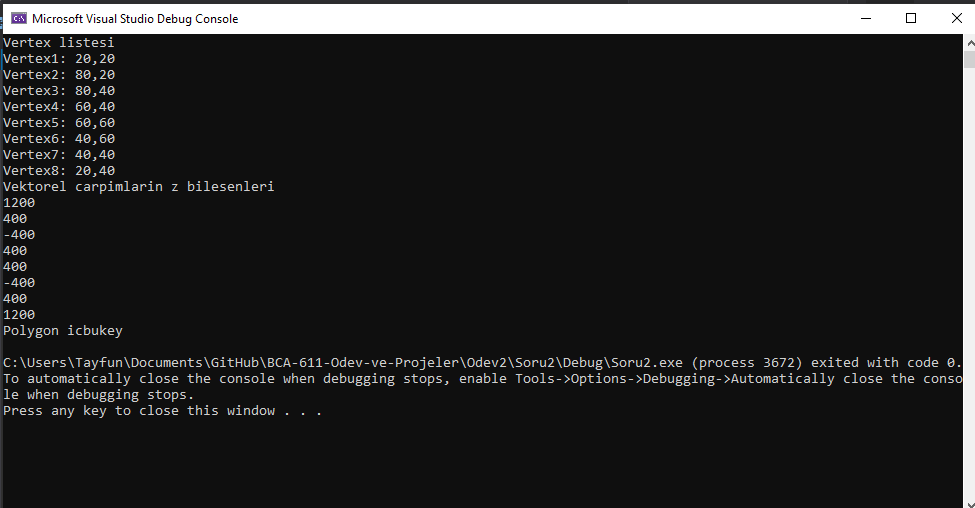
{

cout << "Polygon icbukey" << endl;

}

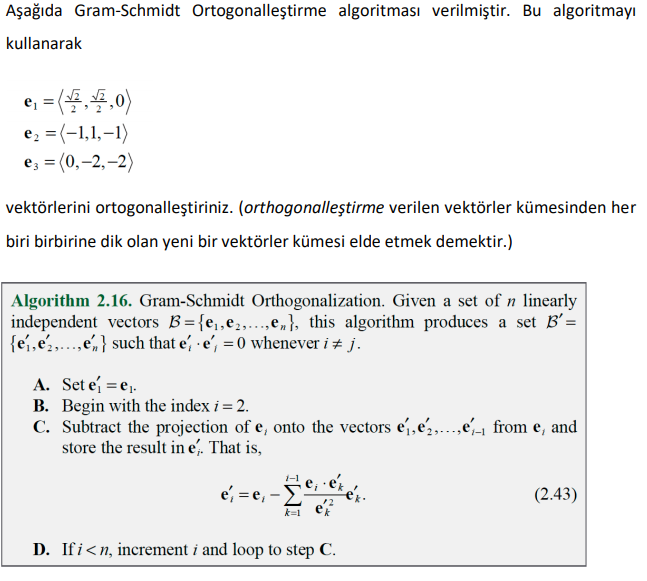
return 0;

}

c)Ekler

Soru2.cpp, Soru2.exe

Soru3)



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 ortoganal vektörleri e\_prime adındaki bir dizide tuttum ve ekrana yazdırdım.

Bunun sonucunda ortogonalleştirilmiş olan vektörler dizisi aşağıdaki gibi çıkmaktadır:

e1’=( 0.707107, 0.707107, 0)

e2’=( -1,1,-1)

e3’=( 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 << ", "

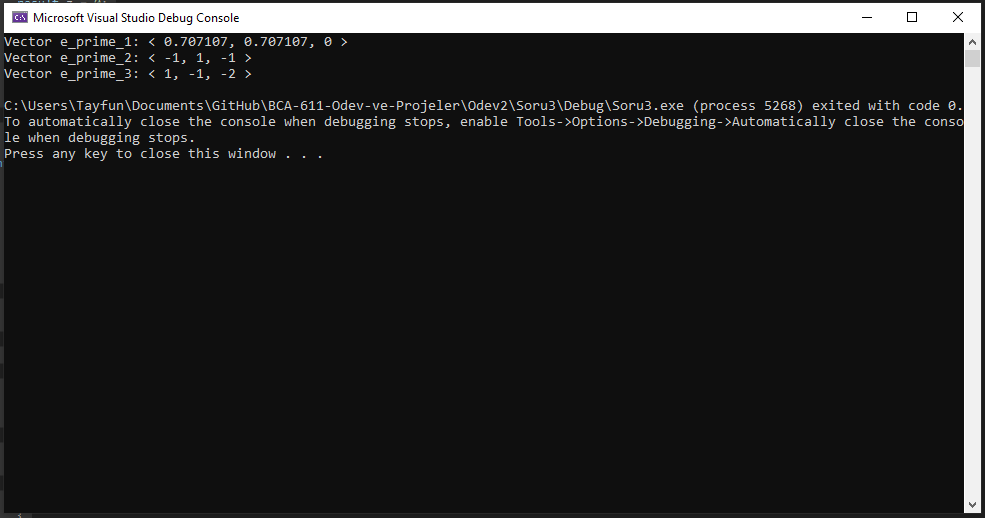
<< e\_prime[i].y << ", "

<< e\_prime[i].z << " >" << std::endl;

}

return 0;

}



c)Ekler

Soru3.cpp, Soru3.exe