16.12.2019

BCA 611 Video Oyunları İçin 3B Grafik

ÖDEV4

Hazırlayan : Tayfun GÜRLEVİK

Öğrenci No: N19139647

Aşama 1:

Bir OpenGL programıyla x, y, z bileşenleri en fazla [-100,100] aralığında olan 20 adet sentetik 3B nokta üretin ve bu noktaların konumlarına birer çok küçük kırmızı küre yerleştirerek noktaların viewport içinde görünmesini sağlayın.

Çözüm:

20 adet küre için bir vector tanımlanmıştır. Bu vektörün için koordinatları -100,100 arasında değişen merkez noktaları push edilmiştir.

```
vector<vec3> kure_merkezler;
int kureSayisi = 20;

int randomGen()
{
	int max = 100, min = -100, range;
	range = max - min + 1;
	return rand() % range + min;
}

for (size_t i = 0; i < kureSayisi; i++)
	{
		vec3 merkez;
		merkez.x = randomGen();
		merkez.y = randomGen();
		merkez.z = randomGen();
		kure_merkezler.push_back(merkez);
	}
```

drawScene Metodunun içinde küreler çizdirilmiştir.

```
for (size_t i = 0; i < kureSayisi; i++)
	{
		glPushMatrix();
		glTranslatef(kure_merkezler[i].x, kure_merkezler[i].y,
kure_merkezler[i].z);
		glutWireSphere(1, 10, 10);
		glPopMatrix();
	}
```

Aşama 2:

Principal Component Analysis (PCA) yöntemini uygulayarak bu noktaların belirlediği özdeğerleri (eigenvalues) ve özvektörleri (eigenvektors) yazdırın.

Çözüm:

Rastgele elde edilen noktaların ortalamaları bulunmuştur.

```
vec3 OrtaNokta(vector<vec3> noktalar)
{
	float x;
	float y;
	float z;
	float sumx=0;
	float sumy=0;
	float sumz=0;
	for (size_t i = 0; i < noktalar.size(); i++)
	{
		sumx += noktalar[i].x;
		sumy += noktalar[i].y;
		sumz += noktalar[i].z;
	}
	x = sumx / noktalar.size();
	y= sumy / noktalar.size();
	z=sumz / noktalar.size();
	vec3 ortanokta;
	ortanokta.x = x;
	ortanokta.y = y;
	ortanokta.z = z;
	return ortanokta;

}
```

Bulunan orta nokta yardımıyla Covaryans matrisi elde edilmiştir.

```
mat3 CovaryansMatrisiOlustur(vector<vec3> noktalar,vec3 m)
{
	mat3 C= mat3(vec3(0, 0, 0), vec3(0, 0, 0), vec3(0, 0, 0));
	size_t N = noktalar.size();
	float sumx = 0;
	float sumy = 0;
	float sumz = 0;
	float sum12 = 0;
	float sum13 = 0;
	float sum23 = 0;
	for (size_t i = 0; i < N; i++)
	{
		sumx += glm::pow(noktalar[i].x - m.x, 2);
		sumy += glm::pow(noktalar[i].y - m.y, 2);
		sumz += glm::pow(noktalar[i].z - m.z, 2);
		sum12 += (noktalar[i].x - m.x) * (noktalar[i].y - m.y);
		sum13+= (noktalar[i].x - m.x) * (noktalar[i].z - m.z);
		sum23 += (noktalar[i].y - m.y) * (noktalar[i].z - m.z);
	}
	C[0].x = sumx / N;
```

```
	C[1].y = sumy / N;
	C[2].z = sumz / N;
	C[0].y = C[1].x = sum12 / N;
	C[0].z = C[2].x = sum13 / N;
	C[1].z = C[2].y = sum23 / N;
	return C;

}
```

Mathematics for 3D Game Programming and Computer Graphics, Eric Lengyel kitabında 16.3 nolu kısımda anlatılan yöntemle özdeğerler ve özvektörler hesaplanmıştır.

```
const float epsilon = 1.0e-10F;
const int maxSweeps = 32;
void CalculateEigensystem(const mat3x3& m, float* lambda, mat3x3& r)
{
	float m11 = m[0].x;
	float m12 = m[1].x;
	float m13 = m[2].x;
	float m22 = m[1].y;
	float m23 = m[2].y;
	float m33 = m[2].z;
	r = mat3x3(1.0f);

	for (int a = 0; a < maxSweeps; a++)
	{
		// Exit if off-diagonal entries small enough.
		if ((abs(m12) < epsilon) && (abs(m13) < epsilon) &&
			(abs(m23) < epsilon)) break;
		// Annihilate (1,2) entry.
		if (m12 != 0.0F)
		{
			float u = (m22 - m11) * 0.5F / m12;
			float u2 = u * u;
			float u2p1 = u2 + 1.0F;
			float t = (u2p1 != u2) ?
				((u < 0.0F) ? -1.0F : 1.0F) * (sqrt(u2p1) - fabs(u))
				: 0.5F / u;
			float c = 1.0F / sqrt(t * t + 1.0F);
			float s = c * t;
				m11 -= t * m12;
			m22 += t * m12;
			m12 = 0.0F;
			float temp = c * m13 - s * m23;
			m23 = s * m13 + c * m23;
			m13 = temp;
			for (int i = 0; i < 3; i++)
			{
				float temp = c * r[i].x - s * r[i].y;
				r[i].y = s * r[i].x + c * r[i].y;
				r[i].x = temp;
			}
		}
		// Annihilate (1,3) entry.
		if (m13 != 0.0F)
		{
			float u = (m33 - m11) * 0.5F / m13;
			float u2 = u * u;
```

```
                        float u2p1 = u2 + 1.0F;
                        float t = (u2p1 != u2) ?
                               ((u < 0.0F) ? -1.0F : 1.0F) * (sqrt(u2p1) - fabs(u))
                               : 0.5F / u;
                        float c = 1.0F / sqrt(t * t + 1.0F);
                        float s = c * t;
                        m11 -= t * m13;
                        m33 += t * m13;
                        m13 = 0.0F;
                        float temp = c * m12 - s * m23;
                        m23 = s * m12 + c * m23;
                        m12 = temp;
                        for (int i = 0; i < 3; i++)
                        {
                               float temp = c * r[i].x - s * r[i].z;
                               r[i].z= s * r[i].x + c * r[i].z;
                               r[i].x = temp;
                        }
                }
                // Annihilate (2,3) entry.
                if (m23 != 0.0F)
                {
                        float u = (m33 - m22) * 0.5F / m23;
                        float u2 = u * u;
                        float u2p1 = u2 + 1.0F;
                        float t = (u2p1 != u2) ?
                               ((u < 0.0F) ? -1.0F : 1.0F) * (sqrt(u2p1) - fabs(u))
                               : 0.5F / u;
                        float c = 1.0F / sqrt(t * t + 1.0F);
                        float s = c * t;
                        m22 -= t * m23;
                        m33 += t * m23;
                        m23 = 0.0F;
                        float temp = c * m12 - s * m13;
                        m13 = s * m12 + c * m13;
                        m12 = temp;
                        for (int i = 0; i < 3; i++)
                        {
                               float temp = c * r[i].y - s * r[i].z;
                               r[i].z= s * r[i].y+ c * r[i].z;
                               r[i].y = temp;
                        }
                }
        }
        lambda[0] = m11;
        lambda[1] = m22;
        lambda[2] = m33;
}
```

b) Kod:

Kodun tamamı aşağıdaki gibidir.

```
#include <iostream>

#include <GL/glew.h>
```

```
#include <GL/freeglut.h>
#include <vector>
#include <glm.hpp>



using namespace glm;
using namespace std;
const float epsilon = 1.0e-10F;
const int maxSweeps = 32;
void CalculateEigensystem(const mat3x3& m, float* lambda, mat3x3& r)
{
	float m11 = m[0].x;
	float m12 = m[1].x;
	float m13 = m[2].x;
	float m22 = m[1].y;
	float m23 = m[2].y;
	float m33 = m[2].z;
	r = mat3x3(1.0f);

	for (int a = 0; a < maxSweeps; a++)
	{
		// Exit if off-diagonal entries small enough.
		if ((abs(m12) < epsilon) && (abs(m13) < epsilon) &&
			(abs(m23) < epsilon)) break;
		// Annihilate (1,2) entry.
		if (m12 != 0.0F)
		{
			float u = (m22 - m11) * 0.5F / m12;
			float u2 = u * u;
			float u2p1 = u2 + 1.0F;
			float t = (u2p1 != u2) ?
				((u < 0.0F) ? -1.0F : 1.0F) * (sqrt(u2p1) - fabs(u))
				: 0.5F / u;
			float c = 1.0F / sqrt(t * t + 1.0F);
			float s = c * t;
				m11 -= t * m12;
			m22 += t * m12;
			m12 = 0.0F;
			float temp = c * m13 - s * m23;
			m23 = s * m13 + c * m23;
			m13 = temp;
			for (int i = 0; i < 3; i++)
			{
				float temp = c * r[i].x - s * r[i].y;
				r[i].y = s * r[i].x + c * r[i].y;
				r[i].x = temp;
			}
		}
		// Annihilate (1,3) entry.
		if (m13 != 0.0F)
		{
			float u = (m33 - m11) * 0.5F / m13;
			float u2 = u * u;
			float u2p1 = u2 + 1.0F;
			float t = (u2p1 != u2) ?
				((u < 0.0F) ? -1.0F : 1.0F) * (sqrt(u2p1) - fabs(u))
				: 0.5F / u;
```

```
                        float c = 1.0F / sqrt(t * t + 1.0F);
                        float s = c * t;
                        m11 -= t * m13;
                        m33 += t * m13;
                        m13 = 0.0F;
                        float temp = c * m12 - s * m23;
                        m23 = s * m12 + c * m23;
                        m12 = temp;
                        for (int i = 0; i < 3; i++)
                        {
                                float temp = c * r[i].x - s * r[i].z;
                                r[i].z= s * r[i].x + c * r[i].z;
                                r[i].x = temp;
                        }
                }
                // Annihilate (2,3) entry.
                if (m23 != 0.0F)
                {
                        float u = (m33 - m22) * 0.5F / m23;
                        float u2 = u * u;
                        float u2p1 = u2 + 1.0F;
                        float t = (u2p1 != u2) ?
                                ((u < 0.0F) ? -1.0F : 1.0F) * (sqrt(u2p1) - fabs(u))
                                : 0.5F / u;
                        float c = 1.0F / sqrt(t * t + 1.0F);
                        float s = c * t;
                        m22 -= t * m23;
                        m33 += t * m23;
                        m23 = 0.0F;
                        float temp = c * m12 - s * m13;
                        m13 = s * m12 + c * m13;
                        m12 = temp;
                        for (int i = 0; i < 3; i++)
                        {
                                float temp = c * r[i].y - s * r[i].z;
                                r[i].z= s * r[i].y+ c * r[i].z;
                                r[i].y = temp;
                        }
                }
        }
        lambda[0] = m11;
        lambda[1] = m22;
        lambda[2] = m33;
}
int randomGen()
{
        int max = 100, min = -100, range;
        range = max - min + 1;
        return rand() % range + min;
}
vec3 OrtaNokta(vector<vec3> noktalar)
{
        float x;
        float y;
        float z;
        float sumx=0;
        float sumy=0;
        float sumz=0;
```

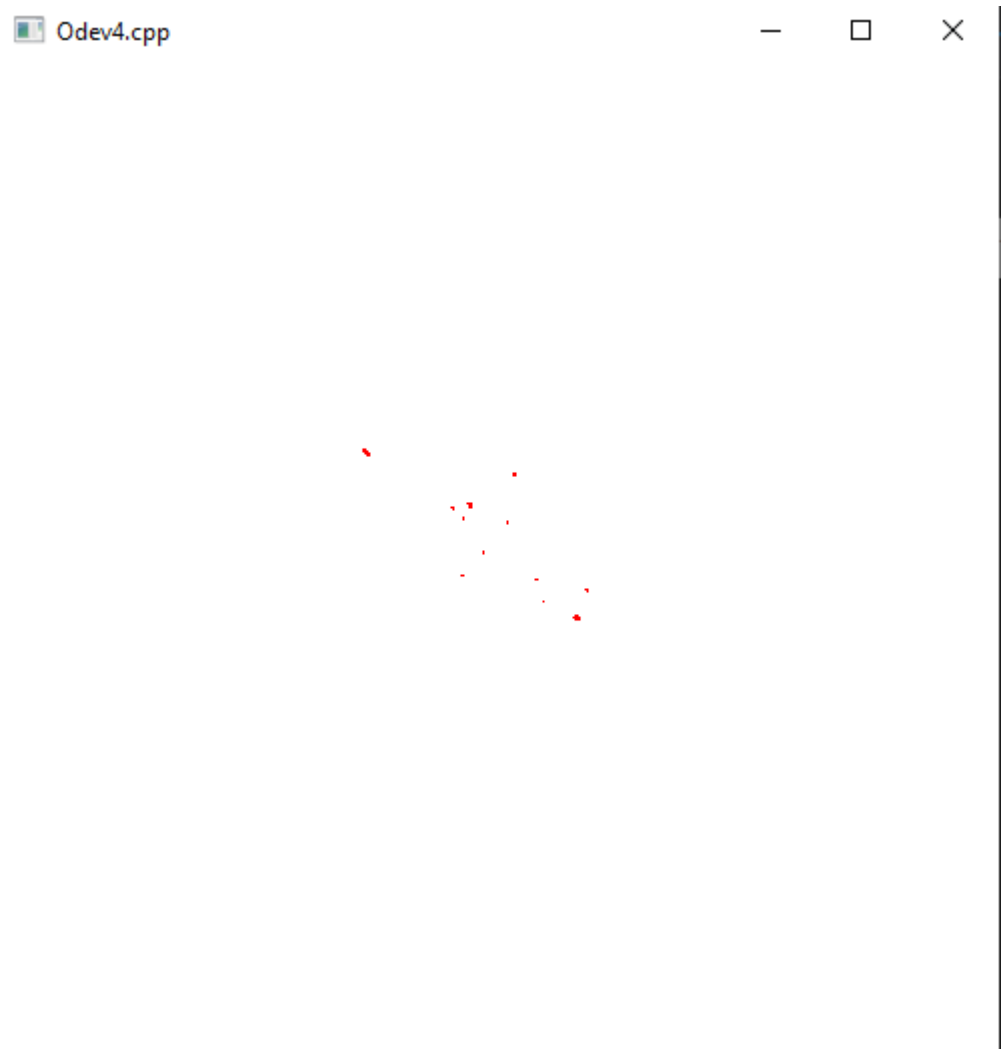
```
	for (size_t i = 0; i < noktalar.size(); i++)
	{
		sumx += noktalar[i].x;
		sumy += noktalar[i].y;
		sumz += noktalar[i].z;
	}
	x = sumx / noktalar.size();
	y= sumy / noktalar.size();
	z=sumz / noktalar.size();
	vec3 ortanokta;
	ortanokta.x = x;
	ortanokta.y = y;
	ortanokta.z = z;
	return ortanokta;

}
mat3 CovaryansMatrisiOlustur(vector<vec3> noktalar,vec3 m)
{
	mat3 C= mat3(vec3(0, 0, 0), vec3(0, 0, 0), vec3(0, 0, 0));
	size_t N = noktalar.size();
	float sumx = 0;
	float sumy = 0;
	float sumz = 0;
	float sum12 = 0;
	float sum13 = 0;
	float sum23 = 0;
	for (size_t i = 0; i < N; i++)
	{
		sumx += glm::pow(noktalar[i].x - m.x, 2);
		sumy += glm::pow(noktalar[i].y - m.y, 2);
		sumz += glm::pow(noktalar[i].z - m.z, 2);
		sum12 += (noktalar[i].x - m.x) * (noktalar[i].y - m.y);
		sum13+= (noktalar[i].x - m.x) * (noktalar[i].z - m.z);
		sum23 += (noktalar[i].y - m.y) * (noktalar[i].z - m.z);
	}
	C[0].x = sumx / N;
	C[1].y = sumy / N;
	C[2].z = sumz / N;
	C[0].y = C[1].x = sum12 / N;
	C[0].z = C[2].x = sum13 / N;
	C[1].z = C[2].y = sum23 / N;
	return C;

}


vector<vec3> kure_merkezler;
int kureSayisi = 20;
float eyeZ=150;
float eyeX = 100;
vec3 m;
mat3 CovarianceMatrix= mat3(vec3(0,0,0), vec3(0, 0, 0), vec3(0, 0, 0));
vec3 e1,e2,e3;
float lambda[3];
// Drawing routine.
void drawScene(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
```

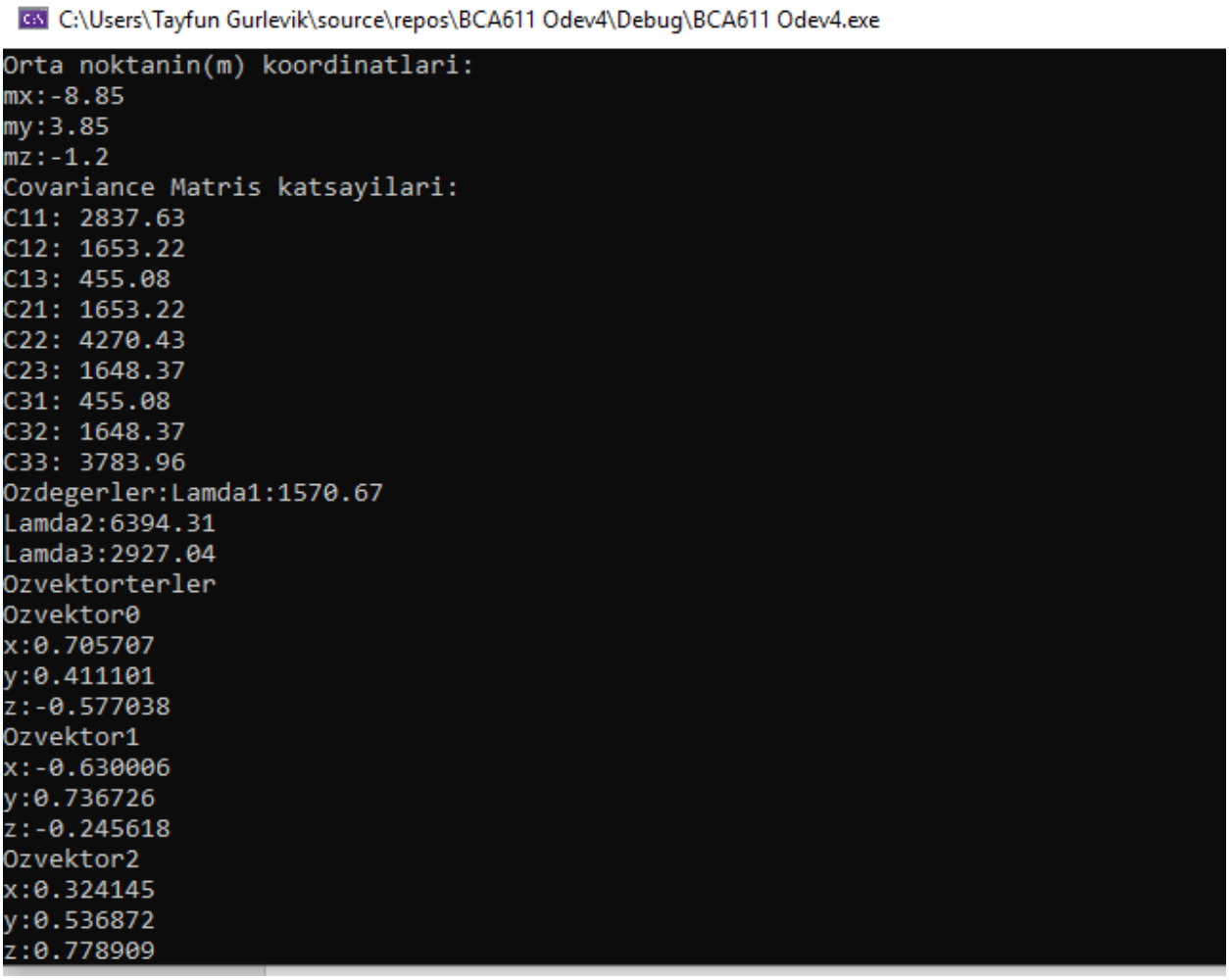
```
	glColor3f(1.0, 0.0, 0.0);
	glLoadIdentity();

	// Viewing transformation.
	gluLookAt(eyeX, 10, eyeZ, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

	for (size_t i = 0; i < kureSayisi; i++)
	{
		glPushMatrix();
		glTranslatef(kure_merkezler[i].x, kure_merkezler[i].y,
kure_merkezler[i].z);
		glutWireSphere(1, 10, 10);
		glPopMatrix();
	}
	glutSwapBuffers();


}

// Initialization routine.
void setup(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
	for (size_t i = 0; i < kureSayisi; i++)
	{
		vec3 merkez;
		merkez.x = randomGen();
		merkez.y = randomGen();
		merkez.z = randomGen();
		kure_merkezler.push_back(merkez);
	}
	m = OrtaNokta(kure_merkezler);
	cout << "Orta noktanin(m) koordinatlari:" << endl;
	cout << "mx:" <<m.x<< endl;
	cout << "my:" << m.y << endl;
	cout << "mz:" << m.z << endl;
	CovarianceMatrix = CovaryansMatrisiOlustur(kure_merkezler,m);
	cout << "Covariance Matris katsayilari:" << endl;
	cout << "C11: " << CovarianceMatrix[0].x << endl;
	cout << "C12: " << CovarianceMatrix[1].x << endl;
	cout << "C13: " << CovarianceMatrix[2].x << endl;
	cout << "C21: " << CovarianceMatrix[0].y << endl;
	cout << "C22: " << CovarianceMatrix[1].y << endl;
	cout << "C23: " << CovarianceMatrix[2].y << endl;
	cout << "C31: " << CovarianceMatrix[0].z << endl;
	cout << "C32: " << CovarianceMatrix[1].z << endl;
	cout << "C33: " << CovarianceMatrix[2].z << endl;



	mat3x3 R= mat3x3(1.0f);
	CalculateEigensystem(CovarianceMatrix, lambda, R);
	cout << "Ozdegerler:";
	cout << "Lamda1:"<<lambda[0] << endl;
	cout << "Lamda2:"<< lambda[1] << endl;
	cout << "Lamda3:" << lambda[2] << endl;
	cout << "Ozvektorterler" << endl;
	for (size_t i = 0; i < 3; i++)
```

```
	{
		cout << "Ozvektor" <<i<< endl;
		cout << "x:" << R[i].x<<endl;
		cout << "y:" << R[i].y << endl;
		cout << "z:" << R[i].z << endl;
	}
	e1=glm::normalize(R[0]);
	e2 = glm::normalize(R[1]);
	e3 = glm::normalize(R[3]);
}

// OpenGL window reshape routine.
void resize(int w, int h)
{
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glFrustum(-10, 10, -10, 10, 1, 150.0);

	glMatrixMode(GL_MODELVIEW);
}

// Keyboard input processing routine.
void keyInput(unsigned char key, int x, int y)
{
	switch (key)
	{
	case 27:
		exit(0);
		break;
	case 'w':
		eyeZ--;
		glutPostRedisplay();
		break;
	case 'a':
		eyeX--;
		glutPostRedisplay();
		break;
	case 'd':
		eyeX++;
		glutPostRedisplay();
		break;
	case 's':
		eyeZ++;
		glutPostRedisplay();
		break;
	default:
		break;
	}
}

// Main routine.
int main(int argc, char** argv)
{
	glutInit(&argc, argv);

	glutInitContextVersion(4, 3);
	glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);
```

```
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("Odev4.cpp");
        glutDisplayFunc(drawScene);
        glutReshapeFunc(resize);
        glutKeyboardFunc(keyInput);

        glewExperimental = GL_TRUE;
        glewInit();

        setup();

        glutMainLoop();
}
```

Ekran görüntüsü:





Ekler:

Odev4.cpp , BCA611 Odev4.exe