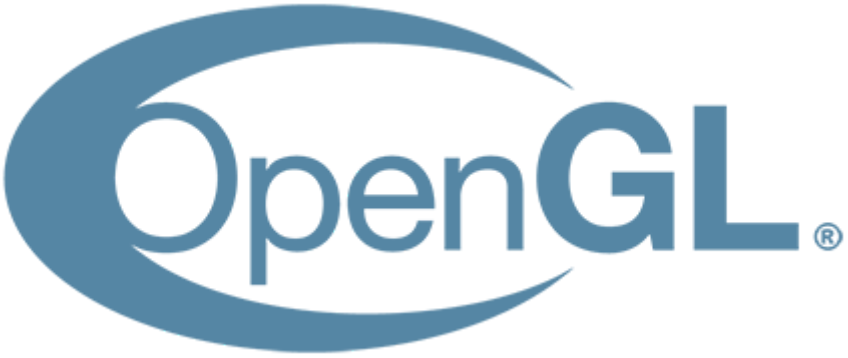

İstanbul Üniversitesi-Cerrahpaşa

Bilgisayar Grafikleri Final Projesi Raporu

Ad-Soyad: Mustafa Çirci
Üniversite: İstanbul Üniversitesi-Cerrahpaşa
Numara: 1306210018

Tema

Bu projenin ana teması satranç olarak belirledim. Küçük yaşlardan itibaren satranca büyük bir ilgi duydum ve bu oyun, hem stratejik düşünmemi hem de keyifli vakit geçirmemi sağladı. Stratejik düşünme, öngörü ve sabır gibi beceriler kazandırması, satrancı benim için sadece bir oyun değil, aynı zamanda öğretici bir uğraş haline getirdi. Bu nedenle, proje konumu belirlerken benim için anlamlı olan ve teknik açıdan da üzerinde çalışmaktan keyif alacağım bir tema olarak satrancı seçtim.

Projenin animasyon sahnesinde ise satranç tarihinin en kısa ve şaşırtıcı matlarından biri olan “Çoban Matı”nı canlandırdım. Bu taktik, ilk öğrendiğim mat yöntemlerinden biriydi ve dört hamlede oyunu bitirme fikri beni çok etkilemişti. Animasyonda, matın nasıl yapıldığını zamana bağlı hareketlerle canlandırarak bu kısa oyunun stratejik etkisini görsel olarak yansıttım. Satranç sevgimi teknik bilgilerle birleştirme fırsatı sunan bu proje, hem kendimi ifade edebileceğim yaratıcı bir alan oluşturdu hem de OpenGL konusundaki becerilerimi geliştirmeme katkı sağladı.

Kullanılan Teknik Bileşenler

Assimp (Open Asset Import Library)

Assimp, farklı 3D model formatlarını uygulamaya kolayca entegre etmemi sağlayan güçlü bir model yükleme kütüphanesidir. Projede .obj ve .fbx gibi yaygın dosya türlerinde hazırlanmış satranç taşlarını ve satranç tahtasını sahneye aktarmak için kullanılmıştır. Bu sayede Blender gibi 3D modelleme yazılımlarında oluşturulan karmaşık geometriler doğrudan OpenGL ortamında kullanılabilir hale geldi. Assimp, model verilerini (vertex, normal, texture koordinatları, vs.) organize ederek OpenGL'e uygun hale getirir ve bu süreçte büyük kolaylık sağlar.

stb_image

stb_image, görsel dosyaları (örn. .png, .jpg gibi formatlar) OpenGL ile kullanılabilir hale getiren, küçük ve etkili bir resim yükleme kütüphanesidir. Bu proje kapsamında, satranç taşlarının ve tahtanın yüzeylerine gerçekçi görünümler kazandırmak için texture dosyaları kullanılmıştır. stb_image sayesinde bu görseller belleğe alınarak shader programlarında kullanılmak üzere OpenGL texture nesnelerine dönüştürülmüştür. Bu da sahnenin daha doğal ve estetik görünmesine katkı sağlamıştır.

GLAD ve GLFW

GLAD, OpenGL fonksiyonlarına erişim sağlayan bir yükleyicidir. OpenGL, platformdan bağımsız bir API olduğu için, çalışma zamanında hangi fonksiyonların kullanılabilir olduğunu belirlemek gerekir. GLAD, bu süreci otomatikleştirerek geliştiricinin manuel olarak fonksiyon işaretçileriyle uğraşmasını engeller.
GLFW ise pencere oluşturma, kullanıcı girişlerini (klavye, fare), zamanlayıcı ve OpenGL bağlamı gibi düşük seviye işlemleri yöneten hafif bir kütüphanedir. Bu projede, uygulamanın penceresi GLFW ile oluşturulmuş ve kullanıcının etkileşimleri (örneğin kamera hareketleri veya mod değişimleri) bu kütüphane aracılığıyla yönetilmiştir.

GLAD ve GLFW birlikte kullanılarak uygulamanın temel yapısı oluşturulmuş ve platform bağımsız, kullanıcıyla etkileşimli bir 3D ortam hazırlanmıştır.

GLM (OpenGL Mathematics Library)

GLM, OpenGL için özel olarak tasarlanmış bir matematik kütüphanesidir ve C++ ile yazılmıştır. Bu projede GLM, tüm vektör ve matris işlemleri için kullanılmıştır.

- Modelleme sırasında objelerin sahneye konumlandırılması, döndürülmesi ve ölçeklendirilmesi gibi dönüşüm işlemleri
- Kamera hareketlerinin ve görüş açısının hesaplanması
- Perspektif ve ortografik projeksiyon matrislerinin oluşturulması

gibi temel işlemler, GLM aracılığıyla gerçekleştirilmiştir. GLM, OpenGL'in sütun öncelikli matris sistemine doğrudan uyum sağladığı için karmaşık dönüşüm işlemlerini sade ve okunabilir bir yapıda uygulamama olanak tanımıştır. Bu sayede animasyonların zamanla uyumlu ilerlemesi ve doğru kamera görünümünün elde edilmesi sağlanmıştır.

Sahne Mimarisi

Bu projede 3D sahne düzeni ve animasyon mimarisi, OpenGL'in temel bileşenleri kullanılarak kurgulanmıştır. Her bir satranç taşları sahnede ayrı bir nesne (model) olarak tanımlanmış, bu nesneler çeşitli dönüşüm matrisleri (model, view, projection) ile sahneye yerleştirilmiştir. Animasyon mantığı ise zaman tabanlı pozisyon güncellemeleriyle kurgulanmıştır.

1. Objelerin Temsili ve Dönüşüm Matrisleri

Her 3D model için sahneye çizilmeden önce glm::mat4 tipinde bir model matrisi oluşturulmuş ve bu matrise sırasıyla:

- translate (taşınacağı pozisyon)
- rotate (dönüş)
- scale (ölçek)

işlemleri uygulanmıştır. Bu matris daha sonra shader'a gönderilerek OpenGL'in çizim pipeline'ında kullanılmaktadır.

```
glm::mat4 model4 = glm::mat4(1.0f);
model4 = glm::translate(model4, glm::vec3(chessPosX + whiteBishopAnim.offsetX, chessPosY, chessPosZ + whiteBishopAnim.offsetZ));
model4 = glm::rotate(model4, glm::radians(chessRotate), glm::vec3(0.0f, 1.0f, 0.0f));
model4 = glm::scale(model4, glm::vec3(chessScale));
shader.setMat4("model", model4);
whiteBishop.Draw(shader);
```

Yukarıdaki örnek, beyaz filin sahneye yerleştirilmesini ve animasyonla pozisyonunun güncellenmesini sağlar.

2. Animasyon Sistemi

Her taş için özel bir Anim yapısı tanımlanarak, taşın sahne içindeki hareketi kontrol altına alınmıştır. Bu yapı; X ve Z eksenlerindeki ofsetleri, maksimum hareket mesafesini, yön bilgilerini ve aktiflik durumlarını içermektedir.

```
struct Anim {
    float offsetZ = 0.0f;
    float offsetX = 0.0f;
    float speed = 10.0f;
    float maxZ = 150.0f;
    float maxX = 150.0f;
    int dirZ = 1;
    int dirX = 0;
    bool active = false;
    bool done = false;
    bool done2 = false;

    void update(float dt) {
        if (active) {
            if (abs(offsetZ) < maxZ)
                offsetZ += speed * dt * dirZ;
            else
                offsetZ = dirZ * maxZ;

            if (abs(offsetX) < maxX)
                offsetX += speed * dt * dirX;
            else
                offsetX = dirX * maxX;

            if (abs(offsetZ) >= maxZ && abs(offsetX) >= maxX) {
                active = false;
                if (done == true)
                    done2 = true;
                else
                    done = true;
            }
        }
    }
};
```

Her update() fonksiyonu, deltaTime (kareler arası süre) değerine göre çağrılarak animasyonların zamanla akıcı şekilde ilerlemesini sağlar.

```
Anim whitePawnAnim = { 0.0f, 0.0f, chessSpeed * chessScale, 72.0f* chessScale, 0.0f* chessScale, 1, 0, false, false };
Anim whiteQueenAnim = { 0.0f, 0.0f, chessSpeed * chessScale, 72.0f* chessScale, 72.0f* chessScale, 1, -1, false, false };
Anim whiteBishopAnim = { 0.0f, 0.0f, chessSpeed * chessScale, 108.0f* chessScale, 108.0f* chessScale, 1, 1, false, false };
Anim blackPawnAnim = { 0.0f, 0.0f, chessSpeed * chessScale, 72.0f* chessScale, 0.0f* chessScale, -1, 0, false, false };
Anim blackPawn2Anim = { 0.0f, 0.0f, chessSpeed * chessScale, 72.0f* chessScale, 0.0f* chessScale, -1, 0, false, false };
Anim blackPawn3Anim = { 0.0f, 0.0f, 200.0f * chessScale, 72.0f* chessScale, 108.0f* chessScale, -1, -1, false, false };
Anim blackKnightAnim = { 0.0f, 0.0f, chessSpeed * chessScale, 72.0f* chessScale, 36.0f* chessScale, -1, 1, false, false };
```

3. Sıralı Hareket Mantığı (Sequential Animation)

sequentialStarted değişkeni true olduğunda, taşların hareket sırası birbirine bağlı olarak tetiklenmektedir. Örneğin bir taş hareketini tamamladığında, bir sonraki taşın active durumu true yapılır:

```
if (sequentialStarted) {
    if (whitePawnAnim.done == true)
        blackPawnAnim.active = true;
    if (blackPawnAnim.done == true)
        whiteQueenAnim.active = true;
    if (whiteQueenAnim.done == true)
        blackKnightAnim.active = true;
    if (blackKnightAnim.done == true)
        whiteBishopAnim.active = true;
    if (whiteBishopAnim.done == true)
        blackPawn2Anim.active = true;
    if (blackPawn2Anim.done == true){
        whiteQueenAnim.maxZ = 216.0f * chessScale;
        whiteQueenAnim.maxX = -72.0f * chessScale;
        whiteQueenAnim.dirX = 1;
        whiteQueenAnim.speed = 72.0f * chessScale;
        whiteQueenAnim.active = true;
    }
    if (blackPawn2Anim.done == true)
        blackPawn3Anim.active = true;
    if (blackPawn3Anim.done == true)
        sequentialStarted = false;
}
```

Bu yapı sayesinde “Çoban Matı” animasyonu hamle hamle oynatılır.

4. Shader Yapısı

Projede aydınlatma ve gölgelendirme işlemleri için basit bir Phong aydınlatma modeli kullanılan bir vertex ve bir fragment shader yazılmıştır.

Vertex Shader:

Her bir vertex’in dünya uzayındaki pozisyonu (FragPos) ve normal vektörü (Normal) hesaplanır:

```
FragPos = vec3(model * vec4(aPos, 1.0));
Normal = mat3(transpose(inverse(model))) * aNormal;
gl_Position = projection * view * vec4(FragPos, 1.0);
```

Fragment Shader:

Işık ve kamera konumuna göre ambient, diffuse ve specular bileşenleriyle birlikte renk hesaplanır. Ayrıca objenin dokusu veya düz rengi kullanılabilir:

```
vec3 finalColor = (ambient + diffuse + specular);
vec3 baseColor = useTexture == 1 ? texture(texture_diffuse1, TexCoords).rgb : materialColor;
FragColor = vec4(finalColor * baseColor, 1.0);
```

5. Sahne Bileşenleri

Sahnedeki ana objeler şunlardır:
Satranç Tahtası: Statik bir zemin objesi
Taşlar: Her biri bağımsız model olarak animasyonlara dahil edilmiştir
Işık Küresi (Light Sphere): Görsel ışık kaynağını temsil eder
Oda (Room): Sahneye hacim kazandıran çevresel model

Karşılaşılan Problemler ve Çözümler

Projenin ilk aşamalarında, Assimp kütüphanesini OpenGL ile entegre etmekte çeşitli zorluklar yaşadım. Model yükleme sürecinde sık sık segmentation fault ve bellek erişim hatalarıyla karşılaştım. Bu hatalar çoğunlukla kütüphaneyi doğru şekilde başlatmamam, model dosyalarının yollarını yanlış belirtmem ya da Assimp ile yüklediğim model verilerini OpenGL tamponlarına aktarırken yapılan hatalardan kaynaklanıyordu.

Assimp'in resmi dökümantasyonu, temel bilgiler açısından yeterli olsa da bazı ileri düzey kullanım senaryolarında oldukça sınırlı kalabiliyor. Özellikle karmaşık model dosyalarını işlerken, normal vektörleri, doku koordinatlarını ya da malzeme özelliklerini doğru bir şekilde ayıklamakta zorlandım. Bu süreçte, bazı veri yapılarını anlamak ve doğru şekilde OpenGL'e aktarmak için birçok farklı örnek kod inceledim ve kendi uygulamamı deneme-yanılma yöntemiyle geliştirdim.

Karşılaştığım bu teknik problemler, başlangıçta beni yavaşlatsa da her birini çözmeye çalışırken Assimp'in veri yapısını ve OpenGL ile ilişkisini daha derinlemesine öğrenme fırsatı buldum. Sonuç olarak, model yükleme sürecini daha stabil ve esnek hale getirdim. Bu süreç, hata ayıklama becerilerimi ve dış kütüphanelerle çalışma pratiğimi ciddi şekilde geliştirdi.