Bilgisayar Grafikleri
Proje 2 - Rapor

HAZIRLAYAN

MUHAMMET TALHA ODABAŞI

İçerik

Hikaye Tanıtımı

Basketbolla tanışmam çok küçük yaşlarda, mahalle aralarında oynadığımız sokak maçlarına dayanıyor. O zamanlar elimde bir top, karşımdaki duvarı pota yapıp saatlerce şut çalışırdım. Topu her elime aldığımda kendimi bir NBA yıldızı gibi hissederdim. Zamanla bu sadece bir oyun olmaktan çıktı, bir tutkuya dönüştü. NBA maçlarını sabahlara kadar izler, oyuncuların hareketlerini tekrar tekrar izleyip taklit etmeye çalışırdım. Bu tutkuyla büyüdüm. Bu yüzden bu projeyi yapma fikri aklıma geldiğinde hiç tereddüt etmedim. Hem çocukluk hayalimi bir şekilde yaşatmak hem de basketbola olan sevgimi teknolojiyle birleştirmek istedim.

Kullanılan Teknolojiler

Grafikler ve Girdi (Graphics & Input)
Bu bölüm, sahnedeki grafiklerin çizilmesi ve kullanıcının klavye, fare gibi giriş araçlarıyla etkileşime geçmesini sağlar.

- **OpenGL (Rendering API):** Ekrana 2D veya 3D grafik çizmek için kullanılan güçlü bir grafik programlama arayüzüdür. Oyunlardaki modellerin çizilmesi, ışıklandırma ve efektlerin uygulanması gibi işlemler bu API ile yapılır.
- **GLFW (Pencere ve Girdi Yönetimi):** OpenGL için pencere oluşturmaya, ekran çözünürlüğünü ayarlamaya ve klavye/fare gibi kullanıcı girişlerini algılamaya yarayan bir kütüphanedir. Yani hem ekranı açar hem de kullanıcıyla etkileşimi sağlar.
- **GLAD (OpenGL Fonksiyon Yükleyici):** OpenGL'in sağladığı fonksiyonlara erişmek için gerekli bir yardımcı kütüphanedir. OpenGL fonksiyonları sistemden sisteme farklılık gösterebildiği için GLAD bu fonksiyonları uygun şekilde yükler ve kullanılabilir hale getirir.

Matematik ve Geometri (Math & Geometry)
3D ortamda nesnelerin döndürülmesi, taşınması, büyütülüp küçültülmesi gibi işlemler bu kısımda yapılır.

- **GLM (OpenGL Mathematics):** Vektör ve matris işlemleri yapmak için kullanılan matematik kütüphanesidir. Örneğin bir nesneyi döndürmek, sahnedeki konumunu değiştirmek veya kamerayı hareket ettirmek için glm::vec3 (vektör), glm::mat4 (4x4 matris) gibi veri tipleriyle işlem yapılır.

Model ve Doku Yükleme (Model & Texture Loading)
3D modellerin ve dokuların (kaplamaların) dosyalardan yüklenerek sahnede görünmesini sağlar.

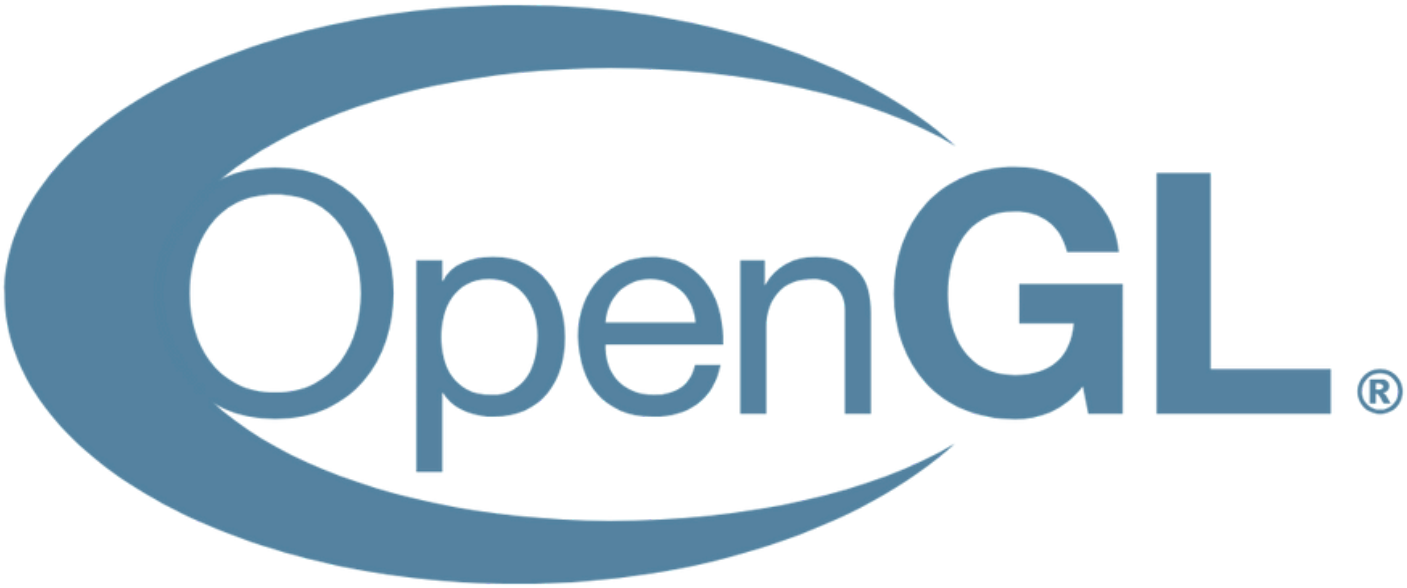
- **Özel OBJ Yükleyici (Custom OBJ Loader):** .obj uzantılı 3D model dosyalarını, herhangi bir hazır kütüphane (örneğin Assimp) kullanmadan, kendinizin yazdığı bir kodla ayrıştırarak sahneye yerleştirmenizi sağlar. Bu dosyalar genelde nesnenin şekli, yüzey normalleri ve doku koordinatlarını içerir.
- **Özel DDS Yükleyici (Custom DDS Loader):** .dds uzantılı doku (texture) dosyalarını yükleyen özel bir sistemdir. Bu dosyalar özellikle oyunlarda kullanılan sıkıştırılmış doku formatlarıdır. S3TC sıkıştırma algoritmaları (DXT1, DXT3, DXT5) ile dosya boyutu düşürülürken görsel kalite korunur.

Animasyon
3D modellerin zaman içinde hareket etmesini sağlayan sistemdir.

- **Anahtar Kare (Keyframe) Tabanlı Animasyon Sistemi:** Belirli zamanlarda tanımlanan pozlar (keyframe) arasında geçiş yapılarak hareket elde edilir. Örneğin bir topun zıplaması, yürüme hareketi gibi animasyonlar AnimationObj sınıfı ve .anm uzantılı dosyalar üzerinden tanımlanır ve yürütülür.

Gölgelendiriciler (Shaders)
Görüntülerin ekran kartı tarafından nasıl çizileceğini tanımlar. Görsel efektler burada işlenir.

- **GLSL Gölgelendiriciler (GLSL Shaders):** OpenGL Shading Language (GLSL) kullanılarak yazılan küçük programlardır. vertexshader dosyası, modelin ekrandaki konumunu hesaplar; fragmentshader ise pikselin hangi renkte görüneceğini belirler. Bu dosyalar LoadShaders() fonksiyonu ile yüklenip derlenerek kullanılır.



Mimari Detaylar

1. Sahne Mimarisi (Scene Architecture)

Objeler (Obj sınıfı ile temsil edilir)

- Objeler .obj formatından manuel olarak yüklenir (loadOBJ()).
- Her obje:
  - 3D modeli (.obj)
  - Doku (texture .dds)
  - Pozisyon (obj.pos)
  - Rotasyon (obj.rot)
  - Opsiyonel animasyon (obj.animation)

load_all() fonksiyonu:

- 10 obje yüklenir: saha, pota, top, koltuk, skybox vb.
- Örnek:

```
objList[i++].set("proj/models/ball.obj", "proj/textures/ball.dds");
objList[i-1].animation = AnimationObj("proj/animations/ball.anm");
```

2. Dönüşümler (Transformations)

Kamera:

- Cam sınıfı içinde:
  - pos, front, up, right, yaw, pitch
- updateMVP():
  - glm::lookAt + glm::perspective
  - Kamera matrisini (MVP) oluşturur.

Obje Pozisyonu ve Animasyon:

- Obj::update() → animasyon varsa pozisyon hesaplanır.
- AnimationObj::getPos(int frame) → iki keyframe arasında lineer interpolasyon.

3. Shader Mimarisi

GLSL Shader'lar:

- Vertex ve Fragment shader'lar LoadShaders() ile yüklenir:

```
LoadShaders("proj/shaders/TransformVertexShader.vertexshader",
            "proj/shaders/TextureFragmentShader.fragmentshader");
```

Uniformlar:

- MVP matrisi shader'a gönderilir:

```
glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
```

- Texture sampler (myTextureSampler) da aktarılır.

4. Animasyon Sistemi (Keyframe Tabanlı)

AnimationObj:

- .anm dosyası okunur → keyframe listesi oluşturulur
- getPos(int frame):
  - Her kare için pozisyon hesaplanır
  - Loop desteği var

5. Render Döngüsü

```
glfwSetKeyCallback(window, Fkey_callback);
while (!glfwWindowShouldClose(window))
{
  if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
    glfwSetWindowShouldClose(window, true);

  // Measure speed
  double currentTime = glfwGetTime();
  nbFrames++;
  if (currentTime - lastTime >= 1.0)
  {
    printf("%d FPS : %f ms/frame\n", nbFrames, 1000.0 / double(nbFrames));
    nbFrames = 0;
    lastTime += 1.0;
  }

  for (int i = 0; i < object_count; i++)
    objList[i].update();


  float currentFrame = glfwGetTime();
  deltaTime = currentFrame - lastFrame;
  lastFrame = currentFrame;

  processInput(window);
  cam.update(objList);
  cam.updateMVP();
```

```
  // also clear the depth buffer now!
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  // Use our shader
  glUseProgram(programID);

  for (int i = 0; i < object_count; i++)
    objList[i].draw(cam.MVP, MatrixID, TextureID);

  /* glfw: swap buffers and poll IO events (keys pressed/released,
   *         mouse moved etc.)
   *         ---------------------------------------------------------*/
  glfwSwapBuffers(window);
  glfwPollEvents();
}
```

6. Detaylı Açıklama

Bu C++ projesi, gerçek zamanlı 3B grafikler üretmek için modern OpenGL kullanılarak geliştirilmiştir. Projede kullanılan temel teknolojiler arasında GLFW (pencere ve kullanıcı giriş yönetimi), GLAD (OpenGL işlev yükleyici), ve GLM (vektör/matris hesaplamaları için matematik kütüphanesi) yer almaktadır. Grafik sahnesi, .obj dosyalarından yüklenen modellerden ve .dds formatında sıkıştırılmış dokulardan oluşmaktadır. Dosyada herhangi bir harici 3B model yükleyici (örneğin Assimp) kullanılmadan, doğrudan dosya okuma ve satır-parselleme ile özel bir .obj dosya ayrıştırıcı geliştirilmiştir. Bu sayede model verileri (köşe noktaları, yüzey normalleri, UV koordinatları) manüel olarak okunur ve belleğe alınır.

Sahne mimarisi, Obj sınıflarının bir dizi olarak tutulması ile yönetilmektedir. Her obje; pozisyon, rotasyon, model verisi ve opsiyonel olarak AnimationObj sınıfı ile tanımlanan bir animasyon taşır. Animasyon sistemi, temel olarak keyframe (anahtar kare) interpolasyonu ile çalışır. .anm uzantılı dosyalardan okunan Keyframe yapıları, sahnede top gibi nesnelerin zamanla pozisyonunun değişmesini sağlar. AnimationObj sınıfı, belirtilen kareler arasında lineer pozisyon interpolasyonu yapar ve döngüsel (loop) animasyon desteği de sunar.

Kamera sistemi Cam sınıfı ile temsil edilir ve hem serbest dolaşma (WASD + fare ile yön kontrolü) hem de FPS (bir nesneyi takip eden) kamera modlarını destekler. MVP (Model-View-Projection) matrisi her çerçevede hesaplanarak shader'a aktarılır. Kamera yaw ve pitch değerleriyle yönlendirilir ve bu değerler fare hareketleriyle güncellenir. FPS moduna geçildiğinde, kamera sahnedeki top nesnesine bağlanır ve yalnızca sağ-sol fare hareketi ile yönlendirme yapılabilir.

Shader mimarisi, dış dosyalarda tanımlı vertex ve fragment shader'ların LoadShaders() fonksiyonu aracılığıyla OpenGL'e yüklenmesi ile kurulur. Shader dosyaları içerisinde MVP matrisi ve doku sampler'ları gibi uniform değişkenler kullanılarak sahneye dokulu modeller çizdirilir. Gerçek zamanlı FPS ölçümü ve temel olay döngüsü de ana main() fonksiyonunda gerçekleştirilir.

Sonuç olarak, proje temel bir oyun motoru iskeleti sunar: özel yazılmış model ve doku yükleyiciler, keyframe tabanlı animasyon sistemi, çoklu nesne yönetimi, shader mimarisi ve kullanıcı kontrollü kamera sistemiyle oldukça modüler ve genişletilebilir bir mimariye sahiptir.

Karşılaşılan Problemler

Proje sürecinde çeşitli teknik zorluklarla karşılaşıldı ve her biri farklı yaklaşımlarla çözüldü. İlk olarak, 3B model yükleme aşamasında Assimp kullanımı denendi ancak bu kütüphane çok karmaşık bir yapı sunduğu ve birçok hata ile karşılaşıldığı için bu yöntem terk edildi. Bunun yerine, internette araştırma yapılarak bulunmuş olan özel olarak yazılmış bir OBJ dosya ayrıştırıcı (custom OBJ loader) tercih edildi ve bu sayede modeller başarıyla yüklenebildi. Derleme ortamında yaşanan OpenGL sürüm uyumsuzlukları ve kütüphane bağımlılıklarını ortadan kaldırmak için CMake kullanılarak yapılandırma dosyası oluşturuldu. Ek olarak, kullanıcı deneyimini kolaylaştırmak adına run.sh betiği oluşturularak tek komutla projeyi derleyip çalıştırmak mümkün hale getirildi.

Animasyon sistemi oluşturulurken de çeşitli zorluklar yaşandı. Özellikle keyframe bazlı animasyonun dosya yapısı ve interpolasyon hesaplamaları karmaşıktı. Bu problem, internette dolaşırken karşılaşılan ve hazır olarak sunulan bir .anm formatının projeye adapte edilmesiyle aşıldı. Daha önce incelenen örnek projeler genellikle GLEW kullanıyordu, ancak bu kütüphane ile yapılan kamera kontrolleri oldukça sınırlı ve yetersizdi. Bu nedenle, OpenGL fonksiyonlarını yüklemek için GLAD tercih edildi ve FPS tarzı bir kamera sistemi manuel olarak geliştirildi. Kamera kontrolü, F tuşuna basıldığında sahnedeki topun bakış açısına geçerek kullanıcıya farklı bir perspektif sunacak şekilde özelleştirildi. Tüm bu çözümler projenin hem işlevsel hem de kullanıcı dostu olmasını sağladı.