Unity ile 3D Oyun Geliştirme – Ders Notu

3D Oyun Stüdyosu Dersi | Sinop Üniversitesi MYO | Öğr. Gör. İsmail Umut DURAN

1. 3D Grafiklerin Temel Kavramları, Sahne ve Transform

1.1 Temel kavramlar

- **Vertex:** 3D uzayda bir köşe noktasıdır (x, y, z). Birkaç vertex bir edge (kenar) oluşturur.
- **Polygon / Face:** Genellikle üçgenlerden oluşan yüzeydir. GPU her şeyi üçgen olarak çizer.
- **Mesh:** Vertex, edge ve yüzeylerin birleşimiyle oluşan 3D nesne geometrisidir. Poligon sayısı arttıkça görüntü detaylanır, performans düşer.
- **UV Haritası:** 3D yüzeyin 2D doku (texture) üzerindeki karşılığıdır.
- **Koordinat sistemi:** Unity'de X sağ, Y yukarı, Z ileri yöndedir (sol elli sistem). Birim 1 = 1 metre kabul edilir.
- **Render hattı (pipeline):** Mesh → vertex işleme → rasterizasyon → piksel (fragment) gölgelendirme → ekran. Unity'de Built-in, URP ve HDRP olmak üzere üç render hattı vardır.

1.2 Unity'de 3D sahne oluşturma ve yönetme

- **Proje oluşturma:** Unity Hub > New Project > 3D (Core) veya 3D (URP) şablonu seçilir.
- **Arayüz:** Hierarchy (sahnedeki nesneler), Scene (düzenleme alanı), Game (oyuncunun gördüğü ekran), Inspector (seçili nesnenin bileşenleri), Project (dosyalar), Console (hata ve log mesajları).
- **GameObject:** Sahnedeki her şeyin temel nesnesidir; kendi başına bir şey yapmaz, işlevini Component'lerden alır.
- **Hazır 3D nesneler:** GameObject > 3D Object menüsünden Cube, Sphere, Capsule, Cylinder, Plane, Quad eklenir.
- **Prefab:** Bir nesnenin yeniden kullanılabilir şablonudur. Hierarchy'den Project'e sürüklenerek oluşturulur; değişiklik tüm kopyalara yansıtılabilir.
- **Tag ve Layer:** Nesneleri gruplamak ve script içinde ayırt etmek için kullanılır (ör. "Player", "Dusman").
- **Sahne yönetimi:** Sahneler File > Save ile .unity olarak kaydedilir; Build Settings'e eklenen sahneler SceneManager.LoadScene ile yüklenir.

1.3 Transform ve 3D uzayda konumlandırma

Özellik	Açıklama
Position	Nesnenin X, Y, Z konumu (Vector3).
Rotation	Derece cinsinden dönüş (Euler açıları). Dahili olarak Quaternion ile tutulur.
Scale	Her eksendeki ölçek; (1,1,1) orijinal boyuttur.

- **Araçlar:** Q (el), W (taşı), E (döndür), R (ölçekle), T (rect).
- **Local / Global:** Parent-child ilişkisinde çocuğun Inspector'daki değerleri ebeveyne göre (local) yazılır. transform.position dünyaya göre, transform.localPosition ebeveyne göredir.
- Pivot / Center ve Local / Global düğmeleri: Araç çubuğundan değiştirilir; döndürmenin nokta ve eksen referansını belirler.
- **Snap:** Ctrl basılı tutularak sabit adımlarla taşıma yapılır.

```
transform.position = new Vector3(0, 1, 5);
transform.Translate(0, 0, 2);            // yerel eksende ilerle
transform.Rotate(0, 90, 0);              // Y ekseninde 90° döndür
transform.localScale = new Vector3(2, 2, 2);
transform.LookAt(hedef);                 // hedefe bak
```

2. 3D Modellerin Import Edilmesi ve Mesh Renderer

- **Formatlar:** .fbx (önerilen; mesh, materyal ve animasyon taşır), .obj (sadece geometri), .blend, .dae, .3ds. Dosya Assets klasörüne sürükle-bırak ile eklenir.
- **Import ayarları (Model sekmesi):** Scale Factor (ölçek), Mesh Compression, Read/Write, Generate Colliders, Normals ve Tangents.
- **Rig sekmesi:** Animation Type seçilir: None, Legacy, Generic, Humanoid. İnsan benzeri karakterlerde Humanoid, animasyonların farklı modellerde kullanılmasını sağlar.
- **Animation sekmesi:** Modelle gelen animasyon klipleri buradan ayarlanır (Loop Time vb.).
- **Materials sekmesi:** Modelin materyalleri çıkarılır (Extract) ve Unity'de düzenlenir.
- **Sık sorunlar:** Model çok büyük/küçük gelirse Scale Factor ayarlanır; dokular eksikse Materials sekmesinden dokular yeniden bağlanır; yüzeyler ters görünüyorsa normaller kontrol edilir.

Mesh Filter ve Mesh Renderer

- **Mesh Filter:** Nesnenin hangi mesh'i kullandığını tutar.
- **Mesh Renderer:** Mesh'in ekrana çizilmesini sağlar. Başlıca ayarlar: Materials (liste), Cast Shadows (gölge düşürme), Receive Shadows (gölge alma), Light Probes, Reflection Probes.
- **Skinned Mesh Renderer:** Kemik (bone) tabanlı animasyonlu karakterlerde Mesh Renderer yerine kullanılır.

İpucu: Mobil ve düşük donanımlı hedeflerde poligon sayısı düşük tutulmalı, gereksiz Read/Write seçeneği kapatılmalıdır.

3. Materyal ve Shader'ların Temelleri

- **Shader:** GPU üzerinde çalışan, yüzeyin ışığa nasıl tepki vereceğini hesaplayan programdır.
- **Materyal:** Bir shader'ı seçip ayarlarını (renk, doku, parlaklık vb.) saklayan varlıktır. Aynı materyal birçok nesnede kullanılabilir.
- **Oluşturma:** Project > Sağ tık > Create > Material. Nesneye uygulamak için materyal nesnenin üzerine sürüklenir veya Mesh Renderer > Materials alanına atanır.
- **Yaygın shader'lar:** Lit / Standard (fiziksel tabanlı), Unlit (ışıktan etkilenmez), Particles, UI, Skybox, Shader Graph ile oluşturulan özel shader'lar.

Özellik	Görevi
Albedo / Base Map	Yüzeyin temel rengi veya doku görseli.
Metallic	Yüzeyin metalik görünme oranı (0–1).
Smoothness	Parlaklık / yansıma keskinliği (0 mat, 1 ayna gibi).
Normal Map	Poligon eklemeden yüzeyde girinti-çıkıntı hissi verir.
Emission	Yüzeyin kendi ışığını yaymasını (parlamasını) sağlar.
Tiling / Offset	Dokunun tekrar sayısı ve kaydırması.

- **Rendering Mode:** Opaque (opak), Cutout, Fade, Transparent. Cam gibi saydam yüzeylerde Transparent seçilir.
- **Script ile materyal değiştirme:**

```
Renderer r = GetComponent<Renderer>();
r.material.color = Color.red;
r.material = yeniMateryal;
```

4. Işıklandırma: Türler, Gölgeler ve Işık Haritaları

Işık türü	Özellik	Kullanım
Directional	Sonsuz uzaklıktan paralel ışık; yönü önemlidir, konumu önemsizdir.	Güneş, ay ışığı. Sahnede genelde bir tane bulunur.
Point	Bir noktadan her yöne yayılır; Range ile menzil ayarlanır.	Lamba, ateş, patlama.
Spot	Koni şeklinde yayılır; Spot Angle ve Range vardır.	El feneri, sahne ışığı, araba farı.
Area	Dikdörtgen/disk yüzeyden yayılır; yalnızca Baked modda.	Pencere, neon, yumuşak aydınlatma.

Işık ayarları

- **Color, Intensity, Range:** Işığın rengi, şiddeti ve menzili.
- **Mode:** Realtime (anlık hesaplanır, maliyetli), Baked (önceden hesaplanır, hızlı), Mixed (ikisinin karışımı).
- **Indirect Multiplier:** Yansıyan (dolaylı) ışığın şiddeti.
- **Environment:** Window > Rendering > Lighting > Environment altında Skybox Material ve ortam ışığı (Ambient) ayarlanır.

Gölgelendirme

- **Shadow Type:** No Shadows, Hard Shadows (keskin, hızlı), Soft Shadows (yumuşak, daha maliyetli).
- Gölge için nesnenin Mesh Renderer'ında Cast Shadows, zemin gibi nesnede Receive Shadows açık olmalıdır.
- Kalite ayarları (Edit > Project Settings > Quality): Shadow Distance, Shadow Resolution, Cascades.

Işık haritaları (Lightmap) ve Baked Lighting

1. Hareket etmeyen nesneler (zemin, duvar) Inspector'da Static (Contribute GI) olarak işaretlenir.
2. Işığın Mode değeri Baked veya Mixed yapılır.
3. Window > Rendering > Lighting > Generate Lighting düğmesine basılır.
4. Hesaplanan ışık ve gölgeler doku (lightmap) olarak sahneye işlenir.

- **Avantaj:** Çalışma anında ışık hesaplanmadığı için performans artar, daha gerçekçi dolaylı ışık elde edilir.
- **Dezavantaj:** Hareketli nesneler lightmap'e dahil olmaz; hesaplama süresi uzayabilir. Hareketli nesneler için Light Probe kullanılır.
- **Reflection Probe:** Çevre yansımalarını yakalayarak parlak yüzeylere gerçekçi yansıma verir.

5. Kamera Türleri, Ayarları ve Script ile Kontrol

- **Projection – Perspective:** Perspektif (derinlik) etkisi verir; 3D oyunların standardıdır.
- **Projection – Orthographic:** Uzaklıktan bağımsız aynı boyut; 2D, izometrik ve strateji oyunlarında kullanılır (Size parametresi ile).
- **Field of View (FOV):** Görüş açısı. Büyük değer geniş açı, küçük değer yakınlaştırma hissi verir (genelde 60 civarı).
- **Clipping Planes:** Near ve Far; kameranın çizeceği mesafe aralığı.
- **Clear Flags:** Arka planın Skybox, Solid Color veya Depth Only olarak nasıl temizleneceği.
- **Viewport Rect ve Depth:** Birden fazla kamerada ekranı bölmek (split-screen, mini harita) için kullanılır.
- **Culling Mask:** Kameranın hangi katmanları (Layer) göreceğini belirler.
- **Cinemachine:** Unity'nin kamera takibi, geçişleri ve sinematik çekimleri için paketidir (Package Manager'dan eklenir).

Kamera hareketleri

- Üçüncü şahıs takip kamerası (LateUpdate'te, karakter hareket ettikten sonra):

```
public Transform hedef;
public Vector3 uzaklik = new Vector3(0, 5, -8);
public float yumusaklik = 5f;

void LateUpdate() {
    Vector3 hedefPoz = hedef.position + uzaklik;
    transform.position = Vector3.Lerp(transform.position, hedefPoz,
                                      yumusaklik * Time.deltaTime);
    transform.LookAt(hedef);
}
```

- **Birinci şahıs (FPS) kamerası:** Fare hareketi kamerayı yukarı-aşağı, karakteri sağa-sola döndürür:

```
public Transform govde;            // oyuncu gövdesi
public float hassasiyet = 2f;
float xDonus = 0f;

void Update() {
    float mx = Input.GetAxis("Mouse X") * hassasiyet;
    float my = Input.GetAxis("Mouse Y") * hassasiyet;
    xDonus = Mathf.Clamp(xDonus - my, -80f, 80f);   // bakış sınırı
    transform.localRotation = Quaternion.Euler(xDonus, 0, 0);
    govde.Rotate(Vector3.up * mx);
}
```

- **Fare imleci:** Cursor.lockState = CursorLockMode.Locked; ile oyun içinde imleç kilitlenir.
- **Kamera değiştirme:** İki kameradan birinin enabled / SetActive durumu tuşla değiştirilerek geçiş yapılır; Camera.main ana kameraya erişim sağlar.

6. Fizik Motoru: Rigidbody, Collider, Çarpışma ve Tetikleyiciler

- **Unity 3D fizik:** Nvidia PhysX tabanlıdır; yerçekimi, kütle, sürtünme ve çarpışmaları simüle eder.

Rigidbody

- **Mass, Drag, Angular Drag:** Kütle ve hava direnci değerleri.
- **Use Gravity:** Yerçekiminden etkilenme.
- **Is Kinematic:** İşaretliyse fizik kuvvetleri etkilemez, nesne yalnızca script ile hareket eder.
- **Constraints:** Belirli eksenlerde hareket veya dönüşü kilitler.
- **Collision Detection:** Hızlı nesnelerde Continuous seçilirse nesnenin duvarın içinden geçmesi önlenir.

```
Rigidbody rb;
void Start() { rb = GetComponent<Rigidbody>(); }
void FixedUpdate() {
    rb.AddForce(Vector3.forward * 10f);          // kuvvet uygula
}
// Zıplama: rb.AddForce(Vector3.up * 5f, ForceMode.Impulse);
```

Collider

- **Türler:** Box, Sphere, Capsule (basit ve hızlı), Mesh Collider (karmaşık şekil; maliyetli, Convex işaretlenmeden hareketli nesnede kullanılamaz).
- **Çarpışma için:** İki nesnede de Collider, en az birinde Rigidbody bulunmalıdır.
- **Physic Material:** Sürtünme (Friction) ve sekme (Bounciness) değerlerini belirler.
- **Statik collider:** Rigidbody'siz zemin ve duvarlar; sahnede hareket ettirilmemelidir.

Çarpışma algılama ve tetikleyiciler

	Collision	Trigger
Ayar	Is Trigger kapalı	Collider'da Is Trigger açık
Davranış	Nesneler birbirini engeller	Nesneler içinden geçer, olay tetiklenir
Olaylar	OnCollisionEnter / Stay / Exit	OnTriggerEnter / Stay / Exit
Kullanım	Duvar, zemin, engel, vurma	Toplanabilir eşya, alan, checkpoint

```
void OnCollisionEnter(Collision c) {
    if (c.gameObject.CompareTag("Dusman"))
        Debug.Log("Düşmana çarptı: " + c.contacts[0].point);
}
void OnTriggerEnter(Collider other) {
    if (other.CompareTag("Altin")) {
        puan++;
        Destroy(other.gameObject);
    }
}
```

- **Raycast:** Bir noktadan ışın göndererek nesne algılar (zemin kontrolü, nişan alma, tıklama).

```
if (Physics.Raycast(transform.position, Vector3.down, out RaycastHit hit, 1.1f))
    yerde = true;
```

Önemli: Fizik işlemleri FixedUpdate içinde, Rigidbody üzerinden yapılmalı; Rigidbody'li nesne transform.position ile taşınmamalıdır.

7. Karakter Hareket Kontrolleri, Animator ve Blend Tree

7.1 Karakter hareketi

- **Yöntem 1 – Rigidbody:** Fizik tabanlı; itme, düşme gibi etkiler doğal olur.
- **Yöntem 2 – Character Controller:** Fizikten bağımsız, çarpışmalara duyarlı hareket (Move, SimpleMove, isGrounded). Eğim, basamak (Step Offset) ayarlarına sahiptir.

```
CharacterController cc;
public float hiz = 5f, ziplama = 6f, yercekimi = -20f;
Vector3 hizVektoru;

void Start() { cc = GetComponent<CharacterController>(); }
void Update() {
    float x = Input.GetAxis("Horizontal");
    float z = Input.GetAxis("Vertical");
    Vector3 yon = transform.right * x + transform.forward * z;
    cc.Move(yon * hiz * Time.deltaTime);

    if (cc.isGrounded && hizVektoru.y < 0) hizVektoru.y = -2f;
    if (Input.GetButtonDown("Jump") && cc.isGrounded)
        hizVektoru.y = Mathf.Sqrt(ziplama * -2f * yercekimi);
    hizVektoru.y += yercekimi * Time.deltaTime;
    cc.Move(hizVektoru * Time.deltaTime);
}
```

7.2 Animator ve Animation bileşenleri

- **Animation Clip:** Tek bir hareketi (yürüme, zıplama) içeren animasyon dosyasıdır. Modelle gelebilir veya Animation penceresiyle (Window > Animation) oluşturulabilir.
- **Animator Controller:** Klipleri durum makinesi (state machine) olarak yöneten varlıktır; nesnedeki Animator bileşenine atanır.
- **State:** Bir animasyon durumu (Idle, Walk, Run, Jump). Turuncu olan varsayılan başlangıç durumudur.

- **Transition:** Durumlar arası geçiştir; Has Exit Time, Transition Duration ve Conditions ile ayarlanır.
- **Parameter:** Geçişleri kontrol eden değişkenler: Float, Int, Bool, Trigger.
- **Animation bileşeni (Legacy):** Eski ve basit sistemdir; yeni projelerde Animator tercih edilir.
- **Avatar / Humanoid:** Humanoid rig, bir karakterin animasyonunu başka insansı karaktere aktarmayı (retargeting) sağlar. Mixamo gibi kaynaklardan hazır animasyonlar kullanılabilir.

7.3 Blend Tree

- **Amaç:** Bir parametrenin değerine göre birden fazla animasyonu yumuşak biçimde harmanlamak (Idle → Walk → Run).
- **Oluşturma:** Animator'da sağ tık > Create State > From New Blend Tree; çift tıklayıp Motion listesine klipler eklenir.
- **Türler:** 1D (tek parametre, ör. Speed), 2D Simple Directional (yön: ileri-geri-sağ-sol), 2D Freeform, Direct.
- **Threshold:** Her kliğin hangi parametre değerinde tam etkili olacağını belirler (ör. Idle 0, Walk 0.5, Run 1).

```
Animator anim;
void Start() { anim = GetComponent<Animator>(); }
void Update() {
    float hiz = new Vector2(Input.GetAxis("Horizontal"),
                            Input.GetAxis("Vertical")).magnitude;
    anim.SetFloat("Speed", hiz, 0.1f, Time.deltaTime);  // yumuşak geçiş
    anim.SetBool("Kosuyor", Input.GetKey(KeyCode.LeftShift));
    if (Input.GetKeyDown(KeyCode.Space)) anim.SetTrigger("Jump");
}
```

8. Temel Script Bilgisi: Klavye ve Fare Girişi

8.1 MonoBehaviour yaşam döngüsü

Metot	Ne zaman çalışır?
Awake()	Nesne yüklendiğinde, ilk olarak (bir kez).
Start()	İlk kare çizilmeden önce (bir kez).
Update()	Her karede; girişler ve genel mantık burada.
FixedUpdate()	Sabit aralıklarla; fizik işlemleri burada.
LateUpdate()	Tüm Update'lerden sonra; kamera takibi için.
OnDestroy()	Nesne yok edilirken.

- **Script oluşturma:** Project > Create > C# Script; adı sınıf adıyla aynı olmalı, nesneye sürüklenir.
- Public değişkenler Inspector'da görünür; [SerializeField] ile private değişken de görünür hale getirilebilir.
- **Time.deltaTime:** Kareler arası süre; hareketi bilgisayar hızından bağımsız yapmak için çarpılır.
- **Debug.Log():** Console'a mesaj yazarak hata ayıklama yapılır.

8.2 Klavye girişi

Komut	Açıklama
Input.GetKeyDown(KeyCode.Space)	Tuşa basıldığı an (tek kare).
Input.GetKey(KeyCode.W)	Tuş basılı olduğu sürece.
Input.GetKeyUp(KeyCode.E)	Tuş bırakıldığı an.
Input.GetAxis("Horizontal")	A-D / ok tuşları; -1 ile 1 arası yumuşak değer.
Input.GetAxisRaw("Vertical")	W-S; yumuşatmasız -1, 0, 1 değeri.

Komut	Açıklama
Input.GetButtonDown("Jump")	Input Manager'da tanımlı buton (Space).

8.3 Fare girişi

Komut	Açıklama
Input.GetMouseButtonDown(0)	Sol tuşa basıldığı an (1 sağ, 2 orta tuş).
Input.GetMouseButton(0)	Sol tuş basılı tutulurken.
Input.GetMouseButtonUp(0)	Tuş bırakıldığında.
Input.mousePosition	İmlecin ekran (piksel) konumu.
Input.GetAxis("Mouse X / Mouse Y")	Fare hareket miktarı.
Input.mouseScrollDelta.y	Fare tekerleği.

8.4 Örnek script: hareket, zıplama ve tıklama

```
using UnityEngine;

public class OyuncuKontrol : MonoBehaviour {
    public float hiz = 5f;
    public float ziplamaGucu = 6f;
    Rigidbody rb;

    void Start() { rb = GetComponent<Rigidbody>(); }

    void Update() {
        // Klavye ile hareket
        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");
        transform.Translate(new Vector3(x, 0, z) * hiz * Time.deltaTime);

        // Zıplama
        if (Input.GetKeyDown(KeyCode.Space))
            rb.AddForce(Vector3.up * ziplamaGucu, ForceMode.Impulse);

        // Koşma
        hiz = Input.GetKey(KeyCode.LeftShift) ? 10f : 5f;

        // Fare ile dönme
        transform.Rotate(0, Input.GetAxis("Mouse X") * 2f, 0);

        // Sol tık: ekrandan 3D uzaya ışın gönderip nesne seçme
        if (Input.GetMouseButtonDown(0)) {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out RaycastHit hit))
                Debug.Log("Tıklanan nesne: " + hit.collider.name);
        }
    }
}
```

Not: Yeni Input System paketi kullanılan projelerde Input.* komutları yerine Keyboard.current ve Mouse.current kullanılır. Eski komutlar için Project Settings > Player > Active Input Handling değeri Input Manager (Old) veya Both olmalıdır.

Kısa Tekrar Soruları

5. Local ve global koordinat arasındaki fark nedir?

6. Mesh Filter ile Mesh Renderer ne işe yarar?
7. Directional, Point ve Spot ışık hangi durumlarda kullanılır?
8. Baked ışık haritası neden performansı artırır; hangi nesneler Static olmalıdır?
9. OnCollisionEnter ile OnTriggerEnter arasındaki fark nedir?
10. Blend Tree ne için kullanılır; Speed parametresi nasıl bağlanır?
11. GetKeyDown ile GetKey arasındaki fark nedir?