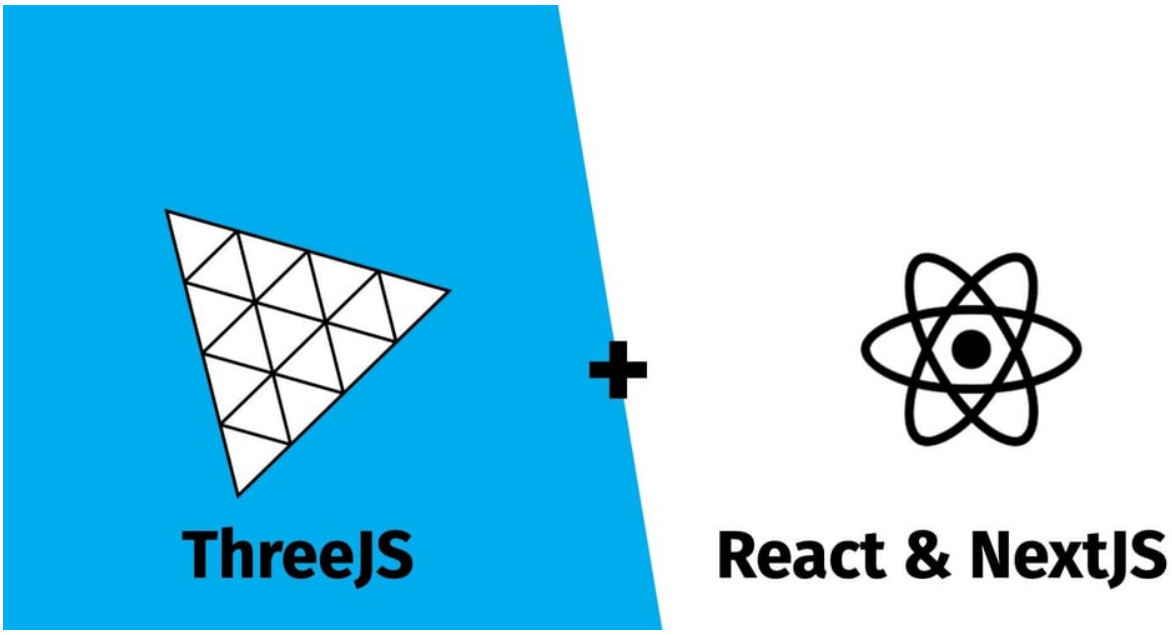
THREEJS ARAŞTIRMA RAPORU



Araştırma Konuları:

UV Mapping

Intersection

Raycaster

Decal Tekniği

Canvas2D’nin 3D Konumlandırılması

Hazırlayan

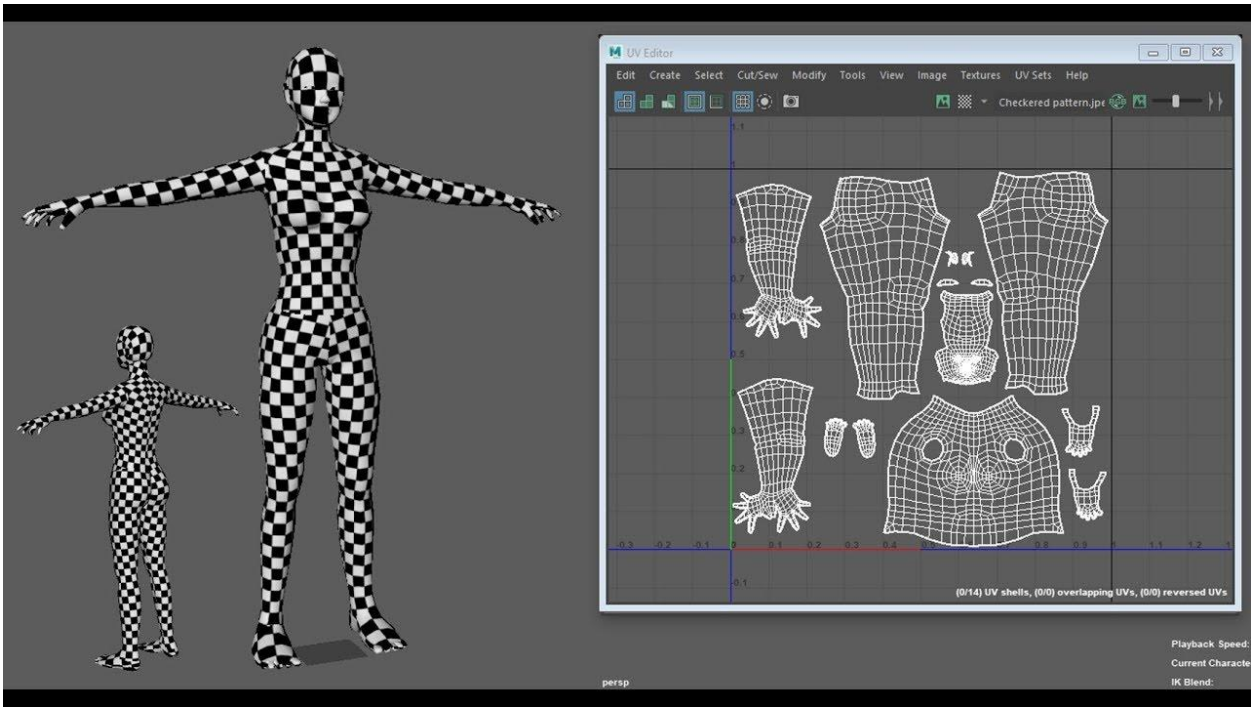
Batuhan Gözükara

1-UV MAPPİNG

UV Mapping , modeli daha da dokulandırmak için bir 3B ağı 3B modelden 2B alana aktarma işlemidir.

UV Haritaları, tüm uygulamalar tarafından kullanılan doku oluşturmanın temel ilkesini temsil eder. UV haritası, çokgen bir 3B model modellendikten sonra oluşturulur ve 3 boyutlu nesne ile aynı ağ yapısına sahiptir, ancak bu çokgenlerin tümü, deforme olabilmeleri için 2B alana dönüştürülür.

Aşağıdaki görsel UV haritasının 3D modeldeki bölümlere karşılık gelen bölümlerini göstermekte:



Bir model tamamen haritalandırılarak numaralandırılır. Bunun sebebi modelin üzerine işlenecek 2B dokunun bu numaralandırılmış bölgelere map edilmesi için gerekli referansı sağlamaktır. ThreeJS projelerinde ekstra UV Map oluşturmaya gerek yoktur. Bunun sebebi .obj gibi modellerin çıktı olarak alındığı modelleme programları bu işi otomatik olarak yaparlar. ThreeJs üzerinde bir modelin yüzeyine 2B bir dokunun işlenmesi için “map” fonksyonunun kullanımı yeterlidir. Ancak eğer bir dokunun hareketi sağlanacaksa dokunun UV koordinatlarının değiştirilmesi ile 2B doku hareketi sağlanabilir.

2-INTERSECTİON - RAYCASTER

Intersection Bilgisayar bilimlerinde kesişim noktasını ifade eder. Pek çok yazılımda farklı bağlamları temsil etsede hepsinde ortak olarak kesişimi ifade eder. ThreeJs üzerinde ise intersection Raycaster ile fare tıklanması sonucu oluşturulan ışının içinden geçtiği, kesiştiği noktaları ifade eden bir terimdir.

Genel olarak ThreeJs üzerinde intersection'ların belirlenmesi olayı raycaster ile oluşturulan ışının mousedown olayında hareket etmesi ve bu ışınının kestiği noktanın yada noktaların raycaster.intersectobject() yada(birden fazla kesişim noktası tutulacaksa) raycaster.intersectobjects([]) fonksyonu ile yakalanması ile olur. Yaklanan noktaların kestiği objeler bir obje içerisine aktarılarak bu kesişim noktalarında bulunan objelerin kimlik bilgilerine ulaşılabilir ve bu objelerin özellikleri değiştirilebilir(örn:Renk, Konum, pozisyon). Ayrıca ulaşılan bu objeler saklanarak başka bir noktadan tekrar erişilebilir.

Görsel olarak açıklamak gerekirse aşağıdaki görselde bir referans içerisinde tutulan modeller arasında bir objeye veya objelere tıklama olayında Raycaster ile bir ışın oluşturulur. Oluşturma sırasında raycaster.current kullanılmasının sebebi bu sayfadan erişilen raycaster ögesinin başka bir sayfada oluşturulup bu sayfadan referans ile ulaşılıyor olmasıdır. Ayrıca intersetcObjects() fonksyonu yerine intersetcObjects([]) kullanılmasının sebebide birden fazla kesişimin tutulmak istenmesidir.

```
const intersectsModels = raycaster.current.intersectObjects(modelsRef.current.children, true);

if (intersectsModels.length > 0) {
    console.log(intersectsModels)
    if (!intersectsModels) console.log('null değer')
    const intersected = intersectsModels[0];
    selectedModelRef.current = intersected.object;
    console.log(selectedModelRef.current)
```

Daha sonrasında bu ışın mouse down olayında sahneye projekte edilir ve eğer projekte edilen ışın, model referansının(modelsRef) tuttuğu objelerden birine veya birkaçı ile kesişirse bu objelerin kimlik bilgileri intersectsModels objesinin içerisine aktarılır.

```
                                                                    src_3b77c8._.js:189
▼ (2) [{…}, {…}] i
  ▶ 0: {distance: 8.665366055286027, point: Vector3, object: Mesh, normal: Vector3, face: {…}, …}
  ▶ 1: {distance: 10.98398085757397, point: Vector3, object: Mesh, normal: Vector3, face: {…}, …}
    length: 2
  ▶ [[Prototype]]: Array(0)
```

Sonrasında talep edilirse ilk kesişen yada 2. Kesişen(talepe bağlı) objenin bilgilerine de ulaşılabilir. Sonrasında bu objeler referanslar içerisinde saklanarak sonradan ulaşılabilir hale de getirilebilir.

Raycaster ile ışın oluşturmak için birden fazla yöntem mevcuttur. Bunlardan ilki yukarda yapıldığı gibi doğrudan raycaster.intersectObject() fonksyonu iledir.

Bir diğer yöntem ise ışın orjinini ve yönünü elle atayıp bunları raycaster'a atamaktır.

```
const origin = new THREE.Vector3(0, 0, 0);
const direction = new THREE.Vector3(1, 0, 0).normalize();
const raycaster = new THREE.Raycaster(origin, direction);
```

Ayrıca raycaster ışının keseceği modeller üzerinde hangi katmanın taranacağıda raycastera belirtilebilir. Ayrıca visible olmayan meshleri tarayamaz. Ayrıca oluşturulan meshlerin özellikleride mesh.visible:false ise true yapılmalıdır.

```
mesh.layers.set(1); // mesh'i 1. katmana ekle
raycaster.layers.enable(1); // raycaster sadece 1. katmanı tarar
```

Bunun dışında raycaster nesnelerin fiziksel sınırlarını test etmek içinde kullanılabilir.

```
mesh.geometry.computeBoundingBox();
const intersect = raycaster.ray.intersectBox(mesh.geometry.boundingBox, new THREE.Vector3());
```

Raycaster ile oluşturulan ışınların mesafeleri sınırlandırılabilri.

```
raycaster.near = 1; //en yakın mesafe
raycaster.far = 100; // en uzak mesafe
```

Kısaca Raycaster ve Intersection'ın doğru ve kontrol edilebilir kullanımı proje üzerinde hakimiyet sağlanabilmesi için çok önemlidir.

3-DECAL TEKNİĞİ

Decal tekniği 3D modeller üzerine dokular, çıkartmalar, logolar eklemek için kullanılan bir tekniktir. Dokular uygulanırken UV haritasının yapısını bozmadan yüzeyin geometrisine ek bir katman olarak uygulanır. Bu sayede büyük modellerdeki uv düzeni korunur. Decaller dinamik ögelerdir. Sahne içerisinde dinamik olarak güncellenebilir, pozisyonu veya açısı değiştirilebilir, kaldırılabilir.

ThreeJS ile kullanımı yine aynı amacı taşır. Modellerin üzerine dokular eklemek için kullanılır bu yapıyı kullanmak için DecalGeometry ögesi projeye ThreeJs'in geometries kütüphanesinden import edilir.

Decal Geometri içerisine uygulanacağı meshi, pozisyonu oryantasyonu ve size'ı alarak new parametresi ile oluşturulur. Daha sonra yapısında kullanılacak material oluşturularak sahneye eklenir.

```
const geometry =  new DecalGeometry( mesh, position, orientation, size );
const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
const mesh = new THREE.Mesh( geometry, material );
scene.add( mesh );
```

Decal geometriler bir modelin yüzeyine uygulanabilmesi için yapısında kullanılacak material oluşturulurken bazı parametreler kullanılır.

```
const material = new THREE.MeshStandardMaterial({
    map: texture,
    transparent: true,
    depthTest: true,
    depthWrite: false,
    polygonOffset: true,
    polygonOffsetFactor: -4,
    side: THREE.FrontSide

});
```

Map parametresi decal için kullanılacak texture'un materiale uygulanması için kullanılır.

Transparent parametresi ise decalin belirli bir yüzeye uygulandığında sadece texture üzerindeki çizim kısmının görünmesi kalan materialin uygulanan yüzeye kamufle olması içindir.

DepthTest parametresi ise decalin uygulanacağı yüzeyin tam üstüne uygulanması içindir. Eğer bu parametre false deeğr alırsa decal yüzeyin arkasına çizilebilir.

Decal, bir yüzeye çok yakın yerleştirilir veya yüzeyle çakışır. Eğer depthWrite açık olursa, decal'in derinlik bilgisi yüzeyin derinlik bilgisinin üzerine yazılır. Bu decalin yüzeyin önünde veya arkasında görünmesine sebep olabilir.

polygonOffsetFactor ve polygonOffset parametreleri ise yüzeyinmi yoksa decalin mi daha önce görüneceğini belirler eğer polygonOffset true ve PolygonOffsetFactor değeri negative olursa decal yüzeye yapışık görünür.

Decal geometri oluşturmak içinde yukardaki bahsettiğim gibi uygulanacak mesh dışında 3 parametre alır bunlardan ilki olan position parametresi decalin modelin neresine uygulanacağını belirler. Eğer intersected objenin fare ile tıklanan noktasına uygulanacaksa intersected.point.clone() fonksyonu kullanılır. Bunun dışında bu nokta ya farklı bir parameterde verilebilir.

Orientation parametresi ise yüzeye sıfırlanması için verilen parametredir. Decalin yüzeye hangi açıda yapışacağını belirler bunun için THREE.Euler() fonkyonu kullanılabileceği gibi bu bilgide elle verilebilir.

Size değeri ise decalin x, y, z eksenlerindeki boyutunu belirlemek için kullanılır.

Uygulamamdan örnek decaller:

4-CANVAS 2D'NİN 3D KONUMLANDIRILMASI

Canvas temelde html'in sunduğu güçlü,esnek bir grafik işleme aracıdır. 2D ve 3D grafikler oluşturmak için yaygın kullanılan bir araçtır.

Çizimler, veri görselleştirme ve resim manipülasyonu gibi amaçlar için kullanılabilir. Piksel seviyesinde kontrol sağlar.

ThreeJS ile kullanımı ise THREE.CanvasTexture() fonksyonu ile sağlanır. Bu fonksyon sahneye bir texture çizer. Bu texture'a ctx parametresi kullanılarak metin, görsel şekil eklenebilir. Ayrıca canvasın oluşturduğu bu texture 3D öğelerin üzerine kaplama olarak eklenebilir.

Canvas texture'ları üzerinde yapılan değişiklikler threejs sahnesine doğrudan yansıtılır bu sayede kayan bilgi panelleri, görsel efektler gibi ögeler canvas ile threejs sahnesinde oluşturulabilir.

Canvas üzerinde piksel seviyesinde çizimler yapılabilir ve bunlar 3D nesnelere uygulanabilir.

Canvas ile veri görselleştirme yapılabilir. Gerçek zamanlı grafikler useEffect kullanımı ile sahneye eklenebilir. Grafik üzerindeki bir veri değiştiğinde grafik güncellenebilir. React, ThreeJs ve canvas üçlüsü bu deneyimi üst seviyeye taşır.

Ayrıca canvasın saydığım özellikleri threeJs'in diğer fonksyonları ile birleştiğinde bir harita üzerinde koordinat işaretlemeside yapılabilir.