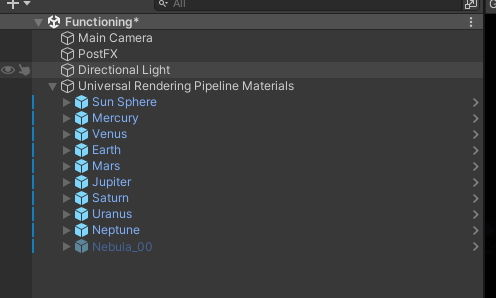
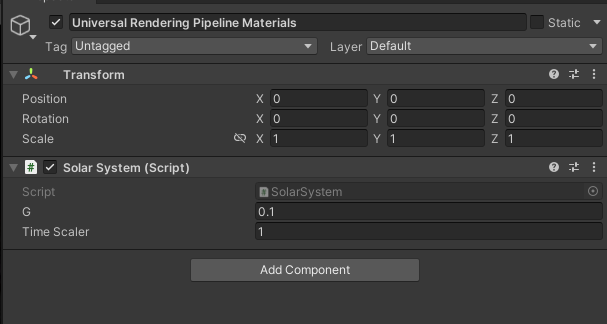
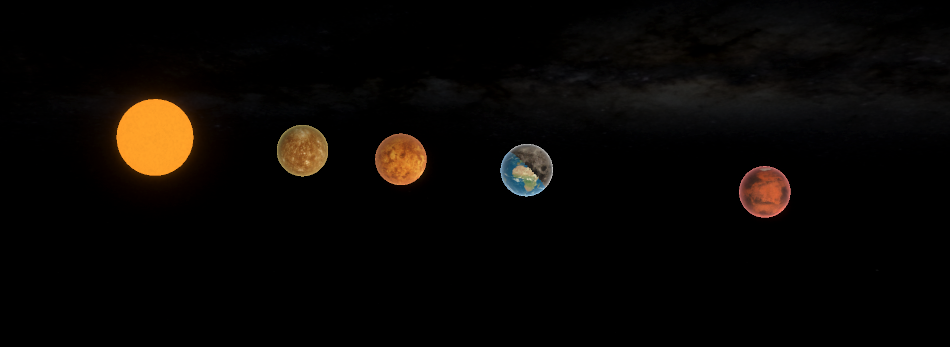
Mixed Reality & Simulation  
Solar System - Physics  
WS 2022/23

AI21M041 – Eric Eckstein

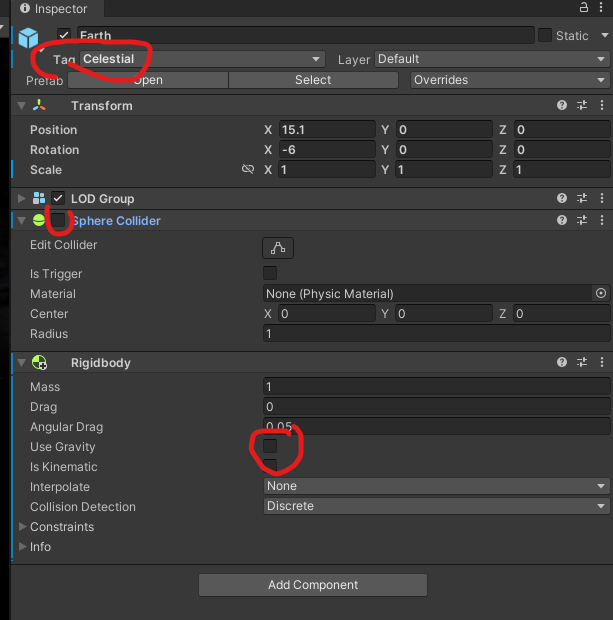


Im Universal Rendering Pipeline Materials (alle Planeten) ist das globale Solar System Script hinterlegt. Hier können die Parameter G und Time-Scaler verändert werden. Im Script werden die Kräfte aller Planeten aufeinander gemessen und gegeneinander aufmultipiziert um die Planeten in einer intakten Umlaufbahn zu halten.



Die Planeten sind aus dem Asset-Store (Mond und Erde überlagern weil sehr groß für Sichtbarkeit)

An jedem Planeten ist der Tag Celestial hinterlegt. Die Werte für Mass wird aus dem Rigidbody genutzt. Die Gravitation ist ausgeschalten.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Orbital period | Measure in seconds | Percentage of earth | Derivation |
| Mercure | 0,241 | 0,93 | 0,24538259 | 102% |
| Venus | 0,615 | 2,02 | 0,53298153 | 87% |
| Earth | 1 | 3,79 | 1 | 100% |
| Mars | 1,881 | 7,87 | 2,07651715 | 110% |
| Jupiter | 11,862 | 42,2 | 11,1345646 | 94% |
| Saturn | 29,457 | 137,84 | 36,3693931 | 123% |
| Uranus | 84,021 | 417,2 | 110,079156 | 131% |
| Neptune | 164,8 | 792,4 | 209,076517 | 127% |

using UnityEngine;

public class SolarSystem : MonoBehaviour

{

//real value of gravitational constant is 6.67408 × 10-11

//can increase to make thing go faster instead of increase timestep of Unity

[SerializeField]

float G = 0.1f;

GameObject[] celestials;

[SerializeField]

float TimeScaler = 1f;

// Start is called before the first frame update

void Start()

{

celestials = GameObject.FindGameObjectsWithTag("Celestial");

SetInitialVelocity();

}

// Update is called once per frame

void FixedUpdate()

{

Time.timeScale = TimeScaler;

Gravity();

}

void SetInitialVelocity()

{

foreach (GameObject a in celestials)

{

foreach (GameObject b in celestials)

{

if (!a.Equals(b))

{

float m2 = b.GetComponent<Rigidbody>().mass;

float r = Vector3.Distance(a.transform.position, b.transform.position);

a.transform.LookAt(b.transform);

//Circular Orbit = ((G \* M) / r)^0.5, where G = gravitational constant, M is the mass of the central object and r is the distance between the two objects

//We ignore the mass of the orbiting object when the orbiting object's mass is negligible, like the mass of the earth vs. mass of the sun

a.GetComponent<Rigidbody>().velocity += a.transform.right \* Mathf.Sqrt((G \* m2) / r);

}

}

}

}

void Gravity()

{

foreach (GameObject a in celestials)

{

foreach (GameObject b in celestials)

{

if (!a.Equals(b))

{

float m1 = a.GetComponent<Rigidbody>().mass;

float m2 = b.GetComponent<Rigidbody>().mass;

float r = Vector3.Distance(a.transform.position, b.transform.position);

a.GetComponent<Rigidbody>().AddForce((b.transform.position - a.transform.position).normalized \* (G \* (m1 \* m2) / (r \* r)));

}

}

}

}

}