

Média- és Oktatásinformatikai Tanszék

Naprendszer Szimuláció Exobolygókkal

Dr. Horváth GyőzőTanszékvezető egyetemi docens (ELTE IK)Nyírő Levente Gyula

Programtervező informatikus MSc

Budapest, 2025

Tartalom

[1. Bevezetés 3](#_Toc195165223)

[2. Irodalmi áttekintés 5](#_Toc195165224)

[2.1. Teljesítményproblémák és optimalizálás JavaScriptben 6](#_Toc195165225)

[2.2. Webes 3D vizualizáció és kihívásai 7](#_Toc195165226)

[2.3. Háromdimenziós vizualizáció egy bútor webáruházban 8](#_Toc195165227)

[2.4. Fényhasználat optimalizálása háromdimenziós megjelenítésben 9](#_Toc195165228)

[2.5. BabylonJS és ThreeJS összehasonlítás 10](#_Toc195165229)

[3. Technikai háttér 11](#_Toc195165230)

[3.1. Felhasznált technológiák 12](#_Toc195165231)

[3.1.1. Angular keretrendszer 12](#_Toc195165232)

[3.1.2. ThreeJS – WebGL alapú háromdimenziós renderelés 13](#_Toc195165233)

[Forrásjegyzék 16](#_Toc195165234)

# Bevezetés

Kétéves programtervező informatikus mesterképzésem keretében az utolsó félévben készítenem kell egy diplomamunkát, ahol bizonyíthatom, hogy képes vagyok az adott szakterület fejlesztésére, illetve kutatására. A témám keretein belül kutatási anyagokat kerestem, azokat dolgoztam fel, majd használtam az így megszerzett tudást, hogy a saját programomba beleépítsem, fejlesszem és teszteljem azt. A dokumentációban szóba fog kerülni a témaválasztás, a probléma, ami után kutatni kell és ennek magyarázata, eredményeket fogok ismertetni, beszélni fogok mások eredményeiről, akár más programnyelvekben szerzett tapasztalataikról, majd ismertetni fogom a technikai hátterét az általam írt programnak és az összeszedett megoldásomat a problémára.

Témakörként a Naprendszer modelleket választottam, ugyanis ennek ábrázolása nemcsak lenyűgöző látványvilágot kínál, hanem lehetőséget ad arra is, hogy a modern webes technológiák határait feszegetve, interaktív és valósághű modelleket hozzak létre. Már két féléve foglalkoztam ezzel a témával, hála az egyetemen elérhető kurzusnak, ahol a kiválasztott projektmunkával foglalkozhat a hallgató a félév során és egy mentor is segít a webes tudás elmélyítésében.

A szoftverfejlesztési pályafutásom során mindig is vonzott a frontend fejlesztés, hiszen itt kerül sor a felhasználói felület tervezésére és megvalósítására, amelynek kiemelt szerepe van a felhasználói élmény alakításában. Ehhez kapcsolódóan szerettem volna foglalkozni a háromdimenziós objektumok ábrázolásával weboldalaknál vagy mobilapplikációknál. Ezt azért tartottam fontosnak, mert a webdesign olyan irányba halad, ahol a vizuális megjelenítés és a felhasználóbarát megoldások rendkívül fontosak. A háromdimenziós modellek használata lehetővé teszi, hogy a felhasználók interaktív módon fedezzék fel a tartalmat, ami különösen fontos lehet olyan területeken, mint a tudományos vizualizáció, játékfejlesztés vagy akár az oktatás.

Technikai szempontból a webapplikáció, amiben a kutatásom készült Angular keretrendszerben készült és ebbe lett integrálva a ThreeJS egy canvas-on keresztül. A webes háromdimenziós képi megjelenítésre sok lehetőség elérhető webapplikációknál, ezek közül a legismertebb a ThreeJS, így ez volt az a technológia, amire a kutatásaimat és az általam megírt programot támasztom. Adatlekéréseket is alkalmazok a megírt programban, mégpedig egy TODO francia API-tól kérem el a bolygóknak a pontos adatait.

A ThreeJS már több, mint egy évtizede velünk van az informatikában. 2010-ben publikálták és azóta rengeteg fejlődésen ment keresztül. Ez a JavaScript könyvtár lehetővé teszi, hogy a böngészőben közvetlenül rendereljünk 3D modelleket, anélkül, hogy külső szoftverekre vagy pluginokra lenne szükség. Közben megjelentek az ismert keretrendszerek is a webtechnológiában, mint a React, vagy az Angular, így elő kellett segíteni, hogy ezen technológiákba is importálható legyen a háromdimenziós csomag.

A diplomamunkám végső célja, hogy kutatást végezzek arról, hogyan lehet hatékonyabbá és optimalizáltabbá tenni a ThreeJS működését Angular keretrendszerben, akár mesterséges intelligencia beépítésével a fent említett Naprendszer modellben. Mindez, azért szükséges, mert a megfigyeléseim szerint az egész ThreeJS hatalmas potenciállal rendelkezhet a jövőben, azonban a komplex modellek és a dinamikus animációk kezelése gyakran hatalmas teljesítményigényeket követelhetnek. Ez különösen igaz lehet olyan esetekben, amikor a felhasználói élmény folyamatosan változik és interaktív háromdimenziós tartalmat igényel, mint például a Naprendszer modellek esetében. A kutatásom során olyan technikai megoldásokat és optimalizációs stratégiákat keresek, amelyekkel javítható a renderelési teljesítmény, csökkenthető a memóriahasználat, és biztosítható a felhasználók számára a zökkenőmentes és élvezetes élmény.

# Irodalmi áttekintés

A ThreeJS és a háromdimenziós ábrázolás optimalizálása nem egy új téma a szakmába. Számos kutató és fejlesztő is foglalkozott már ezzel a kérdéssel az elmúlt években. Háromdimenziós tartalmak webböngészőben történő megjelenítése, különösen komplex modellek és dinamikus jelenetek esetében mindig is kihívást jelentett, hogy a teljesítmény és a felhasználói élmény hasonló legyen, mint egy egyszerűbb modellnél. A ThreeJS, mint a WebGL egyik legnépszerűbb JavaScript könyvtára, lehetővé teszi a háromdimenziós modellek hatékony megjelenítését, de a nagy teljesítményigény és a böngésző korlátai miatt a fejlesztőknek gyakran kreatív megoldásokat kell keresniük a problémákra.

Számos kutatás és cikk foglalkozott már a ThreeJS teljesítményének a javításával. Volt, ahol a geometriák egyszerűsítésével és textúrák optimalizálásával foglalkoztak, mindezzel is csökkenthető volt a renderelési idő és a memóriahasználat. Egy másik gyakori megközelítés az instancing. Ez lehetővé teszi, hogy több objektumot egyetlen renderelési hívással jelenítsünk meg, ezzel jelentősen csökkentve a CPU és GPU terhelését.

A modern webes keretrendszerek (Angular, React) is komoly figyelmet kapott az elmúlt pár évben. A komponensalapú technológia és a state management hatékony használata, hogy a háromdimenziós tartalom zökkenőmentesen fusson együtt a webalkalmazás többi részével.

Több kutatás kitér arra is, hogy miképp lehet a mesterséges intelligencia igénybevételével hozzásegíteni a háromdimenziós ábrázolás teljesítményének a fokozását.

Ebben a fejezetben most egyes kutatások eredményeit fogjuk áttekinteni, amik hatással voltak az általam bemutatott kutatásokra. Azt is be fogom mutatni, hogy miképp volt mindez hatással.

## Teljesítményproblémák és optimalizálás JavaScriptben

Az első bemutatott cikk egy 2016-ban megírt publikáció, amely még nem kapcsolódik a ThreeJS-hez. A háromdimenziós megjelenítés weben a sok teljesítményigénytől fogva optimalizálni szükségeltetik. Ehhez meg kell alapozni azt, hogy magának a JavaScript is (amiben fejlesztünk) feltárjuk a problémáit teljesítmény tekintetében és azt vizsgálhassuk, hogy ezt hogy tudjuk optimalizálni. Ezt a tudást átültethetjük a ThreeJS síkjára is és így kaphatunk egy kielégítő teljesítménnyel futó háromdimenziós megjelenítést.

A tanulmány célja, hogy empirikus módszerekkel elemezze a gyakori teljesítményproblémákat és bemutassa, hogyan lehet ezeket optimalizálni. A kutatás során 98 rögzített teljesítményproblémát elemeztek 16 népszerű JavaScript projektből, mind kliensoldali, mind szerveroldali kódokból. A kutatás öt kérdésre keresett választ.

Az első a JavaScript teljesítményromlásának fő okaira próbál magyarázatot találni. Ezeket elsősorban a nem megfelelő API hívásokban, felesleges adatmásolásokban és ismétlődő műveletekben találta meg. Előbbi az esetek 52%-át tette ki a kutatásban. Ez egy elég nagy arány, hogy erre mindenképp odafigyeljen a fejlesztő.

Második a komplexitását kutatta az optimalizációs változtatásoknak. Itt a cikk leírta, hogy a legtöbb optimalizáció csak pár sor kódot érint, mégis ezek vonhatók felelősségre a JavaScript teljesítményromlásáért.

Harmadik esetben az optimalizálás hatására elért teljesítménynövekedést vizsgálja a cikk. Az eredmény jelentős növekedést mutatott ki (átlagosan 20-75%-os növekedés), azonban az eredmények nem voltak konzisztensek különböző JavaScript motorok között.

Negyedik esetben az előző válaszban található optimalizációknak biztosított konzisztens növekedés mértékét vizsgálták, itt azonban mindössze csak 42,68%-a volt minden motorban.

Az utolsó kérdésben a kutatás az ismétlődő mintázatokra tért ki és arra jutott, hogy az összegyűjtött 29 ismétlődő mintázat más projektben is alkalmazhatóak.

## Webes 3D vizualizáció és kihívásai

Ez a cikk az előző cikkhez képest kifejezetten a webes háromdimenziós implementáció során felmerülő teljesítményoptimalizációról szól. A cikk szintén leírja, hogy ez a technológia egyre elterjedtebb mind a e-kereskedelem, játékfejlesztés, építészeti tervezésben, kutatásokban, ahol a felhasználók interaktív és valósághű élményt várnak. Azonban a webes környezetnek sajátos korlátai és egyúttal kihívásai is vannak. Ezeket figyelembe kell venni a hatékony működés érdekében.

Figyelembe kell venni, hogy a böngészők korlátozott számítási teljesítménye és memóriahasználata miatt a komplex háromdimenziós modellek renderelése lassúvá válhat. A ThreeJS WebGL alapú, ami egy hatékony technológia, de nagy felbontású textúrák és komplex geometriák kezelése esetén nehézségekbe ütközhet. Arra is figyelni kell az implementáció közben, hogy különböző böngészők és eszközök eltérően támogatják a WebGL-t és a 3D grafika egyéb technológiáit, továbbá figyelembe kell venni a régebbi eszközökön való működést is. Továbbá fontos, hogy a felhasználó ne egy hosszú betöltési idővel rendelkező oldalt lásson, mert ez jelentősen ronthatja a felhasználói élményt. Ilyenkor megoldásként a fejlesztők a modellek tömörítésének és a progresszív betöltés használatának irányába mozdulhatnak el. Egy másik fontos tényező, hogy a felhasználói interakciók esetén is optimalizálni kell a renderelési folyamatot.

A cikk megoldásként javasolja a háromdimenziós modellek poligon számának csökkentését, illetve a Level of Detail technika alkalmazását, amely a modell részletességét dinamikusan állítja a kamera távolsága alapján. A másik ilyen hatásos módszer a fényoptimalizálás, fényforrások számának csökkentésével és a statikus árnyéktérképek használatával. További megoldásként javasolható, hogy a modellek és a textúrák tömörítése sokkal hatásosabb GLTF formátumban, mint mondjuk a hagyományosan igénybe vett OBJ formátumban. Mindemellett a progresszív betöltés és a lazy loading használata nagyobb fokú felhasználói élményt adhat főleg, ha a felhasználó már a teljes modell betöltése előtt interakcióba léphet a weboldalon megjelenő modellel. Fontos, hogy az eszköznek a kapacitását is vegyük igénybe úgy, hogy a felhasználó eszközének képességeitől függően dinamikusan változtatjuk a renderelési minőséget.

## Háromdimenziós vizualizáció egy bútor webáruházban

Ez a cikk egy olyan online kereskedési platform fejlesztését mutatja be, ahol a cél az volt, hogy a felhasználóknak minél nagyobb élményt adjanak azáltal, hogy a piacra szánt bútorokat pontosan és informatívan mutassa be. Azonban az ilyen fejlesztéseknek a hátránya az is, hogy nagy teljesítmény szükségeltetik és ehhez egy optimalizált frontendet és backendet kell párhuzamosan létrehozni.

A probléma az volt, hogy sok webáruház kétdimenziós képekre támaszkodik, ezzel viszont nem lehet bemutatni pontosan az eladni szánt termék méreteit, dizájnját és funkcionalitását. Ez viszont rosszul meghozott vásárlási döntésekhez, magasabb visszaküldési arányokhoz és elégedetlen ügyfelekhez vezet, különösen olyan iparágakban, mint a bútorkereskedelem, ahol a térbeli vizualizáció kulcsfontosságú.

A webáruház a ThreeJS integrálása mellett határozta el magát, ezzel elérhetővé téve a felhasználónak a termékek 360 fokos forgatását, közelítést a részletek megjelenítéséhez, illetve valós időben testre szabhatják az anyagokat és a színeket.

További megoldásként a MERN stack-re hivatkozik a cikk, ez annyit takar, hogy ezáltal a fejlesztés során létrejövő architektúra skálázhatóvá és modulárissá válik, mind frontend, backend és adatbázis szinten is. Ezzel elérve, hogy a teljesítmény a legjobbá váljon kevesebb erőforrás igénybevételével. Ez is egy fontos szempont, hogy az ügyfélelégedettség növekedésnek induljon.

A kutatás kimutatta az elért előnyöket is, legyen szó a fokozott felhasználói elköteleződéstől, vagy épp a magabiztosabb vásárlói döntéshozataltól. Jelentősen csökkent továbbá a visszáruk száma is, ugyanis a pontos vizualizáció csökkentette az eltéréseket a várakozások és a valóság között.

## Fényhasználat optimalizálása háromdimenziós megjelenítésben

Ebben a cikkben a fényhasználat optimalizálását vizsgálják. Ez az én kutatásomban szintén egy fontos támaszpont, ugyanis a fénybeállítások általában több erőforrást tudnak igénybe venni és ezáltal nagyobb optimalizációra van szükségünk, hogy egy élvezhető teljesítményélményt lehessen adni a felhasználóknak. A cikkben kitérnek arra is, hogy a megfelelő beállítások a renderelési időt is jelentősen le tudják csökkenteni.

A fényforrások helyes beállítása elengedhetetlen a valósághű árnyékok, tükröződések és fényvisszaverődések létrehozásához. Ennek optimalizálása segít kiemelni a modellek részleteit, egyben mélységérzetet ad és növeli a vizuális hitelességet. Több fénytípust is megkülönböztetünk. Környezeti fény – alapvető megvilágítás, ami az egész teret bevilágítja. Irányított fény – ilyen a napfény is. Pontszerű fény – lámpa fényéhez hasonlítható. Az utolsó ilyen a fókuszfény – ami egy zseblámpa világítását imitálhatja.

Az optimalizálási technikák sokrétűek lehetnek, ezt a kutatás is leírja. Dinamikus árnyékok esetén, ahol a valós idejű árnyékszámítások kellenek a pontosabb megjelenítés céljából erőforrásigényesek lehetnek. Optimalizálás céljából ezt csakis a kritikus területeken érdemes használni. Másik ilyen technika az előre számított fény- és árnyékinformációk. Ezek csökkentik a valós idejű számítások terhelését, ami mindenképp előnyös renderelés szempontjából. Ezen kívül még a fénymintavételezés az, ami segítheti az optimalizálást. Itt a fényforrások hatásának mintavételezése csökkenti a renderelési időt.

Következtetésképpen a fényoptimalizálás, ahogy azt a fenti technikák is leírják, nemcsak a vizuális minőséget javítja, hanem egyben csökkenti is a renderelési időt és a hardverigényt. Arra is érdemes a fejlesztés során figyelni, hogy a felesleges fényforrásokat a fejlesztő eltávolítsa és a fényterheket jelentősen csökkentse, ugyanis ez egyben teljesítménynövekedést is eredményezhet. Ezekre a technikákra nagyon odafigyelnek az olyan gyakorlati területeken is, mint a játékfejlesztés, építészeti vizualizáció, vagy épp a termékdesign.

## BabylonJS és ThreeJS összehasonlítás

Fontosnak tartottam a kutatásom során, hogy egy másik keretrendszerrel való összehasonlítást is alapul vegyek a kutatásaim során. Az ebben a cikkben lévő tanulmány célja, hogy statisztikai adatokkal alátámasztva kiderüljön, hogy melyik WebGL könyvtár teljesít jobban nagy terhelés mellett. Mindez a kétdimenziós Voronoi-diagramokat (szigetvilágokat) háromdimenziós térképpé alakítja át. Három fő szempontból történtek a mérések: GPU, CPU és RAM használat.

Módszertant figyelembevéve a tesztek egy alap konfigurációban készültek, alap processzorral és 8 GB RAM igénybevételével. Ubuntu rendszeren futott és NodeJS-en voltak az automatizált tesztek. A teszt során 10 különböző méretű és komplexitású térképet ugyanazzal a jelenetettel rendereltek mindkét könyvtárat használva (kamera útvonal követése, gyors forgatás). Majd ezután az automatizált tesztrendszer 60 másodperces teszteket futtatott le, 3 másodpercenként rögzítve az erőforrás-használatot. A végén nem normális eloszlású adatok miatt Wilcoxon Rank Sum tesztet alkalmaztak a különbségek szignifikanciájának vizsgálatára.

Eredmények tekintetében érdekes adatok kerültek napvilágra. GPU-terhelés szempontjából a ThreeJS sokkal jobb eredményeket mutatott, átlagosan 20-30%-al kevesebbet a BabylonJS-hez képest. CPU-terhelésnél is a ThreeJS mutatott előnyt, viszont itt „csak” 15%-al. RAM használatot tekintve azonban nem jutott különbségre a két keretrendszer – a ThreeJS talán kicsivel több RAM-ot használt a renderelés közben. Ezekből az adatokból arra a következtetésre jutott a cikk, hogy a ThreeJS nagyobb komplexitású jelenetek mellett sokkal jobb futási időket tud produkálni szemben a BabylonJS-el. Utóbbi előnyére írható, hogy sokkal kezelhetőbb, egyszerűbb implementáció szempontjából, de a teljesítményhátránya nem elhanyagolható.

# Technikai háttér

A Naprendszer szimuláció megvalósítása során számos technológiai és architekturális döntést kellett meghozni, hogy a rendszer hatékonyan működjön, mindeközben valósághű vizualizációt nyújtson, és optimalizált legyen a teljesítmény szempontjából. Ebben a fejezetben részletesen bemutatom a felhasznált technológiákat, az alkalmazott architektúrát, valamint a megvalósítás során alkalmazott főbb módszereket és optimalizációs technikákat.

A modern webes technológiák gyors fejlődése lehetővé teszi, hogy egyre komplexebb háromdimenziós tartalmakat jelenítsünk meg a böngészőkben, azonban ez a lehetőség számos kihívással is jár. A teljesítmény, skálázhatóság és felhasználói élmény szempontjából kritikus volt, hogy a megfelelő eszközöket és módszereket válasszam. A következő fejezetekben bemutatom, hogyan választottam ki a megfelelő keretrendszereket, miként épült fel a rendszer architektúrája, és milyen technikákat alkalmaztam a hatékony működés érdekében.

A fentiek mellett kitérek a külső adatforrások integrálásának folyamatára is, amely lehetővé tette, hogy a szimuláció ne csak vizuálisan legyen valósághű, hanem a tudományos megalapozottságát is biztosítsam. Végül pedig áttekintésre kerül a fejlesztés során felmerülő kihívások és az azokra talált megoldások, amelyek nélkülözhetetlenek voltak a stabil és optimálisabban futó alkalmazás létrehozásához.

Ez a fejezet nem csupán a jelenlegi implementációt mutatja be, hanem betekintést nyújt a jövőbeli fejlesztési lehetőségekbe is, mint például a renderelési folyamat további optimalizálása vagy a virtuális valóság támogatásának bevezetése.

## Felhasznált technológiák

A választott technológiák lehetővé tették egy olyan alkalmazás létrehozását, amely nemcsak jól teljesít, hanem könnyen bővíthető és karbantartható is. Mindezt úgy, hogy a tudományos pontosság és a felhasználói élmény sem került háttérbe.

### Angular keretrendszer

Az alkalmazás alapját az Angular keretrendszer képezi, amelyet számos előnye miatt választottam és mert a régebbi fejlesztések során, amelyek a ThreeJS-re irányultak, ebben a technológiában volt releváns tapasztalatom. A komponensalapú architektúra lehetővé teszi a kód újrafelhasználását és könnyű karbantarthatóságát. Az Angular erős típusossága (TypeScript) pedig segít elkerülni a gyakori programozási hibákat. A keretrendszer beépített eszközei, mint a dependency injection és a reaktív programozás (RxJS) lehetővé tették az alkalmazás hatékony és moduláris kialakítását.

Itt tulajdonképpen az app.component.ts-nek és a illetve a threejs.worker.ts-nek volt főszerepe. Azonban idő kellett, míg kikutattam, hogy hogyan kell a kettőt összekötni, hogy működőképes legyen a háttérben futó szál és a köztük lévő kommunikáció is zavarmentes lehessen. Ennek megvalósításáról egy későbbi fejezetben térek ki.

### ThreeJS – WebGL alapú háromdimenziós renderelés

A projektemben, ahogy azt már a bevezetőben bemutattam, ThreeJS könyvtár segítségével valósítottam meg a Naprendszer interaktív háromdimenziós szimulációját, amely lehetővé teszi a bolygók valósághű megjelenítését és viselkedésének szimulálását. A megoldásom a WebGL technológiára épül, így hardveres gyorsítás kihasználásával képes komplex háromdimenziós jelenetek megjelenítésére közvetlenül a böngészőben anélkül, hogy külső bővítményekre lenne szükség.

A Naprendszer modelljében minden bolygót különálló 3D-s objektumként valósítottam meg gömb alakú geometriákkal (SphereGeometry), amelyekhez valós felvételeken alapuló textúrákat rendeltem. A bolygók anyagait a MeshPhongMaterial segítségével hoztam létre, ami lehetővé teszi a fényvisszaverődés és árnyékolás valósághű megjelenítését. A napot egy különleges SpotLight segítségével modelleztem, amely nemcsak megvilágítja a bolygókat, hanem dinamikusan változtatható árnyékokat is vetít, így a napfogyatkozást is lehet szimulálni. A világítás kiegészítésére AmbientLight-ot is alkalmaztam az alap megvilágítás biztosítására.

A kamera rendszer egy perspektivikus nézetet biztosító PerspectiveCamera-ra épül, amely lehetővé teszi a felhasználó számára, hogy szabadon navigáljon a virtuális Naprendszerben. A kamera mozgását és irányítását finoman hangoltam, hogy természetes és intuitív élményt nyújtson. A felhasználó egérrel forgathatja a jelenetet, billentyűzettel közelíthet és távolodhat, valamint kattinthat az egyes bolygókra a részletesebb megtekintéshez.

A szimuláció magja a bolygók mozgásának pontos modellezése. Minden bolygó saját sebességgel forog a tengelye körül, miközben elliptikus pályán kering a Nap körül. A keringési sebességeket a bolygók tömegéből és pályájának excentricitásából számítja a program, hogy a lehető legvalósághűbb viselkedést érjük el. A Föld holdját külön objektumként valósítottam meg, amely a Föld körül kering, saját sebességgel.

A rendszer kiterjesztett funkciókat is tartalmaz. A felhasználó saját bolygókat adhat hozzá a Naprendszerhez, amelyeket a program automatikusan elhelyez a megfelelő pályán. A bolygók pályáit opcionálisan megjeleníthető vonalakkal jelöltük, hogy segítsék a térbeli tájékozódást. A teljesítmény optimalizálása érdekében a számításigényes műveleteket WebWorker-ekbe szerveztem, így a főszál mindig zökkenőmentesen marad.

A fejlesztés során számos kihívással kellett megküzdenünk. A komplex jelenetek renderelése jelentős terhelést jelenthet a GPU-n, ezért optimalizálni kellett a geometriákat és a renderelési folyamatot. Az interaktív vezérlés érzékenységét és pontosságát több iterációban finomhangoltuk. A valósághűség érdekében minden bolygó paraméterét (méret, forgási sebesség, pálya) tudományos adatok alapján állítottuk be.

A ThreeJS könyvtár kiválóan bizonyult a feladathoz, rugalmasságát és teljesítményét tekintve. A megvalósított megoldás nemcsak vizuálisan lenyűgöző, hanem pontos tudományos alapokon nyugszik, miközben interaktív és könnyen kezelhető felületet biztosít a felhasználóknak. A rendszer kialakítása lehetővé teszi további bővítéseket és fejlesztéseket, így jövőben akár újabb égitestek vagy fizikai jelenségek modellezése is könnyedén hozzáadható a szimulációhoz.

### Webworker integrálása Angularban

Az egyik leglényegesebb dolog, amivel egy jövőbeli fejezetben is fogom bizonyítani, hogy a program teljesítménye és erőforrásigénye valóban javulást mutatott az a Webworker beépítése Angularban. Ezzel rengeteg számítás, illetve programfutást lehet kiszervezni egy mellékszálra, ahelyett hogy a főszálat terhelnénk plusz funkciókkal.

Ehhez elsősorban a „\_workers” nevezetű mappába létre kell hozni a „threejs.worker.ts” fájlt. Majd a létrehozott projektben egy másik fájlt is be kell konfigurálni „tsconfig.worker.ts” néven. Ebbe a fájlba az 1. ábrán látható kódot kell belefoglalni. Majd az „angular.json” fájlon belül hozzá kellett adnom a következő sort a „projects/{projectname}/architect/build/option”-ön belül: "webWorkerTsConfig": "tsconfig.worker.json".

A képen szöveg, képernyőkép, Betűtípus, tervezés látható

Automatikusan generált leírás

1. ábra

Az összekötés html oldalon egy canvas-sel valósul meg, és az angular typescript oldalán inicializálni kell a workert, majd ott át lehet adni postMessage segítségével a canvast, illetve bizonyos eseménykezeléseket a hozzájuk tartozó paraméterekkel. Erről a 2. ábrán láthatjuk a példakódot.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

2. ábra

A workerben insideWorker-t és a THREE csomagot inicializálni kell elsősorban, majd az insideWorkeren belül kell a canvas-re hivatkozva megadni a rendereléssel kapcsolatos adatokat. A self.onmessage kóddal megkaphatjuk az eseményt megnevezve a megfelelő paramétereket, amint felhasználói interakció érkezik a főszálról, ahogy az a 3. ábrán látható.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

3. ábra

### DataService és az API forrás

A modell implementálása közben fontos volt, hogy a webapplikáció olyan adatokat szerezzen meg a futás elején, amely valódi adatokkal ábrázolja a bolygókhoz tartozó információkat, így még inkább valóssá téve a modellt.

Az Angular-on belüli data.service.ts fájl szolgál arra, hogy service layer-ként adatokat húzzon le API-ról és ezt továbbítsa azokhoz a metódusokhoz, ahol a service egyes függvényei meg lettek hívva. Ilyen metódusok az egyes bolygókhoz tartozó lekérések.

Az API viszont egy francia oldalról lett lekérve, ez egy publikusan és reális adatokkal elérhető végpont, amelyet bárki használhat céltól függetlenül. [https://api.le-systeme-solaire.net/rest.php/bodies/{planetName](https://api.le-systeme-solaire.net/rest.php/bodies/%7bplanetName)} címen érhető el „GET” metódus használatával és a „planetName” helyére kell írni a megfelelő bolygó nevét francia nyelven.

# Forrásjegyzék

[1] .NET Entity Framework Core Documentation. <https://learn.microsoft.com/en-us/ef/core/>, 2023. 01. 22.