



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

NUMERIKUS ANALÍZIS TANSZÉK

Stilizált 3D szemantikus látvány adott időpontban, adott GPS lokáción

Szerző:

Poros Tamás Gábor

programtervező informatikus BSc

Belső témavezető:

Fábián Gábor

egyetemi adjunktus, PhD

Külső témavezető:

Hiba Antal

kutató, PhD

Budapest, 2023

Tartalomjegyzék

1. Bevezetés	5
2. Elmélet	8
2.1. Az idő ábrázolása	9
2.1.1. Egyezményes koordinált világidő, UTC	9
2.1.2. Unix-idő	10
2.1.3. Julián-dátum	10
2.2. Koordináta-rendszerek	11
2.2.1. Föld-központú, Földhöz rögzített koordináta-rendszer	11
2.2.2. Földrajzi koordináta-rendszer	11
2.2.3. NED koordináta-rendszer	13
2.2.4. Grafikus API-k koordináta-rendszere	13
2.3. Koordináta-rendszerek közötti transzformáció	13
2.3.1. Transzformáció WGS84-ből ECEF-be	13
2.3.2. Transzformáció ECEF-ből NED-be	14
2.3.3. Sodrásirány megfordítása	14
2.4. Euler-szögek meghatározása forgatási mátrixból	16
2.5. Normálvektorok súlyozása	17
2.6. Nap pozíciója	19
2.6.1. Ekkliptikai koordináták	20
2.6.2. Ekvatoriális koordináták	21
2.6.3. Horizontális koordináták	21
2.7. Árnyalás	22
2.7.1. Fényforrások	22
2.7.2. Megvilágítási modellek	23
2.7.3. Árnyalási modellek	23
2.8. Grafikus csővezeték	24

2.8.1. Input Assembler (IA)	24
2.8.2. Vertex Shader (VS)	25
2.8.3. Rasterizer	25
2.8.4. Pixel Shader (PS)	25
2.8.5. Output Merger(OM)	26
3. Felhasználói dokumentáció	27
3.1. A feladat ismertetése	27
3.2. Hardveres rendszerkövetelmények	28
3.3. Szoftveres követelmények	28
3.4. A program futtatása	28
3.5. A program használatának menete	29
3.6. A felhasználói felület áttekintése	29
3.6.1. Menü	30
3.6.2. Általános beállítási ablak	31
3.6.3. Barangolás mód beállítási ablaka, Explore 3D settings	35
3.6.4. Körséta mód beállítási ablaka, Flythrough settings	36
3.6.5. Irányítás	36
3.7. Támogatott formátumok	37
3.7.1. Felületháló fájl	37
3.7.2. Kamera trajektória fájl	38
3.7.3. Konfigurációs fájl	38
4. Fejlesztői dokumentáció	40
4.1. Funkcionális követelmények	40
4.1.1. Bemeneti adatokra vonatkozó követelmények	40
4.1.2. Szimulációra, megjelenítésre vonatkozó követelmények	41
4.1.3. Kimeneti adatokra vonatkozó követelmények	43
4.2. Nem funkcionális követelmények	43
4.3. Felhasználói eset diagram	44
4.4. Felhasználói történetek	45
4.4.1. Külső állományok betöltése	45
4.4.2. Barangolás	46
4.4.3. Körséta, trajektória lejátszása	46
4.5. Felhasznált fejlesztési eszközök ismertetése	47

4.5.1. Felhasznált könyvtárak bemutatása	49
4.6. Program felépítése	50
4.6.1. A program architektúrája	50
4.6.2. A program belépési pontja	51
4.6.3. Az App osztály	51
4.6.4. A RenderWindow osztály	53
4.6.5. A vezérlő réteg	56
4.6.6. A nézet réteg	62
4.6.7. A modell réteg	67
4.6.8. Perzisztencia	73
4.6.9. Felülethálók, vonalláncok	75
4.6.10. Kamera	76
4.6.11. Barangolás mód kamera kezelése	77
4.6.12. Körséta mód kamera kezelése	77
4.6.13. A fény osztály	79
4.7. Tesztelés	79
4.7.1. Az üzleti logika tesztelése egységesztekkel	80
4.7.2. Modell réteg tesztelése	80
4.7.3. Perzisztencia ellenőrzése	80
4.7.4. Segédosztályok ellenőrzése	81
4.7.5. IRenderable interfész leszármazott osztályainak ellenőrzése . .	82
4.7.6. Kamerakezelés ellenőrzése	83
4.7.7. A fényirány ellenőrzése	85
4.7.8. A modell komponens ellenőrzése	85
4.7.9. A vezérlő réteg tesztelése	87
4.7.10. A megjelenítés ellenőrzése	88
5. Összegzés	90
Köszönyetnyilvánítás	92
Irodalomjegyzék	93
Ábrajegyzék	97
Táblázatjegyzék	98

Algoritmusjegyzék	99
Forráskódjegyzék	100

1. fejezet

Bevezetés

Építésmérnökként szerettem volna az eddigi szakmai pályafutásomhoz kapcsolódó témát választani. Az építészben a térkompozíció alapvető fontosságú, hiszen az épületek és építmények kialakítása során azok formai és térbeli összefüggéseit kell megtervezni. A tervező az ábrázoló geometria szabályaira támaszkodva készíti el a terveket. Az 1960-as évek előtt párhuzamvonalzóval, papírra rajzolt terverek készültek. 1963-ban Ivan Sutherland doktori disszertációjában bemutatta a Sketchpad számítógépes programot, amelyet a mai számítógéppel támogatott tervezési (CAD) programok őseként tartanak számon [1]. Mára pedig eljutottunk az épületinformációs modellezésig (angolul *Building Information Modeling*, továbbiakban BIM), ahol egy létesítmény fizikai és funkcionális jellemzőit digitális formában ábrázoljuk. Jelenleg mind az építészeti tervek, mind a szakági tervek, mind az anyagkimutatás, de a tervek ellenőrzése is a BIM modell segítségével történik. A mérnöki tevékenységek, főként azok ábrázolásai szoros összefüggést mutatnak a számítógépes grafikával. Kézenfekvő volt, hogy a számítógépes grafika témakörében keressek magamnak témát.

A szakdolgozat keretében a SZTAKI önvezető drónok fejlesztésével foglalkozó részlegének munkájába kapcsolódtam be, az általam készített program a kutatás egyik elemét képezi. A szoftvert az automata drónok fejlesztésében részt vevő kutatók arra fogják használni, hogy segítsen a drónok helyzetének validálásában, hibák kiszűrésében. Mindez úgy, hogy a kutatók a program által készített képeket összevetik a drónok által repülés közben készített felvételekkel.

Az önvezető drónok fejlesztése az utóbbi években egyre nagyobb figyelmet élvező kutatási terület. Az ilyen drónok alkalmazása széles körben elterjed, többek között

használják épületek felügyeletére, mezőgazdasági munkák elvégzésére, csomagkézbesítésre, katonai és természetvédelmi feladatokra is. Az önvezető drónok további fejlesztése érdekében számos kutatói csoport dolgozik azon, hogy az automata repülőgépek egyre pontosabbak, hatékonyabbak és biztonságosabbak legyenek.

A kijelölt célterület felett a drónok repülés közben fedélzeti kamerájukkal képeket készítenek, amelyeken kutatók *szemantikus szegmentációt* hajtanak végre. A szemantikus szegmentáció egy olyan képfeldolgozási eljárás, amely során egy adott képen minden egyes pixelt az általa reprezentált tartalomtípus szerint csoportosítanak. A kép pixeleit egy előre meghatározott kategóriához rendelik, mint például növényzet, földfelszín, épületek. Ezeket a kategóriákat szegmentációs osztálynak nevezzük.

A célterületről Lidar, aktív távérzékelési technológiával georeferált pontfelhőt készítenek. A pontfelhő a felszín és a felszínen lévő objektumok (épületek, tátvezetékek, fák stb.) magassági értékeit jelenti. A lézerszkennelt pontfelhőből pedig háromszögelt térbeli hálót generálnak. A háromszögelt, más néven *triangulált felületháló* egy olyan 3D-s modell, amely egy felületet háromszögekkel oszt fel. A háromszögek egymással érintkezve alkotják a felületet, ennek köszönhetően alkalmas az összetett felületek modellezésére. A kutatás jelenlegi állapotában kézzel történik a háromszögelt térháló szemantikus szegmentálása. A jövőben várhatóan egy neurális háló fogja elvégezni ezt a feladatot. Az eljárás során a térbeli modell elemeit a megfelelő logikai osztályokba csoportosítják, és az egyes logikai osztályok külön-külön fájlba kerülnek.

A szakdolgozatban szereplő szoftver feladata, hogy a területről készített 3D-s háromszögelt térhálón, a Nap pozíójának ismeretében, meghatározott *kamera trajektória* mentén szimulációt hajtson végre. A szimuláció során a program a megfelelő fénybeállításokkal felvételeket készít a szegmentált térhálóról. A kamera trajektória magában foglalja a kamerák mozgásának útvonalát, melyet diszkrét pontok sorozatával határozunk meg. minden egyes ponthoz hozzárendelünk egy időpontot, a térbeli koordinátáit, valamint az *Euler-szögeket*, melyek a koordináta-rendszer tengelyei körüli forgatási szögeket jelölnek. Az Euler-szögek tehát segítenek a kamera elmozdulásának pontos meghatározásában.

A szoftvert felhasználva a kutatóknak lehetősége van a drónok által készített felvételek és a program szimulációs képeinek összehasonlítására, így valósítva meg a kamera alapú navigációs szenzor hiba detekcióját. Az összehasonlítást végző szoftver

nem a szakdolgozat kereteiben készül. Az összehasonlítás eredményeként meghatározható, hogy a drón a tervek szerint halad-e, vagy eltér a megírt repülési tervtől.

2. fejezet

Elmélet

A program feladata, hogy egy szemantikusan szegmentált térhálóról stilizált látványt készítsen adott időpontban, adott GPS lokáción. Egy térbeli jelenet képernyőn való megjelenítéséhez meg kell határoznunk:

- megjelenítendő tárgyak geometriai és optikai modelljét,
- a virtuális kamera adatokat,
- a színtér fényforrásait és a megvilágítási adatokat.

A 2. fejezet a téma megértéséhez szükséges elméleti ismereteket részletezi.

A térbeli jelenet három elemét egy közös koordináta-rendszerben helyezzük el. Elsőként a program saját, derékszögű koordináta-rendszerét szükséges definiálnunk, amelyben elhelyezhetjük a jelenet elemeit. Egy derékszögű koordináta-rendszer meghatározza az origójának helyzete és a 3 tengelyének iránya. A különböző koordináta-rendszereket és az ezek közötti transzformációkat mutatja be a 2.2 és a 2.3 fejezet.

A felületháló modellek a saját lokális koordináta-rendszerükben szokás definiálni. A felülethálót lokális és világ koordináta-rendszer közötti transzformációk sorozatával helyezhetjük el a program viszonyítási rendszerében. Ahhoz, hogy a felületháló térbeli viszonyait értelmezni tudjuk, árnyalt megjelenítésre van szükségünk. A térbeli jelenetek árnyalását a 2.7 fejezet ismerteti. A megfelelő árnyalás számításához szükségünk van a felületek normálvektorára. A normálvektorok számításáról és súlyozásáról szól a 2.5 fejezet.

A virtuális kamerát a pozíciójának és tájolásának meghatározásával, valamint a látószöggel definiálhatjuk. A pozíció a világ koordináta-rendszer egy adott pontja, a tájolását pedig a tengelyek körüli forgatási szögek, az ún. Euler-szögek segítsé-

gével határozzuk meg. Ahhoz, hogy a kamerán végzett transzformációkat követően meg tudjuk határozni az Euler-szögeket, a transzformációs mátrixon kell különböző műveleteket végrehajtanunk. Mindezkről a 2.4 fejezetben olvashatunk.

A világkoordináta-rendszerben az időpont és a GPS lokáció alapján szimulálhatjuk a modell környezetét vagyis a színtér fényforrásait. Ehhez a tér és az idő függvényében meghatározzuk a Nap helyzetét, vagyis a Napból származó párhuzamos megvilágítás irányát. A Nap pozíciójának számítását a 2.6 fejezet, az idő ábrázolásának különböző módszereit 2.1 fejezet mutatja be.

A választott grafikus API, DirectX grafikus csővezetéke kis mértékben eltér az OpenGL API-ban látottaktól. A 2.8 fejezet sorra veszi a program által használt csővezeték állomásait.

2.1. Az idő ábrázolása

Az idő ábrázolására többféle rendszer használatos. A különböző ábrázolási módsok eltérő célterületeknek felelnek meg. Az ábrázolási módszerek közötti átjárás biztosított, azonban nem minden esetben lehetséges az egy-egyértelmű megfeleltetés.

2.1.1. Egyezményes koordinált világidő, UTC

Az egyezményes koordinált világidő (Coordinated Universal Time, UTC) az órák és az idő világszerte történő szabályozásának fő szabványa. 1961-ben váltotta fel a greenwichi középidőt (GMT) [2]. Az UTC az egyenletesen mért nemzetközi atomidőből származik [3, 4].

Az UTC nem veszi figyelembe a nyári időszámítást, ezért az nem változik a napfordulók alkalmával. Az egyes világidőzónák pozitív vagy negatív eltolásokkal vannak meghatározva az UTC-hez viszonyítva. Bár az UTC-t nem befolyásolja a nyári időszámítás, azonban a helyi idő változhat a nyári időszámítás miatt. Emiatt az UTC és a helyi idő közötti eltérés évszakonként változhat. Magyarország télen UTC+1 időt, nyáron pedig UTC+2 időt használ.

Az UTC az időt napokra, a napokat óráakra, percekre és másodpercekre bontja fel. minden nap 24 órából áll, és minden óra 60 percből. Azonban az 1 percen belüli másodpercek száma változó lehet. Az UTC napok többségénél minden perc

60 másodpercből áll, azonban bizonyos esetekben egy nap utolsó perce állhat 59 vagy 61 másodpercből is. A másodperc és a másodpercnél kisebb időegységek minden állandó időtartamúak. Az átlagos UTC nap pontosan 86 400 SI másodpercet tartalmaz, percenként pontosan 60 másodperccel. Azonban a szoláris középnap valamivel hosszabb, mint 86 400 SI másodperc. Ezért van szükség arra, hogy időnként az UTC nap utolsó perce 61 másodpercre módosuljon. Az extra másodpercet nevezzük szökőmásodpercnek. Egy UTC nap utolsó perce 59 másodpercet tartalmazhat, hogy fedezze a Föld gyorsabb forgásának távoli lehetőségét, de erre még nem volt szükség. Ennek köszönhetően az UTC körülbelül egy másodpercen belül van a Föld forgásából számolt egyezményes világidőhöz (Universal Time, UT) képest a 0° hosszúságon [2, 3, 4].

Az UTC napok megfeleltethetőek a Gergely-naptár szerint. Azonban a csillagászatban a számításokhoz a Julián-dátumot használják, mert az UTC szökőmásodpercei nehezen kivitelezhetővé tennék a számításokat. Az UTC szabálytalan naphosszai miatt a Julián-dátum és az UTC közötti átváltás bizonyos számításokhoz bonyolult.

2.1.2. Unix-idő

A Unix-idő az UTC szerinti 1970. január 1. 00:00:00 óta eltelt másodpercek száma. A Unix-idő minden nap 86 400 másodperccel nő, a szökőmásodperceket is figyelembe véve. A Unix operációs rendszerek mellett számos alkalmazásban használatos. Az előző fejezetben ismertetett szökőmásodpercek miatt az Unix-idő nem valódi reprezentációja a UTC-nek. A két ábrázolás között nincs egy-egyértelmű megfeleltetés [5].

2.1.3. Julián-dátum

A csillagászati számításoknál használt Julián-dátum a Kr. e. 4713. év első napjától eltelt napok számát és a nap decimális törtrészeit használja az időpontok megadására óra-perc-másodperc helyett. Mint bármely naptártól független periódus, a Julián-dátum közvetítőként használható egyik naptári rendszerről egy másikra való gyors átszámításhoz. A Julián-dátum kezdete Kr. e. 4713. január 1. dele. Ezt azonosítják a 0. Julián-nappal. minden további napot egy egész számmal azonosítanak, amely azt jelzi, hogy az első Julián-naptól az adott napig hány nap telt el. Ezt a egész számot nevezzük Julián-napnak. (Julian day number, JDN). A Julián-dátum

(Julian date, JD) egy Julián-nap plusz a megelőző nap delétől eltelt idő decimális törtrészben kifejezve [6].

2.2. Koordináta-rendszerek

A navigációs rendszerekben a repülőgépek mozgásának leírására többféle koordináta-rendszer használatos. Ennek oka az, hogy a különböző jelenségek leírása egyszerűbb különböző koordináta-rendszereket alkalmazva.

2.2.1. Föld-központú, Földhöz rögzített koordináta-rendszer

A Föld-központú, Földhöz rögzített koordináta-rendszer (Earth-centered, Earth-fixed, ECEF), más néven geocentrikus koordináta-rendszer, egy derékszögű térbeli vonatkoztatási rendszer. A koordináta-rendszer origóját a Föld középpontjához rögzítik, Z tengelye az északi pólus felé mutat és azon halad át, X és Y tengelye az egyenlítő síkjában fekszik és X tengelye a fő meridiánon (greenwichi délkör) halad át. Segítségével Föld felszín alatti és feletti pontok is ábrázolhatók az X, Y, Z koordináták megadásával. A GPS alapvetően ezt használja, mert nem szükséges hozzá egy ellipszoid modell használata, csak a Föld középpont pozíciójának és a rendszer orientációjának ismerete [7].

2.2.2. Földrajzi koordináta-rendszer

A földrajzi koordináta-rendszer (Geographic Coordinate System, GCS) egy ellipszoid koordináta-rendszer, amely a Földön lévő pozíciók mérésére szolgál, szélességi és hosszúsági fokban kifejezve. A referencia modelltől mért távolságot pedig a magassággal jelöljük, jele: h. A három koordinátát a gyakorlatban szokás LLA (Latitude, Longitude, Altitude azaz szélesség, hosszúság, magasság) koordinátáknak nevezni [7].

Földrajzi szélesség

Földrajzi szélesség angolul latitude, jele: ϕ .

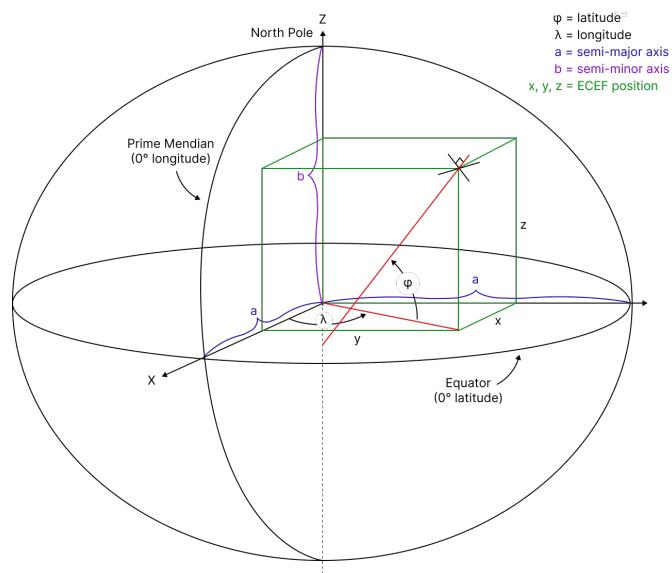
Egy adott P pont szélessége egyenlő a ponton és a Föld középpontján átmenő egyenes és az Egyenlítő síkja által bezárt szöggel fokban kifejezve. Megállapodás alapján északi irányba pozitív, déli irányba negatív az érték előjele. Az azonos szélességű pontok egyazon szélességi körön vannak. A szélességi körök által meghatározott síkok páronként párhuzamosak [8]. Néhány nevezetes szélességi kör: Egyenlítő: $\phi = 0^\circ$, Északi-sark: $\phi = +90^\circ$, Déli-sark: $\phi = -90^\circ$.

Földrajzi hosszúság

Földrajzi hosszúság angolul longitude, jele: λ .

A meridián sík egy adott P pont és a két pólus által meghatározott sík. Megállapodás szerint a kitüntetett kezdő meridián sík, $\lambda = 0^\circ$ a greenwichi obszervatóriumon (Royal Observatory, Greenwich) halad keresztül.

Egy adott P pont földrajzi hosszúsága egyenlő a pont által meghatározott meridián sík és a kitüntetett kezdő meridián sík által bezárt szöggel fokban kifejezve. Megállapodás szerint az érték előjele keleti irányban pozitív, nyugati irányban negatív. Az azonos hosszúságú pontok azonos meridián síkon helyezkednek el. A meridián síkok a definíció miatt nem párhuzamosak [8].



2.1. ábra. Az ECEF koordináták a földrajzi szélességhez és hosszúsághoz viszonyítva
[9]

WGS84

A WGS84 (World Geodetic System) egy geodéziai világrendszer, amely magába foglalja a Föld geometriai, gravitációs (Earth Gravitational Model, EGM) és mágneses erőtér (World Magnetic Model, WMM) modelljét, valamint a földi vonatkoztatási koordináta-rendszert [10]. A WGS84 szabvány a Földet egy ellipszoid-modellel közelíti, az ellipszoid modell paraméterei a 2.3.1-es számú fejezetben olvashatóak [7, 10]. A GPS-műholdak által sugárzott fedélzeti pályaadatok vonatkoztatási rendszere WGS84 geoid modellt alkalmazza [11].

2.2.3. NED koordináta-rendszer

A NED (North, East, Down magyarul észak, kelet, lefelé) a földfelszín egy adott pontjához, (pl. egy adott GPS koordinátához) rögzített helyi jobbsodrású koordináta-rendszer, melynek 1. számú, N tengelye az ellipszoid északi pontja, 2. számú, E tengelye az ellipszoid keleti iránya, 3.számú, D tengelye pedig az ellipszoid belseje felé mutat a helyi normálvektor mentén [7]. Általában kis távolságú, rövid idejű repülések koordináta-rendszereként alkalmazzák.

2.2.4. Grafikus API-k koordináta-rendszerére

A 3D grafikus alkalmazások jellemzően bal, illetve jobbsodrású derékszögű koordináta-rendszert is használnak. A Direct3D alapértelmezett módon balkezes koordináta-rendszert használ, ahol a vízszintes alapsíkot az X és Z tengelyek határozzák meg, a függőleges koordinátákat a felfele mutató Y koordináta tengely mentén ábrázoljuk [12].

2.3. Koordináta-rendszerek közötti transzformáció

2.3.1. Transzformáció WGS84-ből ECEF-be

A transzformáció végrehajtásához ismerni kell a WGS84 által használt ellipszoid modell paramétereit [7]:

- **fél nagytengely:** $a = 6378137.000m$,
- **fél kistengely:** $b = a \cdot (1 - f) = 6356752.314m$,

- **ellipszoid lapultsága:** $f = \frac{1}{298.257\,223\,563}$,
- **első numerikus excentritás:** $e = \sqrt{\frac{a^2 - b^2}{a^2}} = 0.08181919$.

A görbületi sugár számítása az adott ϕ szélességi körön

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}}$$

Koordináták számítása:

$$X = (N + h) \cos \phi \cos \lambda$$

$$Y = (N + h) \cos \phi \sin \lambda$$

$$Z = \left(\frac{b^2}{a^2} N + h \right) \sin \phi$$

2.3.2. Transzformáció ECEF-ből NED-be

A 2.3.1 fejezet számításai alapján rendelkezésünkre állnak az adott P pont ECEF (X , Y , Z) és földrajzi (ϕ , λ) koordinátái. A két derékszögű koordináta-rendszer közötti átváltás egy forgatási és eltolási transzformációval lehetséges [7].

A forgatási transzformáció mátrixa az alábbi, ahol a ϕ a P pont földrajzi szélessége, a λ pedig a földrajzi hosszúsága:

$$R = \begin{bmatrix} -\sin \phi \cos \lambda & -\sin \phi \sin \lambda & \cos \phi \\ -\sin \lambda & \cos \lambda & 0 \\ -\cos \phi \cos \lambda & -\cos \phi \sin \lambda & \sin \phi \end{bmatrix}$$

Ha adottak a P pont ECEF (X , Y , Z) és földrajzi (ϕ , λ) koordinátái, valamint a NED koordináta-rendszer origójának ECEF koordinátái (N_0 , E_0 , D_0), akkor a 2.1 egyenlet segítségével kiszámíthatjuk a P pont N, E, D koordinátáit [7].

$$\begin{bmatrix} N \\ E \\ D \end{bmatrix} = R \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} - \begin{bmatrix} N_0 \\ E_0 \\ D_0 \end{bmatrix} \quad (2.1)$$

2.3.3. Sodrásirány megfordítása

Amint az a 2.2.4 és a 2.2.3 fejezetekben említésre került a két derékszögű koordináta-rendszer sodrásiránya eltérő, amíg a NED jobbsodrású, a Direct3D rendszere alapértelmezetten balsodrású. A NED-ben megadott pozíciókat és a tengelyek

körüli forgatásokat is transzformálnunk kell. A forgatási transformációkat az Euler-szögekkel, a tengelyek körüli forgatási szögekkel határozhatjuk meg [13].

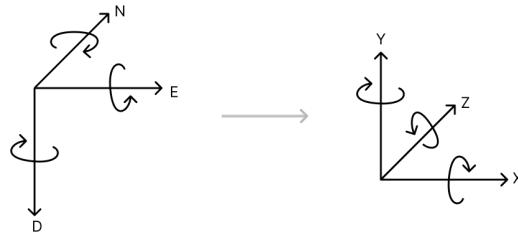
A szögek elnevezései a következők:

- forduló (yaw) a függőleges tengely körüli γ szöggel való forgatás,
- billentés (pitch) az x tengely körüli β szöggel való forgatás,
- csavarás (roll) a z tengely körüli α szöggel való forgatás.

Az Euler-szögekre a továbbiakban az angolszász irodalomban használatos yaw-pitch-roll-ként hivatkozom.

A jobbsodrású koordináta-rendszerben a vektorokat oszlopvektorokként ábrázoljuk és a vektorokon végzett mátrixműveleteket jobbról balra végezzük el.

A balsodrású koordináta-rendszerben a vektorokat sorvektorokként ábrázoljuk és a vektorokon végzett mátrixműveletek végrehajtási sorrendje balról jobbra történik.



2.2. ábra. Jobbsodrású és balsodrású koordináta-rendszer

2.2-es és 2.3-as számú egyenletek segítségével határozhatjuk meg a NED koordinátákból a grafikus rendszer XYZ koordinátáit.

2.4-es egyenletben az Euler szögek transzformálását láthatjuk.

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} N \\ E \\ D \end{bmatrix} = \begin{bmatrix} E \\ -D \\ N \end{bmatrix} \quad (2.2)$$

$$\begin{bmatrix} E \\ -D \\ N \end{bmatrix}^\top = [X \ Y \ Z] \quad (2.3)$$

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} -\beta \\ \gamma \\ -\alpha \end{bmatrix} \quad (2.4)$$

Amennyiben megváltozik a koordináta-rendszer sodrása módosítanunk kell a triangulált felületháló háromszögeinek bejárási irányán is. Ahhoz, hogy megforduljon a körüljárási irány, a csúcsokat fordított sorrendben definiáljuk a balsodrású koordináta-rendszerben.

2.4. Euler-szögek meghatározása forgatási mátrixból

A virtuális kamera útvonalának követése során meg kell határoznunk az Euler-szögeket, amelyek leírják a kamera orientációját a rögzített pontok mentén. Ha a kamera trajektóriát elforgatjuk az eredeti helyzetéhez képest, akkor is meghatározhatjuk a transzformációt követően a kamera szögeit. Ehhez a kamera végső forgatási mátrixából kell kiszámítanunk az Euler-szögeket.

A z tengely körüli forgatási mátrix, ahol az α jelöli z tengely körüli forgatás szögét [13].

$$R_{3DZ} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A x tengely körüli forgatási mátrix, ahol az β jelöli x tengely körüli forgatás szögét [13].

$$R_{3DX} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & \sin(\beta) \\ 0 & -\sin(\beta) & \cos(\beta) \end{bmatrix}$$

A y tengely körüli forgatási mátrix, ahol az γ jelöli y tengely körüli forgatás szögét [13].

$$R_{3DY} = \begin{bmatrix} \cos(\gamma) & 0 & -\sin(\gamma) \\ 0 & 1 & 0 \\ \sin(\gamma) & 0 & \cos(\gamma) \end{bmatrix}$$

A DirectX-ben a forgatásoknak rögzített sorrendje van, először a z, majd az x, majd az y tengely mentén forgatunk [13]. A 2.5 számú egyenlet szerint számíthatjuk ki a forgatási transzformáció mátrixát.

$$\begin{aligned} R &= R_{3DZ} \cdot R_{3DX} \cdot R_{3DY} \\ &= \begin{bmatrix} \sin(\alpha) \sin(\beta) \sin(\gamma) + \cos(\alpha) \cos(\gamma) & \sin(\alpha) \cos(\beta) & \sin(\alpha) \sin(\beta) \cos(\gamma) - \cos(\alpha) \sin(\gamma) \\ \cos(\alpha) \sin(\beta) \sin(\gamma) - \sin(\alpha) \cos(\gamma) & \cos(\alpha) \cos(\beta) & \cos(\alpha) \sin(\beta) \cos(\gamma) + \sin(\alpha) \sin(\gamma) \\ \cos(\beta) \sin(\gamma) & -\sin(\beta) & \cos(\beta) \cos(\gamma) \end{bmatrix} \end{aligned} \quad (2.5)$$

Az Euler-szögek meghatározásához szükséges paraméterek a forgatási mátrix elemei:

$$\begin{aligned} R_{32} &= -\sin(\beta) \\ R_{31} &= \cos(\beta) \sin(\gamma) \\ R_{33} &= \cos(\beta) \cos(\gamma) \\ R_{12} &= \sin(\alpha) \cos(\beta) \\ R_{22} &= \cos(\alpha) \cos(\beta) \end{aligned}$$

Az Euler-szögek meghatározásának első lépése a pitch, azaz az x tengely körüli forgatási szög meghatározása:

$$\beta = \arcsin(-R_{32}) \quad (2.6)$$

A második lépés a yaw, azaz az y tengely körüli forgatási szög meghatározása:

$$\frac{R_{31}}{R_{33}} = \frac{\cos(\beta) \sin(\gamma)}{\cos(\beta) \cos(\gamma)} = \tan(\gamma) \quad (2.7)$$

$$\gamma = \arctan\left(\frac{R_{31}}{R_{33}}\right) \quad (2.8)$$

Végül a roll, azaz a z tengely körüli forgatási szög meghatározása:

$$\frac{R_{12}}{R_{22}} = \frac{\sin(\alpha) \cos(\beta)}{\cos(\alpha) \cos(\beta)} = \tan(\alpha) \quad (2.9)$$

$$\alpha = \arctan\left(\frac{R_{12}}{R_{22}}\right) \quad (2.10)$$

2.5. Normálvektorok súlyozása

Az interpolált árnyaláshoz szükségünk van a triangulált felületháló csúcspontjaiban mért normálvektorokra. A csúcshoz tartozó normálvektor az adott csúcsot tartalmazó oldalak (facet) normálértékeinek súlyozott összege. A súlyok meghatározására többféle módszer létezik, Gouraud [14] egyenlő súlyozást, míg Thürmer és Wülfel [15] a csúcspont szögei alapján történő súlyozást javasol. A Nelson Max [16] tanulmányában egy olyan súlyozást ír le, amely megoldást nyújt arra az esetre, ha a csúcsban találkozó oldalak méretei jelentős eltérést mutatnak. Ezen súlyok képesek egy gömbbe írt felületháló esetén pontosan meghatározni a csúcsokhoz tartozó normálvektorat.

A 2.5 fejezet a Nelson Max-féle tanulmányban leírtakat ismerteti [16].

A bemutatott levezetésben egy gömbbe írt poliéder esetét veszi végig. Adott a poliéder Q csúcsa, a szomszédos csúcsok rendre V_0, V_1, \dots, V_{n-1} . A poliéder helyezzük egy Q középpontú koordináta-rendszerbe, így a szomszédos csúcsokba mutató vektorok rendre V_0, V_1, \dots, V_{n-1} , ahol $V_i = (x_i, y_i, z_i)$. Valamennyi szomszédos csúcsba mutató vektor, meghatároz egy, a vektorra merőleges síkot, az adott sík normálvektoraként. A V_i vektorhoz tartozó sík egyenlete:

$$x_i \cdot x + y_i \cdot y + z_i \cdot z = |V_i|^2/2 \quad (2.11)$$

Tegyük fel, hogy az $n = 3$, ebben az esetben a három síkhöz tartozó egyenletrendszer az alábbi formában írható fel.

$$\begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} |V_0|^2/2 \\ |V_1|^2/2 \\ |V_2|^2/2 \end{bmatrix} \quad (2.12)$$

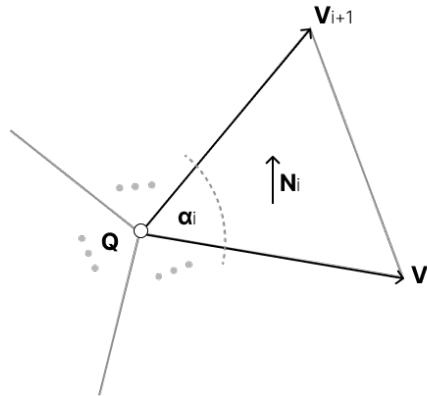
A Cramer [17] szabály szerint az 2.12 egyenletrendszer átalakítva a 2.13 egyenletet kapjuk.

$$C = (x, y, z) = (|V_2|^2 V_0 \times V_1 + |V_0|^2 V_1 \times V_2 + |V_1|^2 V_2 \times V_0) \cdot \frac{1}{2D} \quad (2.13)$$

A 2.13 egyenletben szereplő D a 2.12 egyenlet együttható mátrixának determinánsa. A 2.13 egyenlet C megoldásvektorát a $|V_0|^2 |V_1|^2 |V_2|^2 / (2D)$ értékkel leosztva megkapjuk a Q csúcshoz tartozó súlyozott normálvektort. 2.14 egyenletben szereplő c a normálvektorhoz tartozó súly érték, $c \in \mathbb{R}$.

$$\left(\frac{V_0 \times V_1}{|V_0|^2 |V_1|^2} + \frac{V_1 \times V_2}{|V_1|^2 |V_2|^2} + \frac{V_2 \times V_0}{|V_2|^2 |V_0|^2} \right) = cN \quad (2.14)$$

Az 2.14 egyenletet fejezzük ki a Q csúcsot meghatározó oldalakhoz tartozó normálvektorok lineáris kombinációjaként. A $V_i Q V_{i+1}$ csúcsok által meghatározott sík normálvektora a 2.15 egyenlet szerint számítható, ahol az α_i a V_i és V_{i+1} vektorok közötti szög.



2.3. ábra. $V_i Q V_{i+1}$ csúcsok által meghatározott sík normálvektora

$$N_i = \frac{V_i \times V_{i+1}}{|V_i| |V_{i+1}| \sin(\alpha_i)} \quad (2.15)$$

Átrendezve

$$V_i \times V_{i+1} = N_i |V_i| |V_{i+1}| \sin(\alpha_i) \quad (2.16)$$

2.14 egyenletbe behelyettesítve az N_i értékeket, az 2.17 egyenletet kapjuk.

$$\sum_{i=0}^2 \frac{V_i \times V_{i+1}}{\langle V_i, V_i \rangle \langle V_{i+1}, V_{i+1} \rangle} = \sum_{i=0}^2 \frac{V_i \times V_{i+1}}{|V_i|^2 |V_{i+1}|^2} = \sum_{i=0}^2 \frac{N_i \sin(\alpha_i)}{|V_i| |V_{i+1}|} = cN \quad (2.17)$$

A gyakorlatban a 2.17 egyenlet bal oldali részével célszerűbb számolni, hiszen az nem tartalmaz sem trigonometrikus függvényeket, sem négyzetgyököt.

Teljes indukcióval bizonyítható, hogy bármely $n \geq 3$ -ra teljesül, ezt mutatja a 2.18 egyenlet.

$$\sum_{i=0}^2 \frac{V_i \times V_{i+1}}{|V_i|^2 |V_{i+1}|^2} = \sum_{i=0}^2 \frac{N_i \sin(\alpha_i)}{|V_i| |V_{i+1}|} = cN \quad (2.18)$$

A fenti levezetés csak a gömbökbe írt poliéderekre vonatkozik, és a normálvektor becslés még ellipszoidokra sem ad pontos értéket. A Nelson Max azonban fenntartja, hogy az általa bevezetett súlyozás jobb, mint más becslések, mert kezeli azokat az eseteket, amikor a Q csúcspontba futó élek nagyon eltérő hosszúságúak.

Tanulmány végén a különböző súlyozási módszerrel mért hibákat veti össze. A szerző 1 000 000 különböző felületen végezte a mérést. A felületek egyenlete:

$$f(x, y) = Ax^2 + Bxy + Cy^2 + Dx^3 + Ex^2y + Fxy^2 + Gy^3$$

Az egyenletben szereplő együtthatók egyenletes eloszlású pszeudovéletlen számok, és $A, B, C, D, E, F, G \in [-0.1, 0.1]$. Az összehasonlításban mind a Gouraud módszerénél, mind a Thürmer és Wüthrich módszerénél kisebb hiba értékeket mutatott ki.

2.6. Nap pozíciója

A 2.6 fejezet a "Computing Solar Vector" [18] tudományos cikk alapján készült, a cikk a Nap pozíójának meghatározásának algoritmusát veszi végig. Az algoritmus lépései a cikkben részletesen ki vannak fejtve, ezeket a lépéseket 2.6 fejezet foglalja össze.

A Nap helyzetének meghatározása egy adott helyen, egy adott időpontban az alábbi lépések sorozatával lehetséges.

1. A Nap helyzetének számítása az ekliptikai koordináta-rendszerben. A csillagászatban az ekliptikai koordináta-rendszer egy égi koordináta-rendszer, amelyet általában a Naprendszer objektumai látszólagos helyzetének, pályájának és pólusirányának ábrázolására használnak.
2. Átváltás ekvatoriális koordináta-rendszerbe.
3. Átváltás horizontális koordináta-rendszerbe. A horizontális koordináta-rendszerben a Nap helyzete az adott helyi égbolton jelenik meg, azaz az irányt és a magasságot jelöli meg, ahonnan a Napot megfigyeljük. A számításhoz szükség van a megfigyelő helyének geográfiai koordinátáira, valamint a megfigyelés időpontjának pontos ismeretére.

2.6.1. Ekliptikai koordináták

A 2.6.1 fejezetben az ekliptikai koordináták meghatározására kerül sor. Az ekvatoriális koordináták számításához a Nap ekliptikai hosszúságának (l) és az ekliptika hajlásának (ϵ) ismerete szükséges.

A 2.1.3 fejezetben említésre került, hogy a csillagászatban használt időábrázolási forma az ún. Julián-dátum. Elsőként az univerzális időben (UT) megadott megfigyelési időpont alapján meghatározzuk a Julián-dátumot. A 2.19 egyenlettel a Julián-dátum egész része, a Julián-nap kerül meghatározásra. A 2.19 egyenletben kizárálag egész osztásokat végzünk, az egyenletben szereplő y a megfigyelés UT-ben megadott évét, a m a hónapját, a d a napját jelölik. A 2.20 egyenletben a Julián-dátum tizedes részének meghatározására kerül sor.

A 2.21 egyenlet greenwichi dél óta (2000. január 1.) eltelt napok számát határozza meg a töredéknapot is beleértve. Ezt a számot jelöljük n -nel.

A további egyenletekben használt jelölések:

- L - a perihélium ekliptikai hosszúsága (mean longitude of the Sun),
- g - a Nap közép anomália (mean anomaly of the Sun),
- l - a Nap ekliptikai hosszúsága (the ecliptic longitude of the Sun),
- ϵ - az ekliptika hajlása (obliquity of the ecliptic).

$$\begin{aligned}
 jdn &= (1461 \cdot (y + 4800 + (m - 14)/12))/4 \\
 &+ (367 \cdot (m - 2 - 12 \cdot ((m - 14)/12)))/12 \\
 &- (3 \cdot ((y + 4900 + (m - 14)/12)/100))/4 \\
 &+ d - 32075
 \end{aligned} \tag{2.19}$$

$$jd = jdn + (hour - 12.0)/(24.0) + min/1440.0 + sec/86400.0 \tag{2.20}$$

$$n = jd - 2451545.0 \tag{2.21}$$

$$\Omega = 2.1429 \cdot 0.0010394594 \cdot n \tag{2.22}$$

$$L = 4.895063 + 0.017202791698 \cdot n \tag{2.23}$$

$$g = 6.2400600 + 0.017202791698 \cdot n \tag{2.24}$$

$$\begin{aligned}
 l &= L + 0.03341607 \sin(g) + 0.00034894 \sin(2g) \\
 &- 0.0001134 - 0.0000203 \sin(\Omega)
 \end{aligned} \tag{2.25}$$

$$\epsilon = 0.4090928 - \text{degToRad}(0.0000004) \cdot n + 0.0000396 \cdot \cos(\Omega) \tag{2.26}$$

2.6.2. Ekvatoriális koordináták

Az ekliptikai koordináták átváltása az ekvatoriális koordináta-rendszerbe a 2.27 és a 2.28 egyenletek segítségével történik. Az α a rektaszcenziót (angolul: right ascension), a δ pedig a Nap deklinációját (angolul: declination) jelöli. A deklináció és a rektaszcenzió a II. ekvatoriális koordináta-rendszer két koordinátája.

$$\alpha = \arctan \left(\frac{\cos(\epsilon) \cdot \sin(l)}{\cos(l)} \right) \tag{2.27}$$

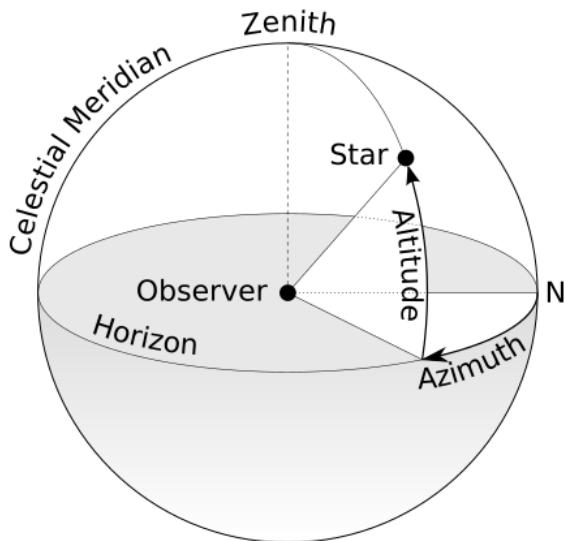
$$\delta = \arcsin (\sin \epsilon \cdot \sin l) \tag{2.28}$$

2.6.3. Horizontális koordináták

A horizontális koordináta-rendszer egy olyan csillagászati vonatkoztatási rendszer, amely szerint az adott megfigyelési pontból az égbolton látható objektumokat egy gömbfelületen látjuk. Az égitestek látszólagos pozícióját ezen a gömbfelületen helyezzük el. A pozíció megadásához két szög koordináta megadására van szükségünk: az azimutra (γ) és a napmagasságra (α_s). A napmagasság helyett szokás a zenitszöget (Θ_z) megadni.

$$\Theta_z + \alpha_s = \frac{\pi}{2} \tag{2.29}$$

A napmagasság a megfigyelő és az objektumot összekötő egyenes és a horizont által bezárt szög. Az azimut az északi irány és megfigyelőt az objektummal összekötő egyenes horizontra vetett képe közötti szög, balsodrású koordináta-rendszerben.



2.4. ábra. Azimut és napmagasság
[19]

A további egyenletekben használt jelölések:

- $gmst$ - greenwichi közép sziderikus idő (Greenwich mean sidereal time),
- $lmst$ - helyi közép sziderikus idő (local mean sidereal time),
- ω - az óraszög (Hour angle).

$$gmst = 6.6974243242 + 0.0657098283 \cdot n + hour \quad (2.30)$$

$$lmst = (gmst \cdot 15 + \lambda) \cdot (\pi/180) \quad (2.31)$$

$$\omega = lmst - \alpha \quad (2.32)$$

$$\Theta_z = \arccos(\cos(\phi) \cos(\omega) \cos(\delta) + \sin(\delta) * \sin(\phi)) \quad (2.33)$$

$$\gamma = \arctan\left(\frac{-\sin(\omega)}{\tan(\delta) \cos(\phi) - \sin(\phi) * \cos(\omega)}\right) \quad (2.34)$$

2.7. Árnyalás

2.7.1. Fényforrások

A grafikus API-kban több típusú fényforrással találkozhatunk, az egyik a szort háttérvilágítás (ambient light) a másik a direkt megvilágítás, ami magába foglalja a pontszerű, a reflektorszerű és a párhuzamos megvilágítást. Ezek közül részletesebben a szort háttérvilágítás és a párhuzamos megvilágítás kerül ismertetésre.

A szort háttérvilágítás esetén a modell minden irányból azonos mértékű, egyenletes megvilágítást kap. Ebben az esetben kizárolag a megvilágítást biztosító fény

színét és intenzitását határozzuk meg.

A modell megvilágítását elsősorban a Nap biztosítja. A számítógépes grafikában a Nap általi megvilágítást a párhuzamos megvilágításként modellezzük. A fénysugarak párhuzamosak és változatlan intenzitásúak. Az árnyalás számítása során az intenzitás változást nem vesszük figyelembe.

2.7.2. Megvilágítási modellek

A megvilágítási modellek segítségével írjuk le a fény és az objektumok interakcióját. A programban megjelenített felülethálókat tökéletesen diffúz anyagként kezeljük. Diffúz tükröződésről akkor beszélünk, amikor a felület a fénysugarakat minden irányban azonos intenzitással veri vissza. A jelenséget a Lambert-féle távolság és koszinuszfüggvényekkel modellezhetjük [13].

A Lambert-féle távolság-függvényt a 2.35 egyenlet írja le. A távolság-függvény szerint az objektumhoz érkező fény intenzitása fordítottan arányos a fényforrás és az objektum távolságának (d) négyzetével, ahol I_0 a fényforrásnál mért fényintenzitás [13]. Párhuzamos megvilágítás esetén feltételezzük, hogy a fényintenzitás változatlan, emiatt a távolság függvény nem kerül számításra.

$$I = \frac{I_0}{d^2} \quad (2.35)$$

A Lambert-féle koszinusz-függényt a 2.36 egyenlet írja le. A koszinusz-függény szerint a visszavert fény intenzitása az objektum felületéhez érkező fény beesési szögétől függ. Matematikai értelemben ezt a felület normálvektorának (n) és a fényvektorának (L) skaláris szorzataként írhatjuk le [13]. A fényvektor a felület pontjából a fényforrás irányába mutató vektor.

$$f(\Theta) = \max(\cos(\Theta), 0) = \max(\langle n, L \rangle, 0) \quad (2.36)$$

2.7.3. Árnyalási modellek

Az árnyalási modellek szerint színezzük a triangulált felületháló háromszögeit. Az árnyalási technikák közül az alábbi kettőt emelném ki.

Konstans árnyalás (flat shading): A csúcspontok közötti területet mindenhol azonos színértékkel töltjük ki.

Interpolált árnyalás (interpolative shading): A csúcspontok közötti területet különböző algoritmusok szerint különböző fényértékekkel számítjuk ki.

A 2.36. egyenlet szerint a diffúz megvilágítási modellben az adott pontnál a kilépő fény intenzitását a felület normálvektora és a fényvektor skaláris szorzata határozza meg. Ahhoz, hogy a modellünket interpolált árnyalással jeleníthessük meg, szükségünk van a felületháló csúcspontjaiban mért normálvektorokra. A normálvektorok számítását a 2.5 fejezet veszi végig.

2.8. Grafikus csővezeték

A grafikus csővezeték leírása az "Introduction to 3D Game Programming with DirectX 11"[12] könyv alapján készült. A grafikus API-k a grafikus adatok képernyőn való megjelenítésének feladatát részfeladatokra bontják. Az adatok egymástól függetlenül haladnak át a grafikus csővezetéken (graphics pipeline), ahol minden egyes állomásnak megvan a maga feladata, a maga felelősségi köre. minden állomás bemenete a megelőző feldolgozó egység kimenete. Vagyis az s_i kimenete az s_{i+1} bemenete. A grafikus csővezeték végzi el a színtér megjelenítését a megadott bemeneti adatok alapján. A csővezeték bemeneti adatai magukba foglalják a megjelenítendő tárgyak geometriai és optikai modelljeit, virtuális kamera adatokat, a képernetet, a színtér fényforrásait és a megvilágítási adatokat. A grafikus API-k lehetővé teszik a fejlesztők számára az egyes állomások finomhangolását és testreszabását. A DirectX több feldolgozó egységgel rendelkezik, amelyek közül az alábbi öt kerül részletes bemutatásra: Input Assembler, Vertex Shader, Rasterizer, Pixel Shader és Output Merger.

2.8.1. Input Assembler (IA)

Az Input Assembler (IA) felelős az adatok összegyűjtéséért és feldolgozásáért, amelyeket aztán továbbít a következő állomásnak. Kiolvassa a csúcspontokat a memoriából, és létrehozza belőlük a geometriát. Valamennyi geometriai elemet egyedi azonosítóval lát el, majd továbbítja a csővezetéken. A feladatok elvégzése nagyrészt automatikusan történik, a fejlesztőnek az alábbi lehetőségei vannak az Input Assembler működésének beállítására.

Az adatok kiolvasásához a fejlesztőnek meg kell határoznia a csúcspont és az index puffer memóriacímét (`IASetVertexBuffers`, `IASetIndexBuffers`). Ahhoz, hogy a memóriában található adatokat a GPU helyesen értelmezze meg kell adnunk úgynevezett Input Layout-ot (`IASetInputLayout`). Itt definiálhatjuk az adatok sorrendjét és típusát, például: az egy csúcsponthoz tartozó adatok: pozícióvektor, normálvektor, színvektor.

A geometriai primitívek topológiájának meghatározásával megadható, hogy a pontok sorozatából milyen geometriát építsen fel az IA, például háromszögek sorozatát, vonalláncot vagy vonalak sorozatát (`IASetPrimitiveTopology`).

2.8.2. Vertex Shader (VS)

A Vertex Shader olyan program, amely minden egyes csúcspontot feldolgoz és kiszámítja annak helyzetét a 3D térben.

A Vertex Shader a bemeneti regiszterekbe várja a csúcspont adatokat. Ezekből a regiszterekből olvassa ki például a csúcsponthoz tartozó pozícióvektort, normálvektort és/vagy színvektort. Egyéb paramétereket, amelyek nem a csúcspont tulajdonságait írják le, például a megvilágítás paramétereit, a konstans és ideiglenes regiszterekből olvassa ki.

A vertex shader eredményeit a kimeneti regiszterekbe helyezi. A kötelezően feltöltendő regiszter az oPos regiszter, amelybe a csúcspont homogén képernyőkoordinátái kerülnek. Ez a pozíció vektoron végrehajtott világ-, nézet- és vetítési transzformációk egymás utáni sorozatával kapjuk meg.

2.8.3. Rasterizer

A Rasterizer a vágóvonalakat számolja ki, amelyek meghatározzák, hogy melyik elem melyik része jelenik meg a képernyőn, majd véglegesíti a takarási viszonyokat. Többek között ebben a szakaszban kerülnek kitöltésre a csúcspontok közötti felületek képpontokkal, interpolációs technikával.

2.8.4. Pixel Shader (PS)

A Pixel Shader a Rasterizer után kerül végrehajtásra. Feladata, hogy kiszámolja minden egyes pixel színét, és itt végezhetők el a textúrázási, pixelenkénti megvilágítási és egyéb utófeldolgozási műveletek, például az elmosódás beállítása. A Pixel

Shader a szín- és textúra-regiszterekből kapja meg az interpolált szín- és textúra adatokat, amelyeket a Rasterizer generál. Az ideiglenes és konstans regisztereket további paraméterek átadására használhatjuk. A Pixel Shader eredménye a pixel színértéke, amely a színpufferbe kerül. A színpuffer a képernyőn ténylegesen megjelenített kép előállításának utolsó fázisa.

2.8.5. Output Merger(OM)

Az Output Merger a grafikus csővezeték utolsó része. Az OM összegyűjti az összes Pixel Shader által előállított szín értékeket, majd azokat összeilleszti a végleges képpé. Az OM-nak van lehetősége a végleges kép felülvizsgálatára és manipulálására is, például a színmaszkok és átlátszóságok beállításával.

3. fejezet

Felhasználói dokumentáció

3.1. A feladat ismertetése

A program célja, hogy adott földrajzi koordinátáknál, adott UTC időpontban szimulációt készítsen egy szemantikusan szegmentált térhálóról. A program szimuláció során a földrajzi koordináták és az időpont függvényében meghatározza az aktuális Nap pozíciót, majd ennek megfelelően árnyalja a betöltött felülethálót.

A program elsődleges felhasználói az önvezető drónok fejlesztésében részt vevő kutatók, akik a szimulációról készített felvételeket felhasználhatják a drónok szenzor hiba detektálására. A kutatásról részletesebben az 1. fejezetben olvashatunk.

A program grafikus felhasználói felületet kínál, ahol a felhasználó kiválaszthatja a szimulációhoz szükséges állományokat és beállíthatja a szimuláció paramétereit. A felhasználó a billentyűzet és az egér segítségével utasításokat is adhat a programnak a szimuláció közben.

A felhasználó a program használata közben kétféle mód közül választhat:

- Explore 3D mode - Barangolás mód vagy
- Flythrough mode - Körséta mód.

A Barangolás módban szabadon mozoghatunk a betöltött terepmodell felett, és különböző szögekből megvizsgálhatjuk a felülethálót. Így ellenőrizhetjük, hogy a megfelelő modellt töltöttük-e be, és kiszűrhetjük az esetleges hibákat. A Barangolás mód az alapértelmezett a program indításakor.

A Körséta módban lejátszhatjuk a betöltött kamera trajektóriát, és a lejátszás közben a szimulált képeket a háttérrára menthetjük. A Körséta mód csak akkor érhető el, ha a programban van már betöltött kamera trajektória.

A fejlesztés eredménye egy C++ programozási nyelven írt, Windows specifikus, DirectX 11 grafikus API alapú felületháló megjelenítő lett.

3.2. Hardveres rendszerkövetelmények

A program rendszerkövetelményei a Windows SDK rendszerkövetelményei alapján kerültek meghatározásra.

- **Processzor:** legalább 1.6 Ghz -es, x64 architektúrájú processzor.
- **Memória:** legalább 1 GB RAM.
- **Videókártya:** DirectX 11-et támogató videókártya.
- **Tárhely:** legalább 100MB szabad tárhely.
- **Operációs rendszer:** Windows 10 (x64).

3.3. Szoftveres követelmények

A program DirectX11 grafikus API-t használ, emiatt szükség van a Windows SDK telepítésére. A Windows 8 operációs rendszertől kezdve a DirectX SDK a Windows SDK része. Korábban a DirectX SDK a Windowson történő játékfejlesztés platformjaként működött. Azonban mára, hogy a számítógépek széles körben rendelkeznek Direct3D támogatással, így az egyszerűbb asztali alkalmazások is kihasználhatják a grafikus hardveres gyorsítást. A Microsoft integrálta a DirectX technológiákat az operációs rendszerbe. A program futtatásához nem szükséges telepíteni a korábbi (legacy) DirectX SDK-t.

3.4. A program futtatása

A mellékelt bináris fájl futtatásához nincs szükség telepítésre, de a program futtatásához legalább Windows 10 operációs rendszerre van szükség. A program a 64 bites architektúrájú Windows operációs rendszereket támogatja.

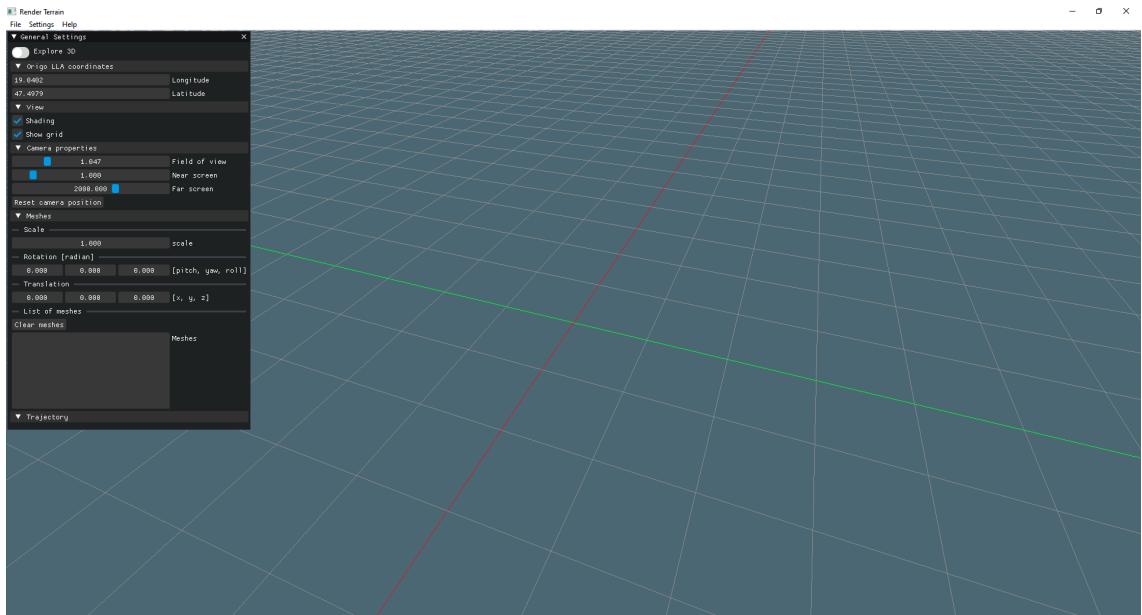
A DirectX Windows-specifikus, ezért a program csak Windows operációs rendszeren futtatható.

3.5. A program használatának menete

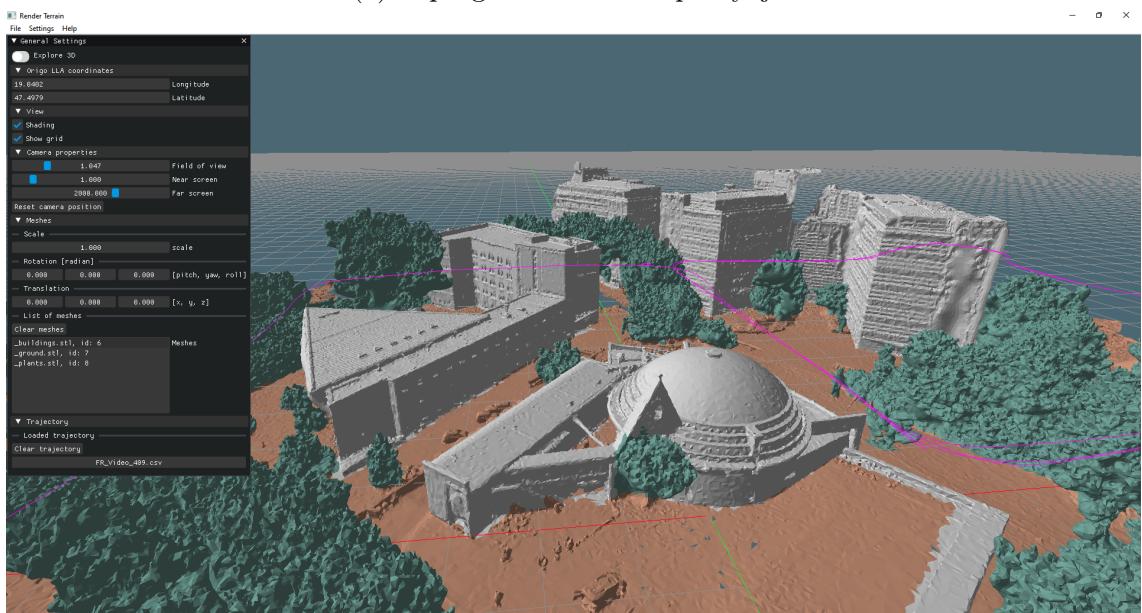
1. Tölts be a felülethálókat a "File > Open Terrain" vagy a "File > Open project" lehetőségek valamelyikével. A felülethálók betöltése során választhatja a "puha" (with Soft edges) vagy "éles" (with Sharp edges) élek opciót. A puha élek az interpolált árnyékoláshoz, míg az éles élek a konstans árnyékoláshoz használhatók. Az alapértelmezett mód a Barangolás mód, ahol a kamerát a billentyűzet és egér segítségével lehet mozgatni.
2. Töltsön be egy kamera trajektória fájlt a "File > Open trajectory" lehetőséggel. Miután a fájl sikeresen betöltődött, átválthat a Körséta módba, ahol lejátszhatja és rögzítheti a kamera útvonaláról készített szimulációt.
3. Tölts be a konfigurációs fájlt a "File > Open configuration file" lehetőséggel, vagy állítsa be a felülethálók és a kamera trajektória pozícióját, színét, egyéb paramétereit a grafikus felhasználói felületen.
4. Állítsa be a kimeneti könyvtár elérési útvonalát a "File > Set output directory" lehetőséggel, ahova a szimuláció képkockáit szeretné menteni.
5. Körséta módban indítsa el a szimulációt.

3.6. A felhasználói felület áttekintése

A program angol nyelvű, így minimális angol nyelvtudásra szükség van a használatához. A program indulásakor megjelenik a fő megjelenítési ablak, ahol láthatjuk a szimulációt, valamint itt jelennek meg a különböző beállítási ablakok is. A program indítását követően az általános beállítási ablak (General Settings) jelenik meg.



(a) A program indítási képernyője



(b) A program betöltött felülethálóval

3.1. ábra. A program megjelenítési ablaka

3.6.1. Menü

A menüsorban érhetjük el a fájlbetöltés funkciókat, megnyithatjuk a különböző beállítási ablakokat és az irányításra vonatkozó súgót.

Az elérhető menüpontok:

- File - Fájlbetöltés funkciók,
- Settings - Beállítási ablakok megnyitása,
- Help - Irányítás leírása.

Fájlbetöltés

A File menüpontra kattintva kiválaszthatjuk, hogy milyen külső állományokat szeretnénk betölteni. A File menüponton belüli további menüpontok közül választhatunk:

- Open terrain - Felületháló betöltése,
- Open project - Több felületháló betöltés egyszerre,
- Open trajectory - Kamera trajektória fájl betöltése,
- Open configuration file - Konfigurációs fájl betöltése,
- Set output directory - Képek mentési könyvtárának beállítása.

A program csak meghatározott formátumú fájlokat tud kezelni, erről részletesebben a 3.7 fejezetben olvashat.

A felülethálók előírt fájltípusa a bináris .stl kiterjesztésű modellek. A felületháló betöltése során két lehetőség közül választhatunk:

- with Sharp Edges - éles élekkel, vagy
- with Soft Edges - puha élekkel való megnyitás.

Amennyiben "Éles kontúr" lehetőséget választjuk a modell triangulált felületként jelenik meg, ekkor konstans árnyalásban (flat shading) látható a modell. Ebben az esetben nincs szükség az .stl fájl átalakítására, így a betöltés gyorsan elvégezhető.

Ha a "Puha kontúr" lehetőséget választjuk, a modellünk interpolált árnyalásban tekinthető meg. Mivel a program ebben az esetben több számítást végez, emiatt a betöltés több időt vesz igénybe.

Beállítási ablakok megnyitása

A "Settings" menüpontra kattintva kiválaszthatjuk, hogy melyik beállítási ablakot szeretnénk megnyitni. A "Settings" menüponton belüli további menüpontok közül választhatunk:

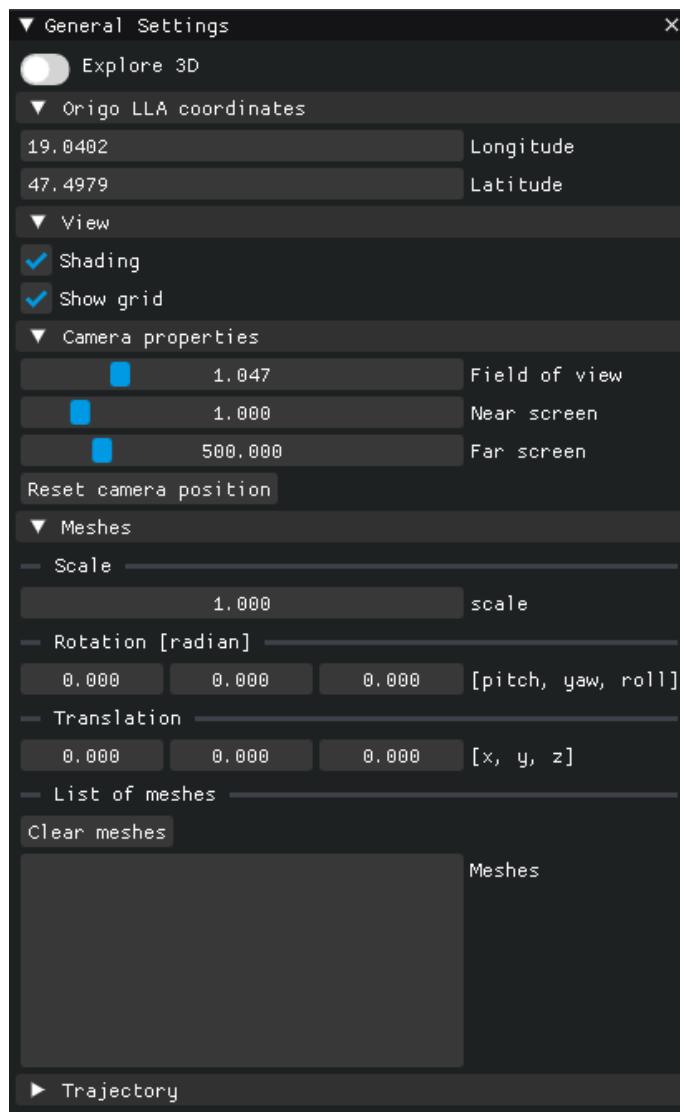
- General Settings - Általános beállítási ablak, 3.6.2. fejezet
- Explore 3D Settings - Barangolás mód beállítási ablaka, 3.6.3. fejezet
- Flythrough Settings - Körséta mód beállítási ablaka, 3.6.4. fejezet

3.6.2. Általános beállítási ablak

Az általános beállítási ablakban érhetjük el a Barangolás és Körséta mód közötti kapcsolót, illetve itt állíthatjuk be mind a két módban érvényes paramétereket.

A beállítási ablakban található kapcsoló (toggle button) segítségével váltani tudunk a Barangolás és a Körséta mód között. A Körséta mód csak akkor érhető el, ha a programban van betöltött kamera trajektória. Ha nincs ilyen betöltve, akkor a kapcsolóra kattintva hibaüzenetet kap a felhasználó, miszerint még nincs betöltött kamera útvonal, emiatt nem választható a Körséta mód. Fontos megjegyezni, hogy a két mód két különböző dátumot és időpontot használ. Ezek a megfelelő ablakban beállíthatóak.

Az Általános beállítási ablakban találhatóak az origó pozíciójának, a nézet paramétereknek, a kamerának, a felülethálóknak és a trajektoriának beállítási lehetőségei.



3.2. ábra. Általános beállítási ablak

Origó földrajzi koordinátái, Origo LLA coordinates

Az "Origó földrajzi koordinátái" lehetőségnél beállíthatjuk az origó hosszúsági és szélességi koordinátáit a földrajzi koordinátarendszerben:

- Longitude - fokban kifejezett hosszúsági koordináta,
- Latitude - fokban kifejezett szélességi koordináta.

Nézet paraméterek, View

A "Nézet paraméterek" lehetőségnél a megjelenítéssel kapcsolatos általános beállításokat állíthatjuk be. Az egyes paraméterek mellett egy-egy jelölőnégyzet található, amely segítségével be- vagy kikapcsolhatjuk a kívánt beállítást. Az alábbi lehetőségek állnak rendelkezésünkre:

- Shading - árnyalás bekapcsolása vagy kikapcsolása,
- Show grid - koordináta-rendszer hálózatának láthatósága vagy láthatatlanná tétele.

Kamera beállítások, Camera properties

A kamera beállításoknál három csúszkát használhatunk:

- Field of view - Függőleges látószög radiánban,
- Near screen - Elülső vágósík pozíciója,
- Far screen - Hátsó vágósík pozíciója.

A "Reset Camera" gomb segítségével visszahelyezhetjük a kamerát a kezdő pozícióba, ha a kamera mozgatása közben eltévedtünk a modell térben.

Felületháló beállítások, Meshes

A Felületháló beállításoknál a betöltött modellek paramétereit módosíthatjuk.

A "Clear meshes" gomb segítségével törölhetjük a betöltött felülethálókat a memóriából.

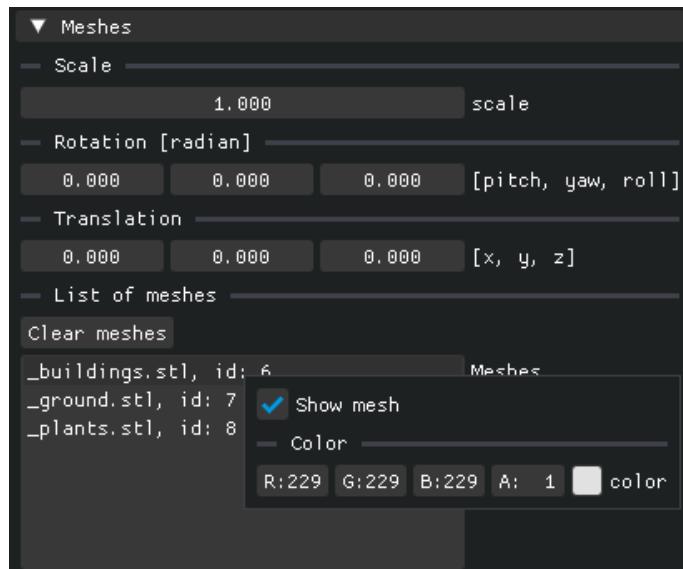
A betöltött felülethálók egy csoportot alkotnak, amelyre vonatkozó transzformációkat állíthatunk be, beleértve a skálázást, forgatást és mozgatást is, melyekhez húzógombos számbevitelt használhatunk. A csoportra vonatkozó transzformációs lehetőségek:

- Scale - Skálázás,
- Rotation - Forgatás radiánban X, Y, Z tengelyek mentén,

- Translation - Mozgatás X, Y, Z tengelyek mentén.

A betöltött felülethálók listáját itt találhatjuk, amennyiben az egyik elemre kattintunk, kiválaszthatjuk azt a listából. Ekkor egy felugró ablak jelenik meg, ahol módosíthatjuk a felületháló beállításait. A beállítási lehetőségek a következők:

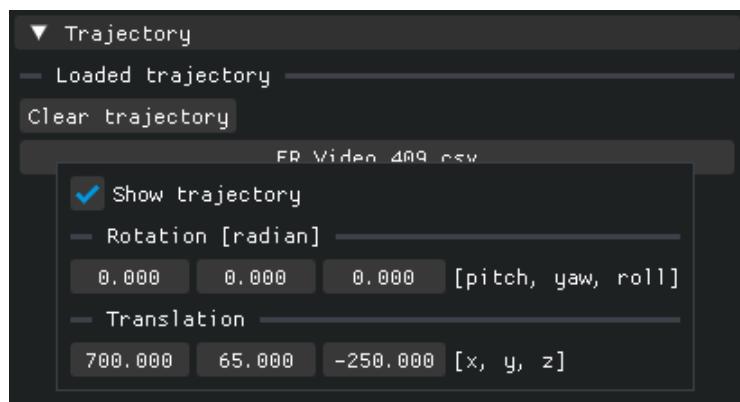
- Show mesh - Felületháló láthatósága,
- Color - Felületháló színe.



3.3. ábra. Felületháló paramétereinek beállítása

Trajektória, Trajectory

A trajektória beállításai hasonlóak a felülethálókéhoz. A "Clear Trajectory" gomb segítségével törölhetjük a memóriából a betöltött kamera útvonalat. Ha a trajektória nevére kattintunk, egy felugró ablakban módosíthatjuk a trajektória beállításait.



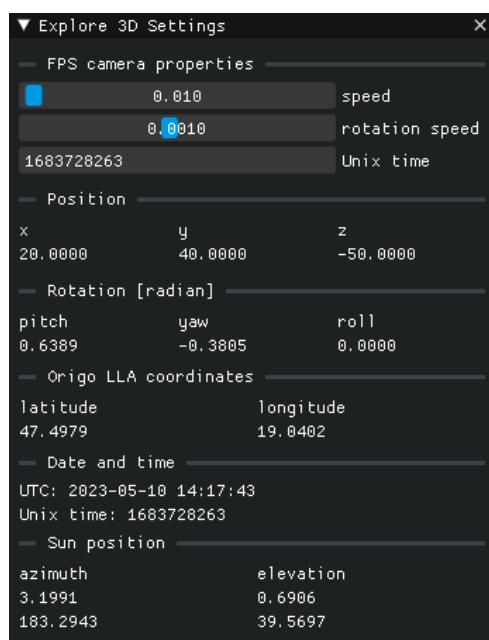
3.4. ábra. Trajektória paramétereinek beállítása

Beállítási lehetőségek:

- Show trajectory - Trajektória láthatósága,
- Rotation - Forgatás radiánban X, Y, Z tengelyek mentén,
- Translation - Mozgatás X, Y, Z tengelyek mentén.

3.6.3. Barangolás mód beállítási ablaka, Explore 3D settings

Fontos megjegyezni, hogy a Barangolás ablak kizárolag a Barangolás módban érhető el. A Barangolás ablakban a kamera mozgatási beállításait végezhetjük el, és megtekinthetjük az aktuális állapot paramétereit. A kamera mozgatásához és forgatásához két csúszka áll rendelkezésre. Az időpontot egy szöveges beviteli mezőben tudjuk beállítani, Unix-időben megadva. A Unix időbelyegek rendelkezésre állnak a kutatók számára a felvételek rögzítése során, a 3.7.2 fejezet tárgyalja az erről szóló információkat.

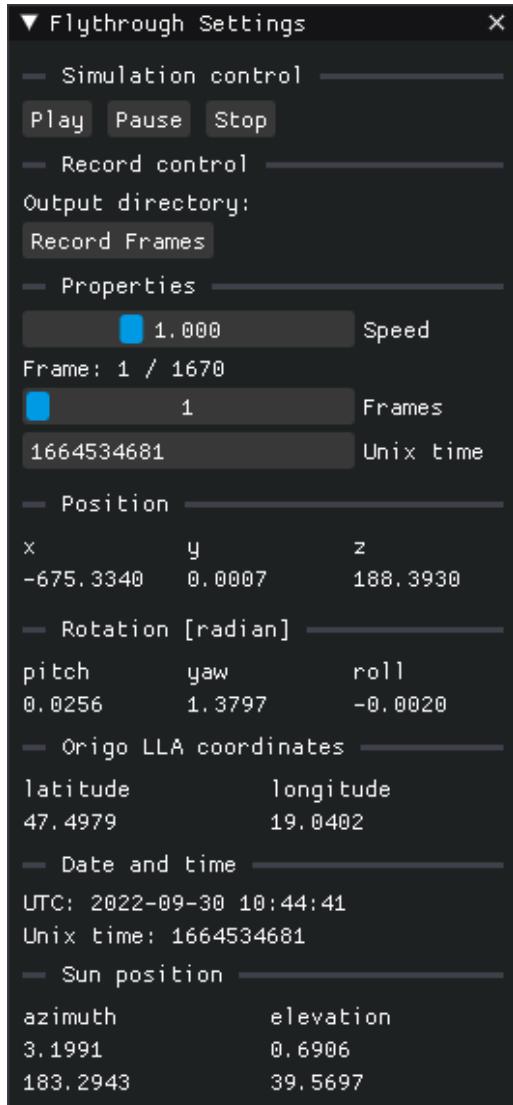


3.5. ábra. Barangolás mód beállítása

A következő státuszparaméterek jelennek meg a felületen:

- Position - Kamera pozíciója X, Y, Z koordinátákban,
- Rotation - Kamera tájolása Euler-szögek szerint, radiánban,
- Origo LLA coordinates - Az origó földrajzi koordinátái,
- Date and time - Az időpont és dátum UTC és Unix idő szerint,
- Sun position - A Nap pozíciója azimut és napmagasság szerint, fokban és radiánban.

3.6.4. Körséta mód beállítási ablaka, Flythrough settings



3.6. ábra. Körséta mód beállítása

A Körséta ablak kizárolag a Körséta módban érhető el.

A Körséta ablakban a trajektória lejátszás beállításait végezhetjük el, és az aktuális állapot paramétereit tekinthetjük meg. A *Play* gombbal indíthatjuk el a lejátszást. A *Pause* gomb segítségével szüneteltethetjük, a *Stop* gombra kattintva pedig megállíthatjuk a lejátszást és a kezdő pozícióba helyezzük a kamerát.

A *Record Frames* gombra kattintva elindul az adott pozícióból a lejátszás és a felvétel. A program a képeket a kiválasztott mappába helyezi. Amennyiben a mentési könyvtár nincs még kiválasztva, a program hibaüzenetet jelenít meg a felhasználónak.

A kezdő időpont megadása és a megjelenített státuszparaméterek megegyeznek a Barangolás beállítási ablakban láthatókkal, amelyet a 3.6.3 fejezetben lehet megtekinteni.

3.6.5. Irányítás

A 3.1 táblázatban szerepelnek a billentyűk leütésével kiadható parancsok Barangolás módban.

A 3.2 táblázatban szerepelnek azok a billentyűparancsok, amelyek kiadhatóak Körséta módban. A táblázat tartalmazza az egyes billentyűk funkcióját is.

Billentyű	Kamera mozgatása
W	Előre
S	Hátra
A	Balra
D	Jobbra
C	Lefele
Space	Felfele

3.1. táblázat. Kamera irányítása barangolás módban

Billentyű	Utasítás
R	Felvétel indítása és megállítása
Space	Lejátszás indítása és megállítása
Esc	Lejátszás megállítása és a kamera kezdő pozícióba állítása

3.2. táblázat. Lejátszás vezérlése Körséta módban

3.7. Támogatott formátumok

3.7.1. Felületháló fájl

Az STL (Standard Triangle Language) fájlformátumot a 3D Systems által készített CAD szoftverekben vezették be, mára számos más gyártó által készített szoftver is támogatja az STL fájlformátumot. A formátum egy háromdimenziós objektum felületi geometriáját írja le, szín, textúra, vagy egyéb CAD programokban használatos attribútum nélkül. A 3.7.1 fejezet az "Analysis of STL files" című tanulmány [20] szerint készült.

Az STL fájlban a triangulált felülethálót térbeli háromszögek sorozatával representáljuk. A háromszögeket a háromszög egység hosszúságú normálisa és a csúcsainak 3 koordinátája határozza meg, a koordináták jobbkéz szabály szerinti sorrendben megadva. Az STL fájlformátum a jobbsodrású koordináta rendszerbe helyezi az objektumot.

Az STL fájlformátum a bináris és ASCII formátumot is támogatja. A szakdolgozat keretében készített program kizárolag a bináris formátumot fogadja el. Ennek oka, hogy a bináris fájlformátumok beolvasása gyorsabb, kisebb erőforrást igényel, mint a szöveges fájl beolvasása.

A bináris STL-fájl egy 80 karakteres fejléccel kezdődik. A fejléc után 4 bájtos little-endian unsigned integer (előjel nélküli egész szám) jelzi, hogy a fájl hány háromszöget tartalmaz. Ezt követi az egyes háromszögek leírása. A fájl az utolsó háromszög meghatározásával végződik.

Leírás	Méret	Típus
Fejléc (Header)	80 bájt	UINT8[80]
Háromszögek száma	4 bájt	UINT32
Háromszögek leírása ciklikusan		
Normálvektor	12 bájt	REAL32[3]
Vertex 1	12 bájt	REAL32[3]
Vertex 2	12 bájt	REAL32[3]
Vertex 3	12 bájt	REAL32[3]
Attribútum	2 bájt	UINT16

3.3. táblázat. STL fájl felépítése

3.7.2. Kamera trajektória fájl

A program csv formátumban megadott kamera trajektóriákat képes beolvasni. A fájlnak tartalmaznia kell a következő fejlécet, bár a sorrend tetszőleges:

```
frame_num;sec;nsec;yaw;pitch;roll;north;east;down
```

A kamera trajektóriát jobbsodrású koordináta-rendszerben szükséges megadni.

Az egyes attribútumok a következőképpen értelmezendők:

- **frame_num**: a képkockák számozása egész számokkal,
- **sec**: az Unix Epoch-tól eltelt idő másodpercben, egész szám,
- **nsec**: az eltelt idő nanoszekundumban, egész szám,
- **yaw**: a kamera függőleges tengely körüli forgatása, radiánban, lebegőpontos szám,
- **pitch**: az East tengely körüli forgatás szöge, lebegőpontos szám,
- **roll**: a North tengely körüli forgatás szöge, radiánban, lebegőpontos szám,
- **north**: a kamera északi koordinátája, lebegőpontos szám,
- **east**: a kamera keleti koordinátája, lebegőpontos szám,
- **down**: a kamera függőleges koordinátája, lebegőpontos szám.

A program jelenleg csak ezeket az attribútumokat kezeli. Bármilyen további attribútumot a program figyelmen kívül hagy. A program a kamera útvonal lejátszása közben a megadott képkockák között lineáris interpolációt hajt végre.

3.7.3. Konfigurációs fájl

A program egy meghatározott formátumú JSON fájlt képes kezelni. A JSON fájl négy fő részből áll: origó, kamera, felületháló és trajektória.

Az origó (origo) részben adható meg a koordináta-rendszerünk origójának hosszúsági és szélességi koordinátája.

A kamera (camera) részben adhatóak meg a virtuális kamera paraméterei, a látószög, az előző és hátsó vágósík.

A felületháló (terrain) részben lehetőség van a betöltött felülethálók skálázására, eltolására és elforgatására a kezdeti helyzetétől. Ezen kívül a szín (color) mezőben a programba betöltött fájlok (buildings, plants, ground) színét lehet egyesével megadni RGBA értékekkel. Amennyiben a felületháló fájlneve tartalmazza a megadott kulcsszót, a program módosítja a felületháló színét.

Végül a trajektória (trajectory) részben lehetőség van az adott kamera útvonalának eltolására és elforgatására. Ez lehetővé teszi a kamera mozgásának finomhangolását és a virtuális környezet még valósághűbb megjelenítését.

```

1 {
2   "origo": {
3     "longitude": 19.040236,
4     "latitude": 47.497913
5   },
6   "camera": {
7     "fieldOfView": 1.0467,
8     "nearScreen": 1,
9     "farScreen": 2000
10  },
11  "terrain": {
12    "scale": [ 1, 1, 1 ],
13    "translation": [ 0, 0, 0 ],
14    "rotation": [ 0, 0, 0 ],
15    "color": {
16      "_buildings": [ 229, 229, 229, 1 ],
17      "_plants": [ 131, 175, 166, 1 ],
18      "_ground": [ 211, 162, 132, 1 ]
19    }
20  },
21  "trajectory": {
22    "translation": [ 700, 65, -250 ],
23    "rotation": [ 0, 0, 0 ]
24  }
25 }
```

3.1. forráskód. Konfigurációs mintafájl

4. fejezet

Fejlesztői dokumentáció

4.1. Funkcionális követelmények

4.1.1. Bemeneti adatokra vonatkozó követelmények

Felhasználói interakció

A program indítását követően a felhasználónak lehetősége van a bemeneti adatokat meghatározni. A bemeneti adatokat a meghatározott formátumú és kiterjesztésű fájlok elérési útvonalának megadásával érheti el.

Bemeneti adatok

- 3D szemantikus térkép (.stl),
- trajektória fájl (.csv),
- konfigurációs fájl (.json).

Felületháló adatai

A felületháló pontjai méterben adottak egy adott GPS koordinátán számolt WGS84 flat-Earth approximációból származó jobbsodrású Descartes koordinátarendszerben. A fájl tartalmazza a felületháló pontjainak az origóhoz viszonyított koordinátáit, a pontok közötti felületek normálvektorait és a szegmentációs osztályt.

A megállapodás értelmében azon felületháló elemek, amelyek egy fájlban vannak, ugyanabba a szegmentációs osztályba tartoznak. Több szegmentációs osztály használata esetén a felületháló elemeit a megfelelő STL fájlba szükséges helyezni.

A program kizárolag a bináris kódolású, .stl kiterjesztésű fájl beolvasását biztosítja.

Kamera trajektória

A program előre meghatározott formátumú kamera trajektóriákat képes betölteni. Az útvonal meghatározása kamera állások sorozatával történik. Egy adott sorozatelem adatai tartalmazzák a kamera pozícióját (North-East-Down, röviden NED), a kamera tájolását (Euler-szögek: yaw-pitch-roll), valamint a GPS időpontot (sec, nsec). Az időpont sec tagja Unix-időben van kifejezve.

Bemeneti fájlok ellenőrzése

A program a bemeneteket ellenőrzi és figyelmezteti a felhasználót, ha a bemeneti fájlok szintaktikai hibákat tartalmaznak. A program a bemenetek megfelelő betöltése érdekében a bevitel megismétlését kéri, ha hibát észlel.

4.1.2. Szimulációra, megjelenítésre vonatkozó követelmények

Térháló beolvasás

A betöltött adatokat a megfelelő formátumra átalakítva felépíti a felülethálót, elhelyezi a saját koordináta-rendszerében. A térhálónak elemeihez tartozó adatok: pozíció, normálvektor, szegmentációs osztály. A felhasználó a grafikus felületen képes módosítani a modell elemeinek színét és láthatóságát, valamint a modell elemek csoportjának pozícióját, tájolását és méretarányát. A felhasználó törölheti a beolvasott modellek listáját.

Fényforrás beállítása

Nap helyzetének és a párhuzamos megvilágítás irányának számítása a megadott UTC időpont és a megadott földrajzi koordináták alapján történik. A szimuláció során azzal a feltételezéssel élünk, hogy a térháló legfeljebb 1-2 km átmérőjű. Emiatt a modell térbeli kiterjedése nem haladja meg azt a léptéket, hogy az aktuális modellben a térbeli helyzet változása módosítaná a Nap állását. Statikusnak vesszük a fény irányát a tér függvényében.

Kamera és nézetkezelés biztosítása

A projekciós mátrixhoz szükséges kamera paraméterek beviteli mezőkön módosíthatóak:

- látómező, *Field of view, FOV*,
- közeli vágósík, *near screen*,
- távoli vágósík, *far screen*.

Kétféle nézetkezelési módszer biztosított:

- kamera trajektória lejátszása, *Körséta mód* vagy angolul *Flythrough mode*,
- szabad barangolás, *Barangolás mód* vagy angolul *Explore 3D mode*.

Kamera trajektória lejátszása

A felhasználó a program nézeti ablakában lejátszhatja a kamera trajektória által meghatározott útvonalat, miközben a program futási időben megjeleníti a felületháló aktuális képét. A felhasználónak lehetősége van módosítani a szimuláció kezdeti időpontját, amely Unix-idő formátumban adható meg egy beviteli mezőben. A Unix időbelyegek rendelkezésre állnak a kutatók számára a felvételek rögzítése során, a 3.7.2 fejezet tárgyalja az erről szóló információkat. Továbbá lehetőség van a modell eltolására és elforgatására, így a felhasználó beállíthatja a trajektória és a modell objektumok elhelyezkedésének egymáshoz való viszonyát. A trajektória lejátszásának sebessége alapértelmezetten a mintavételezési rátától függ, amely általában kb. 50Hz. Azonban a felhasználónak lehetősége van a sebesség módosítására is. A trajektória lejátszása közben egy idővonallal segítségével beállítható az aktuális képkocka. A program egy térbeli vonalláncként jeleníti meg a kamera útvonalát a felületháló fölött, amelynek láthatósága beállítható.

Szabad barangolás

A felhasználó billentyűzet és egér segítségével szabadon bejárhatja a térháló környezetét. A kamera mozgatásának sebességét a felhasználó grafikus felületen módosíthatja.

A Nap állása módosítható egy bemeneti mezőben beállítható időpont megadásával. A szabad barangolás közben a fény iránya statikus marad, az idő függvényében nem változik.

Árnyalás és fényelés

A DirectX 11 SDK által biztosított programozható modell lehetővé teszi a modell paraméterek szerinti árnyalását és fényelését, amelyeket a Vertex és Pixel shaderek biztosítanak. Ennek révén a program képes stilizált megjelenítést biztosítani.

4.1.3. Kimeneti adatokra vonatkozó követelmények

Képek mentése

A szimuláció során az exportálás parancs kiadásával a nézeti ablakban lejátszott képek kimenthetőek .png formátumban.

4.2. Nem funkcionális követelmények

Hatókonysság

- A programnak a betöltött felületháló poligonszámaival arányos processzor, GPU és memória terhelést kell generálnia.
- A programnak a legtöbb funkció esetén minden bevitelre gyors (1 másodperc alatti) válaszidőt kell biztosítania.
- A felületháló modell betöltése a háttértárról a memóriába több időt vehet igénybe, a maximum elfogadható várakozási idő 3 perc.

Megbízhatóság

- A szabványos használat mellett a programnak maximum 1,5 millió poligont kell kezelnie hibamentesen, és nem szabad hibaüzenetet vagy hibajelenséget előidéznie.
- Ha a felhasználó hibás bevitelt ad meg, a programnak hibaüzenetet kell kiadnia, majd lehetőséget kell biztosítania a bevitel megismétlésére.

Biztonság

- A programnak nincsenek biztonsági követelményei.

Hordozhatóság

- A program futtatása legalább Windows 10-es operációsrendszert igényel.

- A programhoz szükséges összes komponens megfelelő használatához telepített Windows SDK-ra van szükség.
- A program nem igényel külön telepítést.

Felhasználhatóság

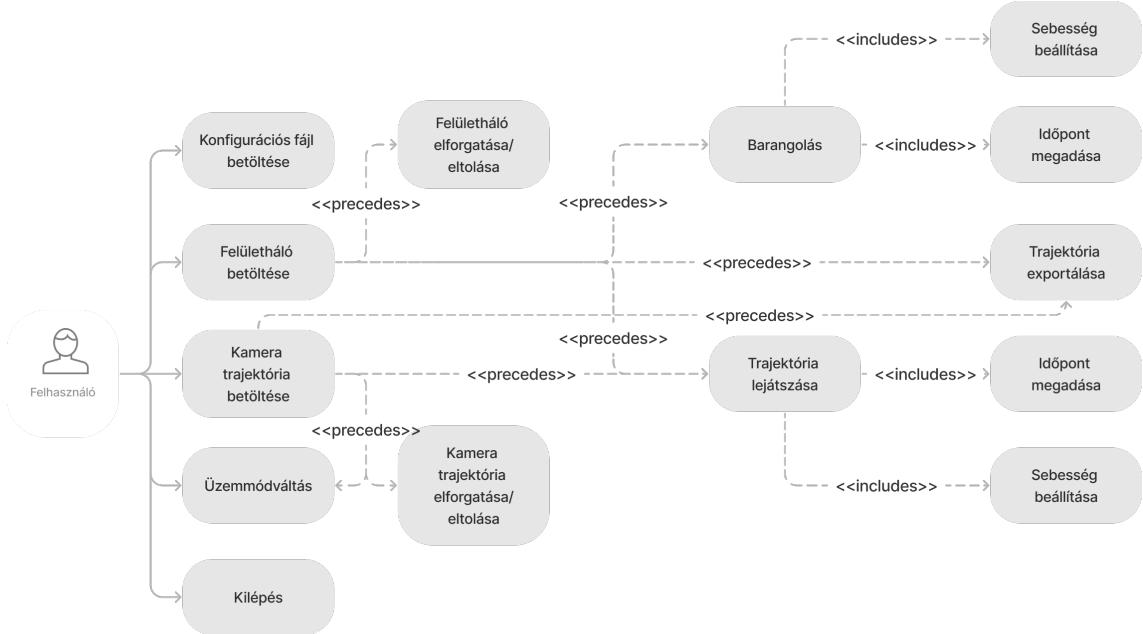
- Intuitív felhasználói felületet kell biztosítani, amely egyszerűen használható és könnyen érthető a felhasználók számára. Az instrukcióknak világosnak és pontosnak kell lenniük, hogy segítsék a felhasználókat a program megértésében és használatában.
- A felhasználói leírásban külön segédletet kell készíteni a használatról. Ez lehetőséget ad a felhasználóknak, hogy részletesebben megismerjék a programot és hogyan használják azt.

Fejlesztés

- A program nyelve a C++.
- A program fejlesztéséhez használt fejlesztői környezet: Visual Studio 2022 IDE.
- A program tervezésére és fejlesztésére használt paradigma objektumorientált. Az OEP könnyen karbantartható, újra-felhasználható és skálázható kódot eredményez.

4.3. Felhasználói eset diagram

A 4.1. ábrán a felhasználói eset diagram mutatja be milyen utasításokat adhat ki a felhasználó.



4.1. ábra. Felhasználói eset diagram

4.4. Felhasználói történetek

4.4.1. Külső állományok betöltése

Mint felhasználó, szeretném betölteni a külső állományokat, hogy inicializáljam a szimuláció paramétereit

- *Amennyiben* a program sikeresen inicializálta a DirectX eszközöket és elindult a *Barangolás* mód, *ha* megadtuk a triangulált terépmagyelérési útvonalát, és az sikeresen betöltődött, *akkor* a program megjeleníti a felülethálót.
- *Amennyiben* a program sikeresen inicializálta a DirectX eszközöket és elindult a *Barangolás* mód, *ha* megadtuk a kamera trajektória elérési útvonalát, és az sikeresen betöltődött, *akkor* a program megjeleníti a trajektória útvonal görbüjét és elérhetővé válik a *Körséta* mód.
- *Amennyiben* a program sikeresen inicializálta a DirectX eszközöket és elindult a *Barangolás* mód, *ha* megadtuk a konfigurációs fájl elérési útvonalát, és az sikeresen betöltődött, *akkor* a program elvégzi a megadott paraméterek alapján a műveleteket.

- *Amennyiben* a program sikeresen inicializálta a DirectX eszközöket és elindult a *Barangolás* mód, *ha* bármelyik külső fájl betöltése sikertelen, *akkor* program figyelmeztet.

4.4.2. Barangolás

Mint felhasználó, *szeretnék* a *Barangolás* módba lépni, *hogy* megállapítsam, hogy a megfelelő modellt töltöttem-e be és kiszűrjem az esetleges modellhibákat.

- *Amennyiben* betöltöttük a modellt, *ha* nem adtunk meg kamera trajektóriát, *akkor* a program *Barangolás* módban van.
- *Amennyiben* *Körséta* mód van beállítva, *ha* *Barangolás* kapcsolóra (toggle button) kattintunk, *akkor* a program elindítja a *Barangolás* módot.

Mint felhasználó, *szeretnék* a *Barangolás* módban szabadon barangolni a terepmóddal felett, *hogy* megállítsam, hogy a megfelelő modellt töltöttem-e be és kiszűrjem az esetleges modellhibákat.

- *Amennyiben* a *Barangolás* mód van beállítva, *ha* a WSAD billentyűket lenyomjuk, *akkor* a program módosítja a kamera pozíóját a megfelelő irány alapján, és megjeleníti az újrarenderelt képet.
- *Amennyiben* a *Barangolás* mód van beállítva, *ha* a bal egérbot lenyomjuk és az egeret mozgatjuk, *akkor* a program módosítja a kamera tájolását, és megjeleníti az újrarenderelt képet.

4.4.3. Körséta, trajektória lejátszása

Mint felhasználó, *szeretnék* a *Körséta* módba lépni, *hogy* megtekintsem a modellt a kamera útvonal mentén.

- *Amennyiben* nincs betöltött kamera trajektória, *ha* *Körséta* kapcsolóra kattintunk, *akkor* a program hibaüzenetet jelenít meg, amely arra kéri a felhasználót, hogy töltön be egy trajektóriát.
- *Amennyiben* van betöltött trajektória és a *Barangolás* mód van beállítva, *ha* a *Körséta* kapcsolóra (toggle button) kattintunk, *akkor* a program elindítja a *Körséta* módot és a kezdő pozícióba helyezi a kamerát.

Mint felhasználó, szeretném a Körséta módban lejátszani a kamera trajektóriát, hogy megtekintsem a modellt a kamera útvonal mentén.

- *Amennyiben van betöltött trajektória és a Körséta mód van beállítva, ha a Play gombra kattintunk, akkor a program elindítja a szimulációt a modell felett a megadott kameraútvonal mentén.*
- *Amennyiben van betöltött trajektória és a Körséta mód van beállítva, ha a SPACE billentyűt leütjük, akkor a program elindítja a szimulációt a modell felett a megadott kameraútvonal mentén.*

Mint felhasználó, szeretném a Körséta módban szüneteltetni a kamera trajektória lejátszását.

- *Amennyiben a program a trajektória útvonala mentén halad, ha a Pause gombra kattintunk, akkor a program szünetelteti a szimulációt.*
- *Amennyiben a program a trajektória útvonala mentén halad, ha a SPACE billentyűt leütjük akkor a program szünetelteti a szimulációt.*

Mint felhasználó, szeretném a Körséta módban megállítani a kamera trajektória lejátszását és a kezdő pozícióba helyezni a kamerát.

- *Amennyiben a program a trajektória útvonala mentén halad, ha a STOP gombra kattintunk, akkor megállítja a szimulációt és a kezdő pozícióba helyezi a kamerát.*

4.5. Felhasznált fejlesztési eszközök ismertetése

A grafikus API-k olyan programozási interfések, amelyek lehetővé teszik a fejlesztők számára, hogy az alkalmazások számára grafikus elemeket hozzanak létre. Ennek köszönhetően lehetőségünk van 2D és 3D modellek, textúrák, vizuális effektek megjelenítésére.

Grafikus API-kat számos programozási nyelvben használhatunk, például C++, C#, Java, Python stb. A C++ programozási nyelv előnye a nyelv sebessége és hatékonysága, emiatt kiülönösen alkalmas grafikus alkalmazások fejlesztésére. A C++ nem támogatja a garbage collector (szemétgyűjtők) használatát, amelyek esetleg

gyengíthetik a futás idejű teljesítményt, a program helyes memória kezelése programozói felelősség. A fordító a forráskódból egy közvetlenül végrehajtható állományt készít, nincs szükségünk például a Java nyelv által biztosított virtuális emuláló környezetben futtatni a kódot. Természetesen a platformspecifikus futtatható állománynak hátránya is van, amennyiben más platformon szeretnénk használni programunkat az adott platform architektúrájára szükséges fordítanunk a forráskódot.

A C++ az egyik legelterjedtebb nyelv a játékfejlesztésben, és számos olyan grafikus API-t támogat, amelyeket a játékfejlesztés során használnak, például a OpenGL-t és a DirectX-t. A két legelterjedtebb grafikus API a nyílt forrású OpenGL és a Windows specifikus DirectX. A DirectX-re esett a választás, mert a programot használni kívánó kutatási csoport eddig is DirectX-ben megírt programokat használnak. A kutatók elvárása egy könnyűsúlyú, Windows specifikus program megírása volt, a platform függetlenség nem volt szempont.

A DirectX-nek mára már több verziója is elérhető, a különbség ezek között a támogatott funkciókban és a támogatott hardverekben rejlik. A DirectX 9 hosszú múltra visszatekintő grafikus API, 2002-től egészen 2011-ig volt népszerű. Emiatt nagy elérhető kódbázissal rendelkezik. Ugyanakkor vannak korlátjai a modern hardverek nyújtotta előnyök kihasználásában. A másik hátránya, hogy a DirectX 10-es rendszertől kezdve átszervezték az API használatát, emiatt úgy találtam, célszerűbb egy újabb rendszer megismerése és elsajátítása. A DirectX 11 az egyik legelterjedtebb verzió, mert számos új funkciót és teljesítményjavítást hozott a DirectX 9-hez képest, miközben jobban kihasználja az új hardverek képességeit. Mivel széles körben népszerű, nagy kódbázissal rendelkezik. Rengeteg elérhető oktatói kód és oktatói videó található az interneten. A DirectX 12 a megnövekedett teljesítménykövetelmények miatt egy alacsonyabb hozzáférést biztosít a fejlesztők számára. Az alacsony szintű vezérlés miatt a grafikus API-k elsajátítására kevésbé ajánlják.

A fejlesztés a Visual Studio fejlesztői környezetben történt. A Visual Studio egy integrált fejlesztői környezet (IDE), amely számos eszközt és funkciót kínál a fejlesztőknek. A Visual Studio előnye, hogy egységes környezetet biztosít a fejlesztéshez, amely magában foglalja a fejlesztői eszközöket, a tesztelő eszközöket és a build rendszereket. A Visual Studio alapértelmezett build rendszere a MSBuild, amely lehetővé teszi a fejlesztők számára a projekt alapvető beállításait és az eredményül kapott futtatható fájlok előállítását. A Visual Studio-nak számos előnye van a többi build rendszerhez képest, például a könnyű használhatóság és a grafikus felület által nyújtott hozzájárulás.

tott egyszerűsített beállítások. Emellett a Visual Studio az egyik legelterjedtebb fejlesztői környezet a Windows platformon.

4.5.1. Felhasznált könyvtárak bemutatása

A szakdolgozatomban a Windows-specifikus programom implementálásához a Win32 API-t [21] választottam, amely az eredeti platform a natív C/C++ Windows alkalmazások fejlesztésére. A Win32 API (más néven Windows API) alkalmas a nagy teljesítményű alkalmazások implementálására, amelyek közvetlen hozzáférést igényelnek a rendszerhardverhez.

A DirectX alkalmazások fejlesztéséhez a Microsoft egy alkalmazáskészletet (tool kit) is biztosít, a DirectXTK-t [22]. A csomag segédosztályok gyűjteménye Universal Windows Platform, Windows 11, Windows 10, Xbox One és Win32 asztali alkalmazásokhoz.

A grafikus motor elkészítéséhez két oktatói oldalt használtam, amelyek segítettek az osztályok megszervezésében és a Direct3D eszközök inicializálásában [23][24][25].

A grafikus felhasználói felületre az ImGui könyvtárat [26] választottam, amely mind DirectX, mind OpenGL esetén használható.

A Nap pozíójának számítására a már 2.6 fejezetben megemlített tanulmányra támaszkodtam. A tanulmány nemcsak a matematikai számításokat veszi végig, hanem egy C++ nyelven implementált forrásfájlt is biztosít [27].

A JSON fájlok beolvasására pedig a nlohmann által biztosított könyvtárat [28] használtam.

A tesztek elvégzéséhez a Google Test [29] könyvtárát használtam, amely egy ingyenesen elérhető és nyílt forráskódú tesztelési keretrendszer C++ nyelvű projektekhez.

A DirectXTK és a Google Test könyvtára a NuGet Manager segítségével lett hozzáadva a projekt fájljaihoz. Az ehhez szükséges könyvtárak és függőségeik automatikusan letöltésre kerülnek a Visual Studio által.

4.6. Program felépítése

4.6.1. A program architektúrája

A program MVC (Model-View-Controller) architektúrára épül, amely három fő réteget tartalmaz [30].

- A modell felelős az adatkezelésért és feldolgozási logika definiálásáért. A modell független a nézettől és a vezérlőtől.
- A nézet az adatok megjelenítéséért felelős. Megjeleníti a modell aktuális állapotát a felhasználó számára és létrehozza a felhasználói felületet. A nézet csak a megjelenítési szabályokat írja le, egyéb számításokat nem végez.
- A vezérlő fogadja a felhasználói bemeneteket, és továbbítja azokat a modellnek és a nézetnek.

A modell és nézet kapcsolata

A modell és a nézet közötti kapcsolatot a Megfigyelő (Observer) tervezési minta biztosítja [30]. Az Alany (Subject) objektum lehetővé teszi, hogy több különböző Megfigyelő (Observer) feliratkozhasson rá. A feliratkozók értesítést kapnak, amennyiben az Alany állapota megváltozik. A modell és nézet kapcsolatában a modell az Alany, a nézet Megfigyelőként jelenik meg. Az Megfigyelő minta lehetővé teszi, hogy a modell állapotának bármilyen változása azonnal értesítse az összes feliratkozót, amelyek így azonnal frissíthetik a megjelenített adatokat.

A nézet és vezérlő kapcsolata

A vezérlők között felépített hierarchia segítségével elérhetjük, hogy a program a felhasználói bemenetekre különböző módon válaszoljon. Az MVC a nézet és vezérlő kapcsolatának kialakítására a Strategy (Stratégia) tervezési mintát javasolja [30]. A Környezet (Context) egy Stratégia (Strategy) objektumra való hivatkozáson keresztül éri el a Stratégia objektumhoz rendelt algoritmust. A Stratégia mintában a Környezet nem ismeri a Stratégia algoritmusát. Tehát, ha módosítjuk a Környezet által tartalmazott referenciát, módosítjuk az algoritmust. A vezérlő és nézet kapcsolatában a vezérlő a Stratégia, a nézet a Környezet szerepét tölti be. A nézet által tartalmazott referencián keresztül lehetőség van arra, hogy futás közben dinamikusan módosítsuk a végrehajtandó algoritmust.

4.6.2. A program belépési pontja

Míg a platformfüggetlen C++ kódok belépési pontja általában az `int main(int argc, char* argv[])` függvény, addig a Win32 API-t használó Windows programok esetében a `WinMain` vagy `wWinMain` függvényt használjuk belépési pontként. Erről bővebb információk találhatók a [31] forrásban.

```
1 int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,  
LPWSTR pCmdLine, int nCmdShow);
```

4.1. forráskód. `wWinMain` függvény deklarációja

wWinMain függvény paraméterei

- `HINSTANCE hInstance` - Az operációs rendszer ezt az értéket használja a futtatható fájl azonosítására, amikor az betöltődik a memóriába.
- `HINSTANCE hPrevInstance` - Mára már nincs használata, a régi 16-bites Windows verziókban volt jelentősége, de most mindenig NULL értéket kap.
- `LPWSTR pCmdLine` - Unicode karakterlánc, amely tartalmazza a parancssori argumentumokat.
- `int nCmdShow` - Az érték segítségével beállítható, hogy az alkalmazás főablaka kicsinyített, maximalizált vagy normál méretű módban jelenjen meg.

4.6.3. Az App osztály

Az App osztály a program főosztálya, felelős a program inicializálásáért, az üzenetek és események fogadásáért, az üzenetek alapján frissítések futtatásáért és a megjelenítésért.

Az App osztály összeköti a modellt, a nézetet és a vezérlőket, így biztosítva a komponensek közötti kommunikációt.

App osztály adattagai

- `RenderWindow m_renderWindow` - Tartalmazza a program megjelenítési ablakát. Részletesebb kifejtése a 4.6.4 fejezetben található.
- `Timer m_timer` - Az időzítőt tartalmazza, amely nyomon követi, hogy legutolsó lekérdezés óta mennyi idő telt el.

- **TerrainModelPtr m_terrainModel** - Az adatkezelésért és feldolgozási logika definiálásáért felelős. Részletesebb kifejtése a 4.6.7 fejezetben található.
- **TerrainViewPtr m_terrainView** - Az adatok megjelenítéséért felelős. Megjeleníti a modell aktuális állapotát a felhasználó számára és létrehozza a felhasználói felületet. Részletesebb kifejtés a 4.6.6 fejezetben található.
- **ControllerRouterPtr m_terrainController** - A felhasználói bemeneteket fogadja és továbbítja azokat a modellnek és a nézetnek. Részletesebb kifejtés a 4.6.5 fejezetben található.
- **IDataAccess m_dataAccess** - A perzisztenciát valósítja meg, felelős a külső állományok betöltéséért. Részletesebb kifejtés a 4.6.8 fejezetben található.
- **MousePtr m_mouse** - Az egérkezelést valósítja meg.
- **KeyboardPtr m_keyboard** - A billentyűzetkezelést valósítja meg.

Az App osztály tagfüggvényei

```

1 private:
2     void Resize(UINT screenWidth, UINT screenHeight);
3     void Update();
4     void RenderFrame();
5     bool ProcessMessages();
6     LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);
7 public:
8     bool Initialize(HINSTANCE hInstance, int screenWidth, int
9         screenHeight);
10    void Run();
11    void Shutdown();

```

4.2. forráskód. App osztály tagfüggvényeinek deklarációja

- **Initialize** - Feladata az adattagok inicializálása, és a rendereléshez szükséges paraméterek beállítása.
- **Run** - A függvény felelős az alkalmazás futtatásáért. A függvényben található egy végtelen ciklus, amelyben a program frissítése és renderelése történik. A ciklus addig fut, amíg a felhasználó be nem zárja az alkalmazást.

- **Update** - Továbbítja a vezérlőnek, hogy mennyi idő telt el az előző ciklus óta. Részletesebb kifejtést a 4.6.5 fejezetben találhat.
- **RenderFrame** - Utasítja a nézetet a kép megjelenítésére. Részletesebb kifejtést a 4.6.6 fejezetben található.
- **ProcessMessage** - Felelős az ablakkezelő üzenetek feldolgozásáért. Részletesebb kifejtést a 4.6.4 fejezetben találhat.
- **Shutdown** - A függvény felelős az alkalmazás bezárásáért, lefoglalt erőforrások felszabadításáért. Leállítja eljárás során a felhasználói felületi elemeket és a vezérlőket.
- **WindowProc** - A függvény a Windows üzenetek kezeléséért felelős. Az eljárás során a különböző felhasználói eseményeket, például a billentyűlétrejük vagy a kurzor mozgatását kezeli. Az üzenetek feldolgozása során a megfelelő vezérlőket aktiválja, hogy megfelelő választ adjanak az eseményekre. Részletesebb kifejtést a 4.6.4 fejezetben találhat.

```

1 void App::Run()
2 {
3     while (this->ProcessMessages())
4     {
5         this->Update();
6         this->RenderFrame();
7     }
8 }
```

4.3. forráskód. App::Run(), az alkalmazás ciklusa

4.6.4. A RenderWindow osztály

A RenderWindow osztály a Win32 API utasításainak segítségével létrehozza a megjelenítési képernyőt.

RenderWindow osztály tagfüggvényei

```
1 public:  
2     bool Initialize(App* app, int width, int height);  
3     bool ProcessMessages();  
4     HWND GetHWND() const;  
5     void Shutdown();  
6 private:  
7     void RegisterWindowClass();
```

4.4. forráskód. RenderWindow tagfüggvényeinek deklarációja

- `RegisterWindowClass` - Ahhoz, hogy ablakot létrehozhassunk elsőként regisztrálnunk kell az ablak osztályt. Az ablak osztály paramétereinek meghatározása után `RegisterClassEx` függvény hívás regisztrálja az operációs rendszer a beállított struktúrát. A struktúra elemei közül a `WinProc` tag részletesebben a fejezet végén kerül kifejtésre.
- `Initialize` - Függvény létrehozza a már regisztrált ablak osztály egy példányát a `CreateWindowEx` rendszerhívással. Majd `ShowWindow` rendszerhívás megjeleníti a képernyőn a létrehozott ablakot.
- `ProcessMessages` - A Windows operációs rendszer különböző eseményekkel és üzenetekkel kommunikál a programunkkal. Az operációs rendszer az üzeneteket egy sorba helyezi el. A `PeekMessage` rendszerhívás kiveszi a sor első üzenetét és értékét átadja az `msg`-nek majd igaz értékkel visszatér. Azonban, ha a sor üres, hamis üzenettel tér vissza, így a ciklus befejeződik. A `TranslateMessage` bizonyos billentyűleütéseket a megfelelő formátumra fordítja. A `DispatchMessage` továbbítja az üzenetet az ablak regisztrálása során meghatározott `WndProc` függvénynek.

```

1 bool RenderWindow::ProcessMessages(){
2     MSG msg;
3     // Initialize the message structure.
4     ZeroMemory(&msg, sizeof(MSG));
5     // Handle the windows messages.
6     while (PeekMessage(&msg, this->_hwnd, 0, 0, PM_REMOVE)){
7         TranslateMessage(&msg);
8         DispatchMessage(&msg);
9     }
10    // Check if the window was closed
11    if (msg.message == WM_NULL && (!IsWindow(this->_hwnd))){
12        this->_hwnd = nullptr;
13        UnregisterClass(this->_window_class.c_str(), this->_hInstance);
14        return false;
15    }
16    return true;
17 }
```

4.5. forráskód. RenderWindow::ProcessMessages tagfüggvény

A WndProc függvény

A WndProc függvény az alkalmazás callback funkciója, amely kezeli az ablakhoz érkező üzeneteket. Az alkalmazás által meghatározott WndProc függvényt a Windows operációs rendszer hívja meg az ablakhoz érkező üzenetek feldolgozására [21]. További részletek a hivatkozásban szereplő linken olvashatók.

Az osztályhoz két WndProc függvény tartozik, a függvények deklarációi:

```

1 static LRESULT CALLBACK MessageHandlerSetup (HWND hwnd, UINT
    umessage, WPARAM wparam, LPARAM lparam);
2 static LRESULT CALLBACK MessageHandlerRedirect (HWND hwnd, UINT
    umessage, WPARAM wparam, LPARAM lparam);
```

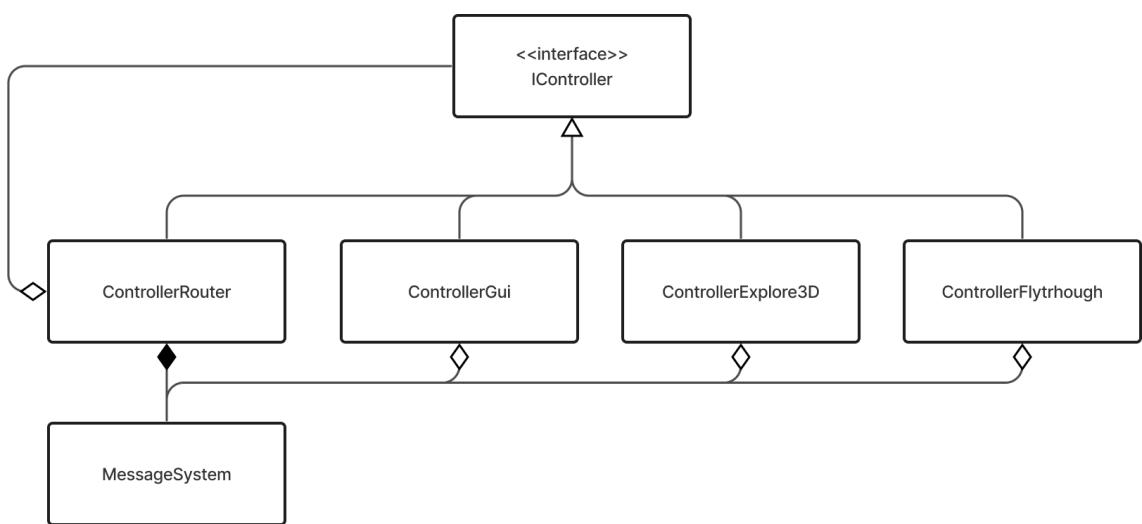
4.6. forráskód. WndProc függvények deklarációja

- **MessageHandlerSetup** - Az ablak regisztrálása során beállított WndProc függvény. Feladata, hogy a megjelenítési ablak létrehozásakor a WndProc függvényt átirányítsa a **MessageHandlerRedirect** függvényre.

- **MessageHandlerRedirect** - A függvény a kapott paramétereit átadja az `App::WindowProc` függvénynek.

4.6.5. A vezérlő réteg

A vezérlő feladata a felhasználótól érkező input fogadása, feldolgozása és a modellkomponens megfelelő frissítése. Az MVC architektúra lehetővé teszi, hogy futás közben a felhasználó utasításaira a program másképp reagáljon. A grafikus felület kezelésének, valamint a Barangolás és Körséta üzemmód megvalósításának érdekében a vezérlők hierarchikus szerkezetben vannak létrehozva.



4.2. ábra. A vezérlők hierarchikus szerkezete

A vezérlők hierarchikus rendszere a Kompozit tervezési mintát követi, ahol a **ControllerRouter** objektum a fő vezérlő, amely referenciával hivatkozik a további vezérlőkre. Feladata, hogy a **MessageSystem** segítségével a további vezérlőknek továbbítsa az üzeneteket, alegységeire delegálja a feladatot. Valamennyi további vezérlő a Levél szerepét tölti be, vagyis nem tartalmaznak további aleemet, a megkapott feladatot nem delegálják tovább.

A vezérlő osztályok

A különböző vezérlő osztályok eltérő felelősségi körrel rendelkeznek, minden egyes osztálynak önálló feladattal rendelkezik.

- **ControllerRouter** - A fő vezérlő osztály, amely referenciaival hivatkozik a további vezérlőkre. Feladata, hogy a **MessageSystem** segítségével a további vezérlőknek továbbítja az üzeneteket. A vezérlő mindenkor aktív.
- **Controller3DExplore** - Az osztály felelős a Barangolás üzemmód irányításáért. A felhasználó által végrehajtott billentyű leütések és egér mozgatás hatására utasítja a modellt, hogy a megfelelő paraméterek alapján módosítsa az állapotát. Az **App::m_Timer** időzítőtől megkapja a megjelenített képek között eltelt időt, amely alapján utasítja a modellt, hogy frissítse az állapotát. A vezérlő aktivitása a program futása közben módosulhat. A **Controller3DExplore** és a **ControllerFlythrough** közül, mindenkor csak pontosan az egyik lehet aktív.
- **ControllerFlythrough** - Ez az osztály a Körséta üzemmód irányításáért felelős. Az **App::m_Timer** időzítő által mért időváltozás alapján frissíti a modellt. Az **App::m_Timer** időzítőtől megkapja a megjelenített képek között eltelt időt, amely alapján utasítja a modellt, hogy frissítse az állapotát.
- **ControllerGui** - Ez az osztály felelős a felhasználói felület felől érkező üzenetek feldolgozásáért. Amennyiben a felhasználói felület utasítása egy másik vezérlőt is érintene, továbbítja az üzenetet a megfelelő vezérlő számára a **MessageSystem** segítségével. A vezérlő mindenkor aktív.

A vezérlő interfész

A vezérlők paraméterekkel kiegészített üzenetek formájában kapja meg az utasításokat a felhasználói felületen keresztül a felhasználótól. A vezérlő az üzenetek feldolgozásáért felelős.

```

1 class IController : public IMModelSubscriber {
2 public:
3     virtual ~IController() {}
4     virtual bool Initialize(IModelPtr pModel, IViewPtr pView,
5                             MousePtr mouse, KeyboardPtr keyboard) = 0;
6     virtual bool HandleMessage(IControllerMessageIDs message, const
7                                 std::vector<float>& fparam, const std::vector<unsigned>&
8                                 uparam) = 0;
9     virtual void Shutdown() = 0;
10    virtual void SetTerrainModel(IModelPtr pModel) = 0;
11    virtual void SetTerrainView(IViewPtr pView) = 0;
12    virtual void SetMouse(MousePtr mouse) = 0;
13    virtual void SetKeyboard(KeyboardPtr keyboard) = 0;
14    virtual void SetMessageSystem(ControllerMessageSystemPtr
15                                  messageSystem) = 0;
16    virtual bool IsActive() const = 0 ;
17 };

```

4.7. forráskód. IController interfész

- **Initialize** - A függvény felelős a vezérlő inicializálásáért.
- **HandleMessage** - A függvény felelős az üzenetek feldolgozásáért és végrehajtásáért. Az üzenet azonosítóját, valamint az üzenethez tartozó lebegőpontos és egész típusú paramétereket kapja paraméterként. A függvényének előképe a WndProc függvény.
- **Shutdown** - A függvény felelős a vezérlő leállításáért és a felhasznált erőforrások felszabadításáért.
- **IsActive** - A függvény visszaadja, hogy a vezérlő aktív-e.

A vezérlő réteg üzenetei

Az enum IControllerMessageIDs azonosítja azokat az üzeneteket, amelyeket a nézet küld a vezérlőnek vagy amelyeket egy vezérlő továbbít egy másik vezérlőnek. A 4.1 táblázat tartalmazza az egyes üzenetek leírását. Az **f** és **u** oszlopok azt jelzik, hogy az adott üzenetben a vezérlő hány darab **float** illetve **unsigned** típusú értéket vár.

A vezérlő réteg üzenetei			
Üzenet azonosító	Leírás	f	u
IDC_TIME_ELAPSED	Az eltelt idő az utolsó képkocka óta.	1	0
IDCC_IS_FLYTHROUGH_MODE_ON	A Körséta mód aktivitásának lekérdezése a vezérlők között.	0	0
IDCC_OUTPUT_DIR_CHOOSED	Az output könyvtár sikeres kiválasztásának jelzése a vezérlők között.	0	0
IDMENU_FILE_TERRAIN_SHARP	Egy felületháló fájl megnyitása éles kontúrral.	0	0
IDMENU_FILE_PROJECT_SHARP	Több felületháló fájl megnyitása éles kontúrral.	0	0
IDMENU_FILE_TERRAIN_SOFT	Egy felületháló fájl megnyitása puha kontúrral.	0	0
IDMENU_FILE_PROJECT_SOFT	Több felületháló fájl megnyitása puha kontúrral.	0	0
IDMENU_FILE_TRAJECTORY	Kamera trajektória fájl megnyitása.	0	0
IDMENU_FILE_CONFIGURATION	Konfigurációs fájl megnyitása.	0	0
IDMENU_FILE_OUTPUT_DIRECTORY	Az output könyvtár kiválasztása.	0	0
IDMENU_HELP	A súgó megnyitása.	0	0
IDMENU_WINDOWS_GENERAL	Az általános beállítások ablakának megnyitása.	0	0
IDMENU_WINDOWS_FLYTHROUGH	A Körséta mód beállítások ablakának megnyitása.	0	0
IDMENU_WINDOWS_EXPLORE3D	A Barangolás mód beállítások ablakának megnyitása.	0	0
IDC_ORIGO_SET_LONGITUDE	Az origó hosszúsági fokának beállítása.	1	0
IDC_ORIGO_SET_LATITUDE	Az origó szélességi fokának beállítása.	1	0
IDC_XZ_PLANE_GRID_SET_ISSEEN	A koordináta-rendszer láthatóságának beállítása.	0	1
IDC_PIXELSHADER_SET_SHADING	A pixelshader árnyékolásának beállítása.	0	1

Üzenet azonosító	Leírás	f	u
IDC_ACTIVATE_FLYTHROUGH_MODE	A Körséta mód aktiválásának üzenete.	0	0
IDC_ACTIVATE_EXPLORE3D_MODE	A Barangolás mód aktiválásának üzenete.	0	0
IDC_E3D_CAMERA_RESET	A kamera kezdőpozícióba helyezése Barangolás módban.	0	0
IDC_E3D_CAMERA_SPEED	A kameramozgatás sebességének beállítása Barangolás módban.	1	0
IDC_E3D_ROTATION_SPEED	A kameraforgatás sebességének beállítása Barangolás módban.	1	0
IDC_SET_CAMERA_FIELD_OF_VIEW	A kamera látószöge beállításának üzenete.	1	0
IDC_SET_CAMERA_NEAR_SCREEN	A kamera elülső vágósíkjának beállítása.	1	0
IDC_SET_CAMERA_FAR_SCREEN	A kamera hátsó vágósíkjának beállítása.	1	0
IDC_SET_TIME_E3D	Időpont beállítása a Barangolás módban.	0	1
IDC_FLYTHROUGH_START	Szimuláció lejátszásának indítása a Körséta módban.	0	0
IDC_FLYTHROUGH_PAUSE	Szimuláció lejátszásának szüneteltetése Körséta módban.	0	0
IDC_FLYTHROUGH_STOP	Szimuláció lejátszásának megállítása Körséta módban.	0	0
IDC_FLYTHROUGH_RECORD_START	Szimuláció képeinek háttértárra mentése Körséta módban.	0	0
IDC_FLYTHROUGH_RECORD_STOP	Szimuláció képeinek háttértárra mentésének megállítása Körséta módban.	0	0
IDC_FLYTHROUGH_SET_SPEED	Szimuláció sebességének beállítása Körséta módban.	1	0
IDC_FLYTHROUGH_SET_FRAME	Szimuláció adott képkockájának beállítása Körséta módban.	0	1

Üzenet azonosító	Leírás	f	u
IDC_SET_START_TIME_TRAJECTORY	Szimuláció kezdő időpontjának beállítása Körséta módban.	0	1
IDC_MESH_SET_ISSEEN	Egy felületháló elem láthatóságának beállítása.	1	1
IDC_MESH_SET_COLOR	Egy felületháló elem színének beállítása.	4	1
IDC_MESH_GROUP_SCALE	A felületháló csoport skálázásának beállítása.	1	0
IDC_MESH_GROUP_ROTATION	A felületháló csoport Euler-szögeinek beállítása.	3	0
IDC_MESH_GROUP_TRANSLATION	A felületháló csoport X, Y, Z koordináták szerinti eltolása.	3	0
IDC_TRAJECTORY_SET_ISSEEN	A kamera trajektória vonallánc láthatóságának beállítása.	0	1
IDC_TRAJECTORY_ROTATION	A kamera trajektória Euler-szögeinek beállítása.	3	0
IDC_TRAJECTORY_TRANSLATION	A kamera trajektória X, Y, Z koordináták szerinti eltolása.	3	0
IDC_CLEAR_MESHES	Betöltött felülethálók törlése a memóriából.	0	0
IDC_CLEAR_TRAJECTORY	Betöltött kamera trajektória törlése a memóriából.	0	0

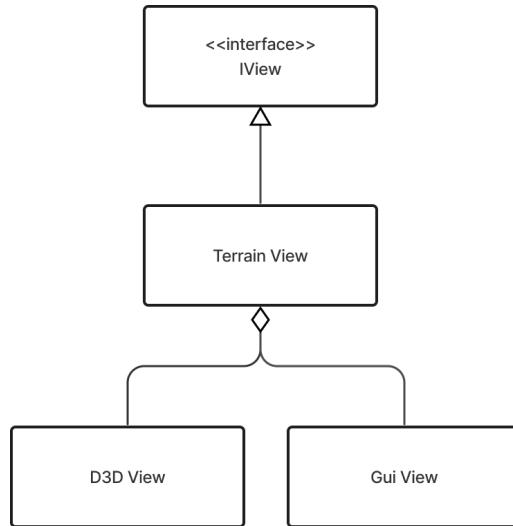
4.1. táblázat. A vezérlő réteg üzenetei, enum IControllerMessageIDs

Kommunikáció a vezérlők között, a MessageSystem osztály

A vezérlők egymás között a Megfigyelő minta segítségével kommunikálnak, melyet a MessageSystem osztály tesz lehetővé. minden vezérlő referenciával rendelkezik a MessageSystem objektumra, és a MessageSystem::Publish metódus segítségével tudnak üzenetet küldeni egy másik, szintén feliratkozott vezérlőnek.

4.6.6. A nézet réteg

A nézet komponens a felhasználói felületért és annak megjelenítéséért, illetve a modell komponens aktuális állapotának megjelenítéséért felelős. A nézet feladata összetett, emiatt a nézet megvalósítása is egy hierarchikus szerkezetben történik.



4.3. ábra. A nézet réteg hierarchikus szerkezete

A nézet osztályok

A különböző nézet osztályok eltérő felelősségi körrel rendelkeznek, minden egyes osztály önálló feladattal rendelkezik.

- **TerrainView** - Az IView interfész egy származtatott osztálya, amely más nézetek aggregálásáért és a teljes tartalom megjelenítéséért felelős a képernyőn. A vezérlőtől származó utasítás hatására frissíti a megjelenített képet a D3DView-on keresztül.
- **D3DView** - A DirectX API használatához szükséges objektumok létrehozásáért, használatáért és az DirectX erőforrások felszabadításáért felel. Az DirectX objektumok segítségével az alacsony szintű grafikus API-n keresztül kommunikál a hardverrel. Az osztály tárolja a megjelenítési felület méretét és felbontását. Valamint az osztály felel a szimuláció képeinek mentéséért.
- **GuiView** - A ImGui könyvtár komponenseit felhasználva, a felhasználói felület megjelenítéséért felelős. A felhasználói felület létrehozásáért és kezeléséért felelős komponenseket inicializálja, például gombokat, szövegmezőket és listákat.

A felhasználói felületen történő eseményeket figyeli, mint például a gombokra kattintásokat vagy a szövegmezőbe írt szövegeket, majd ezeket átadja a **ControllerRouter**-nek.

A nézet interfész

```

1 class IView : public IMModelSubscriber
2 {
3 public:
4     virtual ~IView() {}
5     virtual bool Initialize(HWND hwnd, float screenWidth, float
6         screenHeight, bool fullscreen, bool vsync) = 0;
7     virtual bool RenderFrame() = 0;
8     virtual bool Resize(unsigned screenWidth, unsigned screenHeight)
9         = 0;
10    virtual void Shutdown() = 0;
11    virtual bool HandleMessage(IViewMessageIDs message, const std::
12        vector<std::wstring>& stringParams, const std::vector<float>&
13        fparams, const std::vector<unsigned>& uparams) = 0;
14    virtual void SetController(IControllerPtr terrainController) = 0;
15    virtual void SetModel(IModelPtr terrainModel) = 0;
16    virtual Microsoft::WRL::ComPtr<ID3D11Device> GetDevice() = 0;
17    virtual Microsoft::WRL::ComPtr<ID3D11DeviceContext>
18        GetDeviceContext() = 0;
19 };

```

4.8. forráskód. IView interfész

- **Initialize** - Ez a függvény inicializálja a DirectX 11 illesztőprogramot és a GUI-t a megadott ablakkal, képernyőméréssel, teljes képernyős mód be- vagy kikapcsolásával és a vsync engedélyezésével vagy letiltásával.
- **RenderFrame** - Ez a metódus kirajzolja az aktuális képkockát a képernyőre, miután az összes grafikus adat elkészült.
- **Resize** - Ez a metódus a képernyőméret megváltoztatása után frissíti a hozzá kapcsolódó grafikus objektumokat, amelyek mérete függ a képernyőmérő változásától.
- **Shutdown** - Ez a függvény felszabadítja a felhasznált erőforrásokat.

- `HandleMessage` - A függvény felelős az üzenetek feldolgozásáért és végrehajtásáért, amelyeket a vezérlő küld a nézet felé. Az üzenet azonosítóját, valamint az üzenethez tartozó karakterlánc, lebegőpontos és egész típusú paramétereket kapja paraméterként.

A nézet réteg üzenetei

Az enum `IViewMessageIDs` azonosítja azokat az üzeneteket, amelyeket a vezérlő küld a nézetnek. A 4.2 táblázat tartalmazza az egyes üzenetek leírását. Az utolsó három oszlop azt jelenti, hogy az adott üzenetben a nézet mennyi olyan értéket vár, amely a megadott típusú: **s** esetén `wstring`, **f** esetén `float`, **u** esetén pedig `unsigned`.

Ha a vezérlő üzenet és a nézet üzenet között egyértelmű megfeleltetés van, a `IDC2IDV` függvény végzi a két üzenettípus közötti váltást. Ha nincs egyértelmű megfeleltetés, a függvény `IDV_INVALID` értékkel tér vissza.

A nézet réteg üzenetei					
Üzenet azonosító	Leírás	s	f	u	
<code>IDV_INVALID</code>	Érvénytelen üzenet.	0	0	0	
<code>IDV_SET_OUTPUT_DIRECTORY</code>	Az output könyvtár kiválasztása.	1	0	0	
<code>IDV_SHOW_GENERAL</code>	Az általános beállítások ablakának megnyitása.	0	0	0	
<code>IDV_SHOW_HELP</code>	A súgó megnyitása.	0	0	0	
<code>IDV_SHOW_FLYTHROUGH</code>	A Körséta mód beállítások ablakának megnyitása.	0	0	0	
<code>IDV_SHOW_EXPLORE3D</code>	A Barangolás mód beállítások ablakának megnyitása.	0	0	0	
<code>IDV_CAPTURE_SCREEN</code>	Az adott képkocka mentése a háttér-tárra.	0	0	1	
<code>IDV_FLYTHROUGH_RECORD_START</code>	Szimuláció képeinek háttértárra men-tésének elindítása Körséta módban.	0	0	0	
<code>IDV_FLYTHROUGH_RECORD_STOP</code>	Szimuláció képeinek háttértárra men-tésének megállítása Körséta módban.	0	0	0	

4.2. táblázat. A nézet réteg üzenetei, enum `IViewMessageIDs`

D3DView

A COM (Component Object Model) objektumok biztosítják az alkalmazások közötti kommunikációt. Ezen objektumok az objektumorientált programozás elveit követve, különböző interfésekkel kínálnak, amelyeken keresztül az alkalmazások kommunikálni tudnak egymással.

A Direct3D által biztosított osztályok COM interfész leszármazott osztályai. A COM objektumok biztonságos használatára használhatjuk a ComPtr osztályt, ami egy smart pointer. A ComPtr lehetővé teszi a COM objektumok egyszerűbb kezelését és a memóriakezelési problémák elkerülését.

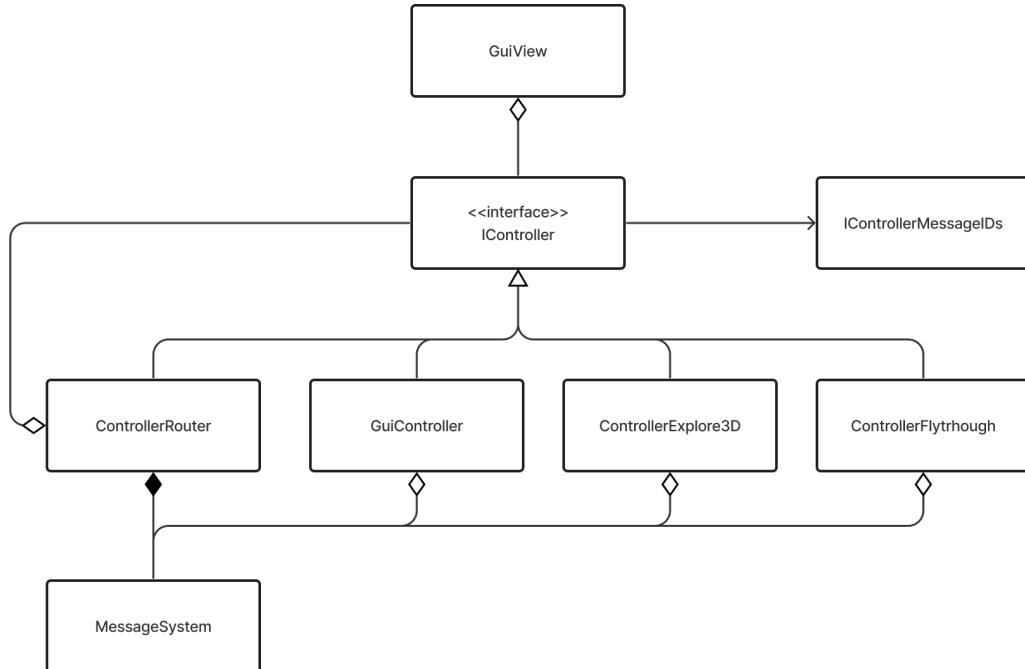
A D3DView osztály publikus függvényei:

- **Initialize** - A függvény a Direct3D COM objektumainak inicializálásért felel. A Direct3D inicializálása a ID3D11Device és a ID3D11DeviceContext létrehozásával kezdődik. Ez a két interfész a fő Direct3D interfész, ezeken az interfésekken keresztül interakcióba léphetünk a hardverrel, és utasításokat adhatunk ki (például erőforrásokat foglalhatunk le a GPU memóriájában, törölhetjük a hátsó puffert, összekapcsolhatjuk az erőforrásokat a különböző folyamatszakaszokkal, és kirajzolhatjuk a geometriát). Az ID3D11Device interfész a szolgáltatások támogatásának ellenőrzésére és az erőforrások lefoglalására szolgál. Az ID3D11DeviceContext interfész a renderelési állapotok beállítására, az erőforrások grafikus folyamathoz való kötésére és a jelenet kirajzolására szolgál. A Direct3D objektumok létrehozásáról részletesebben a "Introduction to 3D Game Programming with DirectX 11"[12] könyvben olvashatunk.
- **Resize** - Ez a függvény a DirectX rendszerhívásait használja a pufferek átméretezéséhez.
- **CaptureScreen** - Ez a függvény a DirectXTK könyvtárban található SaveWICTextureToFile függvényt használja a megjelenített kép háttértárra történő mentéséhez.
- **BeginScene** - A függvény törli a hátsó és mélységi puffereket a renderelés előtt.
- **EndScene** - A függvény megjeleníti a hátsó puffer a képernyőn a renderelés befejeződése után.

- **Shutdown** - A függvény felszabadítja az erőforrásokat.
- **GetDevice** - A függvény visszaadja az `ID3D11Device` objektumot.
- **GetDeviceContext** - A függvény visszaadja az `ID3D11DeviceContext` objektumot.

GuiView

A 4.4. ábrán látható a Stratégia tervezési minta megvalósítása a grafikus interfész (GUI) és a vezérlő között. A komponensek egy üzenet alapú kommunikációt valósítanak meg a felhasználói bemenet kezelésére, a nézet, `GuiView` egy előre meghatározott üzenet típust és paramétereket továbbít az `IController::HandleMessage` metóduson keresztül a vezérlőnek.



4.4. ábra. `GuiView` és a `IController` kapcsolata

Ez a megközelítés lehetővé teszi, hogy a GUI elemek könnyen kommunikáljanak az `IController` objektummal, anélkül, hogy ismeretük lenne egymás belső működéséről. minden GUI elem egyedi azonosítóval rendelkezik. Ezen azonosítók használata lehetővé teszi a grafikus felületen keresztül érkező utasítások azonosítását, így továbbítva a megfelelő üzenetet az `IController` felé. A további paraméterek lehetővé teszik a felhasználói bemenet részletes leírását, amely alapján az `IController` ob-

jektum kiválaszthatja a megfelelő kezelő függvényt és végrehajthatja a megfelelő műveletet.

4.6.7. A modell réteg

A modell réteg tartalmazza az adatokat és azok logikáját, jelen esetben a térbeli jelenet objektumainak leírását.

A modell interfész

```

1 class IModel
2 {
3 public:
4     virtual ~IModel() = default;
5     virtual bool Initialize(IDataAccessPtr persistence, Microsoft::
6         WRL::ComPtr<ID3D11Device> device, int screenWidth, int
7         screenHeight, float screenNear, float screenDepth, float
8         fieldOfView) = 0;
9     virtual bool Resize(unsigned screenWidth, unsigned screenHeight)
10        = 0;
11    virtual void Shutdown() = 0;
12    virtual bool Render(Microsoft::WRL::ComPtr<ID3D11DeviceContext>
13        deviceContext) = 0;
14    virtual bool HandleMessage(IModelMessageIDs messageID, const std
15        ::vector<std::wstring>& stringParams, const std::vector<float
16        >& fparams, const std::vector<unsigned>& uparams) = 0;
17    virtual bool IsTrajectoryInitialized(void) const = 0;
18 };

```

4.9. forráskód. IModel interfész

- **Initialize** - A függvény az IModel objektum adattagjainak inicializálásért felel.
- **Resize** - A függvény átméretezi a virtuális kamera által használt képarány paramétert.
- **Shutdown** - A függvény a lefoglalt erőforrások felszabadításáért felel.

- **Render** - A függvény elvégzi a modell attribútumainak az aktuális értékeinek lekérdezését, amelyeket átad a `ID3D11DeviceContext` objektumnak, a megfelelő rendszerhívások segítségével.
- **HandleMessage** - A függvény felelős az üzenetek feldolgozásáért és végrehajtássáért, amelyeket a vezérlő küld a modell felé. Az üzenet azonosítóját, valamint az üzenethez tartozó karakterlánc, lebegőpontos és egész típusú paramétereket kapja paraméterként.

A sodrásirány megfordításáért az `IModel` interfész leszármazott osztályai felelnek. Ennek a problémának a matematikai megvalósítását a 2.3.3 fejezet fejti ki részleteiben.

A modell réteg üzenetei

Az enum `IModelMessageIDs` azonosítja azokat az üzeneteket, amelyeket a vezérlő küld a modell rétegnek. A 4.3 táblázat tartalmazza az egyes üzenetek leírását. Az utolsó három oszlop azt jelenti, hogy az adott üzenetben a modell mennyi olyan értéket vár, amely a megadott típusú: **s** esetén `wstring`, **f** esetén `float`, **u** esetén pedig `unsigned`.

Ha a vezérlő üzenet és a modell üzenet között egyértelmű megfeleltetés van, a `IDC2IDM` függvény végzi a két üzenettípus közötti váltást. Ha nincs egyértelmű megfeleltetés, a függvény `IDM_INVALID` értékkel tér vissza.

A modell réteg üzenetei					
Üzenet azonosító	Leírás	s	f	u	
<code>IDM_INVALID</code>	Érvénytelen üzenet.	0	0	0	
<code>IDM_ACTIVATE_FLYTHROUGH_MODE</code>	A Körséta mód aktiválásának üzenete.	0	0	0	
<code>IDM_ACTIVATE_EXPLORE3D_MODE</code>	A Barangolás mód aktiválásának üzenete.	0	0	0	
<code>IDM_LOAD_TERRAIN_SHARP</code>	Egy felületháló fájl megnyitása éles kontúrral.	1	0	0	
<code>IDM_LOAD_PROJECT_SHARP</code>	Több felületháló fájl megnyitása éles kontúrral.	N	0	0	

Üzenet azonosító	Leírás	s	f	u
IDM_LOAD_TERRAIN_SOFT	Egy felületháló fájl megnyitása puha kontúrral.	1	0	0
IDM_LOAD_PROJECT_SOFT	Több felületháló fájl megnyitása puha kontúrral.	N	0	0
IDM_LOAD_TRAJECTORY	Kamera trajektória fájl megnyitása.	1	0	0
IDM_LOAD_CONFIGURATION	Konfigurációs fájl megnyitása.	1	0	0
IDM_ORIGO_SET_LONGITUDE	Az origó hosszúsági fokának beállítása.	0	1	0
IDM_ORIGO_SET_LATITUDE	Az origó szélességi fokának beállítása.	0	1	0
IDM_XZ_PLANE_GRID_SET_ISSEEN	A koordináta-rendszer láthatóságának beállítása.	0	0	1
IDM_PIXELSHADER_SET_SHADING	A pixelshader árnyékolásának beállítása.	0	0	1
IDM_E3D_MOVE_FORWARD	A kamera mozgatása előre Barangolás módban.	0	1	0
IDM_E3D_MOVE_BACK	A kamera mozgatása hátra Barangolás módban.	0	1	0
IDM_E3D_MOVE_LEFT	A kamera mozgatása balra Barangolás módban.	0	1	0
IDM_E3D_MOVE_RIGHT	A kamera mozgatása jobbra Barangolás módban.	0	1	0
IDM_E3D_MOVE_UP	A kamera mozgatása feléle Barangolás módban.	0	1	0
IDM_E3D_MOVE_DOWN	A kamera mozgatása lefelé Barangolás módban.	0	1	0
IDM_E3D_ROTATE	A kamera forgatása Barangolás módban.	0	2	0
IDM_E3D_CAMERA_RESET	A kamera kezdőpozícióba helyezése Barangolás módban.	0	0	0
IDM_E3D_CAMERA_SPEED	A kameramozgatás sebességének beállítása Barangolás módban.	0	1	0

Üzenet azonosító	Leírás	s	f	u
IDM_E3D_ROTATION_SPEED	A kameraforgatás sebességének beállítása Barangolás módban.	0	1	0
IDM_SET_CAMERA_FIELD_OF_VIEW	A kamera látószöge beállításának üzenete.	0	1	0
IDM_SET_CAMERA_NEAR_SCREEN	A kamera előlő vágósíkjának beállítása.	0	1	0
IDM_SET_CAMERA_FAR_SCREEN	A kamera hátsó vágósíkjának beállítása.	0	1	0
IDM_SET_TIME_E3D	Időpont beállítása a Barangolás módban.	0	1	0
IDM_FLYTHROUGH_START	Szimuláció lejátszásának indítása a Körséta módban.	0	0	0
IDM_FLYTHROUGH_PAUSE	Szimuláció lejátszásának szüneteltetése Körséta módban.	0	0	0
IDM_FLYTHROUGH_STOP	Szimuláció lejátszásának megállítása Körséta módban.	0	0	0
IDM_FLYTHROUGH_SET_SPEED	Szimuláció sebességének beállítása Körséta módban.	0	1	0
IDM_FLYTHROUGH_SET_FRAME	Szimuláció adott képkockájának beállítása Körséta módban.	0	0	1
IDM_SET_START_TIME_TRAJECTORY	Szimuláció kezdő időpontjának beállítása Körséta módban.	0	0	1
IDM_MESH_SET_ISSEEN	Egy felületháló elem láthatóságának beállítása.	0	1	1
IDM_MESH_SET_COLOR	Egy felületháló elem színének beállítása.	0	4	1
IDM_MESH_GROUP_SCALE	A felületháló csoport skálázásának beállítása.	0	3	0
IDM_MESH_GROUP_ROTATION	A felületháló csoport Euler-szögeinek beállítása.	0	3	0
IDM_MESH_GROUP_TRANSLATION	A felületháló csoport X, Y, Z koordináták szerinti eltolása.	0	3	0

Üzenet azonosító	Leírás	s	f	u
IDM_TRAJECTORY_SET_ISSEEN	A kamera trajektória vonallánc láthatóságának beállítása.	0	1	0
IDM_TRAJECTORY_ROTATION	A kamera trajektória Euler-szögeinek beállítása.	0	3	0
IDM_TRAJECTORY_TRANSLATION	A kamera trajektória X, Y, Z koordináták szerinti eltolása.	0	3	0
IDM_CLEAR_MESHES	A betöltött felülethálók törlése.	0	0	0
IDM_CLEAR_TRAJECTORY	A betöltött trajektória törlése.	0	0	0

4.3. táblázat. A modell réteg üzenetei, enum IMModelMessageIDs

A modell osztály adattagjai

```

1 class TerrainModel : public IMModel
2 {
3 private:
4     Mode    m_mode = Explore3D;
5     LLACoordinate m_llacoordinate = { 47.497913 , 19.040236 };
6     VertexShaderMeshPtr    m_vertexShaderMesh;
7     PixelShaderMeshPtr   m_pixelShaderMesh;
8     VertexShaderPolyLinePtr m_vertexShaderPolyLine;
9     PixelShaderPolyLinePtr m_pixelShaderPolyLine;
10    Light    m_light;
11    CameraPositioner    m_cameraPositioner;
12    CameraTrajectory    m_cameraTrajectory;
13    CompositeRenderable<VertexMesh>    m_meshes;
14    CompositeRenderable<VertexPolyLine>    m_polylines;
15    Microsoft::WRL::ComPtr<ID3D11Device>    m_device;
16    Microsoft::WRL::ComPtr<ID3D11DeviceContext> m_deviceContext;
17    IDataAccessPtr    m_persistence;
18 public:
19     Camera    m_camera;
20     ModelMessageSystem m_modelMessageSystem;

```

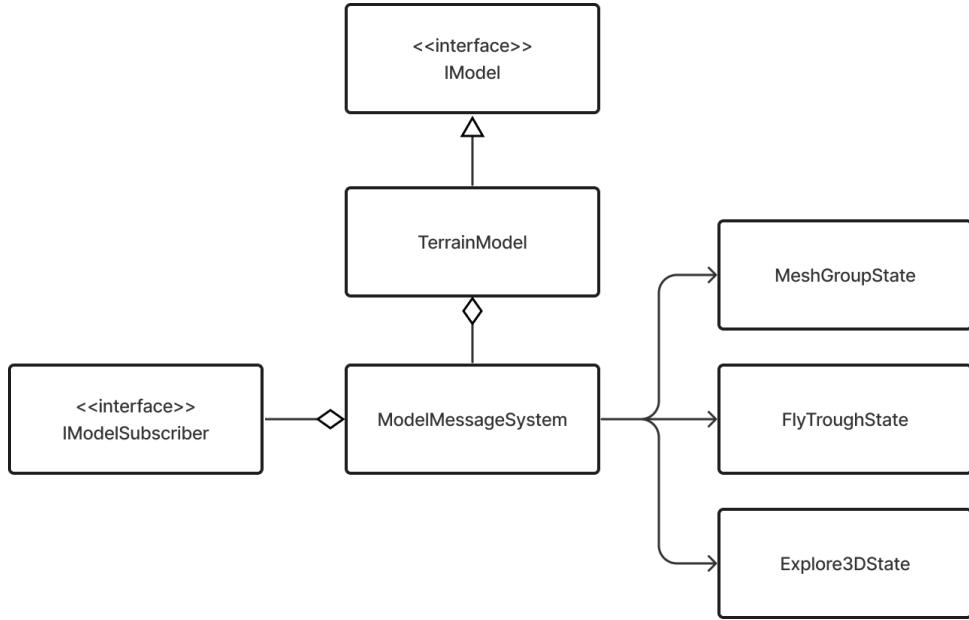
4.10. forráskód. TerrainModel adattagjai

- **Mode** - Felsorolás típus lehetséges értékei: `Explore3D`, `Flythrough`.
- **LLACoordinate** - Struktúra az földrajzi koordináták tárolására.
- **VertexShaderMeshPtr** - A felülethálók Vertex Shader-ét kezelő osztály.
- **PixelShaderMeshPtr** - A felülethálók Pixel Shader-ét kezelő osztály.
- **VertexShaderPolyLinePtr** - A vonalláncok Vertex Shader-ét kezelő osztály.
- **PixelShaderPolyLinePtr** - A vonalláncok Pixel Shader-ét kezelő osztály.
- **Light** - A fény beállításaiért felelő osztály, lásd: 4.6.13. fejezet.
- **CameraPositioner** - A Barangolás üzemmódban a kamera pozícionálásáért felelő osztály, lásd: 4.6.11. fejezet.
- **CameraTrajectory** - A kamera trajektóriát kezelő osztály, lásd: 4.6.12. fejezet.
- **CompositeRenderable<VertexMesh>** - A felülethálókat tartalmazó kompozit osztály, lásd: 4.6.9. fejezet.
- **CompositeRenderable<VertexPolyLine>** - A vonalláncokat tartalmazó kompozit osztály, lásd: 4.6.9. fejezet.
- **IDataAccessPtr** - Perzisztencia kezeléséért felelő osztály, lásd: 4.6.8. fejezet.
- **Camera** - A virtuális kamerát reprezentáló osztály, lásd: 4.6.10. fejezet.
- **ModelMessageSystem** - Az osztály a modellre feliratkozóknak továbbítja a modell aktuális állapotát.

A modell állapotának továbbítása a feliratkozóknak

A modell és a megfigyelők közötti kommunikáció a Megfigyelő (Observer) tervezési minta szerint valósul meg, amint az 4.5. ábrán látható. Az `IModelSubscriber` interfész előírja a `HandleModelState` függvényt, amely képes különböző összetett adatszerkezetek fogadására. Ezekben a struktúrákban tárolja például a nézet a modell aktuális állapotát, majd megjeleníti azt a felhasználói felületen.

Az Megfigyelő tervezési mintát a `ModelMessageSystem` osztály biztosítja. Az objektumra `IModelSubscriber` típusú megfigyelők iratkozhatnak fel. Ha az `IModel` objektum állapota változik, a `ModelMessageSystem` objektum a feliratkozókat értesíti a `PublishModelState` metóduson keresztül.



4.5. ábra. A modell állapotának továbbítása a vezérlő és nézet felé

4.6.8. Perzisztencia

A perzisztencia szerepe az adatok tartós tárolása és visszaállítása a program futása közben. A program perzisztencia rétege biztosítja, hogy a 3.7 fejezetben meghatározott típusú külső állományokat a felhasználó betölthesse a program memóriájába.

A perzisztencia interfész

```

1 class IDataAccess
2 {
3 public:
4     virtual ~IDataAccess() = default;
5     virtual bool LoadTerrain_withSharpEdges(const wchar_t*) = 0;
6     virtual const std::vector<stlFacet>& GetFacets_Sharp(void) = 0;
7     virtual bool LoadCameraTrajectory(const wchar_t*, std::vector<
        CameraPose>&) = 0;
8     virtual bool LoadParameterFile(const wchar_t*, ParameterFile&
        params) = 0;
9     virtual bool LoadTerrain_withSoftEdges(const wchar_t*) = 0;
10    virtual const std::vector<StlVertex>& GetVertices_Soft() = 0;
11    virtual const std::vector<CornerIndices>& GetIndices_Soft() = 0;
12    virtual void Clear() = 0;
13 }
  
```

4.11. forráskód. IDataAccess interfész

- `LoadTerrain_withSharpEdges` - A függvény a háttértárról betölti az adott felülethálót éles élekkel.
- `GetFacets_Sharp` - Amennyiben a `LoadTerrainSharpEdges` függvénnnyel a felületháló betöltése sikeres, a függvény megadja a memóriába töltött háromszögek listáját.
- `LoadCameraTrajectory` - A függvény a háttértárról betölti az adott kamera trajektóriát.
- `LoadParameterFile` - A függvény a háttértárról betölti a konfigurációs fájlt.
- `LoadTerrain_withSoftEdges` - A függvény a háttértárról betölti az adott felülethálót puha élekkel.
- `GetVertices_Soft` - Amennyiben a `LoadTerrainSoftEdges` függvénnnyel a felületháló betöltése sikeres, a függvény megadja a memóriába töltött csúcsok listáját.
- `GetIndices_Soft` - Amennyiben a `LoadTerrainSoftEdges` függvénnnyel a felületháló betöltése sikeres, a függvény megadja a memóriába töltött lapok csúcs-pontjainak indexeit.

Megvalósítás

A megvalósított `BinaryFileDataAccessAsync` osztály a felülethálók betöltését több szálon valósítja meg, multithread programozással. Mind a `LoadTerrainSharpEdges`, mind a `LoadTerrainSoftEdges` függvény elsőként ellenőrzi, hogy a bináris .stl megfelel-e a 3.7 fejezetben leírtaknak, amennyiben nem, akkor a függvény kivételt dob. A nagyméretű .stl fájlok miatt szükség volt a többszálon való betöltésre, a függvények a fájl méretétől függően határozzák meg, hogy hány szálra van szükség a betöltéshez. A szálak kezelésére létrehoztam egy `ICallable` interfész, amely előírja az `operator[]` használatát. A chunk-ok beolvasását `ICallable` leszármazott osztályai valósítják meg.

- `LoadTerrain_withSharpEdges` - A függvény a feldarabolt külső állomány megfelelő részének betöltését a `ReadSTLChunkSharp` objektumoknak delegálja. A szálak külön listákba gyűjtik a beolvasott háromszögeket, így kölcsönös

kizáráusra nincs szükség. A függvény végén a szálak által összegyűjtött háromszöglisták egyesítésre kerülnek.

- **LoadTerrain_withSoftEdges** - A függvény a feldarabolt külső állomány megfelelő részének betöltését a `ReadSTLChunkSoft` objektumokra bízza. A csúcsPontokhoz tartozó normálvektorokat a szálak a 2.5 fejezetben leírtaknak megfelelően határozzák meg, azaz az oldalak normálvektorainak súlyozott összege határozza meg a csúcspontokhoz tartozó értékeket. A csúcspontok összegyűjtése során azonban figyelembe kell venni az oldalak közös csúcspontjait, ezeket egy közös hasítótáblába gyűjtik a szálak, minden új csúcspont pedig egyedi azonosítót kap. A munka során felmerülő kölcsönös kizáráskor miatt az `std::shared_mutex` kerül alkalmazásra.

4.6.9. Felülethálók, vonalláncok

A program a térbeli elemek megjelenítéséhez az `IRenderable` interfész használatát követeli meg. Az interfész egy sablonparamétert (`V-t`) tartalmaz, ami a csúcsPontok típusát határozza meg.

Az `IRenderableBase` osztály gondoskodik arról, hogy minden megjelenítendő objektumnak egyedi azonosítója legyen.

Az `IRenderable` interfésből származó osztályok a Kompozit tervezési mintát valósítják meg. Ennek köszönhetően az `IModel` objektum nem tudja pontosan, hogy milyen megjelenítendő elemeket tartalmaz. A kirajzolás során az `IModel` egyszerűen meghívja az `IRenderable` interfész `Render` függvényét. A megjelenítési feladat azonban már a kompozit osztály felelőssége, amely gondoskodik arról, hogy minden felületháló és vonallánc kirajzolásra kerüljön.

```

1 template <class V>
2 class IRenderable : public IRenderableBase
3 {
4 public:
5     IRenderable() = default;
6     virtual ~IRenderable() = default;
7     virtual bool Initialize(ComPtr<ID3D11Device>, IVertexShaderPtr,
8                             IPixelShaderPtr, V*, unsigned long*, UINT, UINT) = 0;
9     virtual void Shutdown() = 0;
10    virtual void Render(ComPtr<ID3D11DeviceContext>, XMMATRIX,
11                         XMMATRIX, XMMATRIX, const Light&) = 0;
12    virtual void SetName(const std::wstring& name) = 0;
13    virtual void Rotate(float pitch, float yaw, float roll) = 0;
14    virtual void Translate(float x, float y, float z) = 0;
15    virtual void Scale(float x, float y, float z) = 0;
16    virtual void ResetTransformation() = 0;
17    virtual void SetColor(float r, float g, float b, float a) = 0;
18    virtual void SetIsSeen(bool isSeen) = 0;
19    virtual bool IsSeen(void) const = 0;
20    virtual IRenderableState GetState(void) const = 0;
21    virtual std::wstring GetName(void) = 0;
22    virtual XMMATRIX GetWorldMatrix(void) = 0;
23 };

```

4.12. forráskód. IRenderable interfész

4.6.10. Kamera

A program virtuális kameráját a `Camera` osztály valósítja meg. Az osztály adattagjai tartalmazzák a nézet- és projekciós mátrix elkészítéséhez szükséges paramétereket. Ezen attribútumok: a pozíció, a három Euler-szög, a függőleges látószög, a képarány, a távoli és a közeli vágósík távolsága. Az attribútumok beállítására megfelelő setter műveletek rendelkezésre állnak. A `Camera::Render` függvény végzi el a nézet- és vetítési mátrix számítását a megfelelő DirectX rendszerhívások segítségével.

A program közös `Camera` objektumot használ a Barangolás és Körséta módban. A két mód kamera kezelését két különböző osztály valósítja meg.

4.6.11. Barangolás mód kamera kezelése

A Barangolás mód kamera kezelését a `CameraPositioner` osztály valósítja meg, amely aggregálja a `Camera` objektumot és a megfelelő műveleteinek segítségével állítja be a kamera paramétereit. A `Move<direction>` függvények paraméterül a két képfrissítés óta eltelt időt kapják meg, amely alapján kiszámítják a kamera aktuális pozícióját. A `RotatePitchYaw` függvény paraméterül a pitch és yaw szögeket kapja meg, amelyek alapján beállítja a kamera aktuális orientációját.

```

1 void Initialize(Camera* camera);
2 void MoveForward(float dt);
3 void MoveBack(float dt);
4 void MoveLeft(float dt);
5 void MoveRight(float dt);
6 void MoveUp(float dt);
7 void MoveDown(float dt);
8 void RotatePitchYaw(float x, float y);
```

4.13. forráskód. `CameraPositioner` függvényei

4.6.12. Körséta mód kamera kezelése

A Körséta módban a kamerakezelést a `CameraTrajectory` osztály végzi. Az osztály felelősségi köre a 3.7 szakaszban leírt kamera trajektória fájl tartalmának kezelése. A `CameraTrajectory` objektum referencián keresztül hivatkozik a trajektória fájl betöltése során létrehozott vonalláncra, amely a kamera útvonalát jeleníti meg a felhasználó számára.

A `CameraTrajectory` osztály adattagjai:

- `EpochTime m_start` - A kamera trajektória fájl első képkockájához tartozó Unix időpont.
- `double m_elapsedmsec` - A trajektória lejátszása során az első képkockától eltelt aktuális idő milliszekundumban.
- `std::vector<double> m_elapsedmsecs` - A kamera trajektória fájl alapján az első képkockától mért milliszekundumok felsorolása.
- `std::vector<Vector3D> m_positions` - A kamera trajektória fájl alapján a képkockákhoz tartozó térbeli pontok felsorolása.

- `std::vector<Vector3D> m_rotations` - A kamera trajektória fájl alapján a képkockákhoz tartozó Euler-szögek felsorolása.
- `IRenderablePtr<VertexPolyLine> m_polyLine` - A kamera útvonalát representáló vonallánc.
- `CameraPtr m_camera` - Az aggregált kamera objektum.

A Körséta módban a `CameraTrajectory::UpdateCamera(double dt)` függvényen keresztül kapja meg a két képkoca frissítése között eltelt időt, milliszekundumban. A szimuláció lejátszása során az aktuális pozíciót és a forgatási szögeket két külön függvénnyel lehet meghatározni, amelyek az eltelt idő függvényében határozzák meg a függvényértéket. Az eltelt időt a `double m_elapsedmsec` adattag tartalmazza, amely az indulási időponttól eltelt összes időt mutatja.

A interpolációs folyamat során lineárisan közelítjük a direkt pontokat, amelyeket a kamera trajektória fájl alapján ismerünk. Ez lehetővé teszi számunkra, hogy bármely kezdő és vég időpont intervallumba eső t paraméterhez meghatározzuk a függvény értékét.

Fontos megjegyezni, hogy az Euler-szögek értelmezési tartománya $[-\pi, +\pi]$. Ha az interpoláció során az intervallum két végén próbálunk interpolálni, akkor előfordulhat, hogy a kamera hirtelen π fokot fordul. Ez a probléma abból adódik, hogy az interpoláció eredménye 0-hoz közelí szám lesz.

A probléma megoldása érdekében az 1. algoritmus használható, ahol $f = (t - t_i)/(t_{i+1} - t_i)$, t_i és t_{i+1} a két ismert diszkrét időpont, amelyek közé esik a t paraméter. A ϕ_i a t_i időponthoz tartozó elfordulási érték.

1. algoritmus Circular Interpolation algorithm

Func CircularInterpolation(ϕ_i, ϕ_{i+1}, f)

```

1:  $\delta := \phi_{i+1} - \phi_i$ 
2:  $\delta := ((\delta + \pi)\%2\pi) - \pi$ 
3:  $\alpha := \phi_i + \delta \cdot f$ 
4: if  $sgn(\alpha) >= 0$  then
5:   return  $(\alpha + \pi)\%2\pi - \pi$ 
6: else
7:   return  $(\alpha + \pi)\%2\pi + \pi$ 
8: end if
```

4.6.13. A fény osztály

A programban szereplő fényt a Light osztály reprezentálja, ami egy párhuzamos megvilágítást valósít meg. Az osztály két tagváltozója, a `m_ambientColor` és `m_diffuseColor`, az adott fényforrás környezeti és szórt összetevőinek színét tárolják `XMFLOAT4` típusú vektorokban. Az osztály további metódusai lehetővé teszik az adott fényforrás tulajdonságainak lekérdezését, például az `GetAmbientColor`, `GetDiffuseColor` és `GetInverseDirection` metódusokkal.

Az `UpdateSunPosition` metódus számolja ki az adott földrajzi koordinátához és adott Unix-időhöz tartozó Nap azimut és zenit szögértékeit. Ezt a számítást a `SunPos` [18] függvény segítségével végzi. Az eredményül kapott értékeket felhasználja a fény irányának meghatározására a derékszögű koordinátarendszerben, amelyet a `CalculateInverseDirectionBySunPosition` metódus végez el. Ez a metódus a meghatározott gömbi koordinátákból kiszámolja a fény irányvektorát a derékszögű koordinátarendszerben.

Az `UpdateSunPosition` és `SetInverseDirectionBySunPosition` metódusok használatával a Light osztály biztosítja a megfelelő fényirányt, amelyet a Pixel Shader használ az adott jelenet árnyalásának kiszámításához.

4.7. Tesztelés

A validáció és verifikáció során ellenőriznünk kell, hogy a program megfelel a vele szemben támasztott funkcionális és nem funkcionális követelményeknek. A program üzleti logikájának tesztelésére automatikusan végrehajtható egységtesztek születtek, amelyekkel biztosíthatjuk az építőelemek helyes működését mind a vezérlő, mind a modell réteg osztályainak esetében. Az egységtesztek fehér doboz elvét követik, amely azt jelenti, hogy a teszt írása közben a kód ismert a tesztelő számára.

A GUI és a megjelenítés tesztelése kézi módon történt. Azért, hogy a kézi tesztelés a lehető legtöbb hibát felderítse, a teszteket nem csak fejlesztő, hanem fejlesztésben nem résztvevő emberek is elvégezték. Így megvalósítva a fekete dobozos tesztelést. A megjelenítő teszteléséhez egy hasonló 3D-s megjelenítő szoftvert, a Blender-t [32] használtam referenciaként annak ellenőrzésére, hogy a modellek helyesen jelennek-e meg.

4.7.1. Az üzleti logika tesztelése egységesztekkel

Az üzleti logika teszteléséhez egységeszteket alkalmaztam, az egyes komponensekhez külön projekt fájlokat hoztam létre. A tesztelés során az egyes modell és vezérlő réteg elemeit külön-külön teszteltem. Annak érdekében, hogy az osztályokat egymástól függetlenül tesztelhessem, alkalmaztam a Google Test [29] által biztosított Google Mock-ot is. A mock objektumok lehetővé teszik, hogy utánozzuk a valós objektumok viselkedését, ezáltal az egységtesztelés során a különböző osztályokat izoláljuk egymástól. Ez a megközelítés csökkenti a külső függőségek számát a tesztelés során.

4.7.2. Modell réteg tesztelése

A modell réteg a TerrainRender_ModelUnitTest projekt fájlban található. Az egyes osztályok külön forráskódban lettek tesztelve.

4.7.3. Perzisztencia ellenőrzése

FileDataAccessUnitTest.cpp fájl tartalma:

- **Érvénytelen elérési út megadása:** Az osztály kivétel dobással jelzi az érvénytelenséget.

A hozzá tartozó tesztesetek:

- Test_LoadConfigurationFile_InvalidPath,
- Test_LoadCameraTrajectory_InvalidPath,
- Test_LoadTerrain_withSoftEdges_InvalidPath,
- Test_LoadTerrain_withSharpEdges_InvalidPath.

- **Nem megfelelő formátumú fájl betöltése:** Az osztály kivétel dobással jelzi az érvénytelenséget.

A hozzá tartozó teszteset:

- Test_LoadConfigurationFile_Incorrect.
- Test_LoadCameraTrajectory_Incorrect.
- Test_LoadTerrain_withSoftEdges_Incorrect,
- Test_LoadTerrain_withSharpEdges_Incorrect.

- **Megfelelő formátumú konfigurációs fájl betöltése:** A konfigurációs fájl értékei helyesen vannak kiolvasva a fájlból, amelyek különböző Assert-ekkel ellenőriztem.

A hozzá tartozó teszteset:

- `Test_LoadConfigurationFile`.

- **Megfelelő formátumú trajektória fájl betöltése:** A trajektória fájl értékei helyesen vannak kiolvasva a fájlból, amelyek különböző Assert-ekkel ellenőriztem.

A hozzá tartozó teszteset:

- `Test_LoadTrajectoryFile`.

- **Megfelelő formátumú modellfájl betöltése:** A helyes betöltést az ellenőrizte, hogy a program a modellfájl beolvasása során megfelelő mennyiségű csúcsPontot és indexet talál. Az elvárt mennyiséget a MeshLab [33] program segítségével határoztam meg.

A hozzá tartozó tesztesetek:

- `Test_LoadTerrain_withSoftEdges`,
- `Test_LoadTerrain_withSharpEdges`.

4.7.4. Segédosztályok ellenőrzése

A lineáris interpoláció tesztelése a `IInterpolationUnitTest.cpp` fájlból található. A tesztesetek során ellenőriztem, hogy az alkalmazott bináris keresés helyesen működik. Az alkalmazott bináris keresés a legnagyobb keresett elemnél kisebb vagy egyenlő elemet keresi egy rendezett tömbben. Amennyiben nem találja megfelelő elemet, a függvény -1-es index értékkel tér vissza. A tesztesetek a bináris kereső algoritmus helyességét ellenőrzik különböző bemeneti adatokkal, és az algoritmus elvárt kimenetét hasonlítják össze az elvárt indexszámmal.

Ellenőrzésre került, hogy mi történik, ha a keresett elemnél minden elem nagyobb a tömbben, ha a tömb üres, ha a tömbben minden érték egyenlő, ha a tömb egy konkrét értékét keressük, illetve, ha egy olyan elemet keresünk, amelynél van kisebb vagy egyenlő a tömbben.

A `IInterpolationUnitTest.cpp` fájl teszteli a `LinearInterpolation` és `CircularInterpolation` osztályokat is. A tesztesetek különböző értelmezési tar-

tománybeli elemekhez keresik a megfelelő interpolált értéket, amelyeket az elvárt értékkel összehasonlítanak. A tesztesetek az üres tömb esetét is vizsgálják. A **CircularInterpolation** az értelmezési tartomány két széle közötti interpolációt is vizsgálja ($-\pi + \epsilon, \pi - \epsilon$ esete).

Az `EpochTimeUnitTest.cpp` fájlban található tesztek az `EpochTime` osztály tesztelésére szolgálnak. Az osztály egy időpontot reprezentál másodpercekben és nanoszekundumokban. Ellenőrzésre kerültek az `EpochTime`-on végezhető metódusok és a bináris műveletek. Az összes tesztesetben az elvárt értékek összehasonlításra kerültek az aktuális értékkal, így biztosítva a helyes működést.

A műveletekhez tartozó tesztesetek:

- `Test_OperatorPlus`,
- `Test_OperatorMinus`,
- `Test_DiffInMilliSec`,
- `Test_AddMilisec`.

4.7.5. IRenderable interfész leszármazott osztályainak ellenőrzése

A leszármazott osztályokat a `CompositeRenderableUnitTest.cpp`, `PolygonMeshUnitTest.cpp`, `PolyLineUnitTest.cpp` forráskódokban ellenőriztem.

- **Érvénytelen inicializálás:** Az osztály hamis értékkel jelzi az érvénytelenséget.

A hozzá tartozó tesztesetek, mindegyik forrásfájlban megtalálhatóak:

- `Test_InitializeReturnsFalseWithNullVertexShader`,
- `Test_InitializeReturnsFalseWithNullDevice`,
- `Test_InitializeReturnsFalseWithNullPixelShader`.

- **Érvényes inicializálás:** Az osztály igaz értékkel jelzi az érvényességet.

A hozzá tartozó tesztesetek, mindegyik forrásfájlban megtalálható:

- `Test_Initialize`.

- **Transzformációk végrehajtása:** A transzformációk végrehajtása során megadott értékek helyességét különböző Assert-ek segítségével ellenőriztem, össze-

hasonlítva az elvárt értékekkel.

A hozzá tartozó tesztesetek, mind a három forrásfájlban megtalálhatóak:

- `Test_Rotate`,
- `Test_Translate`,
- `Test_Scale`.

- **Kompozit osztály műveletei:** A kompozit osztály rendelkezik olyan egyedi műveletekkel, amelyekkel az interfész nem. Ellenőriztem, hogy a kompozit osztályhoz hozzá lehet-e adni levél elemet, el lehet-e távolítani egy megadott levél elemet. Amennyiben a művelet sikeres, igaz értéket szükséges visszakapni. Ellenőriztem, hogy egy azonosítóval megadott elem módosítható-e a kompoziton belül.

A hozzá tartozó tesztesetek, csak a `CompositeRenderableUnitTest.cpp` forrásfájlban találhatóak:

- `Test_SetColorComponent`,
- `Test_ClearRenderable`,
- `Test_Remove`,
- `Test_Add`.

4.7.6. Kamerakezelés ellenőrzése

A `CameraUnitTest.cpp` fájlban a Camera osztály kezelésével kapcsolatos teszteket találunk.

- **Inicializálás:** Az osztály kivételdobással jelzi az érvénytelen adatokkal történő inicializálást. Illetve igaz értékkel jelzi, ha az inicializálás rendben ment.

A hozzá tartozó tesztesetek:

- `Test_InitializeInvalid`
- `Test_Initialize`

- **Átméretezés:** Az osztály kivételdobással jelzi az érvénytelen adatokkal történő átméretezést. Az átméretezés végrehajtása során megadott értékek helyeségét különböző `Assert`-ek segítségével ellenőriztem, összehasonlítva az elvárt értékekkel.

A hozzá tartozó tesztesetek:

- Test_Resize

- **Transzformáció végrehajtása:** A transzformáció végrehajtása során megadott értékek helyességét különböző Assert-ek segítségével ellenőriztem, összehasonlítva az elvárt értékekkel. Illetve ellenőrzésre került, hogy az aktuális projekciós és nézet mátrixok megfelelnek-e az elvárt értékeknek.

A hozzá tartozó tesztesetek:

- Test_AdjustPosition
- Test_AdjustRotation
- Test_Render

A CameraTrajectoryUnitTest.cpp fájlban a CameraTrajectory osztály kezelésével kapcsolatos teszteket találunk.

- **Inicializálás:** Az osztály kivételdobással jelzi az érvénytelen adatokkal történő inicializálást. Illetve igaz értékkel jelzi, ha az inicializálás rendben ment.

A hozzá tartozó tesztesetek:

- Test_InitializeInvalid
- Test_Initialize

- **Kamera irányítása a kameraútvonal mentén:** A trajektória osztály tartalmazza a kameraútvonal meghatározott pozícióit, az UpdateCamera metódus az idő függvényében frissíti a kamera állapotát. A kamera frissítése során megadott értékek helyességét különböző Assert-ek segítségével ellenőriztem, összehasonlítva az elvárt értékekkel. Amennyiben a megadott időparaméterrel elvégezhető az interpolálás, igaz értékkel tér vissza, az intervallumon kívüli érték esetén hamis értékkel tér vissza a függvény.

A hozzá tartozó tesztesetek:

- Test_UpdateCamera

A CameraPositionerUnitTest.cpp fájlban a CameraPositioner osztály kezelésével kapcsolatos teszteket találunk.

- **Inicializálás:** Az osztály kivételdobással jelzi az érvénytelen adatokkal történő inicializálást. Illetve igaz értékkel jelzi, ha az inicializálás rendben ment.

A hozzá tartozó tesztesetek:

- Test_InitializeInvalid
- Test_Initialize

- **Kamera irányítása a Barangolás módban:** A Barangolás módban a bilentyűzettel és az egérrel irányíthatjuk a kamerát. A kamera frissítése során megadott értékek helyességét különböző Assert-ek segítségével ellenőriztem, összehasonlítva az elvárt értékekkel. A teszteset ellenőrzik, hogy a kamera helyes pozícióban van-e a sebesség, az idő és a mozgás iránya alapján.

A hozzá tartozó tesztesetek:

- Test_MoveForward
- Test_MoveBack
- Test_MoveUp
- Test_MoveDown
- Test_MoveLeft
- Test_MoveRight
- Test_Rotate

4.7.7. A fényirány ellenőrzése

A Light osztály helyes működésének ellenőrzése érdekében különböző funkcióinak tesztelése a LightTest.cpp fájlban kerül végrehajtásra. A Light::UpdateSunPosition metódust véletlenszerű időpontok és földrajzi koordináták megadásával teszteltem. Az egyes tesztesetekben ellenőriztem, hogy a GetAzimuth és GetElevation metódusok visszatérnek-e az elvárt értékkal bizonyos hibahatáron belül, adott időpont és helyszín esetén. Az elvárt értékeket a "NOAA Solar Position Calculator" [34] segítségével határoztam meg.

Ha az ellenőrzések sikeresen teljesülnek, akkor az azt jelenti, hogy az osztály feltehetően helyesen működik, és valószínű, hogy az összes többi helyen is helyesen fog működni.

4.7.8. A modell komponens ellenőrzése

A modell komponens ellenőrzése a TerrainModelTest.cpp fájlban történt. Annak érdekében, hogy a modell a perzisztenciától függetlenül tesztelhető legyen,

létrehoztam egy `MockIDataAccess` osztályt, amivel megszüntettem a perzisztenciától való függőséget.

- **A modell megfigyelése:** A modell tulajdonságainak ellenőrzésére létrehoztam egy csak a tesztelésre használt `TerrainModelSubscriber` osztályt, ami a modell állapotának változásakor értesül az eseményről. Ezáltal a `ModelMessageSystem` is tesztelésre került a `ModelMessageSystemUnitTest.cpp` fájlban.

A hozzá tartozó tesztesetek:

- `Test_SubscribeUnsubscribe`
- `Test_PublishModelState`

- **Inicializálás:** Az osztály kivételdobással jelzi az érvénytelen adatokkal történő inicializálást. Illetve igaz értékkel jelzi, ha az inicializálás rendben ment.

A hozzá tartozó tesztesetek:

- `Test_InitializeInvalid`
- `Test_Initialize`

- **Átméretezés:** Az osztály kivételdobással jelzi az érvénytelen adatokkal történő átméretezést. Az átméretezés végrehajtása során megadott értékek helyességét különböző `Assert`-ek segítségével ellenőriztem, összehasonlítva az elvárt értékekkel.

A hozzá tartozó teszteset:

- `Test_Resize`

- **Üzenetek fogadása:** A modell komponens a `HandleMessage` függvényen keresztül képes fogadni különböző típusú üzeneteket. Az üzeneteket típusuk szerint csoportosítottam, és minden egyes csoporthoz önálló tesztesetet rendeltettem. A tesztesetekben ellenőriztem, hogy az üzenethez megfelelő számú és típusú paraméter tartozik-e. Ha az előírt mennyiségű paraméter nem érkezik meg az üzenettel, a függvény hamis értékkel tér vissza. Az egyes üzenettípusok helyes kezelése az elvárt eredményekkel való összehasonlítással került meghatározásra. Ehhez a `TerrainModelSubscriber` osztályt használtam, amely segítségével lekérdezhettem a modell aktuális állapotát.

- `Test_HandleMessage_IDM_INVALID`

- Test_HandleMessage_IDM_LOAD_TERRAIN_SHARP
- Test_HandleMessage_IDM_LOAD_TERRAIN_SOFT
- Test_HandleMessage_IDM_LOAD_PROJECT_SOFT
- Test_HandleMessage_IDM_LOAD_PROJECT_SHARP
- Test_HandleMessage_IDM_LOAD_CONFIGURATION
- Test_HandleMessage_IDM_LOAD_CAMERA_TRAJECTORY
- Test_HandleMessage_IDM_ORIGO_SET_LONGITUDE_LATITUDE
- Test_HandleMessage_IDM_E3D
- Test_HandleMessage_IDM_SET_CAMERA
- Test_HandleMessage_IDM_FLYTHROUGH
- Test_HandleMessage_IDM_SET_TIME
- Test_HandleMessage_IDM_MESH_GROUP

4.7.9. A vezérlő réteg tesztelése

A vezérlőréteg tesztelése a `TerrainRender_ControllerUnitTest` projekt fájlban található. Az egyes osztályokat külön forráskódban teszteltem. Annak érdekében, hogy a vezérlő osztályok önmagukban ellenőrizhetőek legyenek, létrehoztam `MockIView`, `MockIModel`, `MockKeyboard`, `MockMouse` osztályokat, ezzel csökkentve a külső függőségeket.

A leszármazott osztályok és a hozzájuk tartozó forrásfájlok, amelyek tartalmazzák az osztály teszteseteit:

- `ControllerExplore3D` - `ControllerExplore3DUnitTest.cpp`,
 - `ControllerFlythrough` - `ControllerFlythroughUnitTest.cpp`,
 - `ControllerGui` - `ControllerGuiUnitTest.cpp`,
 - `ControllerRouter` - `ControllerRouterUnitTest.cpp`.
-
- **Inicializálás:** Az osztály hamis értékkel jelzi az érvénytelen adatokkal történő inicializálást. Illetve igaz értékkel jelzi, ha az inicializálás rendben ment.
A hozzá tartozó teszteset valamennyi forrásfájlból:
 - `Test_Initialize`.
 - **Üzenetek fogadása:** A vezérlő komponens a `HandleMessage` függvényen keretrendszerrel képes fogadni különböző típusú üzeneteket. Az üzeneteket típusuk szerint csoportosítottam, és minden egyes csoporthoz önálló Assertet rendeltem.

Ellenőriztem, hogy az üzenethez megfelelő számú és típusú paraméter tartozik-e. Ha az előírt mennyiségű paraméter nem érkezik meg az üzenettel, a függvény hamis értékkel tér vissza. Az egyes üzenettípusok helyes kezelése az elvárt eredményekkel való összehasonlítással került ellenőrzésre. Az ellenőrzés során azt vizsgáltam, hogy a vezérlő a megfelelő üzenetet adja át a mockolt modellnek vagy nézetnek. Ennek érdekében a tesztesetek ellenőrzik, hogy a mock objektumok megfelelő függvényei, a megfelelő paraméterekkel meghívásra kerültek-e.

- Test_HandleMessage

- **Vezérlő hozzáadása a kompozit osztályhoz:** Ennek ellenőrzését kizárolag a ControllerRouterUnitTests.cpp fájlban végeztem el.

A hozzá tartozó teszteset:

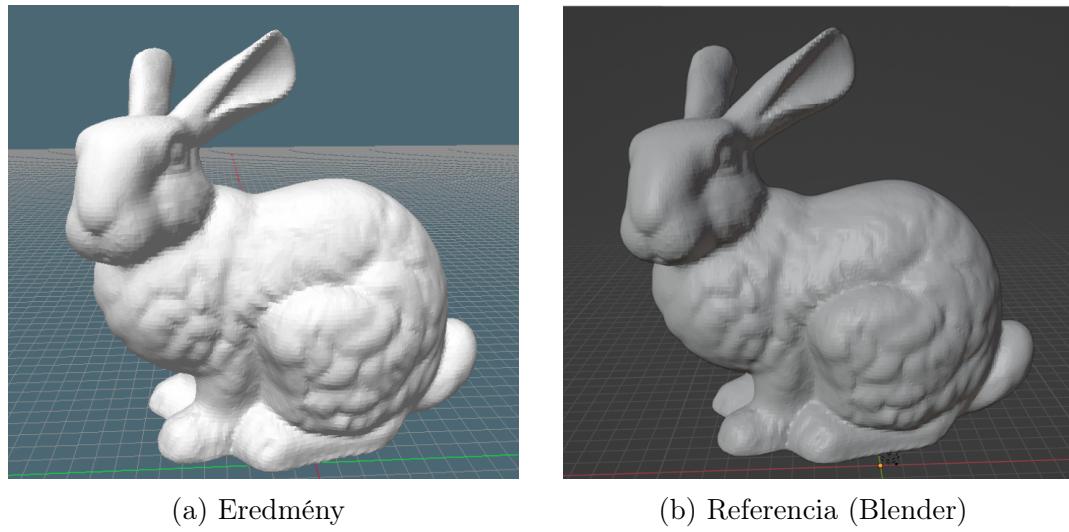
- Test_AddController_RemoveController.

4.7.10. A megjelenítés ellenőrzése

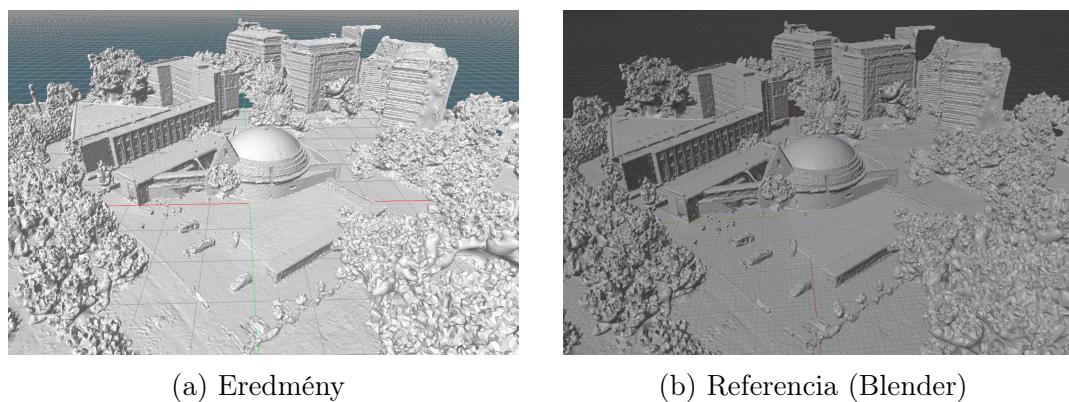
A megjelenítés ellenőrzése során azt vizsgáltam, hogy a program által megjelenített 3D modell megfelelően jelenik e meg. A teszteléshez a Blender [32] nevű 3D-s szoftvert használtam referenciaként, amelynek segítségével ellenőriztem, hogy a program megfelelően jeleníti meg a modelleket.

A teszteléshez készítettem két referencia eredményképet, amelyek segítségével bemutatom, hogy a modellek helyesen jelennek meg. Az eredményképek alapján a program megfelelően kezeli a modellfájlokat és azokat helyesen jeleníti meg a felhasználónak.

A Blender használata referenciaként elősegíti a tesztelés megbízhatóságát és az eredmények pontosabb értelmezését, mivel a Blender egy elismert 3D-s szoftver, amely széles körben használatos a modellek tervezésére és megjelenítésére.



4.6. ábra. Stanford Bunny 70k megjelenítése



4.7. ábra. Budapesti Magyar Szentek templomáról készített felületháló megjelenítése

5. fejezet

Összegzés

Egy tetszőleges térbeli jelenet megjelenítéséhez három tényezőt szükséges meg-
határoznunk: a megjelenítendő tárgy geometriai modelljét, a virtuális kamerát és
a jelenet fényforrásait. A szakdolgozat célja egy olyan szimulációs környezet lét-
rehozása, amelybe a programba betöltött felülethálót úgy helyezhetjük el, hogy a
felhasználó azt érezze, mintha a jelenet adott földrajzi koordinátán és adott UTC
időpontban lenne.

A számítógépes grafika különböző rendszerei gyakran eltérő koordináta-
rendszeret használnak, ezért a jelenetek megjelenítésekor figyelembe kell venni
az eltéréseket, hogy a jelenet megfelelően jelenjen meg a képernyőn. Erre láthattunk
példát, hogy a sodrásirány megváltozása milyen következményeket von maga után.

A térbeli jelenetek modelljeit affin transzformációk segítségével helyezhetjük el
a viszonyítási rendszerben. A modell felületelemeihez tartozó normál vektorok szá-
mítása kulcsfontosságú a modell megjelenítésében. A triangulált felületháló normál-
vektorát könnyű meghatározni, ha a modellt konstans árnyalással szeretnénk meg-
jeleníteni, mert a háromszögek élei egyértelműen meghatározzák a normálvektor
irányát. Az interpolált árnyalás eléréséhez azonban más módszert kell alkalmazni.
Az egy csúcsponthoz tartozó normálvektort a közös csúccsal rendelkező háromszögek
normálvektorainak lineáris kombinációjaként határozhatjuk meg. A súlyok kialakí-
tására több módszer érhető el a szakirodalomban. A program elkészítése során egy
olyan algoritmust választottam, amely segítségével az eltérő méretű háromszögek
találkozása is kezelve van, így egy hiteles közelítést ad a kiszámított normálvektor.

A program lehetővé teszi, hogy a felhasználó egy adott kameraútvonal mentén
tekintse meg a betöltött felületi hálót. A program biztosítja, hogy a trajektória és

a felületháló térbeli viszonyát a felhasználó tetszés szerint beállíthassa. A kamera útvonal transzformációja azonban hatással van a képalkotás folyamatára. Hiszen a képalkotás során felhasznált nézeti transzformációt a kameránk adott pozíciója és orientációja határozza meg. Azonban a kameraútvonal transzformálása során ezeket az értékeket újra meg kell határoznunk, hiszen módosultak a világ koordinátarendszerhez viszonyítva. A módosított orientációt a transzformációs mátrixok összeszorzása után, a végső forgatási mátrixból meghatározhatjuk a kamera megfelelő tájolását.

A számítógépes grafikában a megfelelő megvilágítás segítségével érhetjük el a kívánt hatás, és mondhatjuk, hogy a világítás a grafika lelke. A szimuláció során a pozíció és az időpont figyelembevételével lehet meghatározni a párhuzamos megvilágítás irányát. A megvilágítás irányát felhasználva a megfelelő árnyalásban jeleníthetjük meg a térbeli modellünket.

Köszönyetnyilvánítás

Köszönnettől tartozom Hiba Antalnak, külső konzulensemnek, aki segített nekem témát választani és a dolgozattal kapcsolatos kérdéseimre készségesen válaszolt. A felvázolt szempontok és javaslatok sokat segítettek nekem.

Fábián Gábornak, belső témavezetőmnek köszönöm az irányítását és támogatását, amely nélkül nem tudtam volna a dolgozatomat a tervezett időn belül elkészíteni. A tanácsai és javaslatai segítettek nekem, hogy meghatározzam a dolgozat irányát és céljait.

Köszönöm mindenkit témavezetőmnek, hogy szakértelmükkel és elkötelezettségükkel segítettek nekem abban, hogy átfogó képet alkossak a témáról. Nagyra értékeltem a hozzáállásukat és a rugalmasságukat, amely lehetővé tette számomra, hogy a dolgozat írás során minden kihívással szembenézzek és sikeresen megoldjam azokat.

Végül, de nem utolsósorban, szeretném megköszönni családomnak és barátaimnak a támogatásukat és bátorításukat a szakdolgozatom elkészítése során. Nélkülük nem tudtam volna teljesíteni ezt a feladatot.

Köszönöm még egyszer mindenkinél, akik segítettek nekem a szakdolgozatom elkészítésében.

Irodalomjegyzék

- [1] Ivan E. Sutherland. „Sketchpad: A Man-Machine Graphical Communication System”. New York, NY, USA: Association for Computing Machinery, 1963. ISBN: 9781450378802. URL: <https://doi.org/10.1145/1461551.1461591> (elérés dátuma 2023. 04. 11.).
- [2] E. F. Arias, B. Guinot és T. J. Quinn. „Rotation of the Earth and Time scales”. (2003). URL: https://www.uclick.org/~sla/leapsecs/torino/arias_3.pdf (elérés dátuma 2023. 04. 11.).
- [3] ITU-R. *TF.460 : Standard-frequency and time-signal emissions*. Techn. jel. International Telecommunication Union, 2002-02-12. URL: <https://www.itu.int/rec/R-REC-TF.460-6-200202-I/en> (elérés dátuma 2023. 04. 11.).
- [4] IPSES. *Standard of time definition: UTC, GPS, LORAN and TAI*. Techn. jel. IPSES Srl. URL: <https://www.ipses.com/eng/in-depth-analysis/standard-of-time-definition/> (elérés dátuma 2023. 04. 11.).
- [5] „IEEE Draft Standard for Information Technology-Portable Operating System Interface (POSIX®)”. *IEEE Std P1003.1/D3* (2007). URL: <https://www.open-std.org/jtc1/sc22/open/n4217.pdf> (elérés dátuma 2023. 04. 11.).
- [6] P. Kenneth. Seidelmann, Great Britain. és United States Naval Observatory. *Explanatory supplement to the astronomical almanac / prepared by the Nautical Almanac Office, U.S. Naval Observatory ; with contributions from H.M. Nautical Almanac Office, Royal Greenwich Observatory ... [et al.] ; edited by P. Kenneth Seidelmann*. English. [Rev. ed.]. University Science Books Mill Valley, Calif, 1992, xxviii., 752 p. : ISBN: 0935702687. URL: <https://nla.gov.au/nla.cat-vn3719611> (elérés dátuma 2023. 04. 11.).
- [7] Péter Dr. Bauer. „Koordináta rendszerek és transzformációk”. (2015). URL: http://www.kjitz.bme.hu/images/Tantargyak/Valaszthato_targyak/

- Legi_kozlekedes/Koordinata_rendszerek_es_transzformaciook.pdf (elérés dátuma 2023. 04. 11.).
- [8] Editors of Encyclopaedia Britannica. „Latitude and Longitude”. *Encyclopædia Britannica* (2023). URL: <https://www.britannica.com/science/latitude> (elérés dátuma 2023. 04. 11.).
 - [9] Tshrinivasan. *Ecef coordinates*. 2006. URL: https://en.wikipedia.org/wiki/Earth-centered,_Earth-fixed_coordinate_system#/media/File:Ecef_coordinates.svg (elérés dátuma 2023. 04. 11.).
 - [10] National Geospatial-Intelligence Agency (NGA). *Department of Defense World Geodetic System 1984: Its Definition and Relationships with Local Geodetic Systems, Version 1.0.0*. Edition Date: 8 July 2014. National Center for Geospatial Intelligence Standards (NCGIS), 2014.
 - [11] Psoft Informatikai Fejlesztő és Szolgáltató Kft. *Eov - WGS'84 (GPS) koordináta átszámítás*. URL: <http://www.psoft.hu/szolgaltatasok/eov-wgs84-gps-koordinata-atszamitas.html> (elérés dátuma 2023. 04. 11.).
 - [12] F.D. Luna. *Introduction to 3D Game Programming with DirectX 11*. Mercury Learning Series. Mercury Learning és Information, 2012. ISBN: 9781936420223. URL: <https://books.google.hu/books?id=SqKLpwAACAAJ> (elérés dátuma 2023. 04. 11.).
 - [13] Károly Nyisztor. *Shaderprogramozás - Grafika és játékfejlesztés DirectX-szel*. SZAK Kiadó Kft., 2009. ISBN: 978-963-9863-09-5.
 - [14] Gouraud Henri. „Continuous Shading of Curved Surfaces.”” *IEEE Transactions on Computers* C-20 (1971), 623. old. URL: <https://doi.org/10.1109/T-C.1971.223313> (elérés dátuma 2023. 04. 11.).
 - [15] Grit Thürrner és Charles A. Wüthrich. „Computing Vertex Normals from Polygonal Facets”. *Journal of Graphics Tools* 3.1 (1998), 43–46. old. URL: <https://doi.org/10.1080/10867651.1998.10487487> (elérés dátuma 2023. 04. 11.).
 - [16] Nelson Max. „Weights for Computing Vertex Normals from Facet Normals”. *Journal of Graphics Tools* 4.2 (1999), 1–6. old. URL: <https://doi.org/10.1080/10867651.1999.10487501> (elérés dátuma 2023. 04. 11.).

- [17] Mac Saunders Lane és Garrett Birkhoff. *Algebra*. American Mathematical Society, 1999. ISBN: 978-0-8218-1646-2.
- [18] Manuel Blanco-Muriel és tsai. „Computing the solar vector”. *Solar Energy* 70.5 (2001), 431–441. old. ISSN: 0038-092X. URL: [https://doi.org/10.1016/S0038-092X\(00\)00156-0](https://doi.org/10.1016/S0038-092X(00)00156-0) (elérés dátuma 2023. 04. 11.).
- [19] Wikipedia contributors. *Azimuth-Altitude schematic.svg*. 2023. URL: https://en.wikipedia.org/wiki/Horizontal_coordinate_system#/media/File:Azimuth-Altitude_schematic.svg (elérés dátuma 2023. 04. 11.).
- [20] M. Szilvási-Nagy és Gyula Mátyási. „Analysis of STL files”. *Mathematical and Computer Modelling* 38 (2003. okt.). DOI: 10.1016/S0895-7177(03)90079-3.
- [21] Stevewhims. *Build desktop windows apps using the win32 API - win32 apps*. URL: <https://learn.microsoft.com/en-us/windows/win32/> (elérés dátuma 2023. 04. 11.).
- [22] Microsoft. *Microsoft/DirectXTK: The DirectX Tool Kit (aka DirectXTK) is a collection of helper classes for writing directx 11.x code in C++*. URL: <https://github.com/microsoft/DirectXTK> (elérés dátuma 2023. 04. 11.).
- [23] matt77hias. *Matt77hias/Rastertek: Directx 11 tutorials*. URL: <https://github.com/matt77hias/RasterTek> (elérés dátuma 2023. 04. 11.).
- [24] *DirectX 11 Tutorials*. URL: <https://www.rastertek.com/tutdx11.html> (elérés dátuma 2023. 04. 11.).
- [25] Pindrought. *Pindrought/DirectX-11-ENGINE-VS2017: Directx 11 engine repo for my tutorial series on YouTube. uses visual studio 2017*. URL: <https://github.com/Pindrought/DirectX-11-Engine-VS2017> (elérés dátuma 2023. 04. 11.).
- [26] Ocornut. *OCORNUT/ImGui: DEAR ImGui: Bloat-free graphical user interface for C++ with minimal dependencies*. URL: <https://github.com/ocornut/imgui> (elérés dátuma 2023. 04. 11.).
- [27] *PSA Algorithm Files*. URL: <http://www.psa.es/sdg/sunpos.htm> (elérés dátuma 2023. 04. 11.).
- [28] Nlohmann. *Nlohmann/JSON: JSON for Modern C++*. URL: <https://github.com/nlohmann/json> (elérés dátuma 2023. 04. 11.).

- [29] Google. *Google/googletest: GoogleTest - Google Testing and mocking framework*. URL: <https://github.com/google/googletest> (elérés dátuma 2023. 04. 11.).
- [30] Erich Gamma és tsai. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. kiad. Addison-Wesley Professional, 1994. ISBN: 0201633612. URL: http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1 (elérés dátuma 2023. 04. 11.).
- [31] QuinnRadich. *The winmain application entry point - win32 apps*. URL: <https://learn.microsoft.com/en-us/windows/win32/learnwin32/winmain--the-application-entry-point> (elérés dátuma 2023. 04. 11.).
- [32] Blender Foundation. *Home of the blender project - free and open 3D creation software*. URL: <https://www.blender.org/> (elérés dátuma 2023. 04. 11.).
- [33] Meshlab. URL: <https://www.meshlab.net/> (elérés dátuma 2023. 04. 11.).
- [34] URL: <https://gml.noaa.gov/grad/solcalc/azel.html> (elérés dátuma 2023. 04. 11.).

Ábrák jegyzéke

2.1.	Az ECEF koordináták a földrajzi szélességhez és hosszúsághoz viszonyítva	12
2.2.	Jobbsodrású és balsodrású koordináta-rendszer	15
2.3.	$V_i Q V_{i+1}$ csúcsok által meghatározott sík normálvektora	18
2.4.	Azimut és napmagasság	22
3.1.	A program megjelenítési ablaka	30
3.2.	Általános beállítási ablak	32
3.3.	Felületháló paramétereinek beállítása	34
3.4.	Trajektória paramétereinek beállítása	34
3.5.	Barangolás mód beállítása	35
3.6.	Körséta mód beállítása	36
4.1.	Felhasználói eset diagram	45
4.2.	A vezérlők hierarchikus szerkezete	56
4.3.	A nézet réteg hierarchikus szerkezete	62
4.4.	<code>GuiView</code> és a <code>IController</code> kapcsolata	66
4.5.	A modell állapotának továbbítása a vezérlő és nézet felé	73
4.6.	Stanford Bunny 70k megjelenítése	89
4.7.	Budapesti Magyar Szentek templomáról készített felületháló megjelenítése	89

Táblázatok jegyzéke

3.1.	Kamera irányítása barangolás módban	37
3.2.	Lejátszás vezérlése Körséta módban	37
3.3.	STL fájl felépítése	38
4.1.	A vezérlő réteg üzenetei, enum IControllerMessageIDs	61
4.2.	A nézet réteg üzenetei, enum IViewMessageIDs	64
4.3.	A modell réteg üzenetei, enum IMessageIDs	71

Algoritmusjegyzék

Forráskódjegyzék

3.1.	Konfigurációs mintafájl	39
4.1.	wWinMain függvény deklarációja	51
4.2.	App osztály tagfüggvényeinek deklarációja	52
4.3.	App::Run(), az alkalmazás ciklusa	53
4.4.	RenderWindow tagfüggvényeinek deklarációja	54
4.5.	RenderWindow::ProcessMessages tagfüggvény	55
4.6.	WndProc függvények deklarációja	55
4.7.	IController interfész	58
4.8.	IView interfész	63
4.9.	IModel interfész	67
4.10.	TerrainModel adattagjai	71
4.11.	IDataAccess interfész	73
4.12.	IRenderable interfész	76
4.13.	CameraPositioner függvényei	77