Eötvös Loránd Tudományegyetem
Informatikai Kar
Algoritmusok és Alkalmazásaik Tanszék

Tömbszelvény-rajzoló alkalmazás

SZAKDOLGOZAT

Konzulens:
Baráth Dániel Béla
doktorandusz

Készítette:
Agárdi Norbert Zsolt
programtervező informatikus BSc

Budapest, 2018

Tartalomjegyzék

1. Bevezetés

Első szakként a térképész képzést végeztem el az ELTE-n. A témaválasztásomat főként ez befolyásolta, olyan témát kerestem, amely kötődik az első szakmámhoz. Domborzati modellek matematikai leírása és vizuális megjelenítése érdekelt elsősorban, ezért választásom a számítógépes grafika területére esett.

A célom egy olyan alkalmazás elkészítése volt, amely lehetőséget biztosít a felhasználónak 3D-s felület szerkesztésére, majd annak tereptárgyakkal (pl.: fa, ház) és textúrákkal való kiegészítésére olyan módon, hogy a felhasználó képes legyen egy ábraszintű modell elkészítésére.

A feladat elvégzését Unity környezetben találtam célszerűnek, C# nyelv használatával. A Unity szoftverrel való alkalmazáskészítés napjainkban is népszerű (elsősorban játékokat írnak benne), jól dokumentált, számtalan segédanyag állt rendelkezésemre.

A program elkészítésének fő részei a következők voltak: rajzolófelületek kialakítása, paraméterezése és mozgatása, térgörbék szerkesztése és megjelenítése, görbékre interpolációs felület megadása és annak szerkesztése térgörbék és csomópontok beszúrásával, és a kapott felületre textúrák betöltése, fák és épületek felhelyezésének megoldása. Elkészült továbbá egy random felületgeneráló menüpont, mentési lehetőség, továbbá a program lehetőséget biztosít a felhasználónak, hogy a program fontosabb funkcióinak paramétereit beállíthassa.

Az elkészült program hasznos lehet mindazoknak, akik felületet, domborzati szelvényt, egyszerű terepmodellt szeretnének készíteni. Az elkészült alkalmazás emellett szemléltető eszköze lehet felszínek síkbeli leképezésének bemutatásra is.

2. Felhasználói dokumentáció

Ebben a fejezetben összefoglalom az elkészült szoftver működését, ismertetem a futási környezetet, a használathoz szükséges információkat, a hibaüzeneteket és a teendőket.

2.1. Az elkészült szoftver működésének rövid leírása

A szoftver felülete 2 fő részből áll: egy szerkesztői (szintén 2 részre van osztva: XY sík és az elemek Z metszetét megadó panelje) és egy 3D-s megjelenítő panelből (2–1. ábra). A felhasználó az egyes panelek kameráinak pozícióját, látószögét szabadon változtathatja a program használata közben.

A felhasználónak elsőként létre kell hoznia egy felület (ezt vagy random generálja, vagy 4 határoló polyline-t rajzol, amelyekből a program elvégzi az interpolációt). Amint elkészítette a felhasználó a kiinduló felületet, onnantól a felhasználó elérheti a program összes funkcióját, pl.:

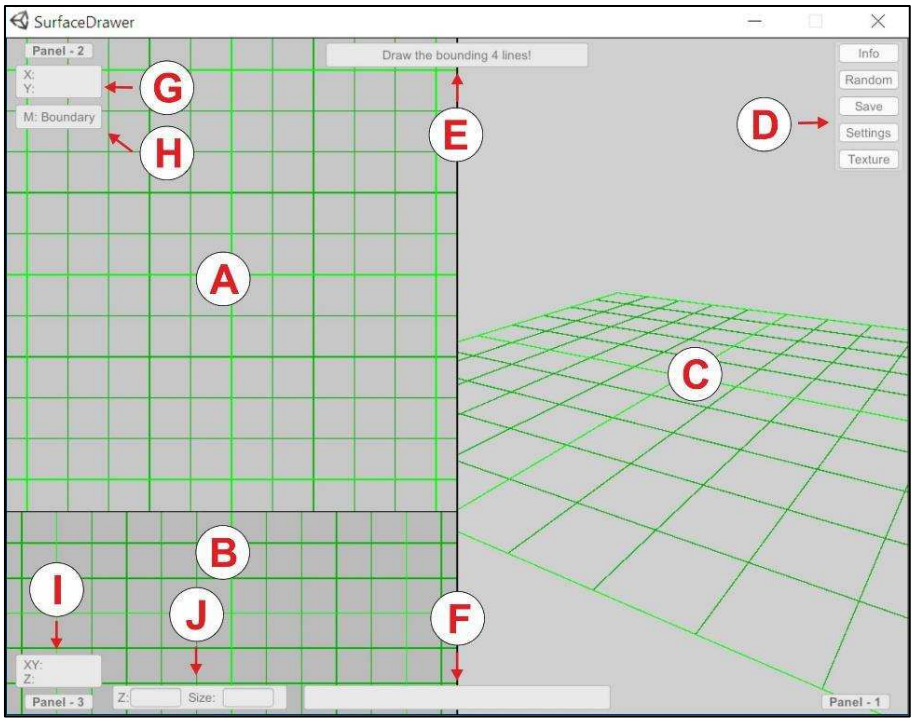
- pontokat és térbeli polyline-okat szúrhat be, amely módosítja a felületet;
- tereptárgyakat tehet fel a felületre;
- textúrát tölthet be a felületre;
- exportálhatja a felületet;
- megnézheti a felület szintvonalas rajzát és grid hálóját.

A program nagyrészt gyorsbillentyűkkel és az egér használatával működik. A felhasználó bármikor újra kezdheti a munkát, valamint egyes részfeladatok törlésére is van mód.

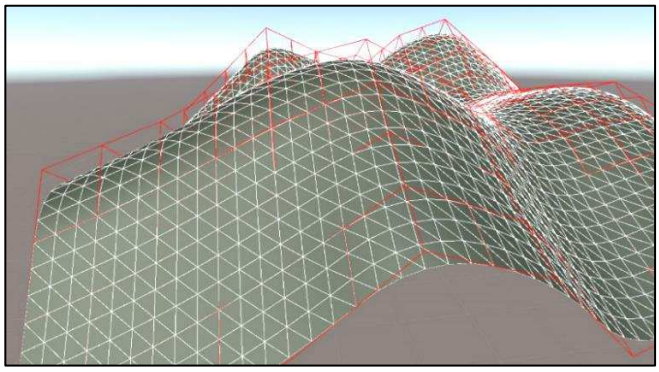
2.2. Felhasznált módszerek röviden

A felületet határoló polyline-ok megadása mindig két lépcsőben történik: először az XY síkbeli vetületét, majd a Z koordinátabéli viszonyait kell megrajzolni, amelyekből az alkalmazás elkészíti a végső polyline-t. Görbe megadása a Catmull–Rom-szplájn illesztésén alapul, azaz a felhasználó által megadott csomópontokra a program elvégezi az interpolációt.

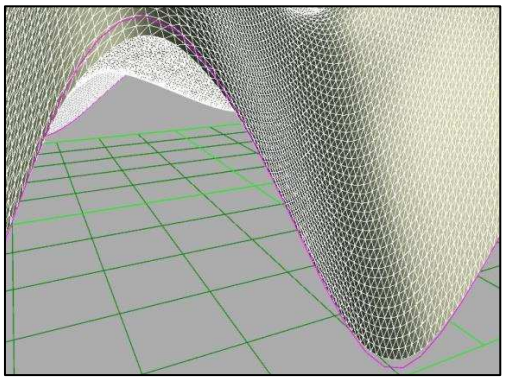
A felületek elkészítésekor a szoftver először a megadott felbontás alapján mintavételezi a határoló térbeli polyline-okat, ezekből interpolál egy Coons-foltot, amelynek ponthálóját a megadott felbontás szerint eltárolja. A program a folytonosság elérése érdekében a háló csomópontjait sűríti, és az így kialakult kontrollpoligonok segítségével szplájn felületeteket illeszt (2–2. ábra). A kiszámolt felület Mesh-ként tárolódik. A létrehozott kontrollpoligonok érintői a kapott felületnek, így előfordul, hogy a kapott felület egyes helyeken nem pontosan illeszkedik a határoló polyline-okra, viszont mindenhol folytonos (2–3. ábra).



2–1. ábra: A szoftver kezelőfelülete
A – XY sík panelje (Panel - 2), **B** – Elemek Z metszetének panelje (Panel - 3); **C** – 3D-s megjelenítő panel (Panel - 1); **D** – Menü; **E** – Segítő, tájékoztató üzenetek; **F** – Hibaüzenetek; **G** – X, Y koordinátákat megjelenítő panel; **H** – Szerkesztési állapotot jelző panel (Boundrary - alapfelület készítése, Inners - belső elemek hozzáadása); **I** – Z koordinátát megjelenítő panel; **J** – Elemek magasságát, kiterjedését megadó panel



2–2. ábra: Kontroll-poligonhálózat és az általuk meghatározott szplájn felület



2–3. ábra: A felület mindenhol folytonos, viszont kis mértékben eltérhet a határoló polyline-októl (a felület felosztása minél nagyobb, a hiba annál kisebb lesz)

A felület szerkesztése során beszúrt térbeli pontok és polyline-ok először a felület kontroll-poligonhálóját módosítják, majd minden esetben újra történik a felület kiszámítása.

A szintvonalas nézet elkészítéséhez a Mesh-ként eltárolt felület háromszögeit használom, síkokkal történő metszetét számolom. A grid nézet előállításakor megadott osztásközökkel függőleges sugarakat indítok, és a felülettel vett metszetét tárolom.

Tereptárgyak felhelyezése a felületre előre elkészített modellek feltevésével történik.

2.3. Program használatához szükséges információk

Használatához szükséges:

- surface.exe fájl futtatásához Windows 7, 8, vagy 10 operációs rendszer szükséges;
- Grafikus kártya hiánya lassíthatja a működést. A program AMD Radeon HD 7800 Series videokártyával rendelkező számítógépen készült;
- a tesztelést Intel(R) Core(TM) i5-6600, 3.30 Ghz processzorú, 16,0 GB memóriájú 64 bites operációs rendszerű számítógépen végeztem, a program gyorsasága nem volt kifogásolható.

Az alkalmazás üzembe helyezése:

- SurfaceDrawe\surface.exe indítása (Windows)

  Szükséges, hogy az .exe állomány mellett megtalálható legyen az UnityPlayer.dll és a surface_Data könyvtár, az állományaival együtt.

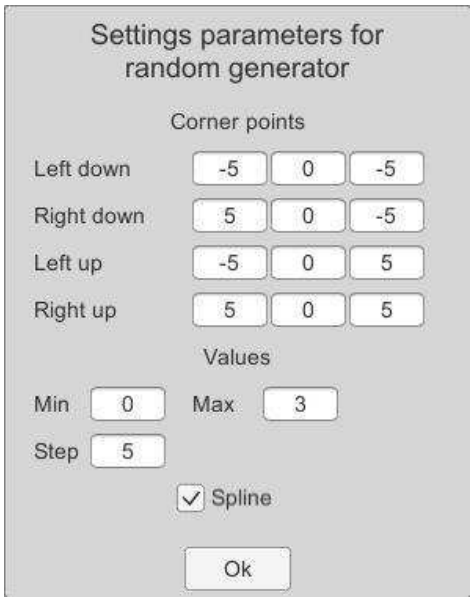
  Ajánlott beállítás: Screen Resolution: minimum 1024x768 (kisebb felbontás esetén a GUI elemek összecsúszhatnak).
- Más operációs rendszeren való futtatáshoz szükséges a Unity3D program (https://unity3d.com/) letöltése és a program újbóli fordítása (build) a megfelelő operációs rendszerre. A fejlesztés Unity 2017-ben történt, korábbi verziók használata gondot jelenthet.

Általános kezelési tájékoztató:

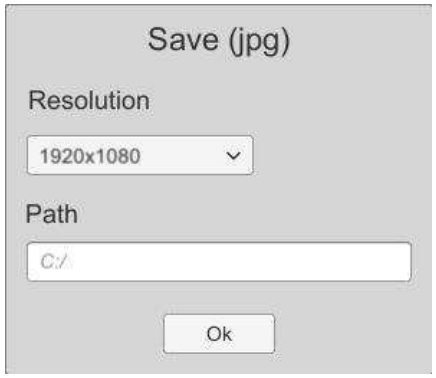
A program indítása után a kezelőfelület (az üres 3D-s panel és a 2D-s rajzolási panelek) jelenik meg (2–1. ábra). Rögtön lehetőség van a rajzolásra, vagy a jobb oldali menü elemei közül választhat a felhasználó:

- Info – a program működéséhez szükséges billentyűzetkiosztás és a szerző adatai olvashatók.

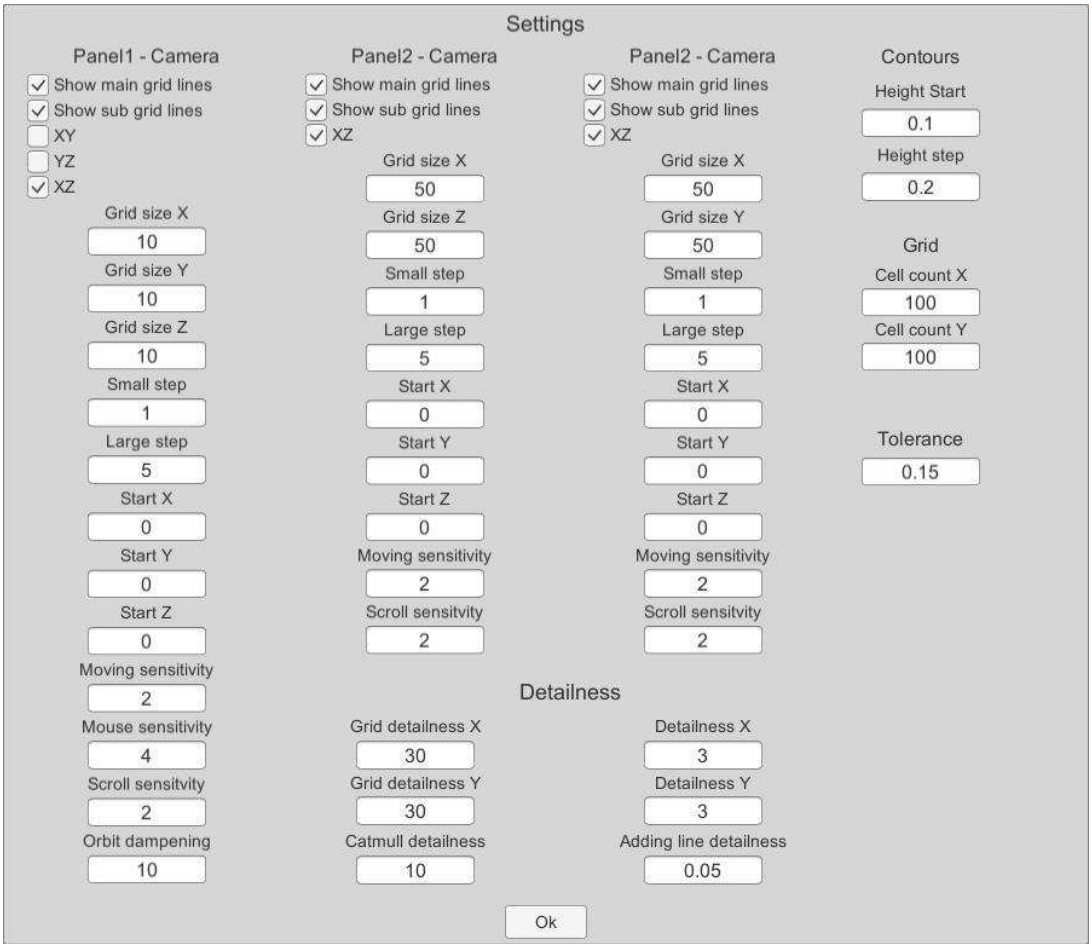
- Random – a megadott paraméterek alapján random felület generálására van mód (2–4. ábra).
- Save – a 3D-s modell .jpg állományba mentését kínálja fel a program (2–5. ábra).
- Settings – a program során előforduló paramétereket lehet beállítani, pl.: a felület finomságát, a koordinátarendszerek beosztását és a kamerák mozgatásainak értékeit (2–6. ábra).
- Texture – a felület textúráját lehet kiválasztani (2–7. ábra).



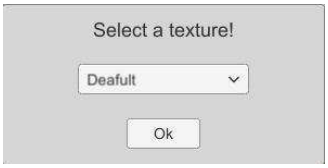
2–4. ábra: A Random panel (beállíthatók a felület sarokkoordinátái, a felület maximum, illetve minimum magassága, az osztópontok száma és az, hogy a határoló polyline-ra illesszen-e a program Catmull-Rom spzplájnt-e, vagy sem)



2–5. ábra: A Save panel (3 felbontás közül lehet választani: 1920x1080, 3840x2160 és 7680x4320, az elérési útvonalban a fájl nevét nem kell megadni, az automatikusan generálódik)



2–6. ábra: A Settings panel (beállítható a panelek koordinátarendszere, a kamerák mozgatásának érzékenysége, a polyline-ok és a felület felbontása, a szintvonalas nézet szintvonalainak osztásköze, a grid nézet beosztása és a snap toleranciája)



2–7. ábra: A Texture panel (a felület textúrája választható ki: alap, füves, homokos, kavicsos textúra közül lehet választani)

A felhasználónak először létre kell hoznia egy kiindulási felületet, amelyet vagy a Random gomb megnyomásával generál, vagy megrajzolja a 4 határoló térgörbéjét, ami alapján a program elkészíti az interpolációs felületet az ***m*** *billentyű* megnyomásával. A 4 határoló görbének a következő tulajdonságokkal kell rendelkeznie:

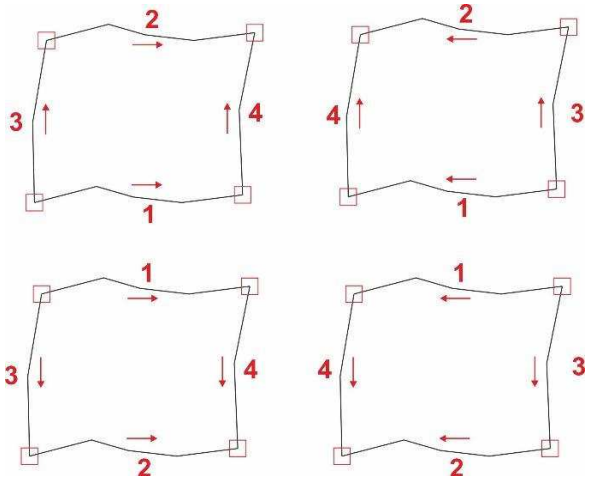
– nem metszhetik egymást;
– végpontjaikban érintkezniük kell (ezt segíti a snap funkció, egy megadott értéken belül – amely állítható a Settings menüpont alatt – a végpontot hozzáilleszti egy másik vonal végpontjához);

– megadott sorrendben és irányban kell megrajzolni őket (2–8. ábra);
– a polyline-oknak „monoton”-nak kell lennie, azaz nem fordulhatnak vissza (2–9. ábra).

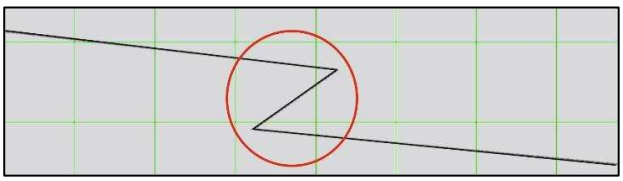
A program a felület interpolációja előtt ellenőrzi ezeket a feltételeket, és amennyiben nem teljesülnek, hibaüzenet ad, és újból kell kezdeni a rajzolást.

Egy térbeli polyline rajzolását a ***p*** *billentyű* lenyomásával lehet elkezdeni, a menete a következő módon történik (ekkor a program többi funkciója inaktívvá válik):

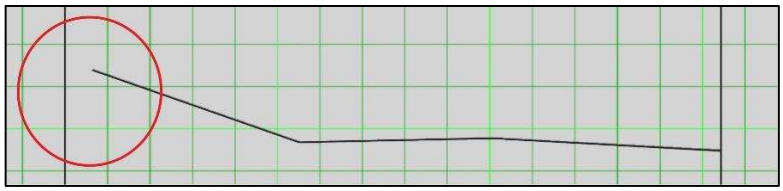
– először az XY síkon kell az egér bal gombjával felvenni a töréspontokat;
– jobb egérgomb lenyomása után befejeződik a polyline XY metszetének szerkesztése (eddig a pillanatig volt még lehetőség a ***p*** billentyű lenyomásával kilépni a szerkesztésből) és aktívvá válik a 3-as panel (automatikusan a vonal hosszához igazodik, és megjelenik a vonal kezdő és végpontja, amelyet egy-egy fekete vonal jelez);
– az egér bal gombjával lehet a vonal Z-béli metszetét megrajzolni;
– az egér jobb gombjának lenyomásával fejeződik be a vonal szerkesztése, a térbeli polyline megjelenik a 3D-s panelen;
– amennyiben a Z-béli metszeten a polyline nem ér el a vonal a kezdőpontjától a végpontjáig (két fekete vonal jelzi), vagy a polyline Z-béli metszete nem „monoton”, akkor a vonalat nem fogadja el a program (2–10. ábra);
– A felhasználó a ***c*** *billentyű* lenyomásával tudja kapcsolni, hogy a rajzolni kívánt polyline csak a felvett pontokból álljon, vagy a program illesszen rá Catmull–Rom-szplájnt.



2–8. ábra: A határoló polyline-ok megrajzolásának lehetséges sorrendje és irányai



2–9. ábra: Hibás, visszaforduló polyline (nem „monoton”)



2–10. ábra: A polyline nem adja meg a teljes polyline Z-béli metszetét

Amennyiben a felhasználó elkészítette a kiinduló felületet, lehető válik a program összes funkciójának használata. A következők közül lehet választani:

- az alapfelülethez további pontokat, polyline-okat lehet hozzáadni, amelyek változtatják a felületet:
  - az ***a*** *billentyű* lenyomásával lehet pontot hozzáadni, a menete a következő:
    - meg kell adni a pont magasságát (2–1. ábra - J) és kiterjedését (mekkora sugarú körben változtassa meg a felületet);
    - bal egérgombbal a felületen belül meg kell adni a pont helyét az XY síkon;
    - amennyiben nincs megadva a pont magassága és kiterjedése, vagy a pont nem a felületen belül helyezkedik el, akkor a pont érvénytelen.
  - a ***p*** *billentyű* lenyomásával belső polyline-t lehet hozzáadni:
    - hasonlóképpen működik, mint a határoló polyline-oknál, csak nem feltétel, hogy a polyline XY síkban „monoton” legyen;
    - a bal felső sarokban (2–1. ábra - H) már Inners felirat szerepel, amely jelzi, hogy már nem az alapfelület határoló polyline-ok szerkesztésén dolgozunk (Boundary).
- a felhasználó tereptárgyakat vehet fel, ehhez először az ***e*** *billentyűt* kell megnyomni:
  - a program „Adding elements mode”-ba kapcsol, a program többi funkciója addig nem érhető el (a menü elemek sem), amíg a felhasználó újból meg nem nyomja az ***e*** *billentyűt*;
  - a ***t*** *billentyű* megnyomásával fát lehet lerakni az XY síkra kattintva, a fa magassága megadható (2–1. ábra - J), ha a fa XY síkbeli koordinátája nem a felületen belül van, akkor a hozzáadás nem történik meg;
  - a ***h*** *billentyű* megnyomásával házat lehet feltenni a fa lerakásához hasonló módon.
- a ***v*** *billentyű* lenyomásával változtathat a felhasználó a felület megjelenési módján:

- a program ekkor „View mode”-ba kapcsol, és a program többi funkciója nem érhető el;
- a ***v*** *billentyű* először lenyomása után a felület szintvonalas képe jelenik meg;
- a ***v*** *billentyű* másodszori lenyomása után a felület grid hálózata jelenik meg;
- a ***v*** *billentyű* harmadszori lenyomása után visszaáll a felület eredeti képe, a program funkciói újból elérhetővé válnak.

A program használata során lehetőség van törlésre is:

- a ***d*** *billentyűvel* mindent ki lehet törölni;
- a ***w*** *billentyűvel* a hozzáadott tereptárgyakat lehet törölni;
- az ***x*** *billentyűvel* az alapfelületet lehet visszaállítani.

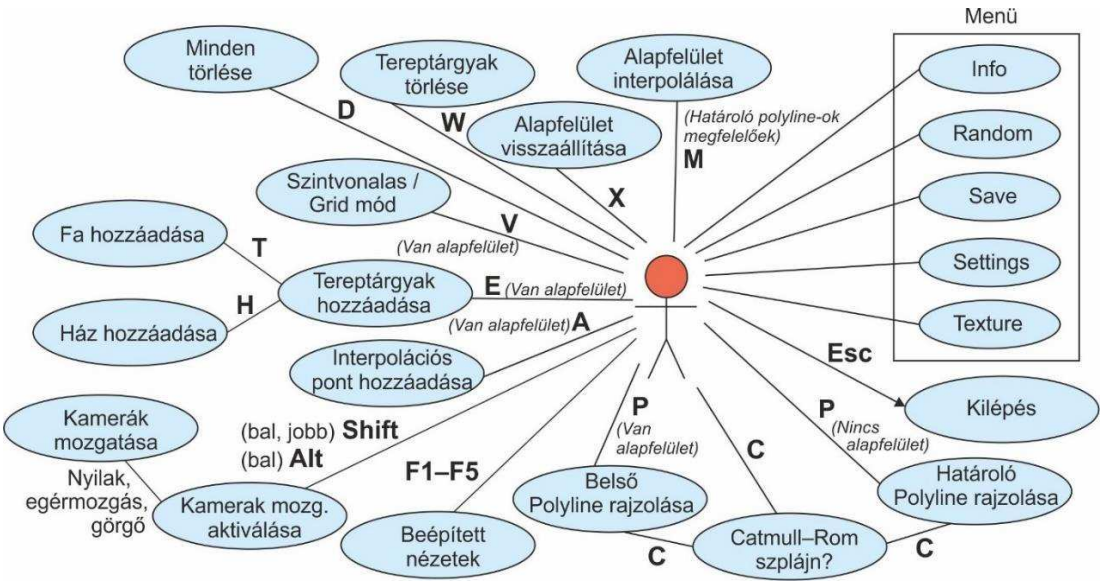
Lehetőség van a három panel nézetének állítására is:

- a ***bal shift*** a 2-es panel állítását teszi aktívvá;
- a ***jobb shift*** az 1-es panel állítását teszi aktívvá;
- a ***bal alt*** a 3-as panel állítását teszi aktívvá;
- a panelek állítása a nyilakkal, az egér mozgatásával és a görgő használatával működik;
- az ***F1***, ***F2***, ***F3***, ***F4***, ***F5*** *billentyű* lenyomásával beépített nézetek választhatók ki az 1-es panel számára.

A programból való kilépésére az ***Esc*** *billentyű* szolgál. A billentyűk kiosztását az 2–1. táblázatban, a program működését a felhasználói esetek diagramján (2–11. ábra) foglalom össze.

A	Interpolációs pont hozzáadása a felülethez
C	Váltás sokszögvonal, illetve Catmull–Rom-szplájn rajzolása között
D	Minden törlése
E	Kapcsolás tereptárgyak hozzáadása módba
H	Ház hozzáadása
M	Határoló polyline-okból interpolációs felület készítése
P	Polyline rajzolása
T	Fa hozzáadása
V	Felület megjelenési módjának kapcsolása
W	Tereptárgyak törlése
X	Alapfelület visszaállítása kiinduló állapotba
F1–F5	Beépített nézetek közötti váltás
Shift, Alt	Kamerák mozgatásának aktiválása
Esc	Kilépés

2–1. táblázat: Billentyűkiosztás



2–11. ábra: Felhasználói esetek diagramja

A program üzenetei

A. Segítő, tájékoztató üzenetek (2–1.ábra - E):

– Add tree (Panel 2)!, Add house (Panel 2)!

T és **H** billentyű lenyomásával fát és házat tudunk elhelyezni a felületre.

– Adding elements mode (T-tree, H-house)

***E** billentyű* megnyomásával lehetőségünk van tereptárgyak felhelyezésére.

– Add point (first write the height value)!

***A** billentyű* megnyomásakor jelenik meg, ha van már alapfelületünk.

– Draw polyline!

***P** billentyű* megnyomására jelenik meg.

– Draw Catmull spline!

***C** billentyűt* megnyomására Catmull–Rom szplájn illesztődik a felvett pontokra polyline rajzolásakor.

– Draw the bounding 4 lines!

Program indulásakor megjelenő üzenet.

– Finish interactions!

Menügombot nyomtunk meg, mialatt más funkciót használtunk.

– Select mode!

Általánosan megjelenő üzenet, éppen semmilyen interakció nincs folyamatban.

– View mode (contour lines), View mode (grid lines)

***V** billentyű* megnyomása után nézet módban vagyunk.

B. Lehetséges hibaüzenetek (2–1.ábra - F):

- Crossing lines!

  Olyan vonalat rajzoltunk, amely keresztez egy másikat.
- First create a surface!

  Olyan feladatot szerettünk volna végrehajtani, amihez szükséges az alapfelület elkészítése.
- Invalid boundary lines (must be monotonics)!

  A határoló polyline-ok nem „monoton”-ok.
- Invalid boundary lines (must be well connected)!

  A határoló polyline-ok nem csatlakoznak.
- Invalid line!

  Nem a feltételeknek megfelelő polyline-t szerkesztettünk.
- Invalid line (Must be monotonic)!

  Nem „monoton” vonalat rajzoltunk.
- Invalid size!

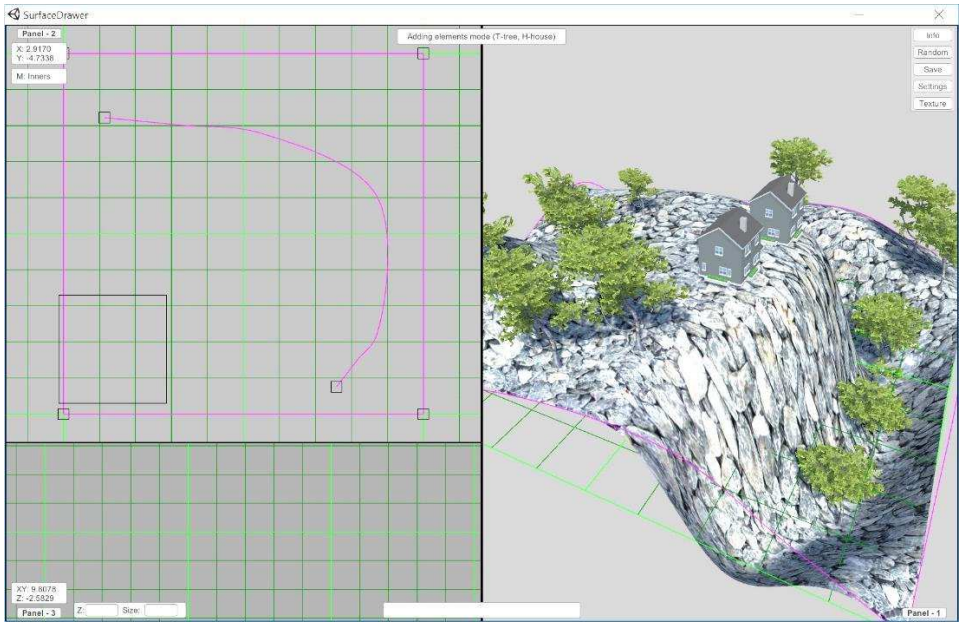
  Nem írtunk be helyes értéket a mérethez.
- Not enough or too many boundary lines!

  Nincs elég határoló polyline.
- Point is not inside the surface!

  A felületen kívüli pontot adtunk meg.
- Surface is alredy valid!

  Alapfelületet csak akkor lehet generálni, ha addig nem készült el.
- Too much boundary lines!

  Már megvan a megfelelő számú határoló polyline.



2–12. ábra: Egy jelenet a programból

3. Fejlesztői dokumentáció

Ez a fejezet tartalmazza a program részletes specifikációját, a feladat megoldásához felhasznált módszerek leírását, a program logikai és fizikai szerkezetének megvalósítását az osztályok és függvények bemutatásával, valamint a tesztelési tervet, fekete- és fehérdoboz-tesztesetek vizsgálatát.

3.1. Részletes specifikáció

A feladat absztrakt, formális specifikálása következik.

3.1.1. A specifikációhoz és a módszerek részletezéséhez használt típusok

- Vektor: $V2 = rec(x{:}\,\mathbb{R}, y{:}\,\mathbb{R})$
  2D-s tér pontja, vektora
- Vektor: $V3 = rec(x{:}\,\mathbb{R}, y{:}\,\mathbb{R}, z{:}\,\mathbb{R})$
  3D-s tér pontja, vektora
- Line: $L = rec(p1{:}\,V3, p2{:}\,V3)$
  3D-s vonalszakasz a kezdőpontjával és a végpontjával
- Polyline: $P = queu(p_i{:}\,V3), i{:}\,\mathbb{N}$
  3D-s polyline a pontjainak felsorolásával
- Triangle: $T = rec(p_1{:}\,V3, p_2{:}\,V3, p_3{:}\,V3)$
  3D-s háromszög a csúcsainak felsorolásával
- B-szplájn felület: $Spline \in V3^{n\times m},\ n, m{:}\,\mathbb{N}$
  A felület pontjai egy 2D-os mátrixban tárolva
- Végső felület: $S \in Spline^{n\times m}, n, m{:}\,\mathbb{N}$
  A program által megjelenített felület B-szplájn felületek hálózata
- Plane: $Plane(h) = rec(x{:}\,\mathbb{R}, h, z{:}\,\mathbb{R}),\ h{:}\,\mathbb{R}$
  XZ sík, amelynek magassága (Y) h;
- Ray: $R = rec(p1{:}\,V3, p2{:}\,V3)$,
  Kibocsátott sugár, amely p1 pontból indul és iránya a p2 vektor;
- Mesh háromszögekből: $Mt \in T^n, n{:}\,\mathbb{N}$
  Unity-ben modellek megjelenítésére szolgáló objektumtípus, most háromszögek halmazával ábrázolom

– Mesh szakaszokból: $Ml \in L^n, n: \mathbb{N}$

  Unity-ben modellek megjelenítésére szolgáló objektumtípus, most vonalszakaszok halmazával ábrázolom

– Kontroll-pontháló: $Grid \in (rec(pos: V2, height: \mathbb{R}))^{n \times m}, n, m \in \mathbb{N}$

  A felület kontroll-poligonhálójának síkbeli(XZ) koordinátáját és magasságát(Y) tárolja;

– Modell: $Mo = rec(p: V3, h: \mathbb{R})$

  Unity-ben megjeleníthető, előre elkészített modell, a modell alappontjával és magasságával jellemzem (az elkészített programban fa és ház között lehet választani)

3.1.2. A specifikációhoz és a módszerek részletezéséhez használt függvények

– Abs(a: $\mathbb{R}$): $\mathbb{R}$

  Az *a* valós szám abszolútértékét adja vissza

– Cross(a:P, b:P): bool

  Igazzal tér vissza, ha a két polyline metszi egymást

– Distance(a:V2, b:V2): $\mathbb{R}$

  Az *a* és *b* 2D-s vektorok közötti távolságot adja vissza

– Distance(a:V3, b:V3): $\mathbb{R}$

  Az *a* és *b* 3D-s vektorok közötti távolságot adja vissza

– Inside(a:V3, p:P): bool

  Igazzal tér vissza, ha az *a* pont síkkoordinátái a *p* polyline síkba vetített koordinátái által meghatározott poligonba esik

– Inter(a:V3, b:V3, c:V3): bool

  Igazzal tér vissza, ha c az a és b által meghatározott vonalszakaszra (*a*-t és *b*-t nem tartalmazza) esik

– Lerp(a:V3, b:V3, f: $\mathbb{R}$):V3, f∈[0,1]

  Lineáris interpolációt végez az f faktor figyelembevételével (az eredményt a következő képlet adja: $(1 - \mathrm{f}) * \mathrm{a} + \mathrm{f} * \mathrm{b}$

– LinePlaneIntersection(a:V3, b:V3, Plane(h)):V3

  Az *a* és *b* által meghatározott térbeli vonalszakasz és a *h* magasságú XZ sík metszetét adja vissza [5]

– Pow(a: $\mathbb{R}$, b: $\mathbb{R}$):

  Az *a* valós számot a *b*-edik hatványra emeli

- Mesh függvényei:
  - bound():rec(min:V3,max:V3)

    A Mesh befoglaló dobozának minimumát és maximumát adja vissza
  - count():ℕ

    A Mesh elemeinek számát adja vissza
  - rayCast(ray:R, hit&:V3):bool

    A ray sugár metszetét vizsgálja a felülettel, ha nem metszi hamissal, ha metszi, igazzal tér vissza, közben a hit vektort beállítja a metszéspontra
  - size():V3

    A Mesh méretét adja vissza
- Polyline függvényei:
  - size():ℕ

    A polyline csomópontjainak számát adja vissza
  - length():ℝ

    A polyline hosszát adja vissza
  - first():V3

    A polyline első pontját adja vissza
  - last():V3

    A polyline utolsó pontját adja vissza
- V3 függvénye:
  - up():V3

    Felfele mutató vektort adja vissza
- S (végső felület) függvényei:
  - calculateSurfaceCornerTiles(detailnessX: ℕ, detailnessY: ℕ, gridPoints: V3xV3)

    A 3–8. ábra szerinti sarok B-szplájn felületeket készíti el
  - calculateSurfaceEdgeTiles(detailnessX: ℕ, detailnessY: ℕ, gridPoints: V3xV3)

    A 3–8. ábra szerinti oldalsó B-szplájn felületeket készíti el
  - calculateSurfaceCenterTiles(detailnessX: ℕ, detailnessY: ℕ, gridPoints: V3xV3)

    A 3–8. ábra szerinti középső B-szplájn felületeket készíti el
- C# List függvényei (magyarázat nélkül):
  - Add(a:<T>)
  - Insert(index: ℕ, a:<T>)

3.1.3. Állapottér

$$\text{Állapottér:} A = \left(\begin{matrix}P\\b_1\end{matrix} \times \begin{matrix}P\\b_2\end{matrix} \times \begin{matrix}P\\b_3\end{matrix} \times \begin{matrix}P\\b_4\end{matrix} \times \begin{matrix}P\\i_1\end{matrix} \times .. \times \begin{matrix}P\\i_n\end{matrix} \times \begin{matrix}V3\\p_1\end{matrix} \times .. \times \begin{matrix}V3\\p_m\end{matrix} \times \begin{matrix}S\\s\end{matrix} \times \begin{matrix}Mt\\m_1\end{matrix} \times \begin{matrix}Ml\\m_2\end{matrix} \times \begin{matrix}Ml\\m_3\end{matrix} \times \begin{matrix}Mo\\mo_1\end{matrix} \times .. \times \begin{matrix}Mo\\mo_k\end{matrix}\right),$$

ahol b_1, b_2, b_3, b_4 a felületet határoló polyline-ok, $i_1, .., i_n$ a felületet módosító polyline-ok, $p_1, .., p_m$ a felületet módosító belső pontok, s a végső felület kontroll-poligonhálóját tároló mátrix, m_1, m_2, m_3 a felület megjelenítésére szolgáló Mesh-ek, $mo_1, .., mo_k$ a felületre elhelyezett modellek.

3.1.4. Specifikációs feltételek

Előfeltétel: a program indulásakor semmilyen objektum nem létezik.

Invariáns állítások:

– A polyline-oknak minimum 2 ponttal kell rendelkezniük:

$\forall i \in P: i.size() > 1$

– A négy határoló polyline-nak megfelelő sorrendben kell érintkeznie:

$b_1.first() = b_3.first() \wedge b_1.last() = b_4.first() \wedge$

$b_2.first() = b_3.last() \wedge b_2.last() = b_4.last()$

– A határoló polyline-oknak „monotonnak” kell lenniük, azaz nem fordulhatnak vissza:

$$\begin{aligned}
&(b_1.first().x < b_1.last().x \wedge\ b_1.p_i.x < b_1.p_{i+1}.x, i \in [1..b_1.size()-1] \wedge\\
&\ b_2.first().x < b_2.last().x \wedge b_2.p_i.x < b_2.p_{i+1}.x, i \in [1..b_2.size()-1] \wedge\\
&\ b_3.first().z < b_3.last().z \wedge b_3.p_i.z < b_3.p_{i+1}.z, i \in [1..b_3.size()-1] \wedge\\
&\ b_4.first().z < b_4.last().z \wedge b_4.p_i.z < b_4.p_{i+1}.z, i \in [1..b_4.size()-1]) \vee\\
&(b_1.first().x > b_1.last().x \wedge\ b_1.p_i.x > b_1.p_{i+1}.x, i \in [1..b_1.size()-1] \wedge\\
&\ b_2.first().x > b_2.last().x \wedge b_2.p_i.x > b_2.p_{i+1}.x, i \in [1..b_2.size()-1] \wedge\\
&\ b_3.first().z > b_3.last().z \wedge b_3.p_i.z > b_3.p_{i+1}.z, i \in [1..b_3.size()-1] \wedge\\
&\ b_4.first().z > b_4.last().z \wedge b_4.p_i.z > b_4.p_{i+1}.z, i \in [1..b_4.size()-1]) \vee\\
&(b_1.first().z < b_1.last().z \wedge\ b_1.p_i.z < b_1.p_{i+1}.z, i \in [1..b_1.size()-1] \wedge\\
&\ b_2.first().z < b_2.last().z \wedge b_2.p_i.z < b_2.p_{i+1}.z, i \in [1..b_2.size()-1] \wedge\\
&\ b_3.first().x < b_3.last().y \wedge b_3.p_i.x < b_3.p_{i+1}.x, i \in [1..b_3.size()-1] \wedge\\
&\ b_4.first().x < b_4.last().y \wedge b_4.p_i.x < b_4.p_{i+1}.x, i \in [1..b_4.size()-1]) \vee\\
&(b_1.first().z > b_1.last().z \wedge\ b_1.p_i.z > b_1.p_{i+1}.z, i \in [1..b_1.size()-1] \wedge\\
&\ b_2.first().z > b_2.last().z \wedge b_2.p_i.z > b_2.p_{i+1}.z, i \in [1..b_2.size()-1] \wedge\\
&\ b_3.first().x > b_3.last().x \wedge b_3.p_i.x > b_3.p_{i+1}.x, i \in [1..b_3.size()-1] \wedge\\
&\ b_4.first().x > b_4.last().x \wedge b_4.p_i.x > b_4.p_{i+1}.x, i \in [1..b_4.size()-1])
\end{aligned}$$

– A polyline-ok Y síkba vetítése során nem keletkezhet egymást fedő vonalszakasz:
  $\forall l \in P: \forall v \in l: \neg Inter(l.p_i, l.p_{i+1}, v), i \in [1..l.count() - 1]$
– Polyline-ok nem keresztezhetik egymást:
  $\forall l \in P: \forall p \in P \backslash \{l\}: \neg Cross(p, l)$
– Minden felületet módosító pontnak és elhelyezett modellnek a felület határain belülre kell esnie:
  $\forall p \in V3: Inside(p, b_1 + b_2 + b_3 + b_4) \wedge \forall m \in Mo: Inside(m, b_1 + b_2 + b_3 + b_4)$

Terminálási feltétel: nincs, mivel a program akár a végtelenségig futhat, amíg a felhasználó ki nem lép.

3.2. A felhasznált módszerek részletesen

Először a térbeli polyline-ok elkészítését írom le, majd a Catmull–Rom-szplájn illesztést. Ezt követi a határoló polyline-okra történő felület megadásának és módosításának részletezése. Utána a szintvonalas és a grid nézet előállításával foglalkozom.

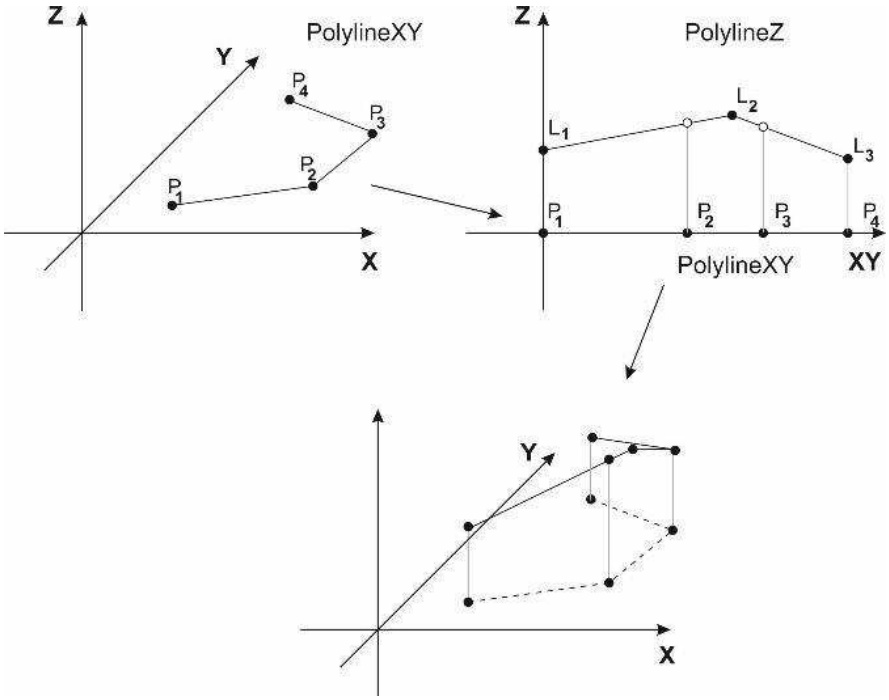
3.2.1. Térbeli polyline-ok elkészítése

A felhasználó felveszi a térbeli polyline XY (polylineXY) és Z (polylineZ) metszetét. Ezekből a program összefésüli a 3D-s polyline-t (3–1. ábra).

Algoritmus (3–2.ábra): az algoritmus lényegében egymásra vetíti a két polyline csomópontjait. Végigmegy a polylineXY összes pontján, ha az adott pont elé esik polylineZ-ből pont, akkor azt hozzáadja az új polyline pontjaihoz, ha nem, akkor a két szomszédos polylineZ-beli pontból kiszámolja a magasságát, és hozzáadja az új pontokhoz. Az algoritmus a polylineXY polyline x, y koordinátáit veszi alapul, azokhoz számolja ki a magassági értékeket, és szúrja be közéjük a polylineZ pontjait.

Invariáns tulajdonságok:

– polylineZ.p[0].x = 0
– polylineZ.p[i].z = 0, i : [0 .. polylineZ.size() – 1]
– polylineXY.length() = polylineZ.length()



3–1. ábra: A polyline-ok összefésülése

```
PolyLine (polylineXY:P, polylineZ:P):P    distance, distanceCurr, h : IR, point : V3, points : queue<V3>

points := <>
distance := 0
j := 0

for i ← 1 to polylineXY.size()

    distanceCurr := Distance(polylineXY.p[i], polylineXY.p[i-1])

    distance += distanceCurr

    while (distance > polylineZ.p[j].x)

        scale := 1 - Abs(distance - polylineZ.p[j])/distanceCurr
        point := new V3(polylineXY.p[i-1].x + (polylineXY.p[i].x-polylineXY.p[i-1].x)*scale,
                        polylineZ.p[j].y,
                        polylineXY.p[i-1].z + (polylineXY.p[i].z-polylineXY.z[i-1].z)*scale)
        points.add(point)
        ++j

    h := polylineZ.p[j-1].y + (polylineZ.p[j].y - polylineZ.p[j-1].y)* (distance - polylineZ.p[j-1].x) /
                              (polylineZ.p[j].x - polylineZ.p[j-1].x)
    point := new V3(polylineXY.p[i].x,
                    h,
                    polylineXY.p[i].z)
    points.add(point)

return new P(points)
```

3–2. ábra: Térbeli polyline elkészítésének algoritmusa

3.2.2. Catmull–Rom-szplájn illesztése

A felhasználónak lehetősége van a polyline rajzolása közben a megadott pontokra Catmull–Rom-szplájnt illeszteni. A Catmull–Rom-szplájn harmadfokú ívekből állítja elő az interpoláló görbét, csak az interpolálandó pontokat kell megadni (3–3.ábra) [1][2]. Az interpoláció 4 kapcsolópontot használ (3–4. ábra).

A következő egyenletek írják le a görbét (több megadási mód is van) [2]:

$$C = \frac{t_2 - t}{t_2 - t_1} B_1 + \frac{t - t_1}{t_2 - t_1} B_2$$

$$B_1 = \frac{t_2 - t}{t_2 - t_0} A_1 + \frac{t - t_0}{t_2 - t_0} A_2$$

$$B_2 = \frac{t_3 - t}{t_3 - t_1} A_1 + \frac{t - t_1}{t_3 - t_1} A_3$$

$$A_1 = \frac{t_1 - t}{t_1 - t_0} P_0 + \frac{t - t_0}{t_1 - t_0} P_1$$

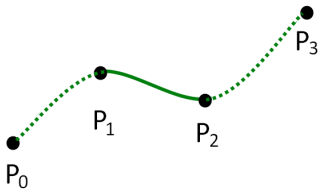
$$A_2 = \frac{t_2 - t}{t_2 - t_1} P_1 + \frac{t - t_1}{t_2 - t_1} P_2$$

$$A_3 = \frac{t_3 - t}{t_3 - t_2} P_2 + \frac{t - t_2}{t_3 - t_2} P_3$$

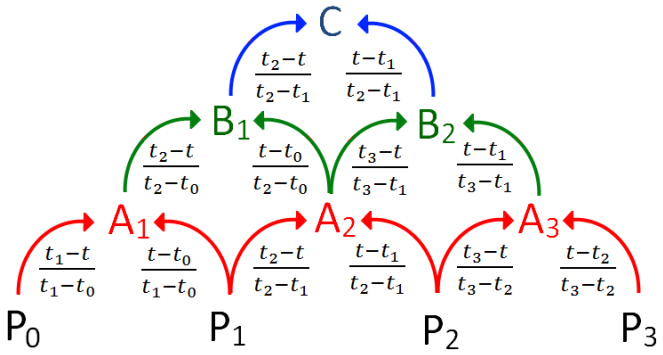
$$t_{i+1} = \left[\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}\right]^{\alpha} + t_i \quad //2D$$

$$t_{i+1} = \left[\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2}\right]^{\alpha} + t_i \quad //3D$$

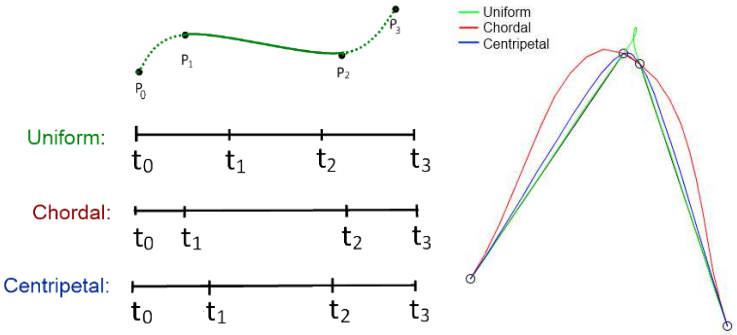
$$\alpha \in \mathbb{R}, \alpha \in [0,1]$$



3–3. ábra: A Catmull–Rom-szplájn [2]



3–4. ábra: Az interpoláció számítása a kapcsolópontokból [2]



3–5. ábra: Catmull–Rom-szplájn fajtái [2]

Az α paraméternek 3 érték közül szoktak választani (3–5. ábra):

- 0 – Uniform Catmull–Rom-szplájn;
- 0.5 – Centripetal Catmull–Rom-szplájn;
- 1 – Chordal Catmull–Rom-szplájn.

A számítógépes grafikában legtöbbször a 0.5-ös értéket választják paraméterül, így én is ezt használtam.

Algoritmus (3–6. ábra): bemenetként a felhasználó által felvett pontok szolgálnak polyline-ként tárolva. Az algoritmus veszi az első négy pontot és interpolálja rá a szplájnt. A pontokat mindig csúsztatja, amíg a polyline végére nem ér. Mivel a szplájn a 4 kontrollpontja közül a második és harmadik pont között ad eredményt (3–3. ábra), ezért a program beszúr egy-egy pontot a polyline első és második, és az utolsó és utolsó előtti pontok alkotta szakasz közepére. Az algoritmus az első és az utolsó pontot az interpolációtól függetlenül hozzáadja az új polyline-hoz. Ezzel értem el, hogy a szplájn átmenjen a kezdő és a végponton is, és a görbe végig folytonos legyen. A kimenet szintén polyline, a felbontása paraméterezhető (detailness).

Invariáns tulajdonság: $p \in P : p.size() > 2$ (ha a csomópontok száma 2, akkor helybenhagyja a program a szakaszt.)

3.2.3. A felület létrehozása

A program a megrajzolt határoló térbeli polyline-okból először Coons-foltot készít. A bilineáris Coons-folt általános leíró képlete [1]:

$$s(u,v) = [1-u \quad u]\begin{bmatrix} s(0,v) \\ s(1,v) \end{bmatrix} + [s(u,0) \quad u(u,1)]\begin{bmatrix} 1-v \\ v \end{bmatrix} - [1-u \quad u]\begin{bmatrix} s(0,0) & s(0,1) \\ s(1,0) & s(1,1) \end{bmatrix}\begin{bmatrix} 1-v \\ v \end{bmatrix}$$

$$s(u,0) = a_1(u), \qquad s(u,1) = a_2(u),$$

$$s(0,v) = b_1(v), \qquad s(1,v) = b_2(v),$$

u, v 0 és 1 között futó paraméter

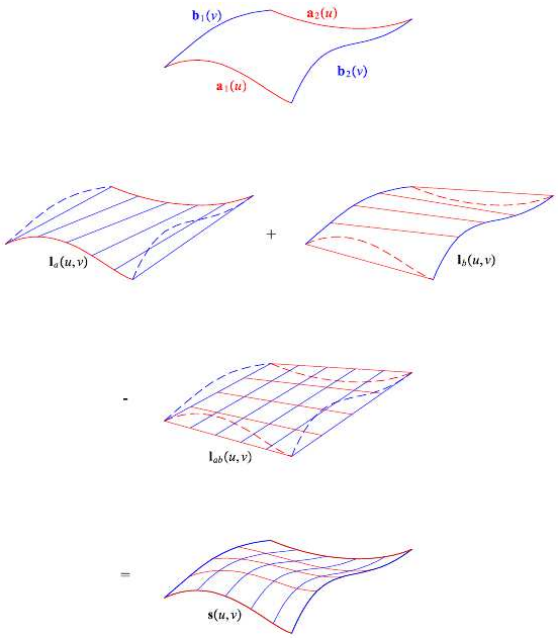
A Coons-folt létrehozását a 3–7. ábra szemlélteti.

```
Catmul(points:P):P
    points.Insert(1, Lerp(points[0], points[1], 0.5))
    points.Insert(points.size()-1, Lerp(points[points.size()-2], points[points.size()-1], 0.5))
    catmulPoints := <>
    catmulPoints.Add(points[0])
    for i ← 1 to points.size()-2
        CatmulRomInterpolation(catmulPoints, points[i-1],points[i],points[i+1],points[i+2])
    catmulPoints.Add(points[points.size()-1])
    return catmulPoints

CatmulRomInterpolation(catmulPoints&:P, p0:V3, p1:V3, p2:V3, p3:V3)
t0, t1, t2, t3, t, detailness:IR, A1, A2, A3, B1, B2, C:V3
    t0 := 0
    t1 := GetT(t0, p0, p1)
    t2 := GetT(t1, p1, p2)
    t3 := GetT(t2, p2, p3)
    t := t1
    while (t<t2)
        A1 := (t1-t)/(t1-t0)*p0+(t-t0)/(t1-t0)*p1
        A2 := (t2-t)/(t2-t1)*p1+(t-t1)/(t2-t1)*p2
        A3 := (t3-t)/(t3-t2)*p2+(t-t2)/(t3-t2)*p3
        B1 := (t2-t)/(t2-t0)*A1+(t-t0)/(t2-t0)*A2
        B2 := (t3-t)/(t3-t1)*A2+(t-t1)/(t3-t1)*A3
        C  := (t2-t)/(t2-t1)*B1+(t-t1)/(t2-t1)*B2
        catmullPoints.Add(C)
        t := t + (t2-t1)/detailness

GetT(t:IR, p0:V3, p1:V3):IR     a, b, c:IR
    a := Pow((p1.x-p0.x), 2) + Pow((p1.y-p0.y), 2) + Pow((p1.z-p0.z), 2)
    b := Pow(a, 0.5)
    c := Pow(b, alfa)
    return c+t
```

3–6. ábra: A Catmull–Rom-szplájn elkészítésének algoritmusa



3–7. ábra: A bilineáris Coons-folt létrehozása [1]

Ahhoz, hogy a program interaktív kontrollpontbeszúrással változtatni tudja a Coons-folttal kapott alapfelületet, egy 2D-s mátrixban eltárolom mintavételezéssel (ennek a sűrűségét a felhasználó paraméterezheti) a Coons-folt pontjait. Ennek a mátrixnak a pontjait felezéssel sűrítem, amely megadja a végleges felület kontroll-poligonhálóját. A kontroll-poligonháló darabjaira B-szplájn felületeket illesztek a 3–8. ábra alapján. Ezzel az elkészült felület folytonosságát biztosítani tudom, cserébe a 2–3. ábrán és a 3–9. ábrán bemutatott hiba fellép. A Coons-folt mintavételezésének sűrítésével csökken a végleges felület eltérése a Coons-folttól.

A B-szplájn felületek előállítására meglévő forráskódot használtam fel [4]. A leíró egyenletek [3]:

$$s(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} b_{ij} F_i(u) G_j(v)$$

A b_{ij} pontokat a felület kontrollpontjainak az $F_i(u)$ és $G_j(v)$ függvényeket bázisfüggvényeknek nevezzük, amelyek jelen esetben B-Szplájn alapfüggvények. A normalizált B-szplájn alapfüggvény [3]:

$$u_i \leq u_{i+1}, u_i \in \mathbb{R}(i = 0, \dots, n+k) skalárok\ csomóértékek(knot\ values)$$

$$N_i^1(u) = \begin{cases} 1, ha\ u_i \leq u \leq u_{i+1}; \\ 0\ egyébként. \end{cases}$$

$$N_i^k(u) = \frac{u - u_i}{u_{i+k-1} - u_i} N_i^{k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} N_{i+1}^{k-1}(u)$$

$\frac{0}{0}$ - *t definíció szerint 0-nak tekintjük*

Az így kapott függvény k – 1-ed fokú polinom

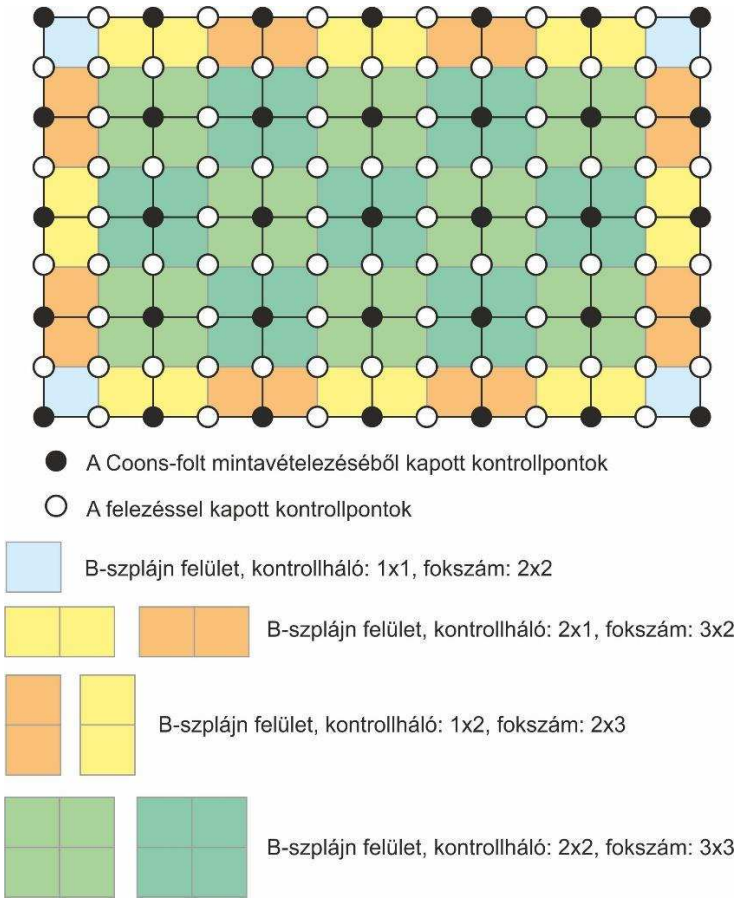
A kontrollpontok és a B-szplájn felület kapcsolatát a 3–10. ábra mutatja.

Unity-ben az elkészült felületből egy Mesh-t készítek, amelyet háromszögek segítségével tárolok.

A felület elkészítését a 3–11. ábra *algoritmus*ával foglalom össze. A gridX, gridY, detailnessX és detailnessY felhasználói paraméterek. A gridX és gridY növelésével a felület jobban közelíti a Coons-foltot, a detilnessX és detailnessY növelésével a felület folytonossága érhető el.

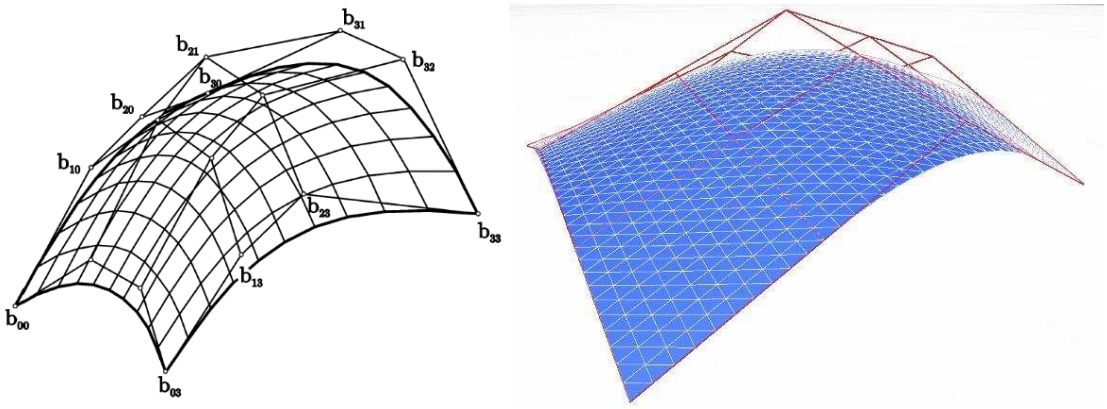
Invariáns tulajdonságok: gridX > 1, gridY > 1, detailnessX > 0, detailnessY > 0,

gridX * gridY * detailnessX * detailnessY < 65512 (Unity-ben a Mesh csomópontjainak száma korlátos).

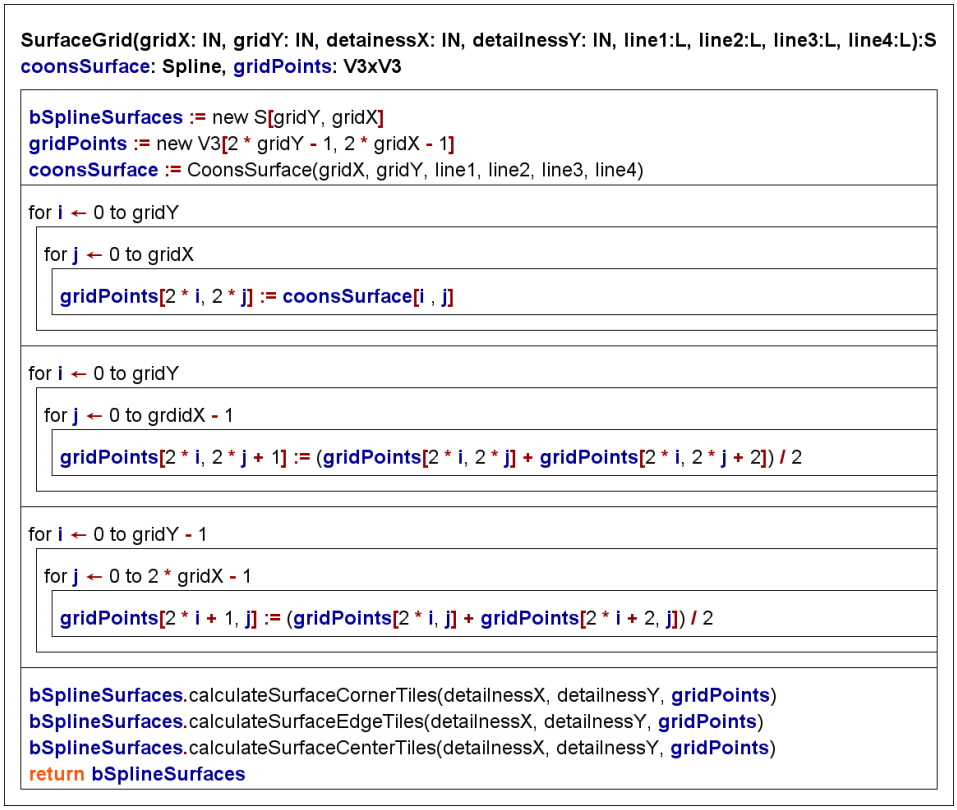


3–8. ábra: A kontroll-poligonháló, és a ráillesztett B-szplájn felületek

3–9. ábra: Az új felület eltérése a Coons-folttól
(A mintavételezés sűrítésével a hiba csökken)



3–10. ábra: A B-szplájn és a kontrollpontok kapcsolata



```
SurfaceGrid(gridX: IN, gridY: IN, detainessX: IN, detailnessY: IN, line1:L, line2:L, line3:L, line4:L):S
coonsSurface: Spline, gridPoints: V3xV3

bSplineSurfaces := new S[gridY, gridX]
gridPoints := new V3[2 * gridY - 1, 2 * gridX - 1]
coonsSurface := CoonsSurface(gridX, gridY, line1, line2, line3, line4)

for i ← 0 to gridY
    for j ← 0 to gridX
        gridPoints[2 * i, 2 * j] := coonsSurface[i , j]

for i ← 0 to gridY
    for j ← 0 to grdidX - 1
        gridPoints[2 * i, 2 * j + 1] := (gridPoints[2 * i, 2 * j] + gridPoints[2 * i, 2 * j + 2]) / 2

for i ← 0 to gridY - 1
    for j ← 0 to 2 * gridX - 1
        gridPoints[2 * i + 1, j] := (gridPoints[2 * i, j] + gridPoints[2 * i + 2, j]) / 2

bSplineSurfaces.calculateSurfaceCornerTiles(detailnessX, detailnessY, gridPoints)
bSplineSurfaces.calculateSurfaceEdgeTiles(detailnessX, detailnessY, gridPoints)
bSplineSurfaces.calculateSurfaceCenterTiles(detailnessX, detailnessY, gridPoints)
return bSplineSurfaces
```

3–11. ábra: A végső felület elkészítésének algoritmusa

3.2.4. Felület módosítása pontok és polyline-ok beszúrásával

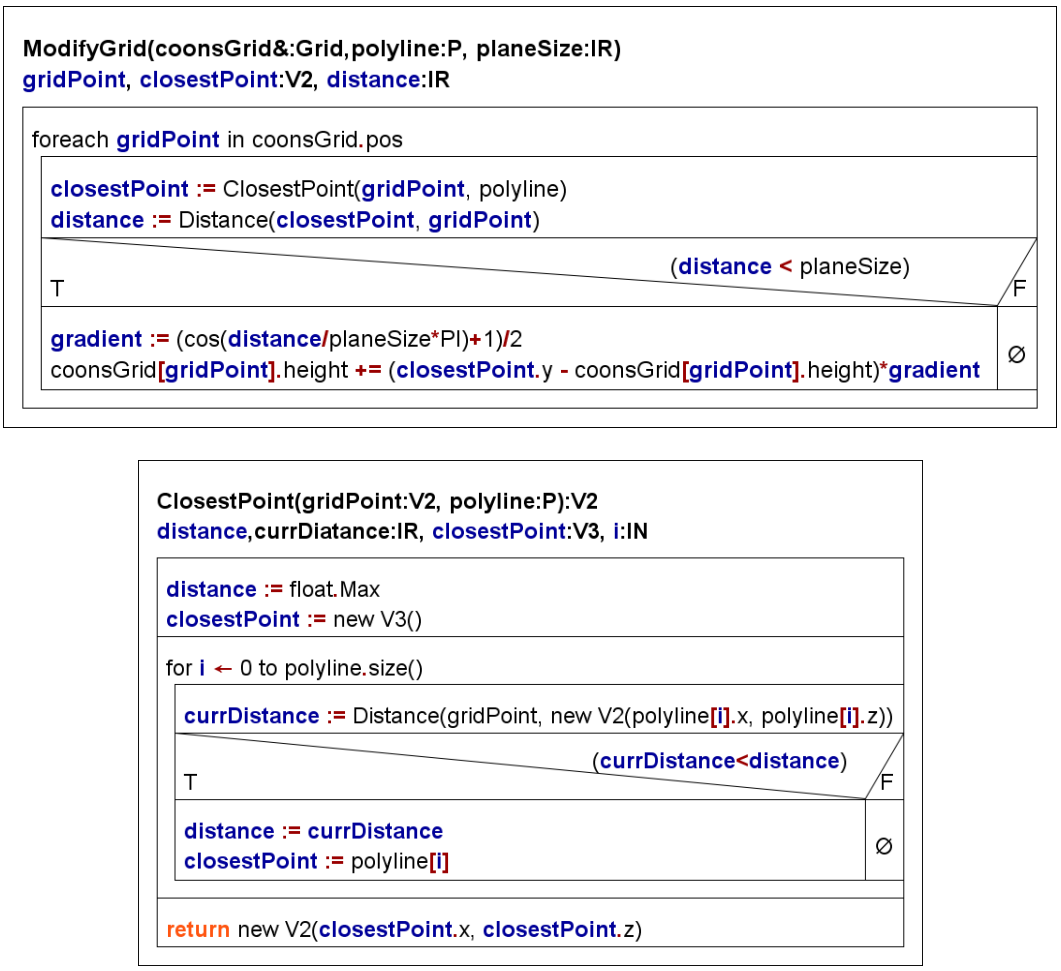
A program ugyanazt az algoritmust követi mindkét esetben, egy különbségtől eltekintve. Pont beszúrásakor a program megkeresi a ponthoz legközelebbi kontrollpontot, és azzal helyettesíti azt (utána egyelemű polyline-ként fog viselkedni).

Algoritmus (3–12. ábra): a program végigmegy az Coons-foltból kapott kontrollpontokon, megkeresi, hogy melyik polyline-beli ponthoz van legközelebb. Amint a felhasználó által megadott méreten belül van a távolság, a kontrollpont magassága módosul a következő függvény segítségével:

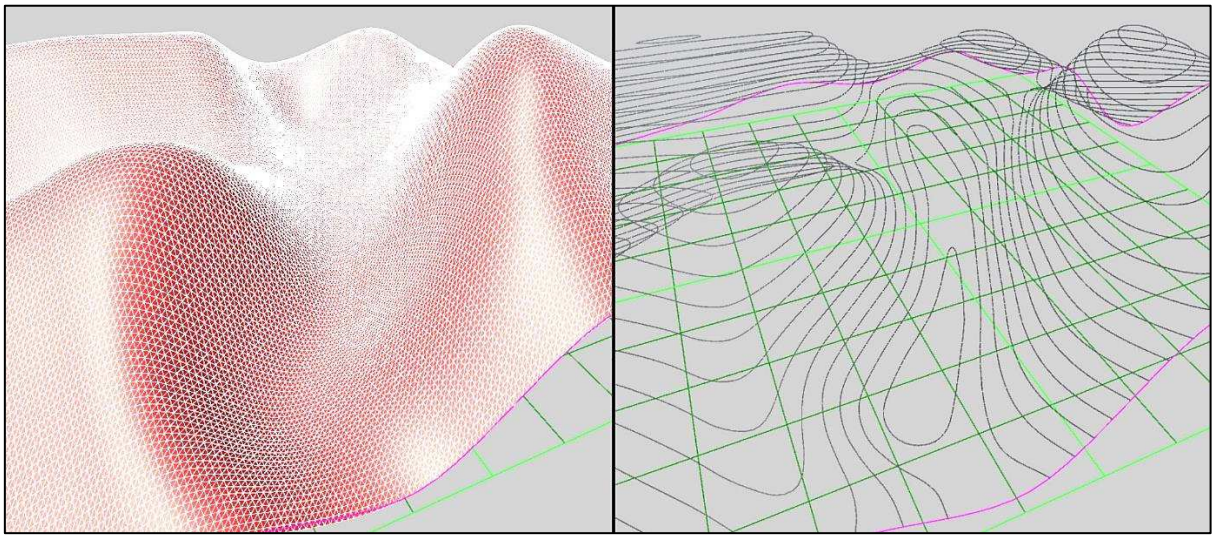
$$P_k.y = P_k.y + (P_l.y - P_k.y) * \frac{\cos\left(\frac{Distance(P_l.y, P_k.y)}{size} * \pi\right) + 1}{2}$$

$$P_k, P_l \in V3, P_k - Kontrollpont, P_l - Legközelebbi\ pont$$
$$size \in \mathbb{R} - felhasználó\ által\ megadott\ paraméter,$$
$$mekkora\ távolságon\ belül\ módosuljanak\ a\ pontok\ magasságai$$

Az algoritmus lefutása után a kontrollpontokból újból elkészül a felület a 3.2.3. fejezet alapján.

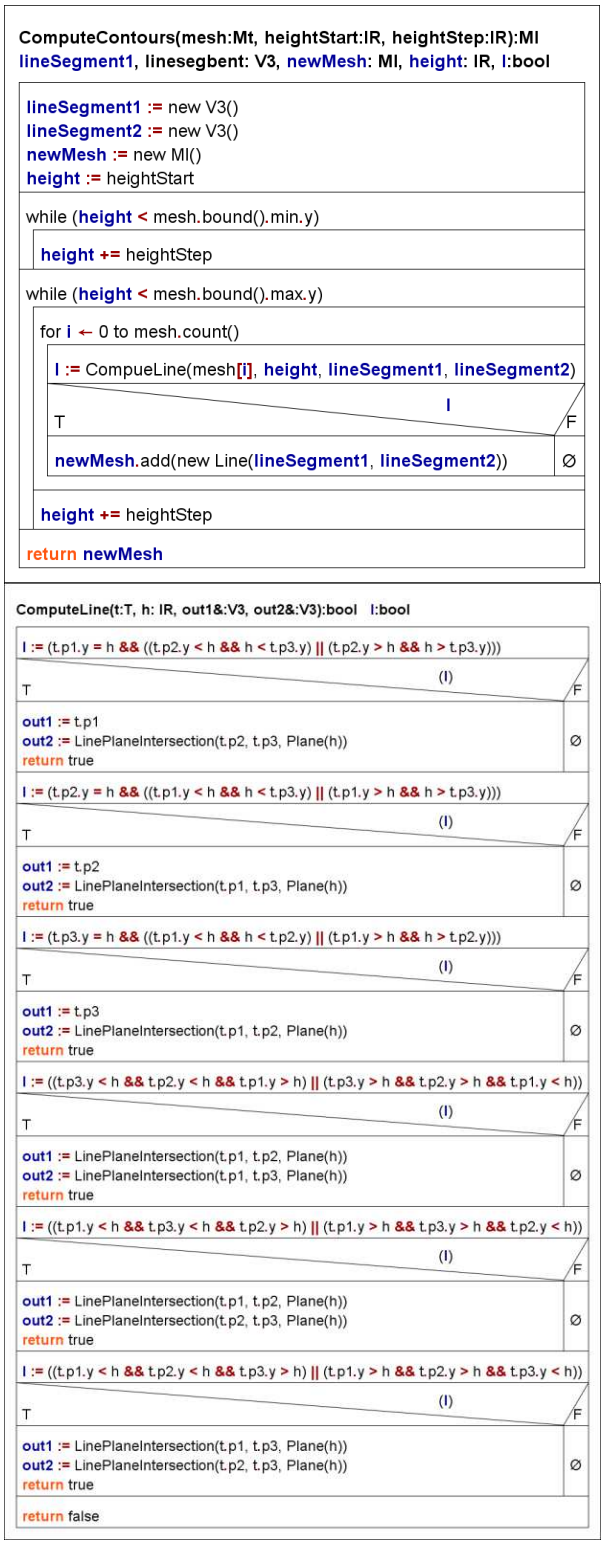


3–12. ábra: A felület módosítása Polyline-nal



3–13. ábra: A szintvonalas nézet

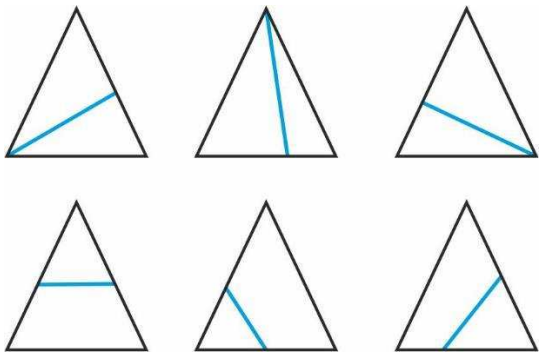
3.2.5. Szintvonalas nézet elkészítése



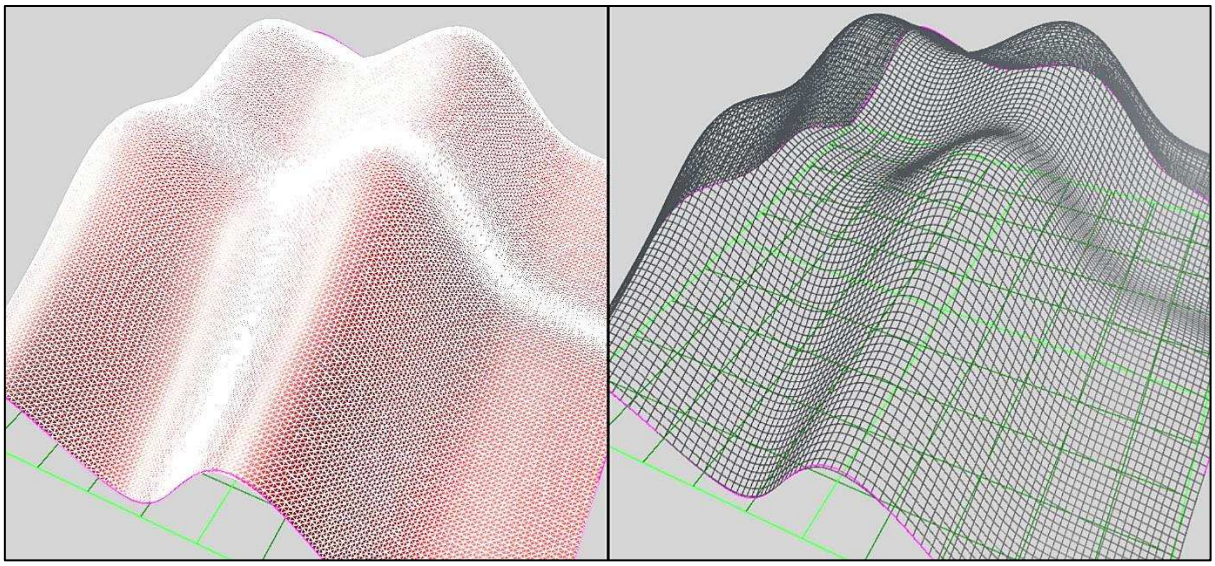
3–14. ábra: A szintvonalak kiszámításának algoritmusa

Az elkészült felületet Mesh-ként tárolom, amelynek alapját háromszögek képezik. Ezekből a háromszögekből számolom ki a szintvonaldarabokat, amelyeket szintén Mesh-ként jelenítek meg, amelynek felépítő elemei most vonalak. A felhasználó paraméterezi a kezdő szintvonal magasságát és az osztásközt (3–13. ábra)

Algoritmus (3–14. ábra): a program felhasználja az eredeti Mesh befoglaló dobozának minimum és maximum magasságát, és e két érték között számolja végig a szintvonalakat. Minden magassági értéknél végigmegy a háromszögeken, és megvizsgálja, hogy a szintvonal átmegy-e rajta. A program 6 esetet vizsgál (3–15. ábra), amint az egyik eset igaz, kiszámolja az adott oldalak és az adott magasságú sík metszéspontjait, és hozzáadja az új Mesh-hez. A LinePlaneIntersection függvényt külső forrásból használtam fel [5].

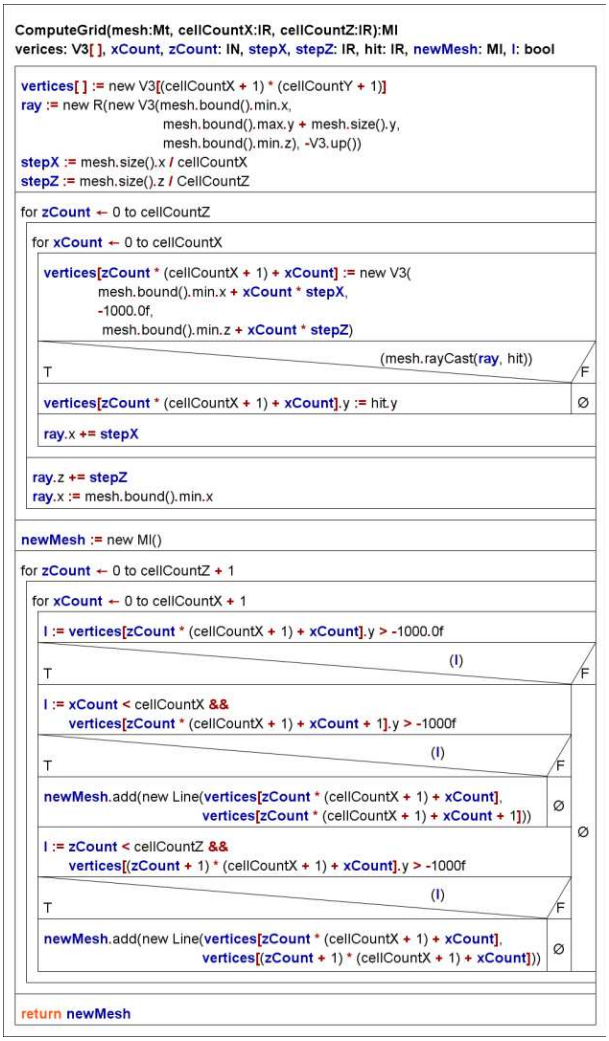


3–15. ábra: Háromszög és vonal metszésének fő esetei



3–16. ábra: A grid nézet

3.2.6. A grid nézet elkészítése



3–17. ábra: A grid nézet elkészítésének algoritmusa

A grid nézet elkészítése sugárkövetésen alapszik, azaz sugarak metszetét vizsgálom a felületemmel [6], [7]. A megjelenítés hasonlóan történik, mint a szintvonalas nézetnél, az eredményt a program Mesh-ként jeleníti meg, amelynek felépítő elemei most is vonalak (3–16. ábra).

Algoritmus (3–17. ábra): az eredeti felületre a felhasználó által megadott sűrűséggel egy rácshálót helyez el a program. A rácsháló minden pontjából egy függőleges sugarat indítok lefelé, a felülettel való metszete megadja a rácspont magasságát. Az így kapott pontokból képzett vonalak alkotják az új Mesh-t.

Invariáns tulajdonság:

mesh.bound().min.y > -1000

3.3. A program fizikai szerkezete

A program fizikai szerkezetének, fájlszerkezetének, a felhasznált moduloknak és eszközöknek ismertetése következik ebben a részben.

3.3.1. A fizikai szerkezet és könyvtárstruktúra

A program fájljai a *szakdolgozat\SurfaceDrawer* könyvtárban találhatók meg. Itt indítható el a surface.exe futtatható fájl (Windows 7, 8, 10 64 bites operációs rendszeren tesztelve). A megírt programkódok, az elkészített egyéb elemek a *szakdolgozat\SurfaceDrawer\ \Assets* könyvtárban helyezkednek el. A többi könyvtár és állomány a Unity szoftverben készített projekt (a projekt a *szakdolgozat\SurfaceDrawer\Assets* könyvtárban lévő surfaceDrawer.unity fájllal indítható, ha a számítógépen telepítve van a Unity 2017) fejlesztése során létrejövő fájlstruktúra. Az Assets könyvtár alkönyvtárai:

- Editor:
  - Object2Terrain.cs: Mesh és Terrain közötti átalakításra szolgál. A Terrain hasznos terepmodellje a Unity-nek. Ez a funkció csak a projekt Unity-ben való futtatása közben használható, nem része a fő programnak. A kód nem saját, külső forrásból származik [8].
- Materials:
  - A programhoz használt materialokat tartalmazza. Ezek biztosítják a Mesh megjelenésének tulajdonságait (a felhasználó a program futása közben választhat kavicsos (rockMaterial.mat), homok- (sandMaterial.mat) és fűtextúra (grassMaterial.mat) között, a szintvonalas és grid nézet „drótvázas” textúrát használ).
  - Alkönyvtárak:
    - Wireframe_UCLUAGameLab: külső forrásból származó drótvázas materialok [9].
    - WireframeMaterials: szintén külső forrásból származik, a szintvonalas és grid nézet materialjait tartalmazza.
- Models:
  - A felületre elhelyezhető tereptárgyak modelljét tartalmazza, külső forrásból, Unity Assets store-ból töltöttem le.
  - Alkönyvtárak:
    - Low Poly Buildings Lite: 2 háztípusnak a modelljét, segédállományait tartalmazza [10].
    - Trees: 4 fának a modelljét tartalmazza [11].

- Scripts:
  - Itt találhatók a program működéséhez szükséges .cs állományok.
  - Alkönyvtárak:
    - Camera2 (minden panel tartalmának megjelenítéséért egy kamera felel):
      - IsOnCamera2.cs: vizsgálja, hogy a kurzor a 2-es panel felett van-e;
      - Draw2D.cs: a 2-es panelen történő rajzolásért felel;
      - Camera2Position.cs: a kamera mozgatásáért felel;
      - Camera2MousePosition.cs: a kurzor pozícióját adja vissza;
      - Camera2Draw.cs: a 2-es panelen lévő elemeket rajzolja ki.
    - Camera3:
      - IsOnCamera3.cs: vizsgálja, hogy a kurzor a 3-as panel felett van-e;
      - DrawZ.cs: a 3-as panelen történő rajzolásért felel;
      - Camera3Position.cs: a kamera mozgatásáért felel;
      - Camera3MousePosition.cs: a kurzor pozícióját adja vissza;
      - Camera3Grid.cs: a koordinátarendszer kirajzolása a feladata;
      - Camera3Color.cs: a 3-as panel alapszínének változtatásáért felel.
    - MainCamera:
      - MainCameraPosition.cs: a fő kamera (1-es panel) mozgatásáért felel;
      - MainCameraDraw.cs: az 1-es panelen lévő elemeket rajzolja ki.
    - LinesPoints:
      - CreatePoint.cs: pontok felvételét valósítja meg;
      - CreateLines.cs: polyline-ok felvételét valósítja meg.
    - MeshTerrain:
      - RandomGenerator.cs: felület random generálását végzi;
      - CreateTerrain.cs: terrain készítéséért felel (csak Unityben);
      - CreateMesh.cs: elkészíti a felületet, majd pontjaiból a Mesh-t;
      - WireSurface:
        - WireSurfaceGrid.cs: a grid nézethez szükséges Mesh-t készíti;
        - WireSurfaceContours.cs: a szintvonalas nézethez szükséges Mesh-t készíti.
    - Models:
      - AddTree.cs: fák felhelyezéséért felel;
      - AddHouse.cs: házak feltevéséért felel.

- Othes:
  - ActionText.cs, ErrorText.cs, ModeText.cs: az üzenetek kiírásáért felelnek;
  - CoordinataXY.cs, CoordinataZ.cs: a koordináták kiírásáért felelnek;
  - InputPlaneSize.cs, InputZ.cs: beviteli mezőkért felelnek;
  - Interactions.cs: a program vezérlése a feladata;
  - ScreenRecorder.cs: a mentést végzi;
  - Panels alkönyvtár fájljai a menüpontok működéséért felelnek: BringToFront.cs, InfoPanel.cs, RandomPanel.cs, SavePanel.cs, SetInfoPanel.cs, SetRandomPanel.cs, SetSavePanel.cs, SetSettingsPanel.cs, SetSubmitPanel.cs, SetTexturePanel.cs, SettingsPanel.cs, SubmitPanel.cs, TexturePanel.cs.
- Utility:
  - Boundary.cs: a felület határát tárolja;
  - Catmul.cs: 2D-s Catmull–Rom-szplájnt valósítja meg;
  - Catmul3D.cs: 3D-s Catmull–Rom-szplájnt valósítja meg;
  - Geometry.cs: a program működéséhez használt geometriai elemek tárolója;
  - Polyline.cs: a Polyline adatszerkezetet valósítja meg;
  - TangentPlane.cs: az érintő síkokat tárolja;
  - Utils.cs: a program működéséhez szükséges változókat és függvényeket tárol.
  - Surface alkönyvtár:
    - BSplineMath.cs: szplájn elkészítéséhez segédfüggvények [4];
    - BSplineSurface.cs: B-szplájn felület elkészítését valósítja meg;
    - CoonsSurface.cs: Coons-folt elkészítéséért felel;
    - PointData.cs: grid pontot 2D-s és magassági értékkel tároló struktúra;
    - SurfaceGrid.cs: a végső felület megvalósításáért felel.
  - Spline_Ryan:
    - kubikus szplájn leírását tartalmazza, a programban végül nem használtam.
- Shaders:
  - A materialokhoz felhasznált shader állományokat tartalmazza. Külső forrásból származnak, shader programozással nem foglalkoztam [9].

3.3.2. Felhasznált eszközök, kész modulok

A program elkészítéséhez használt eszközök, modulok:

- Unity 2017.3.0f3 (64 bit): a programom Unity-ben készült, Unity elemek, könyvtárak felhasználásával;
- Microsoft Visual Studio 2017: a programozás, .cs fájlok fejlesztése itt történt;
- Unity Asset Strore: interneten elérhető modulok gyűjteménye Unityhez, a következő funkciókhoz használtam meglévő forrást:
  - drótvázas material [9];
  - házak modelljei [10];
  - fák modelljei [11].

Egyéb internetes forrásból használt modulok:

- Catmull–Rom-szplájn forráskódja [3];
- B-szplájn felületek kiszámolása [4];
- 3D-s függvények használata [5];
- kamerák mozgatása [12];
- mentés fájlba [13].

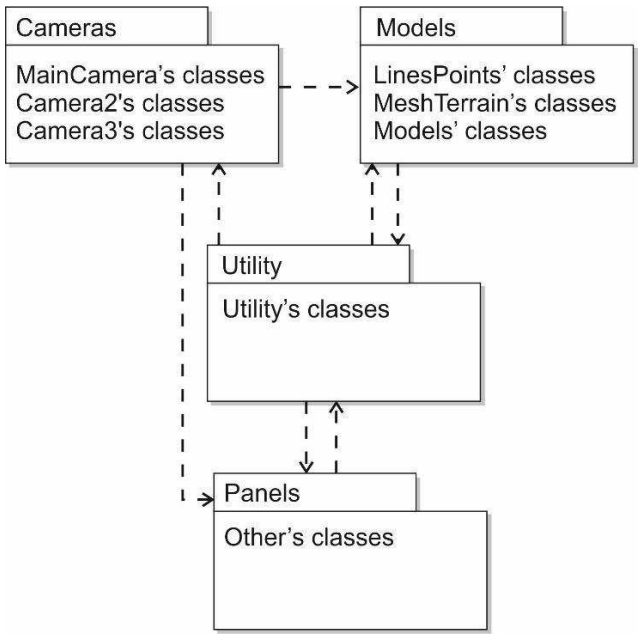
A dokumentáció elkészítéséhez használt programok:

- Microsoft Word 2016: a dokumentáció szövege és tördelése készült wordben;
- Microsoft Visual Studio 2010: UML diagramok készítése;
- Structorizer: stuktogramok készítésére;
- Adobe Photoshop, CorelDraw: ábrák szerkesztése.

3.4. A program logikai szerkezete

A program osztályait működést tekintve a következő 4 rétegbe soroltam (3–18. ábra):

- Kamerák: a kamerák rétegbe tartozó osztályok felelnek a paneleken lévő elemek kirajzolásáért, a polyline-ok és pontok interaktív felvételéért és a panelek nézetének változtatásáért.
- Modellek: ezen réteg osztályai gondoskodnak az alkalmazásban előforduló objektumok elkészítéséért. Ehhez információt gyűjtenek a kamerák rétegről (pontok, poyline-ok térbeli adatai) és a Utility rétegről (osztályleírások, felhasználói paraméterek, már tárolt geometriai objektumok). Az elkészült objektumok a Utility rétegen tárolódnak.



3–18. ábra: A rétegek kapcsolata és osztályaik

– Utility: ez a réteg felel a program vezérléséért. Kezeli az interakciókat, eltárolja az elkészült objektumokat, továbbítja a kamerák rétegre, kapcsolatban áll a panelek réteggel, ahol a felhasználó állíthatja a paramétereket.
– Panelek: ez a réteg közvetíti az üzeneteket, kéri be a paramétereket, azaz kapcsolatot teremt a felhasználóval.

3.4.1. A kamerák réteg

A kamerák rétegének osztályait 3 elkülöníthető csoportba soroltam. A csoportok egyenként felelnek a 3 panel működésért (3–19. ábra). Kapcsolatban a 3. panel áll a 2. panellel, mivel a 2-es panelre felvett polyline automatikusan változtatja a 3-as panel tulajdonságait.

Osztályok:

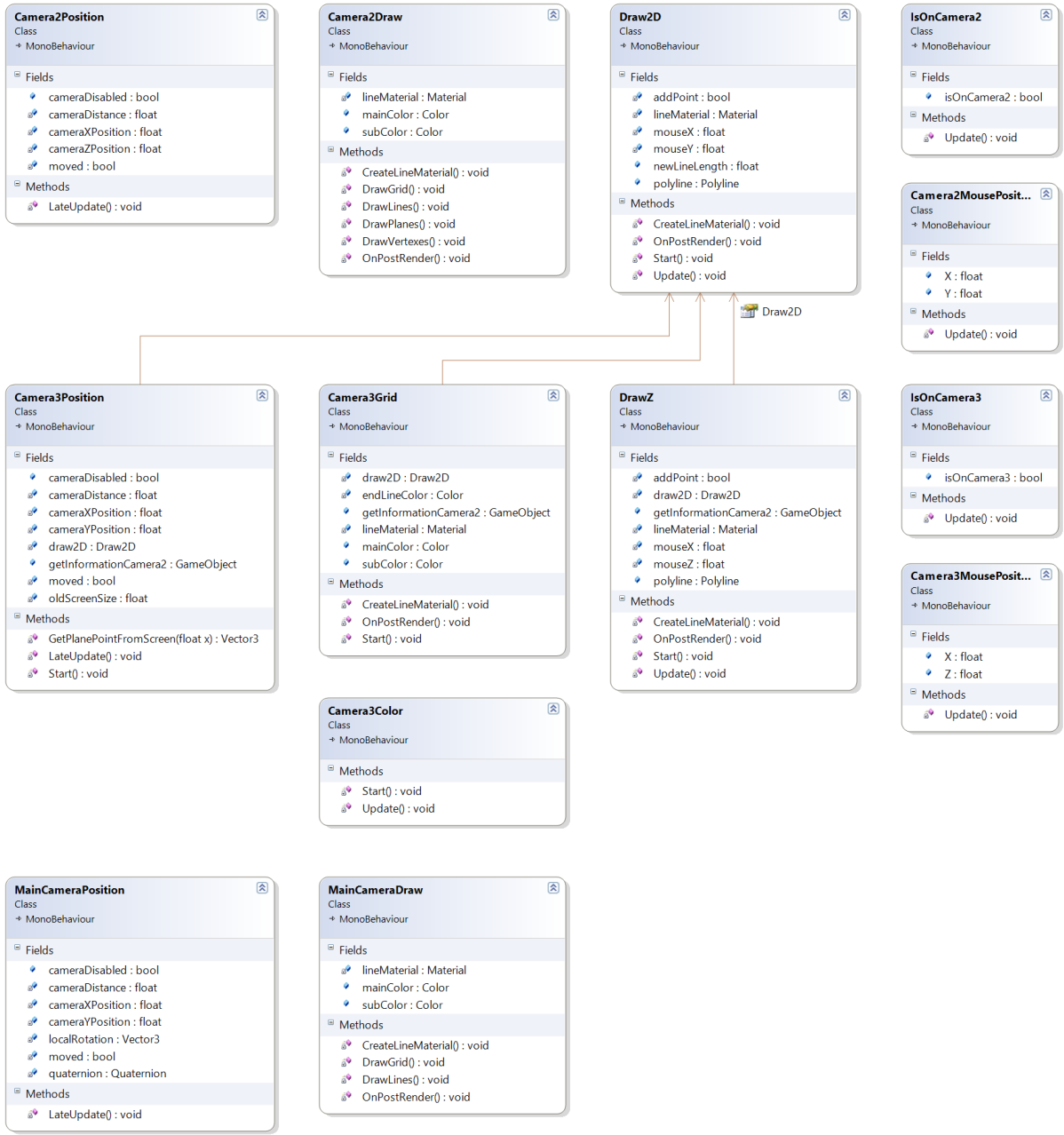
– Camera2Position: a 2-es kamera pozíciójáért felel.

Adattagok:

– cameraDisabled: logikai érték, ha igaz, a kamera mozgatható.
– cameraDistance: a kamera távolsága az XZ síktól.
– cameraXPosition: a kamera X koordinátája.
– cameraZPosition: a kamera Z koordinátája.
– moved: logikai érték, amely figyeli, hogy mozgattuk-e a kamerát.

Metódusok:

– LateUpdate(): bal shift billentyű lenyomása kapcsolja a kamera mozgathatóságát. Amint a kamera pozíciója állítható, figyeli a billentyűzet és az egér használatát, és annak megfelelően állítja a kamera állását.



3–19. ábra: A kamerák réteg osztályai

– Camera2Draw: a 2-es panelre történő kirajzolást végzi.

  Adattagok:

  – lineMaterial: a kirajzolt elemek materialja.
  – mainColor: a fő koordinátavonalak színe.
  – subColor: a mellék koordinátavonalak színe.

  Metódusok:

  – OnPostRenderer(): az osztály többi függvényét hívja meg, felel azért, hogy a kirajzolandó objektumok folyamatosan megjelenjenek a képernyőn (GL könyvtárat használtam hozzá).

  - CreateLineMaterial(): elkészíti a materialt, ha hiányzik.
  - DrawGrid(): kirajzolja a koordinátarendszert.
  - DrawLines(): kirajzolja a már megrajzolt polyline-okat (Utility-ben tárolom: Geometry.lines és Geometry.boundaryLines).
  - DrawPlanes(): kirajzolja a már felvett felületet módosító pontokat (Utility: Geometry.tangentPlanes).
  - DrawVertexes(): kirajzol a polyline-ok végpontjához egy kis négyzetet a felhasználó által paraméterezhető toleranciaérték felhasználásával (Polyline rajzolása során az ezen belül felvett végpont hozzáigazodik a megfelelő polyline végpontjához).
- Draw2D: a 2-es panelen történő interaktív rajzolást teszi lehetővé.

  Adattagok:
  - addPoint: logikai érték, amely igaz, ha a felhasználó megnyomja a bal egérgombot.
  - lineMaterial: a polyline materialja.
  - mouseX: a kurzor X koordinátája.
  - mouseY: a kurzor Y koordinátája.
  - newLineLength: a megrajzolt polyline hossza.
  - polyline: az új polyline.

  Metódusok:
  - Start(): létrehozza az új Polyline objektumot.
  - Update(): a polyline XZ síkbéli koordinátáinak interaktív felvételéért felel. A felhasználó a billentyűzet segítségével aktiválja a funkciókat és az egér használatával teszi le a csomópontokat. Az egér jobb gombjának lenyomásával a polyline szerkesztése a 3-as panelen folytatódik, a polyline hossza átadódik.
  - CreateLineMaterial(): elkészíti a materialt, ha hiányzik.
  - OnPostRender(): megjeleníti a rajzolás közben lévő polyline-t.
- IsOnCamera2: azt vizsgálja, hogy az egér a 2-es panel felett helyezkedik-e el.

  Adattagok:
  - isOnCamera2: logikai változó, amely akkor igaz, ha a kurzor a 2-es panel felett van.

  Metódusok:
  - Update(): folyamatosan figyeli az egér pozícióját.
- Camera2MousePosition: az egér helyzetének síkbéli koordinátáit számolja, ha az egér a 2-es panel felett helyezkedik el.

  Adattagok:
  - X: az egér X koordinátája.
  - Y: az egér Y koordinátája.

Metódusok:

  - Update(): folyamatosan figyeli és számolja az egér pozícióját.

- Camera3Position: a 3-es kamera pozíciójáért felel.

  Adattagok:

  - cameraDisabled: logikai, ha igaz a kamera mozgatható.
  - cameraDistance: kamera távolsága a síktól.
  - cameraXPosition: a kamera X koordinátája.
  - cameraYPosition: kamera Y koordinátája.
  - draw2D, getInformationCamera2: Camera2 GameObjecttel biztosítja a kapcsolatot.
  - moved: logikai érték, amely figyeli, hogy mozgattuk-e a kamerát.
  - oldScreenSize: a 3-as panel rajzolás előtti mérete.

  Metódusok:

  - Start(): beállítja draw2D adattagot a Camera2 Draw2D osztályára.
  - LateUpdate(): bal alt billentyű lenyomása kapcsolja a kamera mozgathatóságát. Amint a kamera mozgatható, figyeli a billentyűzet és az egér használatát, és annak megfelelően állítja a kamera pozícióját. Amint a felhasználó szerkeszteni kezd a 3-as panelen, a kamera pozíciója automatikusan megváltozik, a 2-es panelen megrajzolt polyline hosszához igazodik.
  - GetPlaneFromScreen(float x): a 3-as panel rajzolás előtti méretének kiszámolásához használt függvény.

- Camera3Grid: a 3-es panelre történő kirajzolást végzi.

  Adattagok:

  - draw2D, getInformationCamera2: Camera2 GameObjecttel biztosítja a kapcsolatot.
  - endLineColor: a polyline elejét és végét jelző vonal színe.
  - lineMaterial: kirajzolt elemek materialja.
  - mainColor: a fő koordinátavonalak színe.
  - subColor: a mellék koordinátavonalak színe.

  Metódusok:

  - Start(): beállítja draw2D adattagot a Camera2 Draw2D osztályára.
  - CreateLineMaterial(): elkészíti a materialt, ha hiányzik.
  - OnPostRenderer(): kirajzolja a koordinátarendszert (folyamatosan megjeleníti a képernyőn) és a 2-es panelen felvett vonal elejét és végét egy fekete vonallal.

- Camera3Color: a 3-as panel színét változtatja.

  Metódusok:

  - Start(): beállítja a 3-as panel alapszínét.
  - Update(): amint a felhasználó a 3-as panelen rajzolhat, megváltoztatja a panel színét.

- DrawZ: a 3-as panelen történő interaktív rajzolást teszi lehetővé.
  Adattagok:
  - addPoint: logikai érték, amely igaz, ha a felhasználó megnyomja a bal egérgombot.
  - draw2D, getInformationCamera2: Camera2 GameObjecttel biztosítja a kapcsolatot.
  - lineMaterial: a polyline materialja.
  - mouseX, mouseZ: a kurzor koordinátái.
  - polyline: az új polyline.

  Metódusok:
  - Start(): beállítja draw2D adattagot a Camera2 Draw2D osztályára.
  - Update(): a polyline Z síkbéli koordinátáinak interaktív felvételéért felel. A felhasználó a billentyűzet segítségével aktiválja a funkciókat és az egér használatával teszi le a csomópontokat. Az egér jobb gombjának lenyomásával fejezi be a poyline rajzolását. A program ellenőrzi, hogy a feltételeknek megfelelő vonalat adott-e meg a felhasználó, ha igen, továbbítja a modell rétegre, ahol elkészül a térbeli polyline és hozzáadódik a geometriai elemekhez.
  - CreateLineMaterial(): elkészíti a materialt, ha hiányzik.
  - OnPostRender(): megjeleníti a rajzolás közben lévő polyline-t.
- IsOnCamera3: azt vizsgálja, hogy az egér a 3-as panel felett helyezkedik-e el.
  Adattagok:
  - isOnCamera2: logikai változó, amely akkor igaz, ha a kurzor a 2-es panel felett van.

  Metódusok:
  - Update(): folyamatosan figyeli az egér pozícióját.
- Camera3MousePosition: az egér helyzetének síkbéli koordinátáit számolja, ha az egér a 3-as panel felett helyezkedik el.
  Adattagok:
  - X, Z: az egér koordinátái.

  Metódusok:
  - Update(): folyamatosan figyeli és számolja az egér pozícióját.
- MainCameraPosition: a fő kamera pozíciójáért felel (1-es panel).
  Adattagok:
  - cameraDisabled: logikai érték, ha igaz, akkor a kamera mozgatható.
  - cameraDistance: kamera távolsága az XY síktól.
  - cameraXPosition: kamera X koordinátája.
  - cameraYPosition: kamera Y koordinátája.
  - localRotation: a kamera elforgatási vektorát tárolja.
  - moved: logikai érték, amely figyeli, hogy mozgattuk-e a kamerát.

- quaternion: Quaternion típusú (Unity) adattag.

Metódusok:

- LateUpdate(): jobb shift billentyű lenyomásával kapcsolja a kamera mozgathatóságát. Amint a kamera mozgatható, figyeli a billentyűzet és az egér használatát, és annak megfelelően állítja a kamera pozícióját. Az F1–F5 billentyűk lenyomásával a felhasználó beépített nézetek között kapcsolhat, ekkor automatikusan megváltozik a kamera állása.

– MainCameraDraw: az 1-es panelre történő kirajzolást végzi.

Adattagok:

- lineMaterial: a kirajzolt elemek materialja.
- mainColor: a fő koordinátavonalak színe.
- subColor: a mellék koordinátavonalak színe.

Metódusok:

- OnPostRenderer(): az osztály többi függvényét hívja meg, felel azért, hogy a kirajzolandó objektumok folyamatosan megjelenjenek a képernyőn.
- CreateLineMaterial(): elkészíti a materialt, ha hiányzik.
- DrawGrid(): kirajzolja a koordinátarendszert.
- DrawLines(): kirajzolja a már megrajzolt határoló polyline-okat (Utility-ben tárolom: Geometry.boundaryLines).

3.4.2. A modell réteg

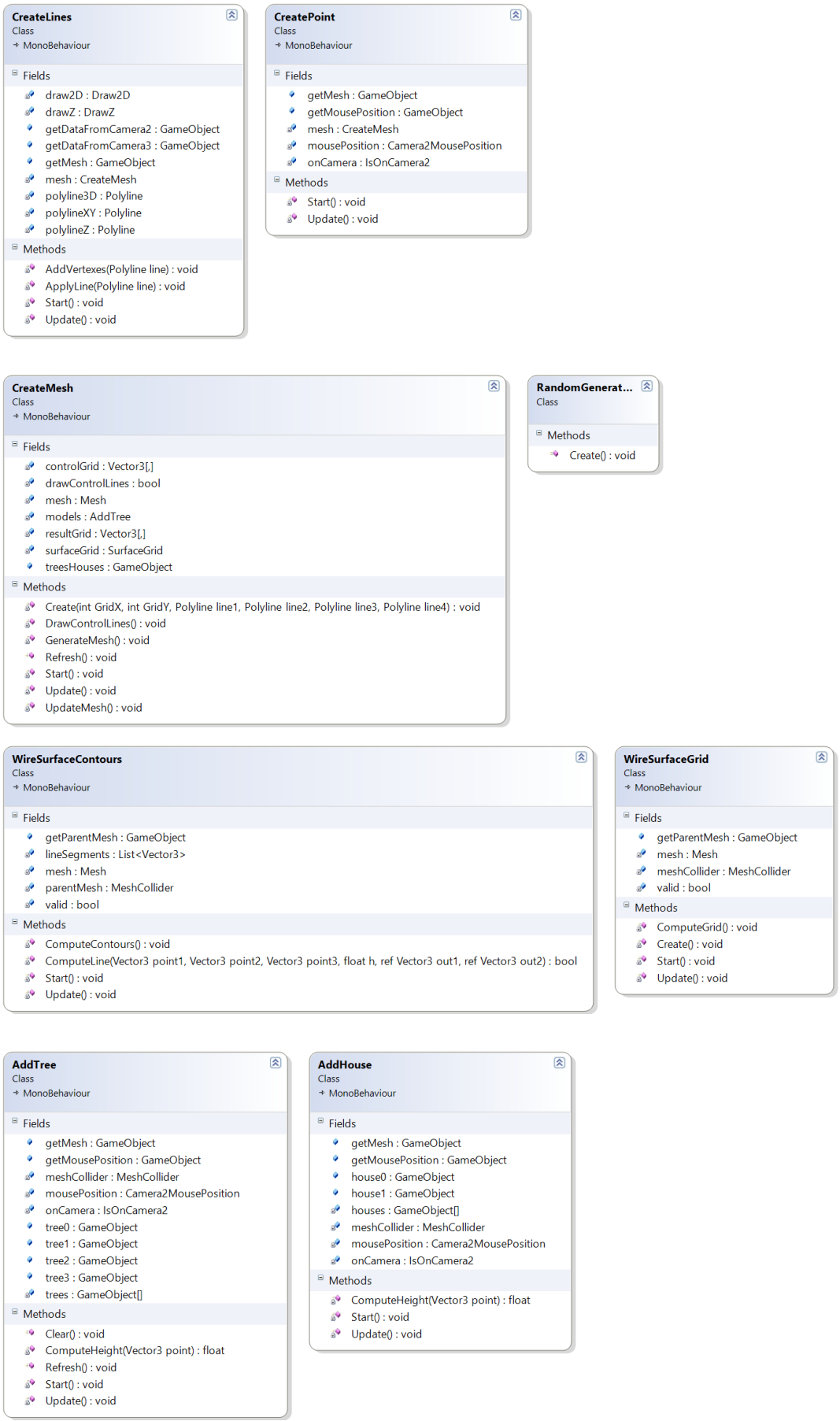
A modell réteg osztályai (3–20. ábra) gondoskodnak az alkalmazásban előforduló objektumok elkészítéséért, módosításáért. A modell réteg folyamatos kapcsolatban áll a kamerák és a utility rétegekkel.

Osztályok:

– CreateLines: polyline-ok elkészítését megvalósító osztály.

Adattagok:

- draw2D, getDataFromCamera2: Camera2 GameObjecttel biztosítja a kapcsolatot.
- drawZ, getDataFromCamera3: Camera3 GameObjecttel biztosítja a kapcsolatot.
- mesh, getMesh: Mesh GameObjecttel biztosítja a kapcsolatot.
- polyline3D: 3D-s polyline, amelyet a paneleken rajzolt polyline-okból készít a program.
- polylineXY: 2-es panelen rajzolt polyline.
- polylineZ: 3-as panelen rajzolt polyline.



3–20. ábra: A modell réteg osztályai

Metódusok:

- Start(): beállítja draw2D adattagot a Camera2 Draw2D, a drawZ adattagot a Camera3 DrawZ osztályra és a mesh adattagot a Mesh objektumra.
- Update(): mindig akkor válik aktívvá, amikor a felhasználó befejezte a szerkesztést a 3-as panelen. A függvény összefésüli a polylineXY-t és a polylineZ-t, és hozzáadja a megfelelő geometriai elemekhez: határoló polyline-okhoz (Geometry.boundaryLines) vagy belső polyline-okhoz (Geometry.lines). A függvény ellenőrzéseket végez (pl.: polyline-ok metszik-e egymást), ha hibát talál, a polyline-t törli. Meghívja az AddVertexes(Polyline line) függvényt, és ha belső polyline-ról van szó, akkor az ApplyLine(Polyline line) függvényt is.
- AddVertexes(Polyline line): az új polyline végpontjain végrehajtja a snap funkciót a beállított toleranciaértékkel. Ellenkező esetben hozzáadja az új polyline végpontjait a Geometry.vertexes, vagy a Geometry.cornerVertexes elemeihez.
- ApplyLine(Polyline line): a függvény módosítja a felületet (mesh-t) a polyline-nal.

– CreatePoint: a felületet módosító pontok hozzáadását megvalósító osztály.

Adattagok:

- mesh, getMesh: Mesh GameObjecttel biztosítja a kapcsolatot.
- mousePosition, onCamera, getMousePosition: Camera2 GameObjecttel biztosítja a kapcsolatot.

Metódusok:

- Start(): beállítja mouesPosition adattagot a Camera2 Camera2MousePosition, az onCamera adattagot a Camera2 IsOnCamera2 osztályra és a mesh adattagot a Mesh objektumra.
- Update(): közvetlenül akkor válik aktívvá, mikor a felhasználó felvett egy pontot az egérrel a 2-es panelen. A függvény ellenőrzi, hogy a pont a felület határain belül helyezkedik-e el, és ha igen, módosítja a felületet (mesh-t), és a pontot hozzáadja a Geometry.tangentPlanes elemeihez.

– CreateMesh: a felület elkészítését megvalósító osztály.

Adattagok:

- controlGrid: a felület kontroll-poligonhálója mátrixban tárolva.
- drawControlLines: logikai változó, amely megmondja, hogy a program kirajzolja-e a felület kontroll-poligonhálóját (nem fut game módban).
- mesh: Mesh típusú adattag.
- resultGrid: a felület pontjait tartalmazó mátrix.
- surfaceGrid: SurfaceGrid típusú adattag.
- models, treesHouses: AddTree GameObjecttel biztosítja a kapcsolatot.

Metódusok:

- Start(): beállítja a models adattagot az AddTree osztályra, a felületet (Mesh-t) elérhetővé teszi a mesh adattagként.
- Update(): az **m** billentyű lenyomására elkészíti, a **d** billentyű lenyomására törli és az **x** billentyű lenyomására alaphelyzetbe állítja a felületet. Ezek szerint hívja meg az osztály többi függvényét.
- Create(int GridX, int GridY, Polyline line1, Polyline line2, Polyline line3, Polyline line4): elkészíti a felületet a határoló polyline-okból (előtte ellenőrzést végez).
- DrawControlLines(): kirajzolja a felület kontroll-poligonhálóját (nem fut game módban).
- GenerateMesh(): a felület kiszámolt pontjaiból elkészíti a Mesh szerkezetét.
- Refresh(): újrakészíti a Mesh-t a resultGrid alapján, a modellek pozícióját frissíti.
- UpdateMesh(): a Mesh hálójának megadja a háromszögeit.

– RandomGenerator: random felületet generál.

Metódusok:

- Create(): a random menüpanel kitöltése után hívódik meg a függvény. A felhasználó által megadott paraméterek felhasználásával elkészíti a határoló polyline-okat, és utána a CreateMesh osztály Update() függvénye válik aktívvá.

– WireSurfaceContours: a szintvonalas nézetet készíti el (módszer leírása: 3.2.5. fejezet).

Adattagok:

- parentMesh, getParentMesh: az eredeti Mesh GameObjecttel biztosítja a kapcsolatot.
- lineSegments: a szintvonalas mesh vonalszegmensei.
- mesh: Mesh típusú adattag (az új szintvonalas mesh).
- valid: logikai változó, amely jelzi, hogy elkészült-e az új mesh.

Metódusok:

- Start(): beállítja a parentMesh adattagot a szülő, a mesh adattagot a szintvonalas Mesh objektumra.
- Update(): amint a felhasználó szintvonalas megjelenítésre vált, a metódus elkészíti a szintvonalas Mesh-t a ComputeContours() függvény meghívásával.
- ComputeContours(): kiszámolja az új mesh vonalszakaszait a szülő mesh háromszögei alapján.
- ComputeLine(Vector3 point1, Vector3 point2, Vector3 point3, float h, ref Vector3 out1, ref Vector3 out2): háromszög metszését vizsgálja egy adott magasságú síkkal.

- WireSurfaceGrid: a grid nézetet készíti el (módszer leírása: 3.2.6. fejezet).
  Adattagok:
  - meshCollider, getParentMesh: az eredeti Mesh GameObjecttel biztosítja a kapcsolatot.
  - mesh: Mesh típusú adattag (az új grid mesh).
  - valid: logikai változó, amely jelzi, hogy elkészült-e az új mesh.

  Metódusok:
  - Start(): beállítja a meshCollider adattagot a szülő, a mesh adattagot a szintvonalas Mesh objektumra.
  - Update(): amint a felhasználó grid megjelenítésre vált, a metódus elkészíti a szintvonalas Mesh-t a Create() majd a ComputeGrid() függvény meghívásával.
  - Create(): ellenőrzi a csomópontok számát.
  - ComputeGrid(): elkészíti a grid mesh-t a 3.2.6. fejezet alapján.
- AddTree: felületre fák felhelyezését megvalósító osztály.
  Adattagok:
  - meshCollider, getMesh: az elkészült Mesh GameObjecttel biztosítja a kapcsolatot.
  - mousePosition, onCamera, getMousePosition: a Camera2 GameObjecttel biztosítja a kapcsolatot.
  - tree0, tree1, tree2, tree3: gameObject típusú adattag.
  - trees: gameObject típusú adattag tömbje.

  Metódusok:
  - Start(): beállítja a meshCollider adattagot a szülő Mesh objektumra, a mousePosition adattagot a Camera2 Camera2MousePosition és az onCamera adattagot a Camera2 IsOnCamera2 osztályára. Eltárolja a tree objektumokat a trees tömbbe.
  - Update(): a megfelelő módban az egér kattintása hatására fát helyez fel a felületre, amit random módon választ ki a trees tömbből.
  - ComputeHeight(Vector3 point): a fa felületre helyezésénél a magassági pozícióját kiszámoló függvény.
  - Clear(): kitörli az összes objektumot.
  - Refresh(): a felület módosításakor a modellek (házak is) pozícióját újraszámolja.
- AddHouse: felületre házak felhelyezését megvalósító osztály.
  Adattagok:
  - meshCollider, getMesh: az elkészült Mesh GameObjecttel biztosítja a kapcsolatot
  - mousePosition, onCamera, getMousePosition: a Camera2 GameObjecttel biztosítja a kapcsolatot.
  - house0, house1: gameObject típusú adattag.
  - houses: gameObject típusú adattagok tömbje.

Metódusok:

- Start(): ugyanúgy működik, mint az AddTree osztály Start() metódusa.
- Update(): ugyanúgy működik, mint az AddTree osztály Update() metódusa.
- ComputeHeight(Vector3 point): ugyanúgy működik, mint az AddTree osztály ComputeHeight () metódusa.

3.4.3. A utility réteg

A utility réteg statikus osztályokat is tartalmaz, először a nem statikus osztályokat mutatom be (3–21. ábra). Az itt található osztályok definiálják azokat az objektumokat és metódusokat, amelyeket az előző két réteg osztályai felhasználnak.

A utility réteg osztályai által elkészült geometriai objektumok a Geometry statikus osztályban létrehozott adattagokban tárolódnak, így a program többi részében szabadon elérhetők.

A utility réteg Utils statikus osztálya tartalmazza a globális adattagokat és függvényeket.

Osztályok:

- Boundary: a felület határát, 2D-s befoglaló poligonját megvalósító osztály.

  Adattagok:

  - polyline1, polyline2, polyline3, polyline4: Polyline típusú adattagok.
  - polygon: a felület 2D-s határpontjai.

  Metódusok:

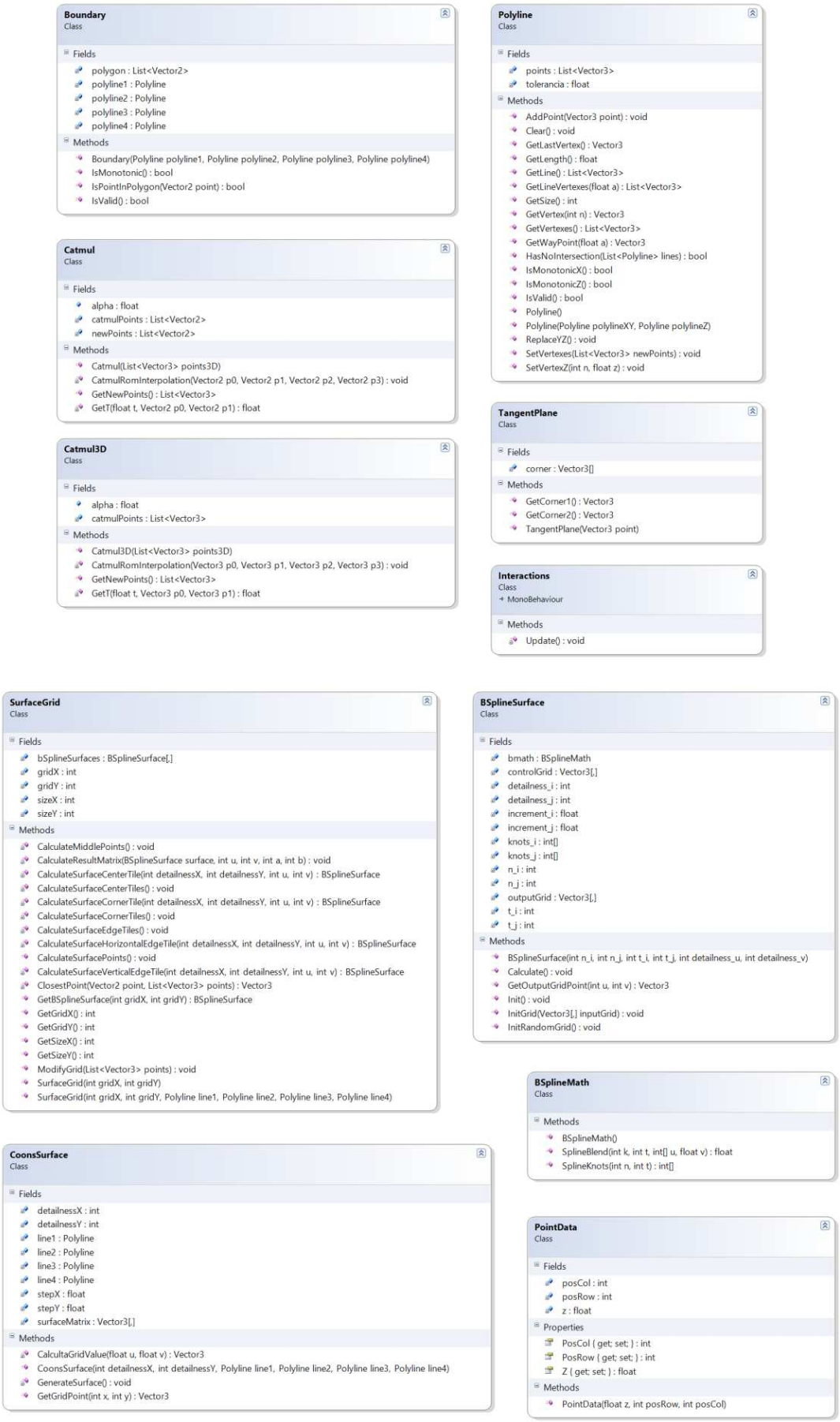
  - Boundary(Polyline polyline1, Polyline polyline2, Polyline polyline3, Polyline polyline4): konstruktor, inicializálja a határoló polyline-okat, kiszámolja a határoló poligont.
  - IsValid(): igazzal tér vissza, ha a polyline-ok megfelelően csatlakoznak.
  - IsMonotonic(): igazzal tér vissza, ha a polyline-ok monotonok.
  - IsPointInPolygon(Vector2 point): igazzal tér vissza, ha a megadott pont a poligonon belül van.
- Catmul és Catmul3D: Catmull–Rom-szplájnt megvalósító osztályok (módszer: 3.2.2. fejezet).

  Adattagok:

  - alpha: a szplájn típusát meghatározó érték.
  - catmulPoints: a szplájn pontjai.

  Metódusok:

  - Catmul3D(List<Vector3> points3D): konstruktor, a kontrollpontokból kiszámolja a szplájnt.



21. ábra: A utilty réteg nem static osztályai

  - CatmulRomInterpolation(Vector3 p0, Vector3 p1, Vector3 p2, Vector3 p3), GetT(float t, Vector3 p0, Vector3 p1): szplájn számításához szükséges segédfüggvények.
  - GetNewPoints(): visszaadja a szplájn pontjait.
- Polyline: polyline-t megvalósító osztály.

  Adattagok:
  - points: a polyline csomópontjainak listája.

  Metódusok:
  - Polyline(): konstruktor, létrehozza a points listát.
  - Polyline(Polyline polylineXY, Polyline polylineZ): konstruktor, a 3.2.1.-es fejezet alapján létrehozza az új polyline-t.
  - IsValid(): igazzal tér vissza, ha a polyline nem metsz másik polyline-t.
  - IsMonotonicX(): igazat ad, ha a polyline x koordinátái monotonok.
  - IsMonotonicZ(): igazat ad, ha a polyline y koordinátái monotonok.
  - HasNoIntersection(List<Polyline> lines): igazzal tér vissza, ha polyline-nak nincs metszete az argumentumban megadott polyline-okkal.
  - ReplaceYZ(): felcseréli az y és z koordinátákat.
  - GetLineVertexes(float a): felosztja a polyline-t az argumentumban kapott „a” osztásközzel és visszaadja az új pontokat.
  - GetWayPoint(float a): az „a” paraméter 0 és 1 közötti érték, kettévágja a polyline-t „a” arányában, és visszaadja az így kapott pontot.
  - AddPoint(Vector3 point): új pontot hozzáad a polyline-hoz.
  - Clear(): törli a polyline csomópontjait.
  - GetLength(): visszaadja a polyline hosszát.
  - GetVertex(int n), GetVertexes(), GetLastVertex(), GetLine(): getterek.
  - SetVertexes(List<Vector3> newPoints), SetVertexZ(int n, float z): setterek.
- TangentPlane: érintő síkot megvalósító osztály.

  Adattagok:
  - corner: a sík sarokpontjai.

  Metódusok:
  - TangentPlane(Vector3 point): konstruktor, kiszámolja a sík sarokpontjait a kapott ponthoz a felhasználó által megadott mérettel.
  - GetCorner1(), GetCorner2(): getterek, visszaadják az adott sarokpontot.

- Interactions: a program vezérlését megvalósító osztály.
  Metódusok:
  - Update(): figyeli a billentyűk lenyomását, és aszerint állítja át a Utility.drawMode értéket. A program különböző részei, osztályai ezt a változót figyelik, és lesznek aktívak, ha a Utility.drawMode megfelelő számra változik.
- SurfaceGrid: a programban megjelenített felületet megvalósító osztály (részletezése: 3.2.3. fejezet).
  Adattagok:
  - gridX, gridY: a felület felosztása B-szplájn felületekre.
  - sizeX, sizeY: a végső felület mátrixának mérete.
  - bSplineSurfaces: a felületet felosztó B-szplájnok.

  Metódusok:
  - SurfaceGrid(int gridX, int gridY): konstruktor, amely a megadott felbontás alapján kiszámolja a felületet tartalmazó mátrixokat.
  - SurfaceGrid(int gridX, int gridY, Polyline line1, Polyline line2, Polyline line3, Polyline line4): konstruktor, amely a megadott határoló polyline-okból elkészíti a Coons-foltot, amelynek rácspontja a felület kontroll-poligonhálójának alapját fogja képezni.
  - CalculateSurfacePoints(): kiszámolja a felület pontjait az adott segédfüggvényekkel.
  - CalculateMiddlePoints(): sűríti a rácspontokat, elkészíti a felület kontroll-poligonhálóját.
  - CalculateSurfaceCornerTiles(), CalculateSurfaceEdgeTiles(), CalculateSurfaceCenterTiles(), CalculateSurfaceCornerTile(), CalculateSurfaceHorizontalEdgeTile(), CalculateSurfaceVerticalEdgeTile(), CalculateSurfaceCenterTile(), CalculateResultMatrix(): kiszámolja az adott típusú B-szplájn-felület pontjait.
  - GetBSplineSurface(), GetSizeX(), GetSizeY(), GetGridX(), GetGridY(): getterek.
  - ModifyGrid(List<Vector3> points), ClosestPoint(Vector2 point, List<Vector3> points): statikus metódusai az osztálynak, amelyek lehetővé teszik a felület külső módosítását pontok és polyline-ok segítségével a 3.2.4. fejezet alapján.
- BSplineMath: BSplineSurface segédfüggvényeit tartalmazó osztály [4].
  Metódusok:
  - SplineKnots(int n, int t), SplineBlend(int k, int t, int[] u, float v): normalizált B-szplájn alapfüggvények és csomópontjaik számítását végzi.
- BSplineSurface: B-szplájn-felületet megvalósító osztály [4].
  Adattagok:
  - n_i, n_j, controlGrid: kontrollpontok adatai.
  - t_i, t_j: az illesztett felület fokai.

  - detailness_i, detailness_j: a felület felbontása.
  - outputGrid: elkészült felület mátrixa.
  - knots_i, knots_j: csomópontok adatai.
  - increment_i, increment_j: osztásközök értékei.
  - bmath: BSplineMath típusú adattag a segédfüggvények meghívásához.

  Metódusok:

  - BSplineSurface(int n_i, int n_j, int t_i, int t_j, int detailness_u, int detailness_v): konstruktor, beállítja a paramétereket, elvégzi az inicializálást (Init()).
  - Init(): kiszámítja a felület kontrollpontjait, az osztásközök és a mátrixok méretét.
  - InitRandomGrid(): random értékekkel tölti fel a kontrollpontok magassági értékeit.
  - InitGrid(Vector3[,] inputGrid): feltölti a kontrollpontok magassági értékeit.
  - Calculate(): kiszámolja a B-szplájn-felület pontjait.
  - GetOutputGridPoint(int u, int v): a felület adott pontját adja vissza.
- CoonsSurface: Coons-foltot megvalósító osztály (részletezése a 3.2.3. fejezetben).

  Adattagok:

  - detailnessX, detailnessY, stepX, stepY: a felület felbontása, az osztásközök mérete.
  - line1, line2, line3, line4: a határoló polyline-ok.
  - surfaceMatrix: a felület pontjainak mátrixa.

  Metódusok:

  - CoonsSurface(int detailnessX, int detailnessY, Polyline line1, Polyline line2, Polyline line3, Polyline line4): konstruktor, elvégzi az inicializálást és kiszámolja a felület pontjait (GenerateSurface()).
  - GenerateSurface(), CalcultaGridValue(float u, float v): a Coons-folt kiszámításának metódusai.
  - GetGridPoint(int x, int y): a felület adott pontját adja vissza.
- PointData: a felület pontjainak a magassági értékét és a felület mátrixában elfoglalt helyét tároló osztály. A felület pontjainak XY síkbéli kereséséhez létrehozott szótár adatstruktúrához (Geometry.mainPoints) használtam fel, amely lehetővé teszi, hogy a kiválasztott pont adatai az XY koordinátái alapján elérhetők legyenek.

  Adattagok:

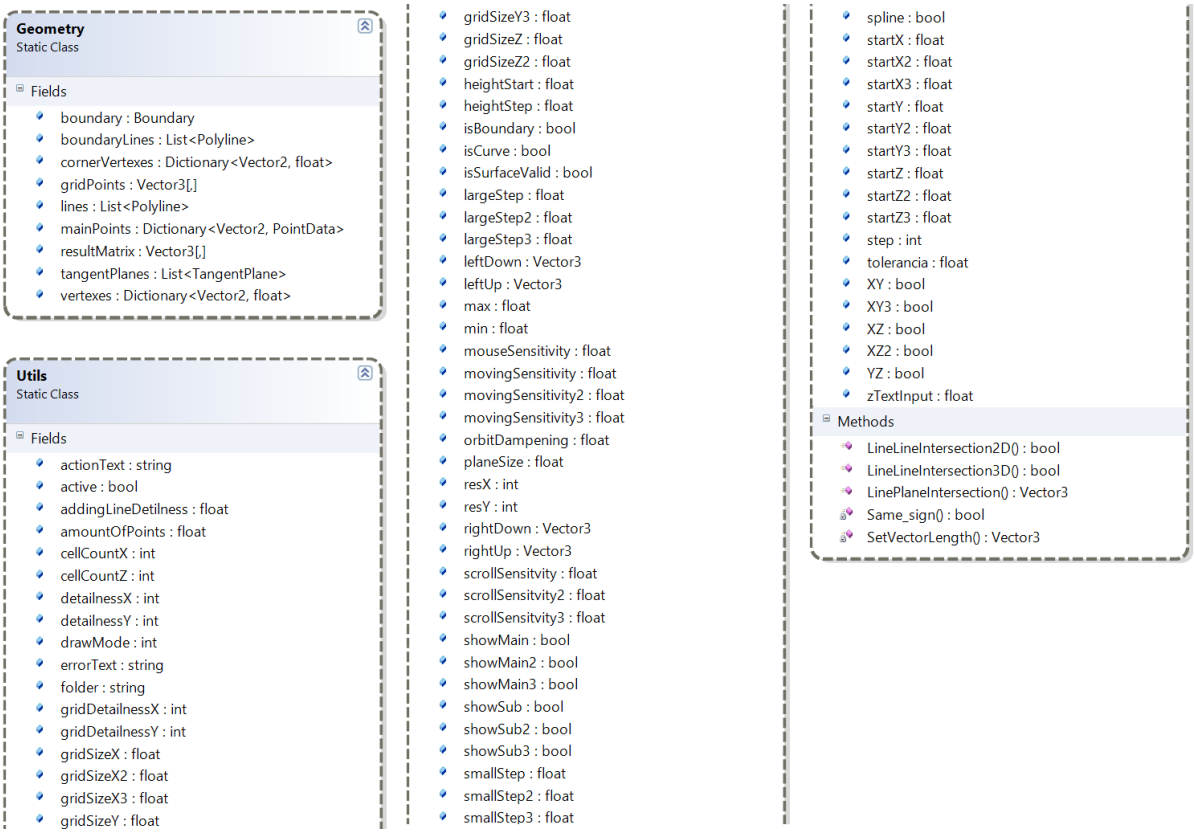
  - posRow, posCol: pontnak a felület mátrixában elfoglalt helye.
  - z: a pont magassági értéke.

  Metódusok:

  - PointData(float z, int posRow, int posCol): konstruktor.
  - PosCol, PosRow, Z: az adattagok elérésére szolgáló tulajdonságok.

Statikus osztályok (3–22. ábra):

– Geometry: a program geometriai eleminek tárolását megvalósító osztály.
  Adattagok:
  – boundary: a felület határoló poligonját tárolja.
  – boundaryLines: a felület határoló polyline-jait tárolja.
  – lines: a felületet módosító belső vonalakat tárolja.
  – tangentPlanes: a felületet módosító pontokat (érintő síkokat) tárolja.
  – gridPoints: a felület kontroll-poligonhálóját tárolja.
  – resultMatrix: a felület kiszámolt pontjait tároló mátrix.
  – mainPoints: egy szótár adatszerkezet, amely a gridPoints síkbéli koordinátáit és a ponthoz tartozó adatokat tartalmazza, egyszerűvé teszi egy adott ponthoz a legközelebbi rácspont megkeresését.
  – cornerVertexes, vertexes: a határoló polyline-ok és a belső polyline-ok végpontjait tárolja elősegítve a snap funkciót.
– Utils: a globális adattagok és függvények tárolását megvalósító osztály.
  Adattagok:
  – drawMode: a program vezérléséért felelős változó.
  – errorText, actionText, zTextInput, planeSize: a paneleken lévő üzenetekért és input mezőkért felelnek.
  – isBoundary: logikai érték, igaz, ha a felvett polyline határoló polyline, különben hamis.
  – isCurve: logikai érték, ha igaz, akkor Catmull–Rom-szplájn illesztés történik a polyine pontjaira.
  – isSurfaceValid: logikai érték, igaz, ha elkészült az alapfelület.
  – active: logikai változó, ha a felhasználó menüelemre kattint, inaktívvá válik a szerkesztői felület.
  – resX, resY, folder: a mentés menüpont változói.
  – showMain, showSub, XY, YZ, XZ, gridSizeX, gridSizeY, gridSizeZ, smallStep, largeStep, startX, startY, startZ, movingSensitivity, mouseSensitivity: a fő kamera koordinátarendszerének változói, mozgathatóságának paraméterei.
  – showMain2, showSub2, XZ2, gridSizeX2, gridSizeZ2, smallStep2, largeStep2, startX2, startY2, startZ2, movingSensitivity2, mouseSensitivity2: a 2-es kamera koordináta-rendszerének változói, mozgathatóságának paraméterei.
  – showMain3, showSub3, XY3, gridSizeX3, gridSizeY3, smallStep3, largeStep3, startX3, startY3, startZ3, movingSensitivity3, mouseSensitivity3: a 3-as kamera koordinátarendszerének változói, mozgathatóságának paraméterei.



3–22. ábra: A utility réteg statikus osztályai

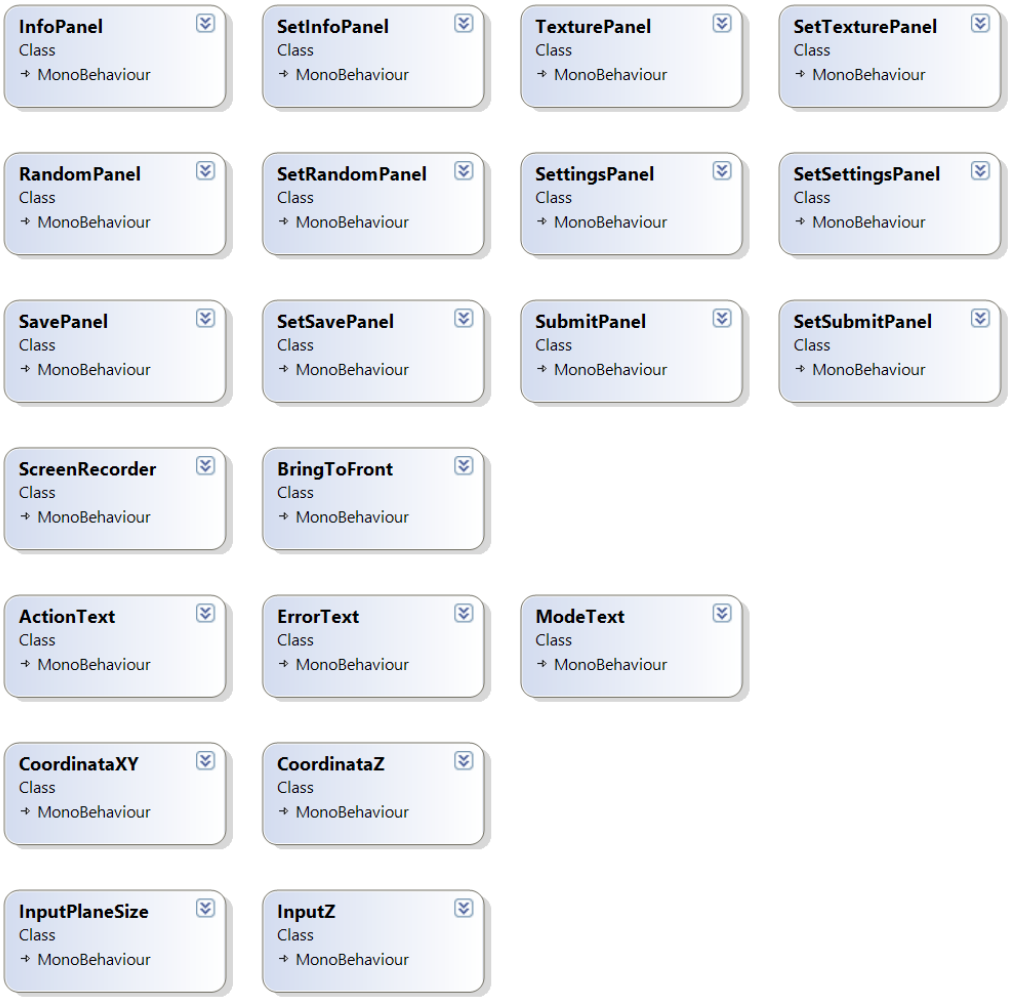
– gridDetailnessX, gridDetailnessY, detailnessX, detailnessY: a felület felosztásának paraméterei.
– amountOfPoints: a Catmull–Rom-szplájn felbontása.
– addingLineDetilness: a felületet módosító belső vonalak felbontása.
– tolerancia: a snap funkció tolerancia értéke.
– heightStart, heightStep, cellCountX, cellCountX: a nézetek paraméterei.
– leftDown, rightDown, leftUp, rightUp, min, max, step, spline: a random generálás paraméterei.

Metódusok:

– LinePlaneIntersection(Vector3 linePoint, Vector3 lineVec, Vector3 planeNormal, Vector3 planePoint), SetVectorLength(Vector3 vector, float size) [5]: vonal és felület metszéspontját adja vissza.
– LineLineIntersection3D(Vector3 linePointA1, Vector3 linePointA2, Vector3 linePointB1, Vector3 linePointB2), LineLineIntersection2D(Vector3 lineA1, Vector3 lineA2, Vector3 lineB1, Vector3 lineB2) [5]: igazzal tér vissza, ha a két vonal metszi egymást.
– Same_sign(float a, float b): igazzal tér vissza, ha a két szám előjele azonos.

3.4.4. A panel réteg

A panelek rétegének osztályai (3–23. ábra) a felhasználóval való kapcsolatot tartják fent. A program működésében csak a paraméterek megadását, a menüelemek vezérlését végzik, ezért az osztályok leírását nem részletezem.



3–23. ábra: A panel réteg osztályai

3.5. Tesztelési terv

A tesztelés során fekete- és fehérdoboz-tesztelést végeztem:

- Feketedoboz-tesztelés: a program felhasználó általi vezérlését teszteltem a Felhasználói esetek diagramja alapján (2–11. ábra).
- Fehérdoboz-tesztelés: a felhasználó által megadható paraméterekre vizsgáltam a program futását.
- Tesztkörnyezet:
  - operációs rendszer: Windows 10 Pro 64-bit;

- videókártya: AMD Radeon HD 7800 Series;
- processzor: Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz, 3301 Mhz., 4 mag, 4 logikai processzor;
- memória: 16,0 GB.

3.5.1. Feketedoboz-tesztelés

A következő eseteket vizsgáltam:

- a program megfelelően elvégzi a billentyűk lenyomására az adott feladatokat;
- a shift, alt billentyűk lenyomására a kamerák pozíciója szabadon állítható;
- a program az egér használatára mindenhol helyesen reagál;
- a menüpontok kiválasztásánál a panelek inaktívvá válnak, nem reagálnak felhasználói utasításokra, az adott menüpont panelje válik aktívvá;
- a mentés, random generálás, a menüpontok funkciói helyesen működnek;
- létrejönnek a megfelelő objektumok, ha a felhasználó a programot rendeltetésszerűen használja;
- polyline-ok rajzolása helyesen működik, a 3-as panel a megfelelő pillanatban válik aktívvá;
- a felület objektum csak akkor jön létre, ha a feltételek helyesek, nincs még kész felület;
- a felületet módosító elemek és a felületre elhelyezhető objektumok csak akkor hozhatók létre, ha már létezik felület;
- az elemek törölhetők, a törlési funkciók jól működnek;
- a felhasználó hibaüzenet kap, amint helytelen művelet végez;
- teszteltem, hogy a felhasználó megkapja-e a felhasználói dokumentációban leírt segítő, tájékoztató és hibaüzeneteket.

3.5.2. Fehérdoboz-tesztelés

Minden paraméterbeállítási lehetőségnél megvizsgáltam, hogy a program helyesen működik-e. A paramétereknek megadtam egy minimum és egy maximum értéket, ha a felhasználó kisebb vagy nagyobb számot ír be ezeknél, akkor automatikusan a minimum vagy a maximum érték válik megadottá (3–2. táblázat). Mindenhol teszteltem a szélsőértékek helyes működését. Mivel a Unity-ben egy Mesh objektum csomópontjainak száma korlátos, így ennek megfelelően engedi a program a felület felbontásának megválasztását (ha átlépi az új Mesh a 65000 csomópontot, akkor a Utils.detailnessX és Utils.detailnessY 1-re állítódik).

3–1. táblázat: Változók minimum és maximum értékei

Változó	**min**	**max**	**Változó**	**min**	**max**
1-es panel kameráinak paraméterei			3-es panel kameráinak paraméterei		
Utils.gridSizeX	1f	200f	Utils.gridSizeX3	1f	200f
Utils.gridSizeY	1f	200f	Utils.gridSizeY3	1f	200f
Utils.gridSizeZ	1f	200f	Utils.smallStep3	0.1f	100f
Utils.smallStep	0.1f	100f	Utils.largeStep3	0.1f	100f
Utils.largeStep	0.1f	100f	Utils.startX3	-100f	100f
Utils.startX	-100f	100f	Utils.startY3	-100f	100f
Utils.startY	-100f	100f	Utils.startZ3	-100f	100f
Utils.startZ	-100f	100f	Utils.movingSensitivity3	0.1f	50f
Utils.movingSensitivity	0.1f	50f	Utils.scrollSensitvity3	0.1f	50f
Utils.mouseSensitivity	0.1f	50f			
Utils.scrollSensitvity	0.1f	50f	Felületek felbontása		
Utils.orbitDampening	0.1f	50f	Utils.gridDetailnessX	2	120
			Utils.gridDetailnessY	2	120
2-es panel kameráinak paraméterei			Utils.detailnessX	1	120
Utils.gridSizeX2	1f	200f	Utils.detailnessY	1	120
Utils.gridSizeZ2	1f	200f			
Utils.smallStep2	0.1f	100f	Polyline-ok felbontása		
Utils.largeStep2	0.1f	100f	Utils.amountOfPoints	2	100
Utils.startX2	-100f	100f	Utils.addingLineDetilness	0.01f	10f
Utils.startY2	-100f	100f	Sznap toleranciája		
Utils.startZ2	-100f	100f	Utils.tolerancia	0.1f	10f
Utils.movingSensitivity2	0.1f	50f			
Utils.scrollSensitvity2	0.1f	50f	Nézetek (szintvonalas és grid) paraméterei		
			Utils.heightStart	-100f	100f
Elemek kiterjedése és magassága			Utils.heightStep	0.05f	100f
Utils.planeSize	0.1f	100f	Utils.cellCountX	2	1000
Utils.zTextInput	-100f	100f	Utils.cellCountZ	2	1000

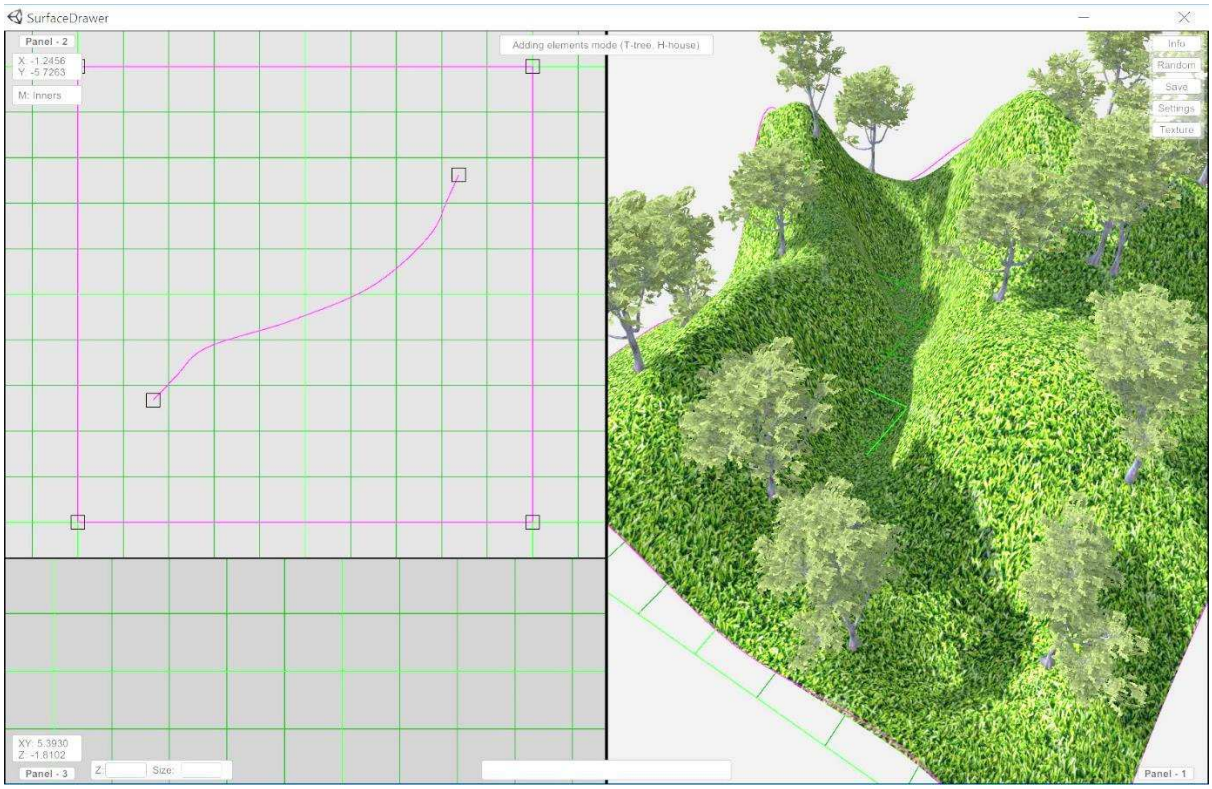
4. Összefoglalás

Az elkészült program teljesíti a kitűzött feladat specifikációját. A számítógépes grafika eszközeivel létrehoztam egy 3D-s felületet megjelenítő és szerkesztő alkalmazást, amiben a felhasználó gyorsan készíthet egy ábraszintű tömbszelvényt, miközben mindvégig állíthatja az egyes kamerák pozícióját.

A program írása során vizsgáltam és implementáltam fontosabb 3D-s felületek (Coons-folt, B-szplájn-felület) matematikai leírását. Sikerült elérnem, hogy az elkészült felületet a felhasználó interaktív módon alakíthassa térbeli pontok és polyline-ok beszúrásával, és modelleket (fákat, házakat) helyezhessen el a felszínre.

A fejlesztés során megismerkedtem a C# programozási nyelv és a Unity fejlesztőkörnyezet alapjaival.

Befejezésül megköszönöm konzulensemnek, Baráth Dánielnek, jó tanácsait és a témavezetés elvállalását, Koczó Fanninak, hogy nyelvi szempontból átnézte dolgozatomat, továbbá köszönöm az egyetemi oktatóimnak, hogy a késői órákban is lelkiismeretesen megtartották a tanórákat.



5. Irodalomjegyzék

[1] Juhász Imre: Görbék és felületek modellezése, Kelet-Magyarországi Informatikai Tananyag Tárház, 2001–2004
https://www.tankonyvtar.hu/hu/tartalom/tamop425/0046_gorbek_es_feluletek_modellezese/ch08.html

[2] Catmull–Rom-szplájn interpoláció
https://en.wikipedia.org/wiki/Centripetal_Catmull%E2%80%93Rom_spline

[3] Kovács Emőd: Komputergrafika – Matematikai alapok, Eszterházy Károly Főiskola, Matematikai és Infromatikai Intézet, 2011
http://aries.ektf.hu/~hz/pdf-tamop/pdf-01/html/

[4] B-Szplájn-felületek forráskódja
http://paulbourke.net/geometry/spline/
https://forum.unity.com/threads/b-spline-surface.68832/

[5] 3D Math funtions (C#)
http://wiki.unity3d.com/index.php/3d_Math_functions

[6] Dr. Szirmay-Kalos László: Számítógépes grafika, Computerbooks, 2000
http://sirkan.iit.bme.hu/~szirmay/grafika/graf.pdf

[7] Antal György–Csonka Ferenc–Dr. Szirmay-Kalos László: Háromdimenziós grafika, animáció és játékfejlesztés, Computerbooks, 2003
http://sirkan.iit.bme.hu/~szirmay/3Dgraf.pdf

[8] Eric Haines: Object2Terrain (Mesh–Terrain átalakító programkód)
http://wiki.unity3d.com/index.php/Object2Terrain

[9] UCLA Game Lab: UCLA Wireframe Shader (drótvázas shader)
https://assetstore.unity.com/packages/vfx/shaders/directx-11/ucla-wireframe-shader-21897

[10] Házak modelljei
https://assetstore.unity.com/packages/3d/environments/low-poly-buildings-lite-98836

[11] Fák modelljei
https://assetstore.unity.com/packages/3d/vegetation/trees/realistic-tree-9-rainbow-tree-54622

[12] Kamerák mozgatásának forráskódja
https://pastebin.com/raw/2RX8fpJ3

[13] Mentés fájlba opció forráskódja
https://answers.unity.com/questions/22954/how-to-save-a-picture-take-screenshot-from-a-camer.html

A honlapok ellenőrzésének dátuma: 2018.05.05.