**UNIVERSITATEA DIN BUCUREȘTI**

**FACULTATEA**

**DE**

**MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de licență**

**DRUMURI DE LUNGIME MINIMĂ PE SUPRAFETE TRIANGULATE**

**Absolvent**

**Duncea Vlad-Alexandru**

**Coordonator științific**

**Lect. Dr. Stupariu Mihai-Sorin**

**București, iunie 2021**

**Rezumat**

**Va fi redactat în limbile română și engleză și, în aproximativ o pagină, va prezenta**

**rezumativ și explicit obiectivele și relevanța temei lucrării de finalizare, metodele**

**utilizate și rezultatele obținute și implicațiile asupra domeniului ales;**

# Cuprins

Cuprins 2

Introducere 2

I. Tehnologii utilizate 2

1. Blender 2

2. Unity 2

II. Algoritmi bazați pe grafuri 2

1. Notații și definiții 2

2. Dijkstra 2

3. Dijkstra Bidirecțional 2

4. A\* 2

5. Rafinare iterativa 2

III. Algoritmi pe fețe 2

IV. Utilizarea aplicației 2

V. Concluzii 2

Bibliografie 2

# Introducere

**Problema găsirii celui mai scurt drum pe o suprafață este .** Problema calculării distanței minime între două puncte pe o suprafață triangulată este trivială daca știm cel mai scurt drum. În lucrare voi considera drumurile și distanțele pe suprafețe discrete, triangulate, în spațiul tridimensional. Cel mai scurt drum dintre două puncte pe o suprafață se numește drum geodezic, iar cea mai scurtă distanță, distanța geodezică.

După cum putem vedea în (Bose, et al., 2011) problema găsirii celui mai scurt drum pe o suprafață are multe aplicabilități, precum: proiectarea circuitelor, sisteme informatice geografice, tratamente radiologice în medicină sau grafica pe calculator. Exemple de utilizări practice sunt: analiza curgerii apei, controlarea unui braț robotic pentru a ocoli diverse obstacole în spațiul de lucru sau studiul și optimizarea traficului. O altă problemă interesantă este găsirea unui drum care să respecte anumite condiții din punctul de vedere al altitudinii, mai exact: Putem găsi un drum intre două puncte care să rămână sub o anumită altitudine ? sau: Daca avem două puncte, care este drumul care minimizează schimbările de altitudine? (de Berg & van Kreveld, 1997).

În lucrare am prezentat două abordări pentru a găsi drumul și distanța geodezică: Algoritmi bazați pe teoria grafurilor și Algoritmi bazați pe desfășurarea suprafeței într-un plan 2D.

**De introdus rezumat pe capitole(structura).**

**tipul lucrării şi subdomeniul specific în care se încadrează tema,**

**■ prezentarea generală a temei,**

**■ scopul şi motivația alegerii temei,**

**■ contribuţia proprie în realizarea lucrării (pe scurt),**

**■ structura lucrării, cu descrierea succintă a fiecărui capitol,**

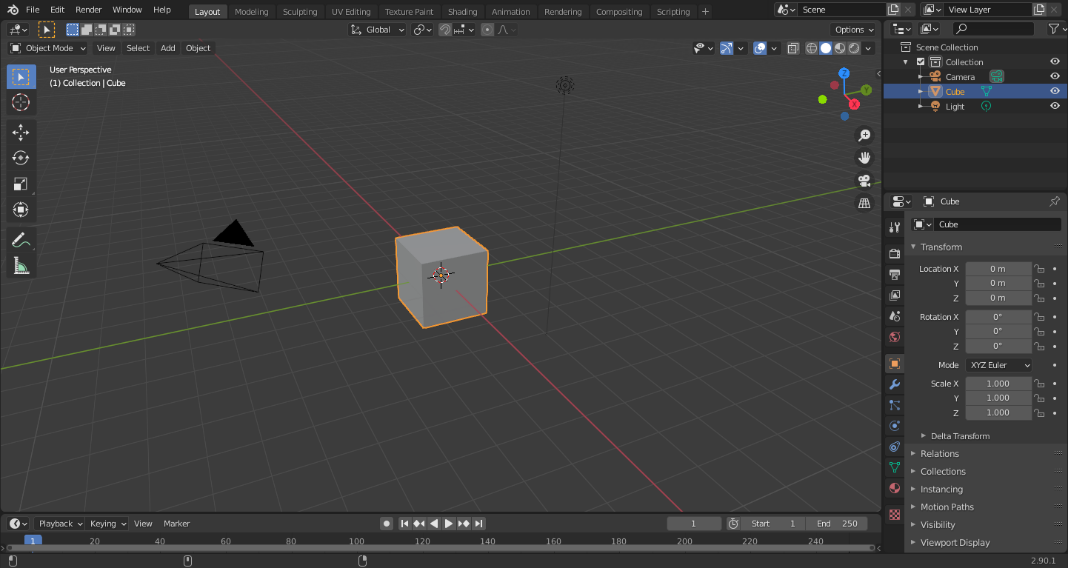
**■ câteva repere istorice relativ la temă şi rezultate cunoscute (starea actuală a**

**domeniului eventual).**

# Tehnologii utilizate

## Blender

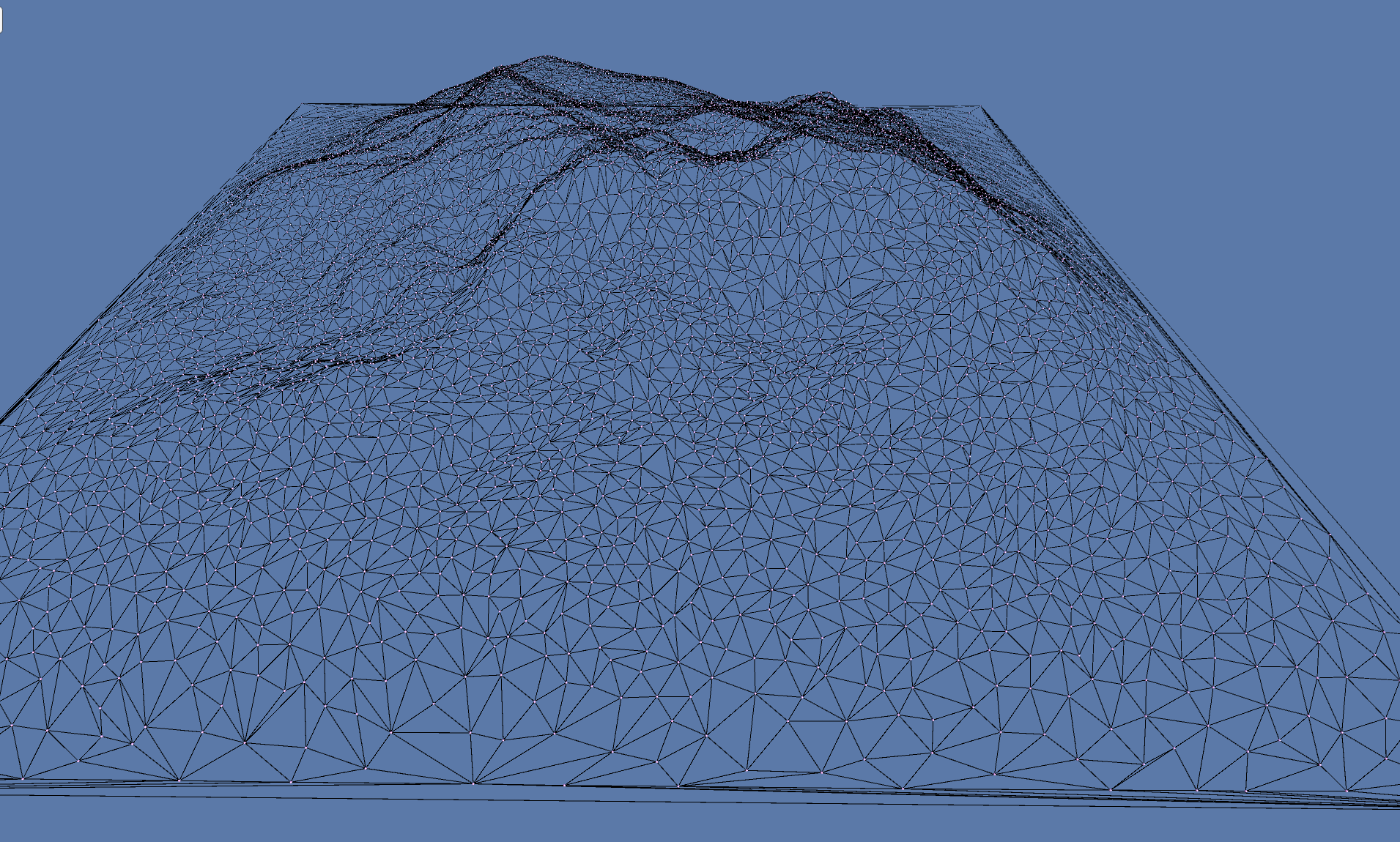
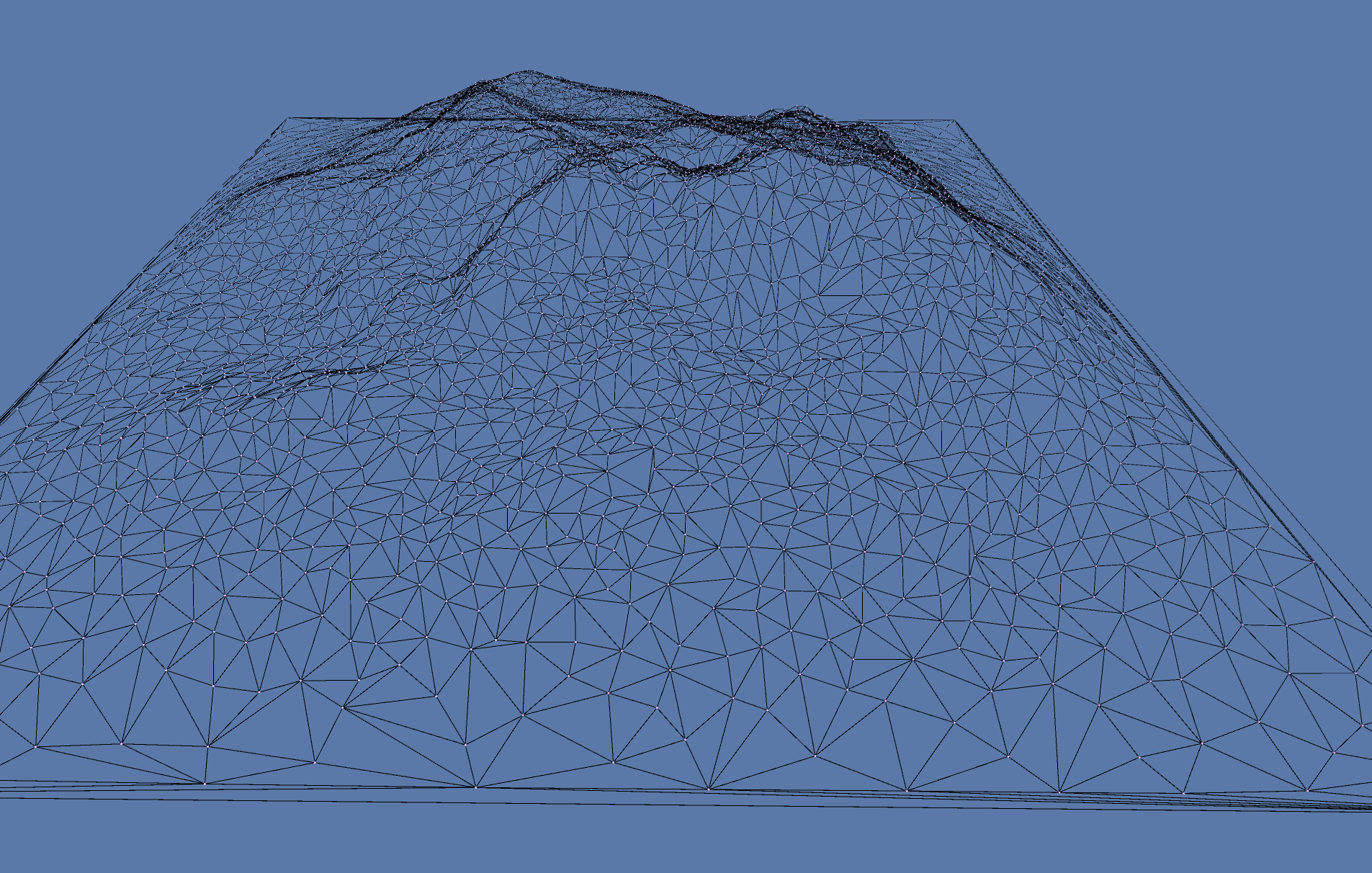
Blender este o aplicație gratuită, cu sursă deschisă, de creare a obiectelor 3D. Conține unelte pentru toate etapele de dezvoltare a produselor 3D: modelare, animație, simulare, redare și compoziție. În ultimele versiuni au fost adăugate și funcționalități de editare a videoclipurilor și creare a jocurilor (Blender Foundation, 2021).



Figură I.1 Interfață Blender (din manual)

Am utilizat aplicația Blender pentru a construi sau modifica obiectele 3D pe care le-am exportat în formatul *.obj* (Murray & Van Ryper, 1996) pentru a le folosi în aplicația proprie. Asupra obiectelor 3D de bază am aplicat diverse modificări precum decimarea, pentru a varia numărul de puncte și muchii ale diferitelor obiecte/suprafețe, triangularea, pentru a pregăti obiectele pentru utilizarea în aplicația proprie și diferite translatări și scalări pentru a ușura vizualizarea și pentru a fi mai intuitiv prezentate.

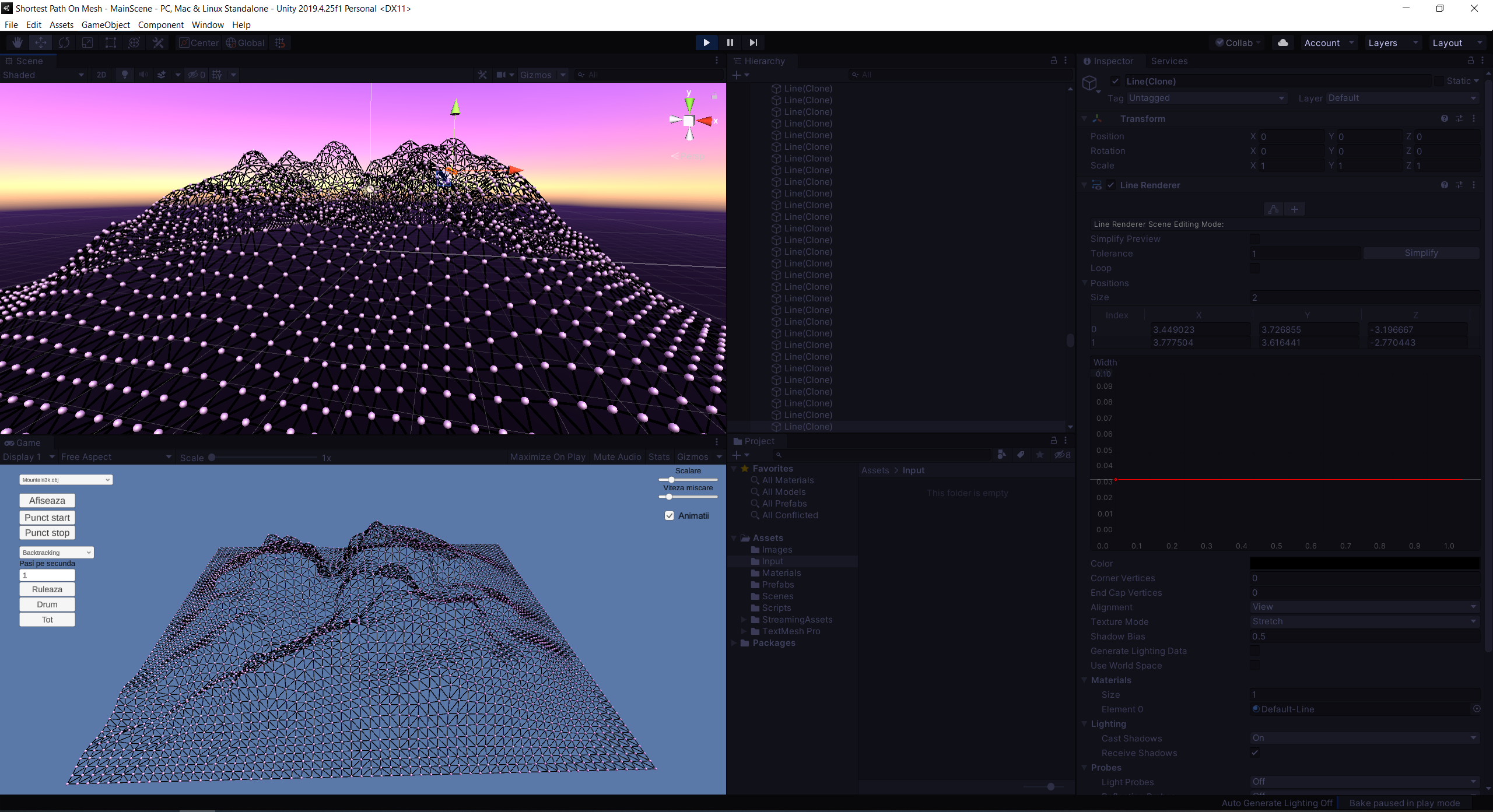
Un exemplu concret de utilizare este fișierul din aplicație denumit *MunteDoi6k.obj*. Obiectul 3D inițial avea 66 de mii de puncte și 200 de mii de muchii. Asupra acestuia am aplicat modificatorul *Decimate* din Blender cu rația 0.1, adică se vor păstra din punctele inițiale alese astfel încât să se păstreze cât mai mult din detaliile obiectului. După am aplicat *Triangulate Faces* pentru a triangula suprafața și am exportat ca fișier *.obj* . Prin acest procedeu am generat mai multe obiecte pentru a testa algoritmii pe diferite grade de complexitate dimensională.



Figură I.2 Comparație între aceeași suprafață cu 3 mii de puncte (stânga), respectiv 6 mii (dreapta).

## Unity

Unity este o platformă pentru dezvoltarea proiectelor 3D și 2D cu redare în timp real precum jocuri, filme, animații sau chiar augmentarea realității. Aplicația prezentată în licență a fost dezvoltată în Unity folosind limbajul de programare C#. Am ales să folosesc această platformă pentru că oferă o multitudine de librării pentru partea vizuală a aplicației, putând astfel să mă concentrez pe dezvoltarea algoritmilor, de asemenea, testarea codului este facilitată de integrarea cu Visual Studio, iar modificarea diferiților parametrii poate fi făcută în timpul rulării programului. Un alt beneficiu este utilizarea gratuita chiar și pentru aplicații comerciale, cât timp nu se depășește o anumită sumă, deci utilizarea în scopul dezvoltarea lucrării de licență este gratuită.



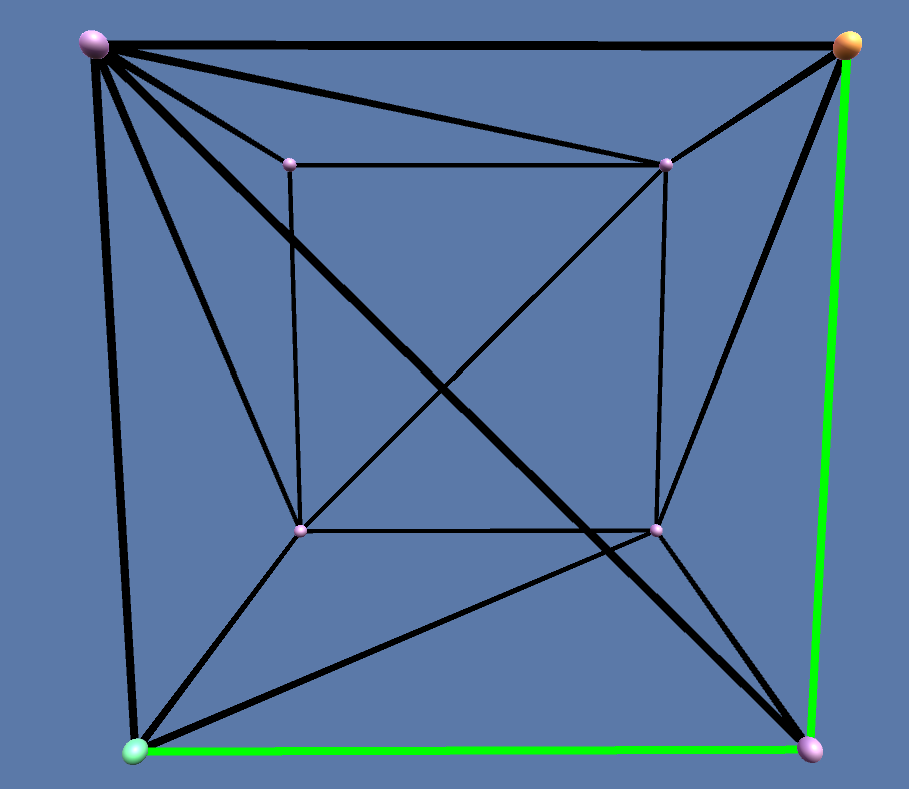
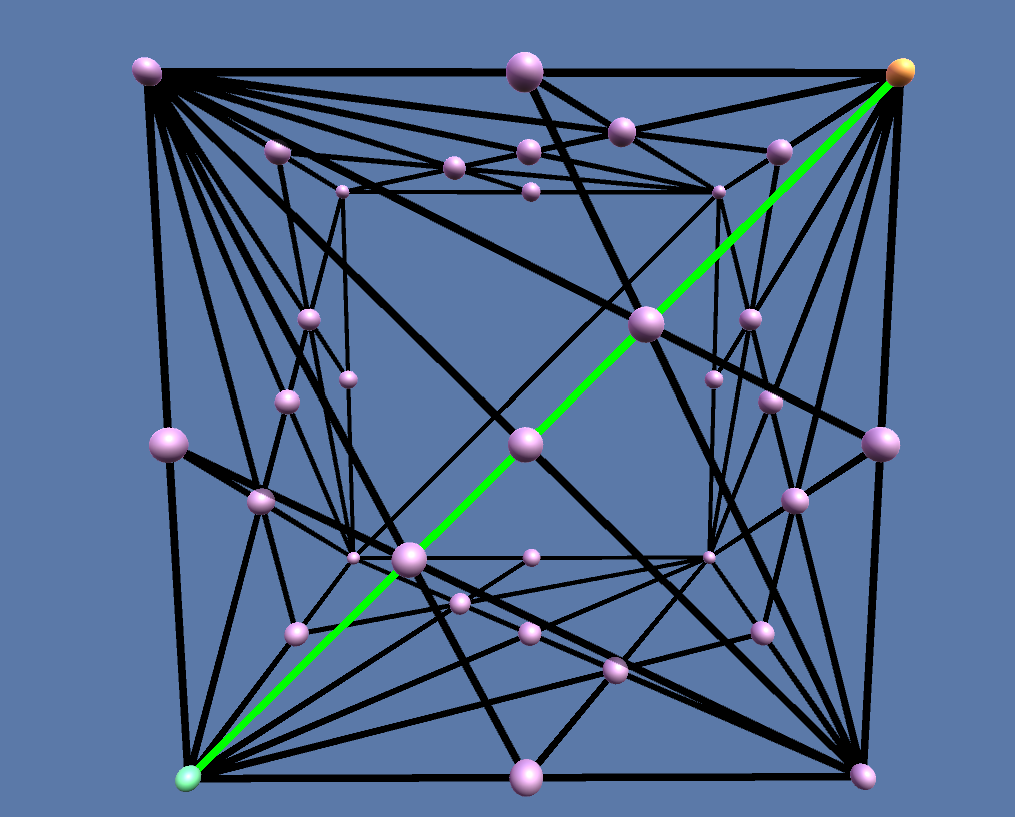
Figură I.3 Interfața aplicației Unity în timpul rulării.

# Algoritmi bazați pe grafuri

În acest capitol voi prezenta aplicarea algoritmilor din teoria grafurilor pentru rezolvarea problemei celui mai scurt drum. Voi prezenta patru algoritm în ordinea complexității implementării acestora.

O suprafață triangulată poate fi transformată cu ușurință într-un graf direcțional prin transformarea punctelor și a laturilor de pe suprafață în nodurile, respectiv muchiile grafului. Costul parcurgerii unei muchii va fi distanța Euclidiană dintre coordonatele punctelor pe care le unește. Pe acest graf vom aplica diverși algoritmi bine studiați din teoria grafurilor pentru a obține o aproximare a distanței geodezice, .

Acești algoritmi calculează cel mai scurt drum mergând doar pe muchiile suprafeței. Din acest motiv triangularea inițială poate avea un impact asupra rezultatului obținut, în cazuri triviale, precum o suprafață plană triangulată convenabil, putem obține chiar distanța si drumul geodezic, pe cazul general, însă, vom obține doar o aproximare. In cazul algoritmilor care folosesc rafinarea iterativa, impact unei triangulări nefavorabile poate fi minimizat prin introducerea de noi muchii așa cum se poate observa în (Figură II.1).

Figură II.1 – Comparație drumuri. **Stânga**: cel mai scurt drum pe obiectul inițial, **Dreapta**: cel mai scurt drum după aplicarea unui pas de rafinare iterativa

Primii trei algoritmi pe care îi voi prezenta pot fi aplicați pe orice suprafață, nu neapărat una triangulată. Algoritmul de rafinare iterativă necesită ca suprafața să fie triangulată în prealabil pentru a putea aplica reguli de divizare ale fețelor acesteia.

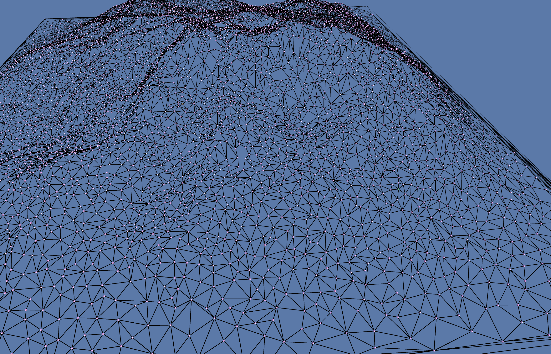
## Notații și definiții

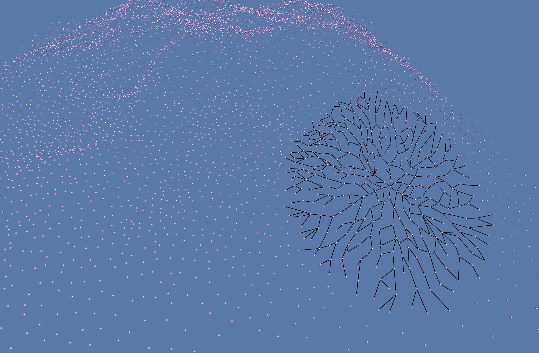
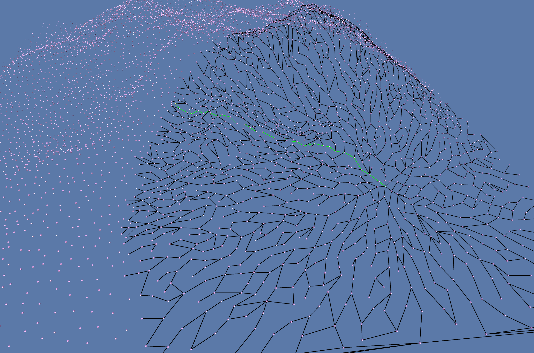
Voi nota graful direcțional cu , unde *V* este mulțimea vârfurilor și *M* mulțimea muchiilor cu . Numărul vârfurilor va fi notat cu și numărul muchiilor va fi notat cu . Graful invers îl voi nota cu , unde este mulțimea muchiilor inverse, adică . Un drum *D* este o secvență de vârfuri cu proprietatea că . Costul unei muchii este . Costul unui drum *D* cu *n* vârfuri este suma muchiilor drumului: . Cel mai scrut drum dintre două vârfuri *start*, *scop* este D, dacă . Voi nota cu costul celui mai scrut drum dintre .

## Dijkstra

Algoritmul lui Dijkstra rezolva problema „single source shortest path” (SSSP) pe un graf orientat cu muchii cu ponderi pozitive (Dijkstra, 1959). Algoritmul se bazează pe construcția unei liste de noduri a căror distanță de la nodul de start este cunoscută. Pornind de la nodul de start, cu distanța 0, încercăm la fiecare pas să îmbunătățim distanțele cunoscute. Observăm că acest algoritm va prelucra fiecare nod o singură dată, adică odată extras din lista de noduri, distanța și drumul de la nodul de start la acesta sunt cunoscute, deci putem să oprim algoritmul fie când extragem nodul dorit, fie îl lăsăm să ruleze până extrage toate nodurile pentru a afla distanțele de la nodul de start la toate celelalte noduri.

Algoritmul este ușor de implementat, utilizând un min-heap pentru a sorta distanțele aproximate și are o complexitate timp de O(mlogn) unde m este numarul muchiilor și n este numarul nodurilor. Având în vedere extragerea nodurilor după distanța de la nodul de start, dacă afișăm nodurile calculate la diferite momente de timp, vom observa o avansare in formă de sferă.





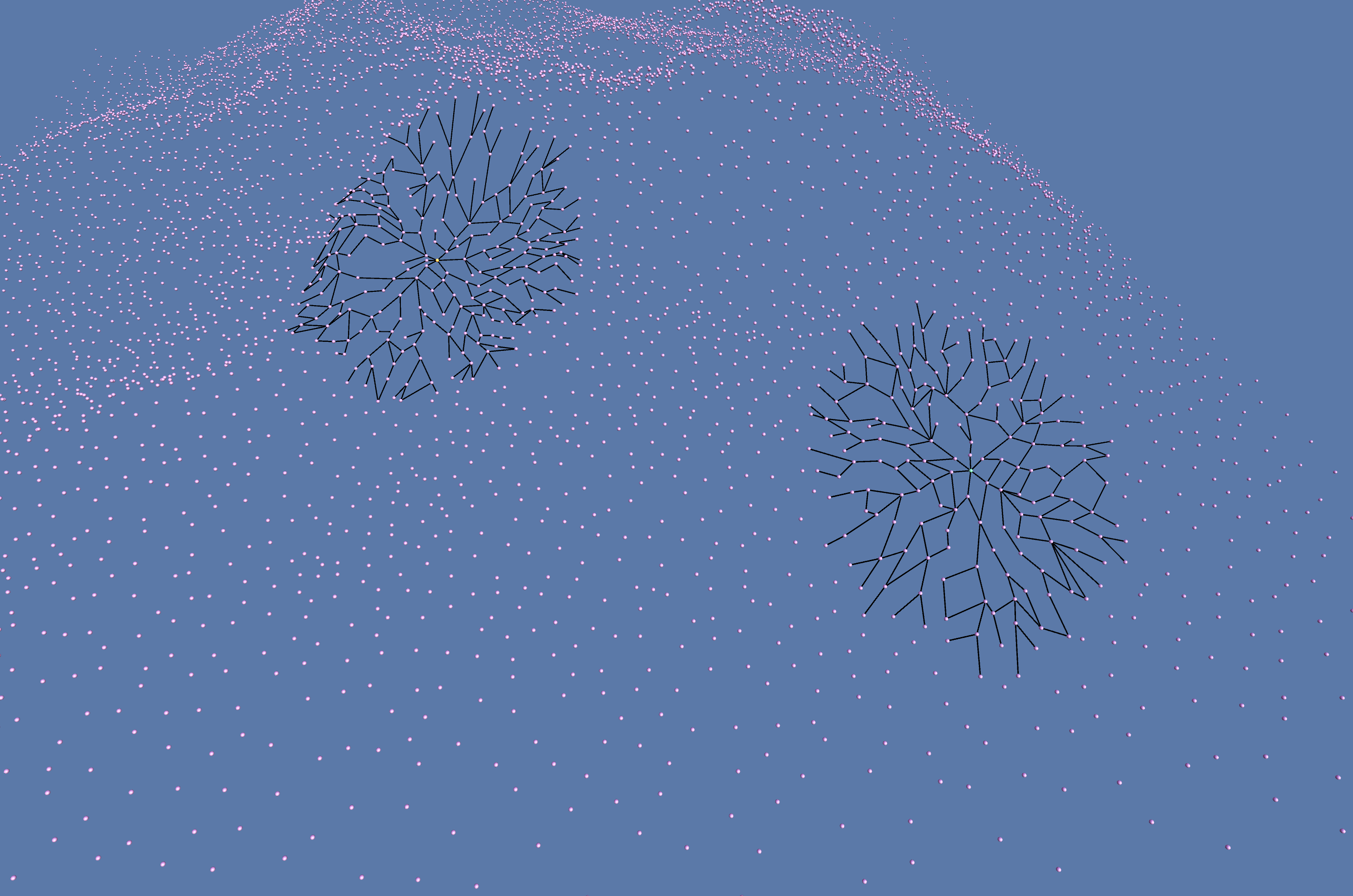
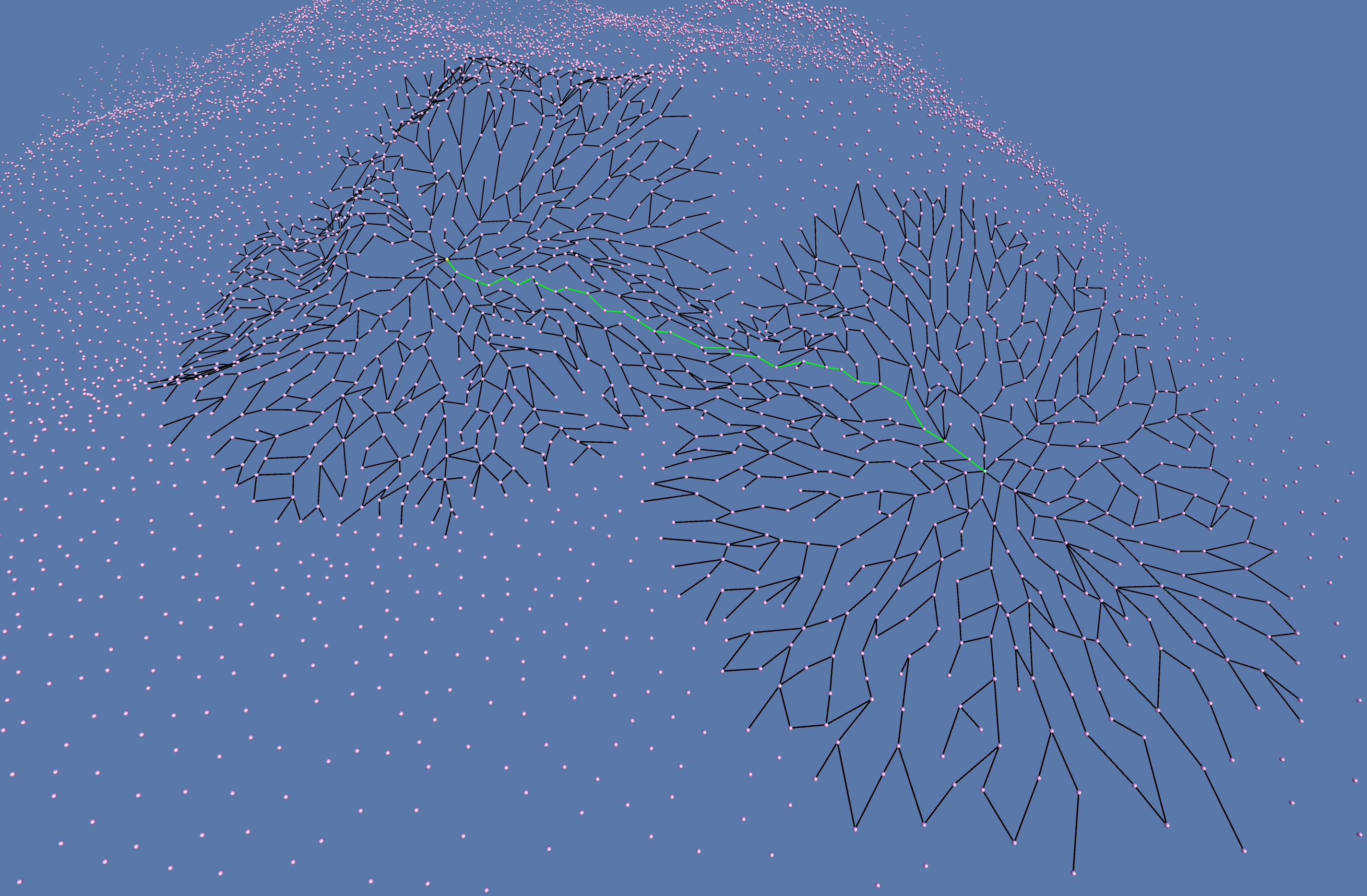
Figură II.2 Propagarea algoritmului Dijkstra. **Sus**: Terenul inițial. **Stânga**: Noduri calculate după 500 de pași. **Dreapta**: Noduri calculate la finalul algoritmului.

Prin modificarea propagării s-a obținut o îmbunătățire a algoritmului lui Dijkstra și anume Dijkstra Bidirecțional (Capitol II.3).

## Dijkstra Bidirecțional

O modificare adusă algoritmului lui Dijkstra. Plecăm atât din punctul de start cât și din punctul scop. Facem acest lucru pentru a micșora numărul de noduri verificate. Folosind intuiția de la (Figură II.2) dacă vizualizăm nodurile verificate in un anumit punct vom avea vizual două sfere, una cu centrul in nodul de start, una cu centrul in nodul scop.

**Bidirectional Search is a basic approach to accelerate Dijkstra’s algorithm for point-topoint queries. It performs simultaneously a backward search from the target and a forward search from the source. The procedure can be finished if both search spaces have met. For our case where graphs represent road networks we can expect a search space reduction of about factor two. We can imagine the search space of Dijkstra’s algorithm as a ball around the source node with radius r, so about r 2 nodes are visited. If we have two touching balls of half the size, the search space reduces to about 2 · ( r 2 ) 2 = 1 2 · r 2 nodes.**

Figură .

Algoritmul va folosi două structuri de tip min-heap, una pentru procesarea nodurilor plecând din punctul de start (not cu *mst*), iar cealaltă una pentru procesarea plecând din nodul scop (not cu *msc)*. Pentru inițializarea algoritmului vom introduce în *mst* nodul de start cu distanța 0 și în *msc* nodul scop cu distanța 0. La fiecare pas vom extrage din structura min-heap care are în vârf nodul cu distanța mai mică.

O abordare naivă ar fi să adaptăm și condiția de oprire de la algoritmul simplu al lui Dijkstra, adică să ne oprim când am extras un nod pornind din nodul de start și din nodul scop. Acest nod va fi intuitiv la mijlocul celor două puncte și vizual am putea zice că este punctul în care se unesc sferele. Această condiție de oprire nu garantează că alegem cel mai scurt drum. După cum observăm în (Figură II.4) primul nod extras de ambele căutări este nodul *M* aflat la distanță 9 atât de *Start* cât și de *Scop***.** Drumul *(Start, A, M, B, Scop)* are lungimea 18 și este incorect ales ca cel mai scurt drum pentru că *(Start, A, B, Scop)* are lungimea 16.



Figură II.4 Exemplu de drum incorect ales de condiția naivă.

Pentru a ne opri doar în momentul în care suntem siguri că am găsit drumul de lungime minimă, va trebui să adăugăm o variabilă nouă care să memoreze cel mai scurt drum găsit până în acest punct (notăm cu *distMinGasit)*, vom inițializa variabila cu valoarea *inf*. Pentru a simplifica explicația vom lua cazul în care extindem o muchie cu capetele *(x,* *y)* cu nodul *x* extras din *mst*. Dacă nodul *y* a fost extras din *msc*, atunci știm cel mai scurt drum de la *y* la nodul scop. Verificăm dacă *distMinGasit* mai mare decat distst(x) + distsc(y) + dist(x,y), dacă da, actualizăm *distMinGasit* cu această valoare. Ne vom opri când suma distanțelor celor două noduri din varful lui *mst* respectiv *msc* sunt mai mari ca *distMinGasit*. Demonstrație**: Why does it work? – Suppose there exists an s-t path P with length less than µ. – There must be an arc (v, w) on this path such that: ∗ dist(s, v) < topf and ∗ dist(w, t) < topr. – Both v and w have been scanned already. – When the second of these was scanned, it would have found the P. ∗ Contradiction: P cannot exist.**

**De gasit referinta.**

## A\*

Algoritm simplu de implementat (complexitate medie)

Necesita o functie de aproximare. Avand in vedere ca noi suntem intr-un spatiu tridimensional Euclidian, putem cu usurinta sa folosim distanta euclidiana a doua puncte care va fi intotdeauna mai mic sau egal cu distanta pe suprafata noastra.

Spre deosebire de efectul de propagare al algoritmului Dijkstra, A\* are o propagare directionala, la fiecare pas incercand sa se apropie de nodul tinta. **De adaugat poza cu propagarea**

Pe suprafete mici unde nu poate folosi avantajul de a merge directionat, sau unde intalneste obstructii, A\* se va comporta mai prost decat Dijkstra. (De vazut unde apare aproximativ aceasta diferentiere)

## Rafinare iterativa

Referinta (Bose, et al., 2011)

# Algoritmi pe fețe

Conventii:

{n = numarul de noduri, m = numarul de muchii}

Cei trei algoritmi functioneaza atat pe suprafete traingulate, dar si pe orice forma geometrica formata din puncte si legaturi intre acestea. Vom observa ca algoritmii mai complecși care se bazeaza pe ‚intinderea’ suprafetelor sau pe rafinarea punctelor vor putea rula doar pe suprafete/obiecte triangulate, adica nu putem avea intersectii de muchii sau fete cu mai mult sau mai putin de 3 muchii.

**Idei:**

* Algoritm care ia drumul minim constrans la muchii, ia triunghiurile care au muchii in drum, construieste in mijlocul fiecarui triunghi un punct, genereaza noua triangulare(o vom numi rafinarea trangularii) si repeta. Va genera un drum de lungime aproximativa, care nu este constrans la muchiile initiale, insa nu va gasi drumul de lungime minima absoluta. Putem impune o limita de rafinare pentru avea un timp de rulare decent.
* Transformarea unei suprafete traingulate in spatiu 3D la o suprafata in spatiul 2D, acum, pentru a gasi drumul minim e suficient sa unim punctele, pentru ca ne aflam in un spatiu 2D. Complexitate foarte mare. Inca nu stiu cum sa fac asta. Alegem o secventa de triunghiuri si muchii, a.i. sa avem triungi,muchie,triunghi,muchie,... si fiecare muchie dintre doua fete este exact muchia comuna. Parcurgem lista si rotim dupa muche a.i. sa ajungem la coplanaritate, repetam pentru fiecare muchie pana cand toate triunghiurile sunt coplanare.
* De adaugat o pagina in care utilizatorul poate selecta fisiere noi pe care sa le adauge in aplicatie (.txt/.obj)
* De discutat la partea de alg cu rafinare de diferenta dintre rafinarea in 3 fete versus 6 fete.

**Gasite in articole**

[A survey of geodesic paths on 3D surfaces]

Because we can triangulate every face of the terrain in linear time [3] we may assume without loss of generality that every face of F is a triangle(Trekking in the Alps Without Freezing or Getting Tired1 M. de Berg2 and M. van Kreveld2)

# Utilizarea aplicației

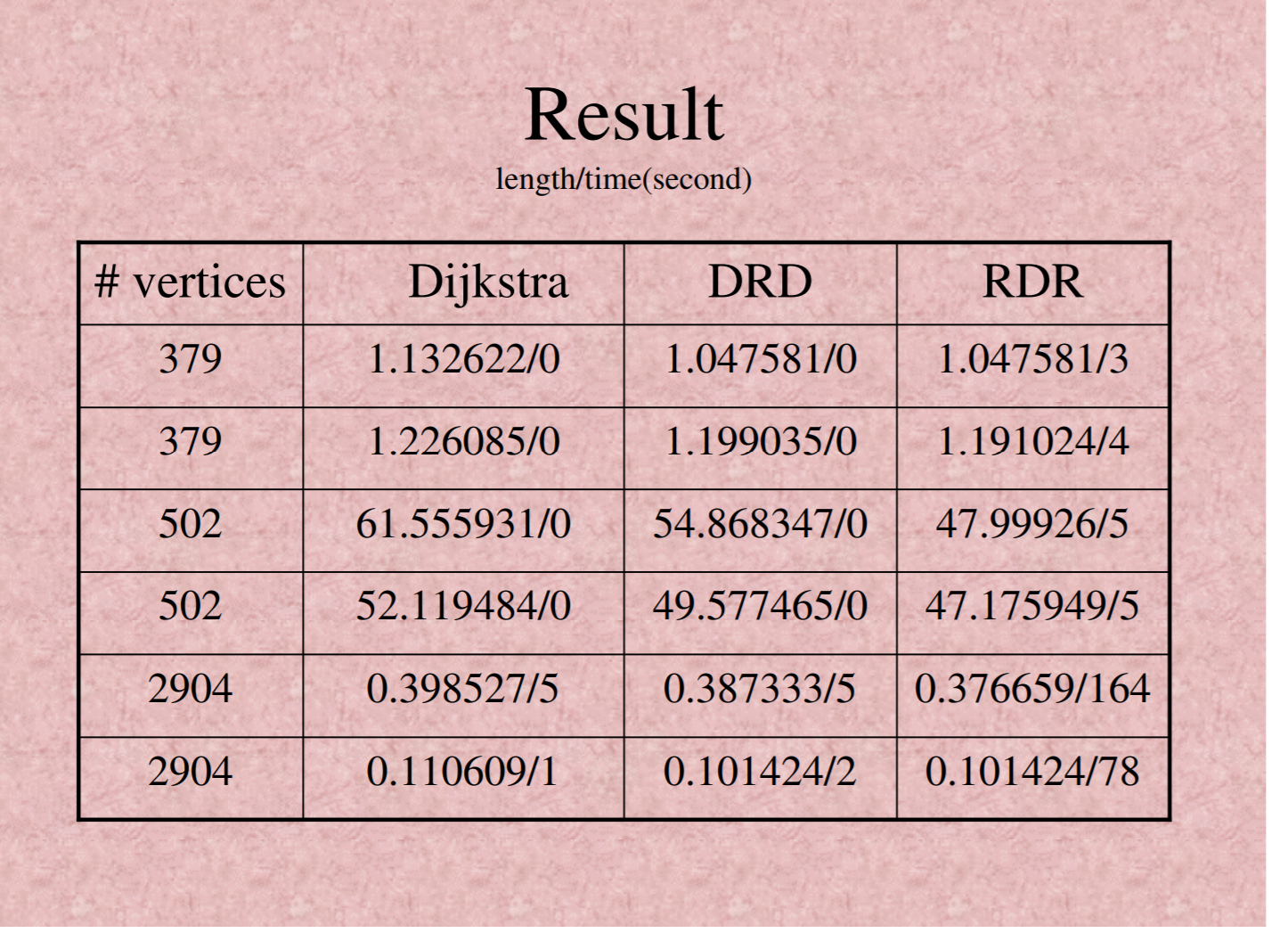
* Taste utilizate:
  + Tastele W, S deplaseaza camera in fata, respectiv in spate.
  + Tastele A, D deplaseaza camera la stanga, respectiv la dreapta.
  + Tastele Spatiu, C deplaseaza camera in sus, respectiv in jos.
  + La apasarea tastei Shift, Cursorul dispare si utilizatorul poate orienta camera folosind miscari ale Mouse-ului. Prin reapasarea tastei Shift cursorul va aparea si Mouse-ul nu va mai influenta orientarea camerei.

Date de intrare:

* Fisiere txt cu urmatorul format: Pe prima linie se afla numarul de puncte, il vom numi *n*. Pe urmatoarele *n* linii se vor afla 3 numere in format intreg sau flotant. Pe urmatoarele *n* vom avea listele de adiacenta cu formatul: Doua numare, primul reprezentand nodul (primul nod este 0) pentru care urmeaza lista de vecini, al doilea reprezentand numarul de vecini ai nodului (notam cu *v*), apoi *v* numere reprezentand vecinii. Toate cele *n* noduri trebuie sa aiba o linie, daca un nod nu are vecini va aparea ca `index\_nod 0`.
* Vreau sa pot introduce in aplicatie si fisiere de tipul *.obj*. Fisierele au un format simplu de parcurs, folosind primul caracter al fiecarei linii pentru a specifica ce va urma pe aceasta. De exemplu:
  + Liniile care incep cu ‚#’ sunt comentarii si vor fi ignorate
  + Liniile care incep cu ‚v’ sunt puncte geometrice cu minim 3 coordonate in ordinea x,y,z. A patra coordonata este optionala.
  + Liniile care incep cu ‚f’ sunt urmate de indicii punctelor care formeaza o fata. Exista mai multe metode prin care pot fi date, de ex se pot specifica si coordonate de textura sau normale, insa in scopul aplicatiei acestea vor fi ignorate.
  + In standard mai exista ‚l’ pentru linii, ‚vt’ pentru coordonate de texturi, ‚vn’ pentru coordonatele normalelor, ‚vp’ pentru puncte in spatiu parametrizat(de ex multimea punctelor de pe un plan sau o dreapta) si multe altele.
  + Aplicatia va prelucra doar liniile incepand cu ‚v’ si ‚f’ pentru a putea construi suprafata, fie ea triangulata sau nu.

# Concluzii

**De facut un tabel comparativ(Backtracking VS Dijkstra VS A\*) cu distante,timpi de rulare, muchii verificate pe diferite forme triangulate**

**EX: **

# Bibliografie

Blender Foundation, 2021. *Blender Reference Manual.* [Interactiv]   
Available at: https://docs.blender.org/manual/en/latest/  
[Accesat 20 Mai 2021].

Bose, P., Maheshwari, A., Shu, C. & Wuhrer, S., 2011. A survey of geodesic paths on 3D surfaces. *Computational Geometry,* 44(9), pp. 486-498.

Chen, J. & Han, Y., 1990. Shortest Paths on a Polyhedron, Part I: Computing Shortest Paths. *International Journal of Computational Geometry & Applications.*

de Berg, M. & van Kreveld, M., 1997. Trekking in the Alps Without Freezing or Getting Tired. *Algorithmica,* Issue 18, p. 306–323.

Dijkstra, E. W., 1959. A note on two problems in connexion with graphs. *Numerische Mathematik,* Volumul 1, p. 269–271.

Murray, J. D. & Van Ryper, W., 1996. *Encyclopedia of Graphics File Formats, 2nd Edition.* Sebastopol: O'Reilly & Associates, Inc..

Surazhsky, V. și alții, 2005. Fast Exact and Approximate Geodesics on Meshes. *ACM Transactions on Graphics,* Issue 24, pp. 553-560.

Unity Technologies, 2021. *Manual Unity.* [Interactiv]   
Available at: https://docs.unity3d.com/Manual/index.html  
[Accesat 10 Mai 2021].