

UNIVERSITATEA POLITEHNICA DIN BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE



PROIECT DE DIPLOMĂ

Simularea unui mediu post-apocaliptic

Vlad Marius-Cătălin

Coordonator științific:

Conf. dr. ing. Anca Morar

BUCUREŞTI

2020

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT



DIPLOMA PROJECT

Post-apocalyptic environment simulation

Vlad Marius-Cătălin

Thesis advisor:
Conf. dr. ing. Anca Morar

BUCHAREST

2020

CUPRINS

1 Introducere	1
1.1 Context	1
1.2 Problema	1
1.3 Obiective	1
1.4 Soluția propusă	2
1.5 Rezultatele obținute	2
1.6 Structura lucrării	2
2 Motivație	3
3 Metode Existente	4
3.1 Motoare grafice	4
3.1.1 Unreal Engine	4
3.1.2 Unity	4
3.1.3 CryEngine	5
3.1.4 Godot	5
3.2 Fragmentare	5
3.3 Creștere vegetație	6
4 Soluția Propusă	8
4.1 Fragmentarea și explozia clădirilor	8
4.2 Creșterea vegetației	9
4.3 Sistemul de degradare	10
5 Detalii de implementare	12
6 Evaluare	14

7 Concluzii	21
Anexe	24

SINOPSIS

În această lucrare propun un model pentru simularea unui mediu în care civilizația dispare. În prezent nu există astfel de model. Mediile post-apocaliptice sunt create manual pornind de la o scenă intactă. În acest model mă voi concentra pe distrugerea clădirilor și creșterea vegetației. Distrugerea clădirilor se va realiza prin împărțirea triunghiurilor obiectului 3D folosind diagrame Voronoi [1], iar creșterea vegetației folosind generatorul de iederă al lui Thomas Luft [4]. Aplicația va genera o aproximare în timp real a evoluției mediului și va permite utilizatorul să modifice anumite parametri, înainte de simulare, ce vor influența creșterea vegetației și fragmentarea clădirilor.

ABSTRACT

I propose a model to simulate the evolution of an environment in the event of disappearance of human civilization. Currently, there is no such model which will accomplish such task. Such environments must be manually created from an unaltered scene. In this model, I will focus on building destruction and vegetation growing. Building destruction will be accomplished by fragmenting the triangles of the 3D object using Voronoi diagrams [1] and vegetation growing through Thomas Luft's ivy generator [4]. The application gives a rough estimation in real time of the evolution of such environment and allows the user to control, to some extent before the actual simulation, the properties of vegetation growing and fragmentation.

1 INTRODUCERE

1.1 Context

Ficțiunea post-apocaliptică este un subgen al ficțiunii științifico-fantastice în care sunt descrise evenimente care au loc după un incident apocaliptic. Un astfel de incident duce, în general, la sfârșitul civilizației. O posibilă clasificare a acestor incidente este:

- climatice (ex. schimbări bruse ale climei);
- astronomice (ex. impactul pământului cu corperi cerești, invazii extraterestre);
- distructive (ex. supraexploatarea resurselor, războaie nucleare);
- medicale (ex. pandemii, invazii zombie);
- tehnologice (ex. revoluții cibernetice);

Datorită evoluției tehnologice din ultimele decenii, efectele speciale din filme și grafica jocurilor video au devenit din ce în ce mai realiste¹. Această evoluție a dus la crearea unor simulări a lumii fictive cât mai realiste. Ficțiunea care implică lumi post-apocaliptice a început să devină populară în domeniile tehnologice axate pe crearea de conținut de divertisment (filme, jocuri video, animații etc.).

1.2 Problema

În momentul de față, generarea de scene post-apocaliptice implică un proces manual de distrugere a unei scene și de creștere a vegetației. Când se creează o scenă post-apocaliptică, se pleacă de la o scenă normală apoi se distrug elemente din scenă și se adaugă vegetație excesivă.

1.3 Obiective

Proiectul va încerca să automatizeze generarea de scene post-apocaliptice. Dându-se o scenă normală, aplicația va modifica scena astfel încât să semene cu un peisaj dintr-o lume în care civilizația a dispărut.

¹<https://www.siliconrepublic.com/play/video-game-graphics-evolution>

1.4 Soluția propusă

Proiectul va fi o aplicație care va lua o scenă 3D cu clădiri, copaci și elemente decorative și va degrada scena în timp real. Prin degradare se înțelege distrugerea treptată a clădirilor, creșterea de iederă în jurul clădirilor, creșterea de mușchi pe suprafața clădirilor și creșterea copacilor existenți în scenă.

1.5 Rezultatele obținute

Aplicația funcționează conform cerințelor, scena se degradează în timp real. O dată cu creșterea vegetației și a nivelului de distrugere din scenă, numărul de cadre pe secunde scade și aplicația va avea nevoie de mai multe resurse hardware.

1.6 Structura lucrării

În capitolul Motivație am analizat situația curentă și am descris de ce ar fi nevoie de o astfel de aplicație. Mai departe am analizat unele dintre motoarele grafice populare existente, metode de distrugere în timp real și algoritmi de creștere a vegetației. Am descris aplicația cu componente sale principale și un algoritm propriu pentru distrugerea clădirilor în timp real. Am specificat niște detalii de implementare și momente dificile întâmpinate în implementare și am evaluat performanța aplicației măsurând numărul de cadre pe secunde și cât de mult sunt utilizate anumite componente hardware ale sistemului.

2 MOTIVATIE

În ultimele decenii, tehnologia a devenit din ce în ce mai performantă și accesibilă publicului. Această evoluție a deschis orizonturi noi, de cercetare și explorare. Unul dintre acestea este de simulare a mediului înconjurător. În ziua de astăzi, astfel de simulări sunt folosite în industria de divertisment în realizarea filmelor, animațiilor și a jocurilor video sau în domenii specializează în cercetare și testare pentru „înțelegerea și evaluarea mediilor reale sau abstractive”¹.

În acest proiect se dorește realizarea unui simulator de mediu post-apocaliptic în care civilizația dispare de pe planetă. Scopul acestuia este de a observa evoluția mediului în astfel de situație. Se pleacă de la o scenă care să semene cu o porțiune dintr-o așezare umană (sat, comună, oraș etc.) care conține clădiri. Aceste clădiri vor fi deteriorate pe parcursul timpului. Se vor murdări, partii din ele se vor distrugere.

Pe lângă distrugerea clădirilor, vegetația din scenă va crește într-un mod cât mai realist. Pe clădiri vor crește plante care să vor cățăra pe acestea. Se vor lua în calcul coliziunile, forma clădirii, eventualele găuri create de distrugeri și lumina din scenă. Un exemplu de scenă se află în Figura 1 din jocul video NieR:Automata(2017).



Figura 1: Scenă post-apocaliptică din jocul NieR:Automata(2017)²

În prezent, generarea acestor scene este în mare parte manuală. Se creează modelele 3D ale cădirilor intace, se modifică manual modelele pentru a arăta deteriorate, se aşază în scenă, se adaugă vegetația și se retusează scena. Proiectul își propune să automatizeze partea de distrugere și creștere a vegetației, să se ia o scenă cu clădiri și vegetație, să o degradeze (să distrugă din clădiri și să crească din vegetație) și, noua scenă rezultată, să poată fi exportată.

¹<http://web.cs.mun.ca/~donald/msc/>

²<https://ro.pinterest.com/pin/250935010474647060/>

3 METODE EXISTENTE

3.1 Motoare grafice

Motorul grafic este componenta principală a unei aplicații grafice. Există motoare specializate pe anumite tipuri de jocuri / aplicații și motoare de uz general. Motoarele de uz general pot fi clasificate în motoare 2D, 3D sau hibrid. Motoarele 3D sunt folosite pentru a genera efecte vizuale cât mai realiste. Unele dintre cele mai populare astfel de motoare sunt **Unreal Engine**, **Unity**, **CryEngine** și **Godot**. O comparație între aceste motoare se află în Tabela 1

3.1.1 Unreal Engine

Este un motor grafic 3D creat de către compania Epic Games¹. Ultima versiune, Unreal Engine 4, a fost lansată pe data de 19 martie 2014. Pentru folosirea motorului, creatorii de conținut trebuiau să plătească un abonament lunar de 19\$ pe lună și 5% din profit să fie redirecționat către Epic Games². Din martie 2015, s-a renunțat la abonamentul lunar³.

Motorul oferă funcționalități moderne precum rasterizare fotorealistă, ray tracing, efecte de pre și post procesare, simularea mișcărilor hainelor și a părului, distrugere în timp real, suport pentru VR etc. Motorul este scris în C++, iar codul este accesibil pentru producătorii de conținut.

3.1.2 Unity

Unity este un motor grafic hibrid creat de Unity Technologies⁴. Inițial, motorul a fost lansat pentru Mac OS X, dar în prezent există suport și pentru Windows și Linux. În Unity se pot crea jocuri 2D, 3D, pentru realitate virtuală și augmentată. Inițial Unity funcționa pe bază de licențe plătite. Din 2016, compania a adoptat un sistem pe bază de abonament⁵. În prezent, există licență gratuită pentru companii care câștigă anual mai puțin de 100 000\$ și licențe plătite care vin cu beneficii precum interfață întunecată, relații cu clienții, acces la

¹<https://www.unrealengine.com/>

²[https://arstechnica.com/gaming/2014/03/unreal-engine-4-now-available-as-19month-subscription-with-5-royalty/](https://arstechnica.com/gaming/2014/03/unreal-engine-4-now-available-as-19-month-subscription-with-5-royalty/)

³<https://www.ign.com/articles/2015/03/02/unreal-engine-4-is-free-for-everyone>

⁴<https://unity.com/>

⁵<https://www.gamesindustry.biz/articles/2016-12-14-unity-dropping-major-updates-in-favour-of-date-based-model>

codul sursă etc.

Motorul este popular în rândul jocurilor Indie, este scris în C++, creatorii de conținut pot scrie cod în C# sau JavaScript și oferă suport pentru peste 24 de platforme.

3.1.3 CryEngine

Este un motor grafic 3D creat de Crytek⁶. În ultima versiune a motorului, CryEngine V, s-a trecut de la modelul „pay what you want” la plata a 5% din profitul total obținut de produsele care folosesc motorul⁷.

Câteva funcționalități importante ale acestui motor sunt path-finding integrat, simularea apei realistă, ceată volumetrică, simularea îndoirii funiei, iluminare dinamică și crearea de vehicule.

3.1.4 Godot

Godot este un motor grafic hibrid open-source creat inițial de către Juan Linietsky și Ariel Manzur⁸. Codul sursă al motorului a fost publicat în 2014 pe Github cu licență MIT. Proiectul este unul nonprofit, costurile de producție sunt acoperite de donații din partea membrilor și a companiilor.

Motorul are funcționalități de bază și alte funcționalități mai importante precum sistem de particule, navigație pe obiecte, redare de videoclipuri și sistem de plugin-uri.

3.2 Fragmentare

Fragmentarea este o tehnică de a împărți un obiect 2D sau 3D în bucăți mai mici. În jocuri video există de 2 tipuri de fragmentări: fragmentare **precalculată** și fragmentare **în timp real**.

În fragmentarea **precalculată**, obiectul va fi fragmentat înaintea rulării programului. Este o metodă statică, se rulează o singură dată, nu are un impact asupra performanței aplicației, dar poate să nu ofere efecte realiste într-un mediu dinamic. Obiectul va avea mereu același număr de fragmente, iar fragmentele vor avea mereu aceleași forme.

În fragmentarea **în timp real**, obiectul va fi fragmentat în timpul rulării. Metoda este dinamică, se rulează de fiecare dată când se dorește fragmentarea obiectului, este intensiv computațională, dar este mai flexibilă și poate oferi efecte mai realiste.

Un obiect 3D este format din triunghiuri. O metodă rapidă, dar nu realistă, este de a subdiviza recursiv triunghiurile obiectului 3D în triunghiuri mai mici până se depășește un

⁶<https://www.cryengine.com/>

⁷<https://www.gamesindustry.biz/articles/2018-03-20-crytek-adopts-royalties-model-as-cryengine-5-5-arrives>

⁸<https://godotengine.org/>

Tabela 1: Comparatie motoare grafice 3D

Specificație	Unreal Engine	Unity	CryEngine	Godot
Platforme de dezvoltare	Windows, Mac OS X, Linux	Windows, Mac OS X, Linux	Windows	Windows, Mac OS X, Linux
Limbaje programare	C++, Blueprints	C#, JavaScript	C++, C#, Lua	C++, C#, GDscript
Limbaje shading	Limbaj propriu, HLSL, GLSL	Gg, HLSL, GLSL	Limbaj propriu similar cu HLSL, CgFX	Limbaj propriu similar cu GLSL ES 3.0
Platforme pentru desktop	Windows, Mac OS X, Linux	Windows, Mac OS X, Linux	Windows, Linux	Windows, Mac OS X, Linux, BSD
Platforme pentru mobile	iOS, Android	Windows Phone, iOS, Android, BlackBerry 10, Tizen		iOS, Android, BlackBerry 10
Platforme pentru console	PlayStation 4, Xbox One, Nintendo Switch	Xbox 360, Xbox One, Wii U, PlayStation 3, PlayStation 4, PlayStation Vita, Nintendo Switch, Nintendo 3DS	PlayStation 4, Xbox One	Xbox One
Platforme pentru VR	Magic Leap One, HTC Vive, Oculus Rift, PlayStation VR, Google Daydream, OSVR, Samsung Gear VR	Magic Leap One, Oculus Rift, PlayStation VR, Google Cardboard, Steam VR, Gear VR	Oculus Rift, OSVR, PSVR, HTC Vive	Google Cardboard, Google Daydream, Oculus Rift

prag (adâncime maximă a recursivității, aria triunghiului este într-un interval etc.) [2]. Fiecare triunghi fragment va fi transformat într-o prismă. Un exemplu de subdivizare se află în figura 2.

O metodă mai realistă este prezentată în „Real-time Mesh Destruction System for a Video Game” [3]. Obiectul 3D este segmentat în părți convexe. Pentru fiecare parte convexă se generează în interior un număr de puncte de control pentru o diagramă Voronoi [1] 3D, se împarte spațiul într-o diagramă Voronoi 3D și se intersectează spațiul cu partea convexă. Vor rezulta un număr de fragmente mai mic sau egal cu numărul de puncte de control.

3.3 Creștere vegetație

Simularea creșterii vegetației se poate face **manual** sau **automat**. În simularea **manuală**, se creează manual modele ale plantelor în diferite stagiile creșterii. La rulare, la

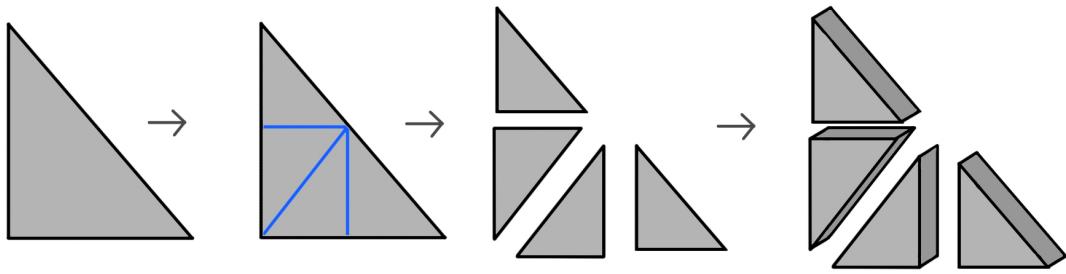


Figura 2: Subdivizarea unui triunghi și transformarea fragmentelor în prisme

diferite momente de timp, se schimbă modelul plantei cu un model dintr-un stagiu anterior al creșterii. Metoda este ușor de implementat, nu oferă complexitate mare la rularea aplicației, dar este foarte costisitoare din punct de vedere al timpului și al memoriei, nu oferă un efect realist pentru creșterea în timp real Există două moduri de generare și creștere a vegetației în mod **automat**, prin **gramatici formale** [5] sau prin **sisteme de particule** [4].

Gramaticile formale sunt, în general, mai ușor de folosit datorită simplității, însă rezultatele pot fi particulare, și nerealiste [5]. O gramatică este formată dintr-un alfabet (set de caractere permise), regulile de start și regulile de producție. Pentru generarea de plante, regulile de start sunt asociate rădăcinilor, regulile de producție sunt asociate crengilor, frunzelor și ramificațiilor, iar alfabetul este asociat rădăcinilor, crengilor și ramificațiilor. Un exemplu de plantă generată folosind o gramatică se află în Figura 3.

Într-un **sistem de particule**, fiecare ramificație este un sistem de particule independent, particulele fiind vârfurile ramurilor. Fiecare sistem de particule are informații locale, creșterea sistemului se face pe baza acestor informații și a interacțiunii cu celelalte sisteme.

În ambele cazuri se pot introduce factori care influențează creșterea și care dă un efect realist (coliziunile cu mediul, lumina din scenă, atracția gravitațională, vânt etc.).



Figura 3: Tulpina unei plante generată cu gramatica formala descrisă de Johan Knutzen [5]

4 SOLUȚIA PROPUȘĂ

În acest proiect am simulat situația post-apocaliptică în care civilizația dispare de pe pământ. Proiectul este realizat folosind motorul grafic Unity. Există o scenă cu câteva clădiri și elemente decorative. Simularea distrugerii se va face prin fragmentarea și explozia clădirilor, iar creșterea vegetației prin creșterea iederei în jurul clădirilor, creșterea copacilor din scenă și creșterea de mușchi pe suprafața clădirilor.

La pornirea aplicației, va apărea un meniu unde utilizatorul va putea varia numărul de clădiri în scenă, numărul de puncte de control pentru diagramele Voronoi [1], viteza de creștere a iederei și dacă copaci vor crește. În scena în care mediul se degradează, utilizatorul va putea deplasa camera, va putea scala trecerea timpului și va putea salva scena în forma unui obiect 3D.

4.1 Fragmentarea și explozia clădirilor

Algoritmul propus este o versiune îmbunătățită a algoritmului de triangularizare [2]. La momentul primei distrugerii, modelul 3D va fi fragmentat complet și se va crea o explozie în zona distrugerii. Pentru distrugerile consecutive, se vor crea doar explozii, nefiind nevoie de refragmentarea obiectului.

În algoritmul de fragmentare, fiecare triunghi al obiectului 3D se împarte în diagrame Voronoi [1]. În interiorul fiecărui triunghi se vor genera un număr de puncte de control (site-uri). Pentru fiecare astfel de punct se va genera un poligon, iar fiecare poligon va fi transformat într-un trunchi de piramidă. Nu va fi transformat într-o prismă pentru a preveni efectul de z-fighting.

După fragmentarea clădirii, modelul 3D al clădirii va fi sters și va fi înlocuit cu modelele 3D ale fragmentelor rezultate. Pentru redarea unui efect realist, fragmentele vor fi legate între ele cu articulații (joint-uri). Fragmentele vor forma un graf neorientat unde nodurile vor fi fragmentele, iar muchiile vor reprezenta calea către fragmentele vecine cu care se ating.

Graful va avea definită o zonă de fragmente ancorate. Fragmentele care intră în această zonă, vor fi ancorate. Astfel de fragmente vor fi considerate imobile (nu se vor mișca decât după ce vor fi explodate). Restul fragmentelor vor fi afectate normal de fizica scenei.

În momentul unei explozii, se iau fragmentele din zona punctului exploziei și se elimină din graf. Se sterg articulațiile sunt lăsate să cadă liber. După o explozie, se aplică algoritmul *flood fill* care pornește de la fragmentele ancorate. Insulele izolate, care nu sunt conectate la cel puțin un fragment ancorat, vor fi eliminate din graf, li se vor sterge articulațiile și vor fi lăsate liber.

Atât fragmentarea cât și explozia sunt configurabile. Pentru fragmentare se pot alege

dimensiuni pentru grosimea fragmentelor, numărul de puncte de control maxim pentru diagramele Voronoi [1], densitatea fragmentelor (util pentru simularea forțelor de interacțiune dintre fragmente într-o manieră mai realistă), punctul și raza exploziei.

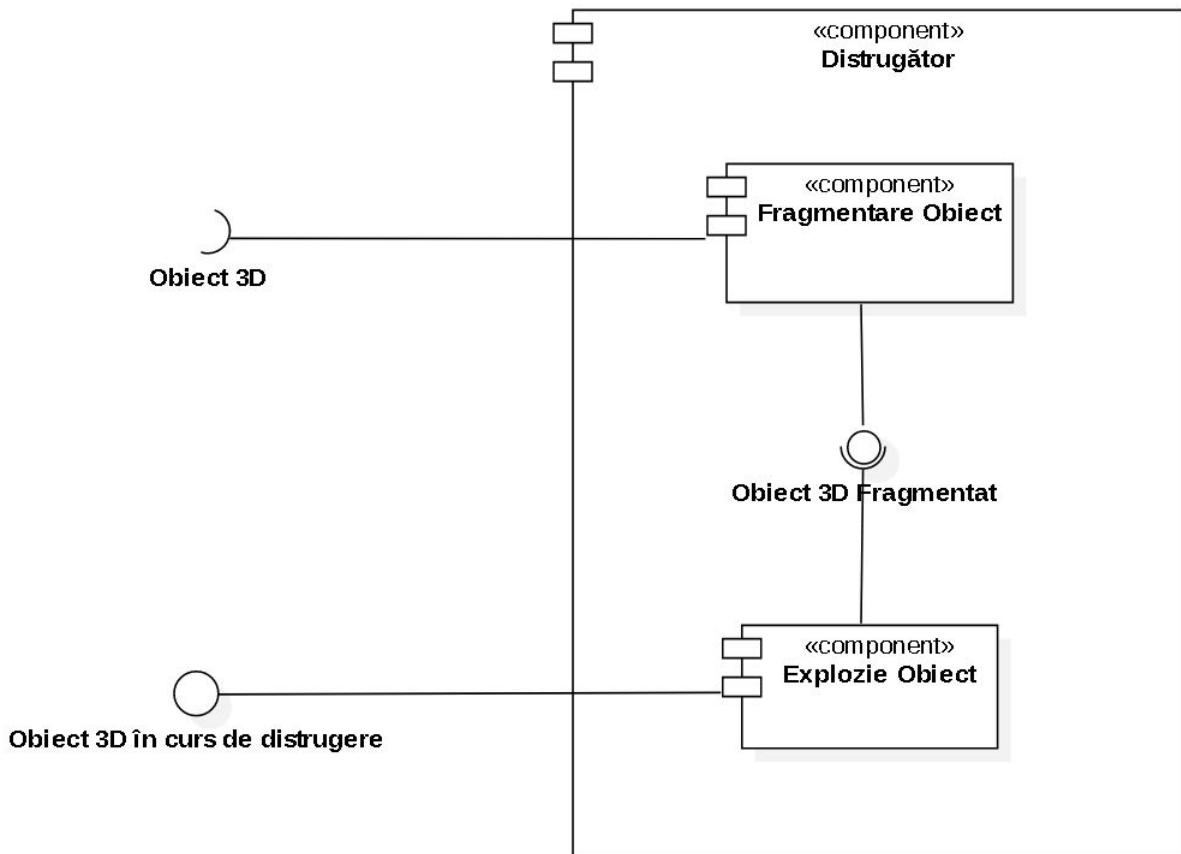


Figura 4: Componenta care realizează distrugerea clădirilor

4.2 Creșterea vegetației

Pentru creșterea iederei, am folosit generatorul de iederă implementat de Thomas Luft [4]. Am folosit o versiune adaptată pentru motorul grafic Unity¹. Sistemul funcționează pe bază de particule. Fiecare ramificație a iederei este reprezentată ca un arbore în rădăcina este punctul de ramificație (în coordinate globale, world space), iar copii sunt puncte în spațiul local al rădăcinii. În momentul creșterii, frunzele arborilor vor fi ramificate și se va crea un arbore nou.

În procesul de creștere, se iau în calcul mai mulți factori:

- direcția creșterii nodurilor anterioare;
- direcția luminii din scenă;

¹<https://github.com/radiatoryang/hedera>

- obiectele din scenă;
- atracția gravitațională.

Acești factori sunt introdusi într-o euristică pentru a genera pozițiile noilor noduri și dacă unele noduri se vor bifurca.

În proiect, am generat rădăcinile initiale ale iederelor în jurul clădirilor pentru a simula cătărarea iedei pe fațadele clădirilor și le-am pus să crească la un anumit interval de timp.

Pentru a adăuga realism, am adăugat un efect de creștere a mușchilor și a lichenilor pe fațadele clădirilor. Creșterea este progresivă, de jos în sus. De asemenea, am adăugat în scenă și copaci care cresc progresiv, la un anumit interval de timp.

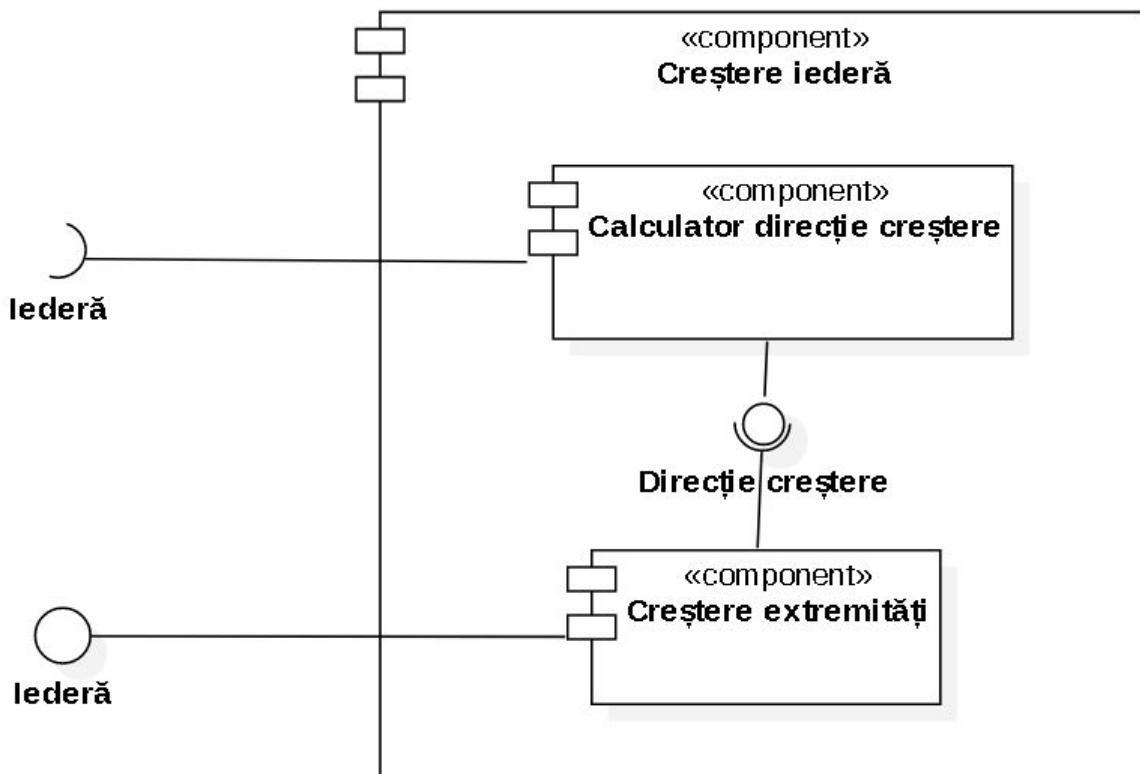


Figura 5: Componenta care realizează creșterea iederei

4.3 Sistemul de degradare

Pentru a simula mediul post-apocaliptic, am combinat cele două componente descrise anterior. Pentru distrugerea unei clădiri, am împărțit modelul 3D al clădirilor în **obiecte mici** și **obiecte mari**.

Obiectele mici sunt părțile de dimensiuni reduse ale clădirilor care se distrug mai repede (în cazul clădirilor din scenă uși și ferestre). Aceste obiecte vor fi fragmentate și explodate complet în primele zeci de secunde ale rulării aplicației.

Obiectele mari sunt părțile de dimensiuni sporite ale clădirilor care se distrug mai lent (în cazul clădirilor din scenă pereti, podele și acoperișuri). Aceste obiecte vor fi fragmentate și explodate parțial de mai multe ori până la distrugerea completă.

Fiecare clădire va avea un număr de rădăcini de unde va crește iedera. La un interval de câteva zecimi de secunde, din scenă se aleg un număr de rădăcini care vor fi crescute. Plantele rezultate se vor cătăra pe clădiri chiar și după fragmentare. Toată iedera din scenă este randată folosind un singur obiect 3D static.

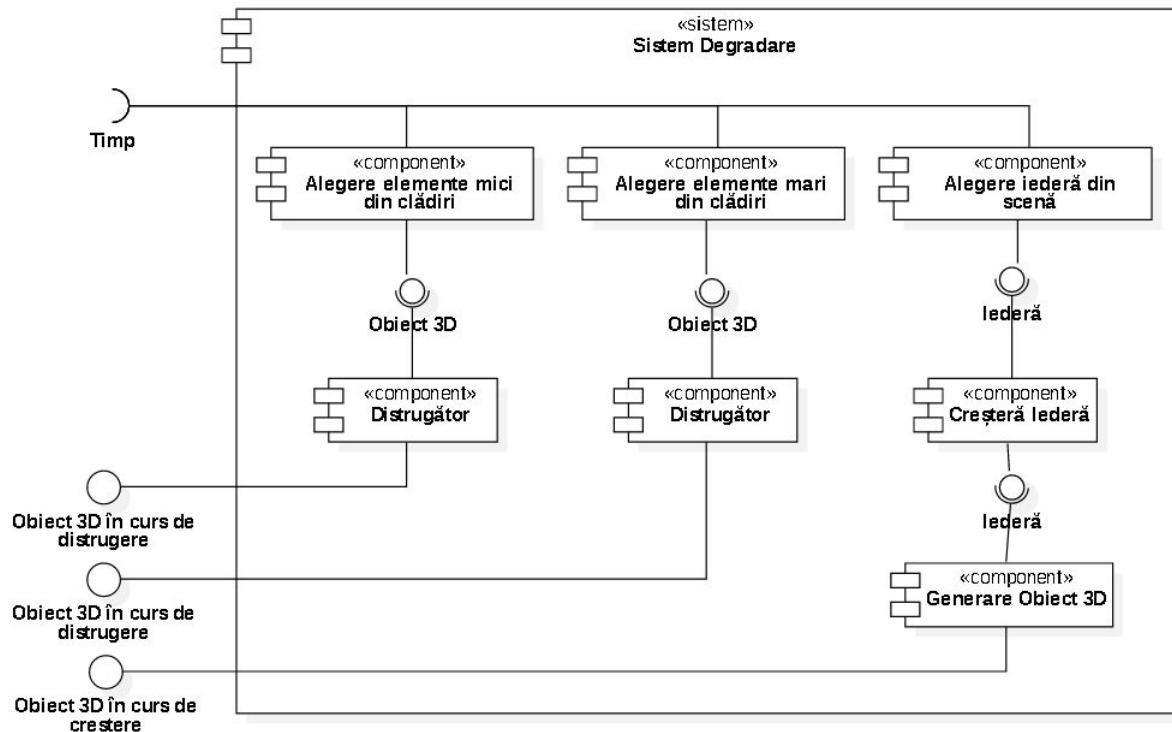


Figura 6: Sistemul de degradare al scenei

5 DETALII DE IMPLEMENTARE

Implementarea a fost făcută în 2 etape. Prima etapă a constat în dezvoltarea componente de **distrugere a clădirilor**. Am pornit de la codul proiectului ScamScatter¹ de unde am ales să implementez 2 metode statice, una pentru fragmentare, alta pentru explozie, după care am introdus clasele pentru calculele cu diagrame Voronoi [1]². O primă problemă a fost că clasele Voronoi produceau rezultate eronate pentru numere în virgulă mobilă mici. Soluția a fost ca în timpul împărțirii triunghiurilor în poligoane Voronoi să măresc obiectul 3D (l-am făcut de 500 de ori mai mare).

Ambele metode, de **fragmentare și explozie**, durau destul de mult timp și pentru obiecte cu număr mari de fețe aplicația se bloca. Am reușit să fac aceste 2 metode corutine. Funcțiile au alocat un anumit timp maxim de rulare. Dacă se depășește acest timp, se ieșe din funcție și se intră de unde a rămas la următorul apel. Avantajul metodei este că frame-rate-ul se menține și pentru un număr mai mare de clădiri.

Metoda de fragmentare folosește triunghiurile modelelor 3D. Metoda de fragmentare nu funcționează foarte bine pentru triunghiuri care au o latură mult mai mică decât celelalte 2 laturi. În acest caz, triunghiul respectiv nu se fragmentează, triunghiul va fi considerat un fragment întreg.

Pentru **creșterea iederei**, a trebuit să rezolv câteva probleme cu framework-ul Hedera. Codul funcționa doar în Editorul Unity, nu și în versiunile compilate. Am folosit directive de preprocesare pentru a converti codul și a-l putea compila. De asemenea, am întâmpinat probleme la generarea automată a rădăcinilor, pe lângă generarea pozițiilor rădăcinilor, trebuiau generate și normalele la suprafetele clădirilor. Am ales să generez manual, pentru fiecare clădire, un set de poziții și normale și să le folosesc pe acelea la rulare.

Creșterea mușchilor pe suprafetele clădirilor am implementat-o la nivel de shader. Se ține cont de coordonata cu direcția în sus (în Unity coordonata y) a fragmentului (din banda grafică). Se calculează un factor de interpolare între textura originală și textura cu mușchi după formula

$$f = s \cdot t \cdot \left(1 - \frac{y - y_{min}}{y_{max} - y_{min}}\right) \quad (1)$$

unde f este factorul de interpolare, s este factorul de scalare al creșterii mușchiului, t este timpul raportat la începutul rulării programului, iar y_{min} și y_{max} sunt coordonatele minimă și maximă din spațiul lume al obiectului 3D. În final, se face o interpolare liniară pe fiecare canal de culoare.

$$c_{fin} = c_{tex} + (c_{mos} - c_{tex}) \cdot f \quad (2)$$

¹<https://github.com/danbystrom/ScamScatter>

²<https://github.com/OskarSigvardsson/unity-delaunay>

unde c_{fin} este canalul culorii finale, c_{tex} este culoare fragmentului după interpolarea coordonatelor UV ale texturii originale ,iar c_{mos} este culoarea fragmentului după interpolarea coordonatelor UV ale texturii cu mușchi.

Factorul de interpolare crește cu trecerea timpului și scade cu creșterea coordonatei y. Astfel, fragmentele din partea de jos a obiectului vor avea culoarea mai apropiată de textura mușchilor, iar cele de sus vor avea culoarea texturii originale. Efectul generat de acest shader este de o creștere de jos în sus a unor mușchi pe suprafața obiectului 3D. În Figura 7 se află o comparație între o suprafață normală și una cu shader-ul aplicat.

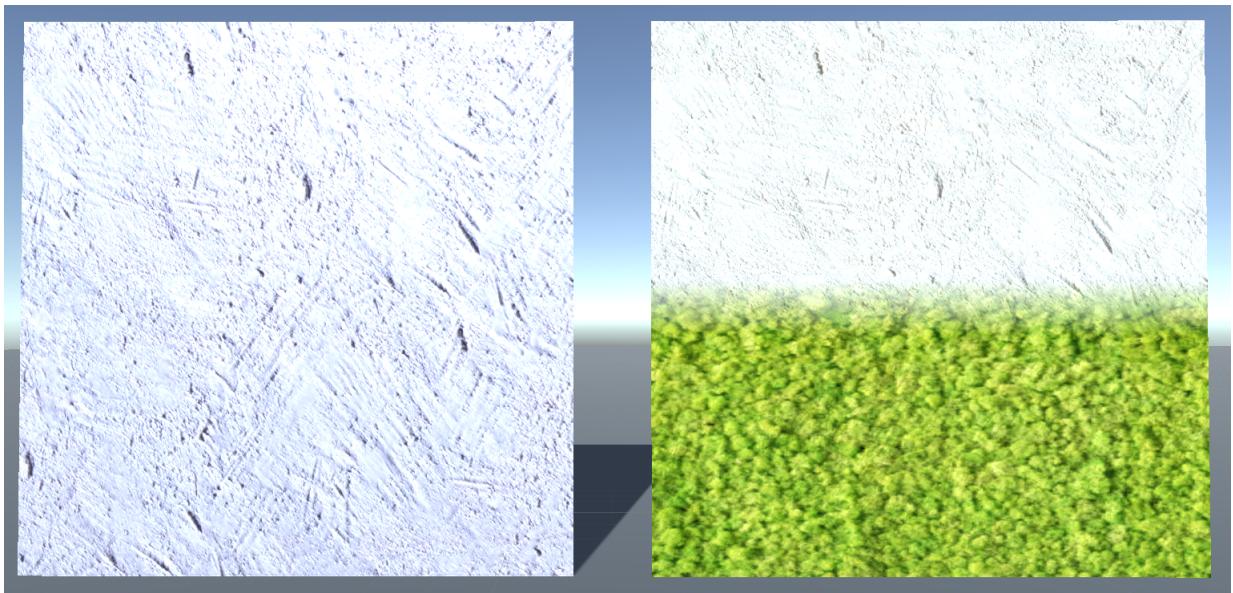


Figura 7: Comparație între un plan cu textura normală (stânga) și un plan cu textura combinată cu textura mușchilor(dreapta)

În Unity există generatoare de copaci. Pentru un efect mai realist, am generat copaci în scenă pe care îi cresc la un anumit interval de timp. Creșterea copacilor este limitată doar în interiorul editorului. În timpul rulării nu se pot accesa și modifica parametrii copacilor generați. Pentru a reda un efect de creștere al copacilor, am luat fiecare copac, am generat un număr de copii și am modificat parametrii astfel încât modelele să fie în stagii progresive al unei creșteri. În timpul rulării, schimb modelele 3D ale copacilor pentru a genera un astfel de efect.

6 EVALUARE

Algoritmul de fragmentare este făcut să funcționeze în timp real. La fiecare apel Update al motorului, acesta rulează o cantă de timp după care se ieșe din funcție. La următorul apel se intră în funcție se pornește de unde s-a ieșit anterior. Pentru a compara cu alți algoritmi, am eliminat această constrângere temporar, pentru a calcula timpul exact de rulare al algoritmului.

Pentru acest proiect am creat 3 modele 3D de dimensiuni diferite prezentate în Tabela 2. Algoritmul de fragmentare l-am comparat cu funcția „cell fracture” din Blender care fragmentează un obiect 3D folosind generatoare de părți convexe¹. Algoritmul generează puncte în volumul încadrator al obiectului 3D, împarte volumul în fragmente convexe și face intersecția fragmentelor cu obiectul original.

Testele au fost rulate pe laptop-ul personal care are următoarele specificații:

- CPU: Intel® Core™ i5-5200U cu 2 nuclee și 4 thread-uri;
- RAM: 8 GB DDR3 1600 MHz;
- GPU: NVIDIA GeForce 920MX 2 GB GDDR3.
- OS: Windows 10 x64

Am rulat fiecare algoritm o singură dată și am trecut rezultatele în Tabela 2. Algoritmul propus în secțiunea Soluția Propusă este dependent de numărul de triunghiuri, iar cel din Blender de volumul încadrator. Din cauza numărului mic de triunghiuri al primelor 2 clădiri, algoritmul propus este mai rapid și compensează pentru intersecțiile de volum din algoritmul din Blender. În schimb, în cazul clădirii 3 fațada are un volum încadrator simplu, dar un număr mare de triunghiuri.

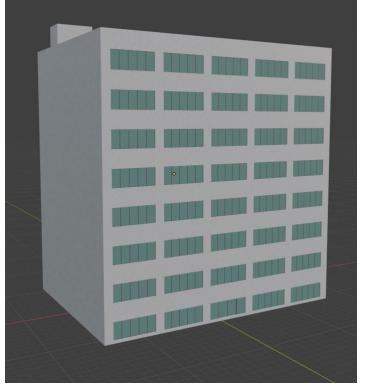
Pentru evaluarea întregii aplicații, am creat o hartă (Figurile 8, 9) care să conțină clădiri, vegetație și elemente decorative. Fiecare casă se distrugă la un moment aleator de timp, iar vegetația crește la un anumit interval stabilit. Pentru testare, am variat numărul de case din scenă (1, 3, 5, 9) și intervalele de creștere ale iederelor (0,7s; 0,5s; 0,3s; 0,1s). Am lăsat simularea să meargă 3 - 5 minute și am salvat evoluția performanței sistemului (framerate, cpu, gpu, ram usage) folosind programul MSI Afterburner². Am prelucrat fișierul cu log-urile despre performanță în Python pentru a realiza niște grafice comparative.

La prima rulare, cu o singură casă și creșterea iederei înceată, s-a observat o decădere a performanței într-un punct (Figura 10). În acel punct clădirea a început să se fragmenteze și, după terminarea fragmentării, în scenă au fost create sute de fragmente. Aceste fragmente ocupă memorie RAM și sunt afectate de fizică. Creșterea în folosirea procesorului este datorată

¹<https://github.com/scorpion81/blender-fracture/issues/5#issuecomment-170898611>

²<https://www.msi.com/page/afterburner>

Tabela 2: Clădirile folosite în proiect și timpii de fragmentare

Clădire	Număr Vârfuri	Număr Triunghiuri	Număr Etaje	Timp frag-mentare în proiect (secunde)	Timp frag-mentare în Blender (secunde)
	140	254	1	1.39	10.19
	168	272	2	1.95	11.35
	12 377	23 120	8	300.469	37.25

de simularea coliziunilor și legăturilor dintre fragmente. Această creștere a dus la micșorarea numărului de cadre pe secundă și reducerea folosirii plăcii grafice.

Fragmentele distruse vor fi eliminate din scenă după un anumit timp, ceea ce a dus la scădere bruscă a memoriei RAM ocupată. Creșterea care urmează este datorată creșterii iederei. Obiectul 3D se mărește progresiv, numărul de vârfuri și fețe crește și se ocupă mai multă memorie.

Am rulat testul cu o singură casă și cu o creștere mai rapidă a iederei și am comparat rezultatele în Figura 11. Se observă că fragmentarea clădirii a început mai târziu după creșterea bruscă a utilizării procesorului și a scăderii bruste a numărului de cadre pe secunde. De asemenea, se observă că memoria folosită a crescut semnificativ.

Ultimul test cu o singură clădire a fost cu creșterea iederei rapidă (la fiecare 0,1 secunde). Rezultatele se află în Figura 12. Deoarece pentru creșterea vegetației nu se folosește un thread

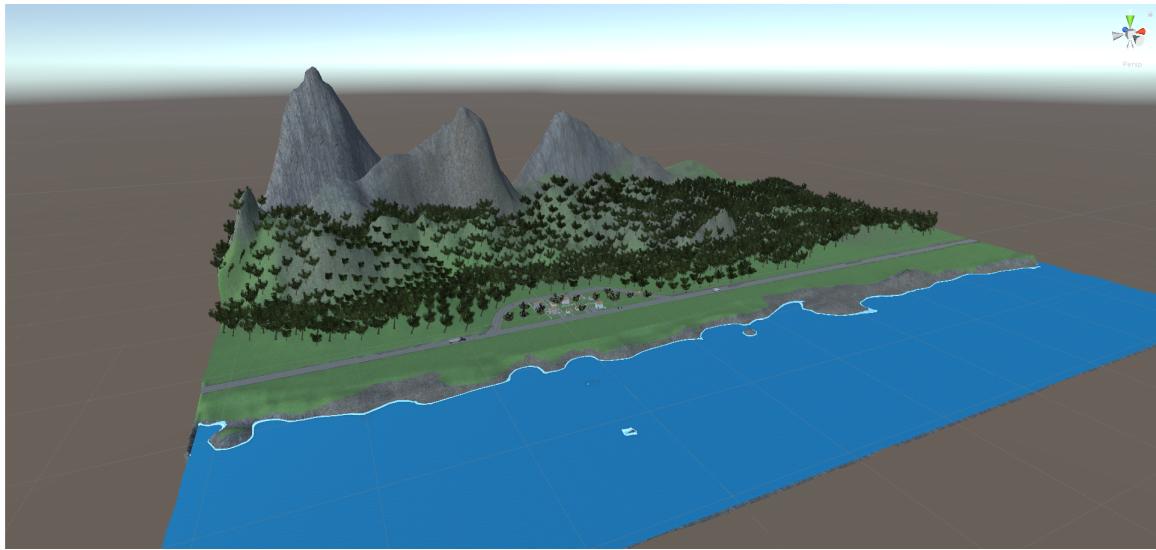


Figura 8: Harta folosită în etapa de testare văzută de departe



Figura 9: Harta folosită în etapa de testare văzută de aproape

nou (comparativ cu fragmentarea și explozia), operațiile se fac serial pe thread-ul principal. Pentru că aplicația funcționează la un număr mic de cadre pe secunde, nu se ajunge în stadiul de fragmentare a clădirii chiar dacă este rulată pentru 3 - 5 minute. De asemenea, datorită creșterii rapide a modelului 3D al iederei, memoria RAM folosită crește progresiv.

Folosind scena completă, cu toate cele 9 clădiri, și cu creștere lentă a iederei am obținut rezultatele din Figura 13.

Comparativ cu celelalte teste, se alocă mai multă memorie RAM pentru că mai multe clădiri trebuie fragmentate și se alocă memorie pentru mai multe obiecte 3D. Comparativ cu scena cu o singură clădire (Figura 14), framerate-ul scade mai devreme datorită volumului mare de iederă care trebuie crescută.

În final, în Figura 15 se observă cum memoria RAM scade, dar în scenă numărul de obiecte crește. Acest lucru este datorat de Garbage Collector-ului din Unity, care dezalocă

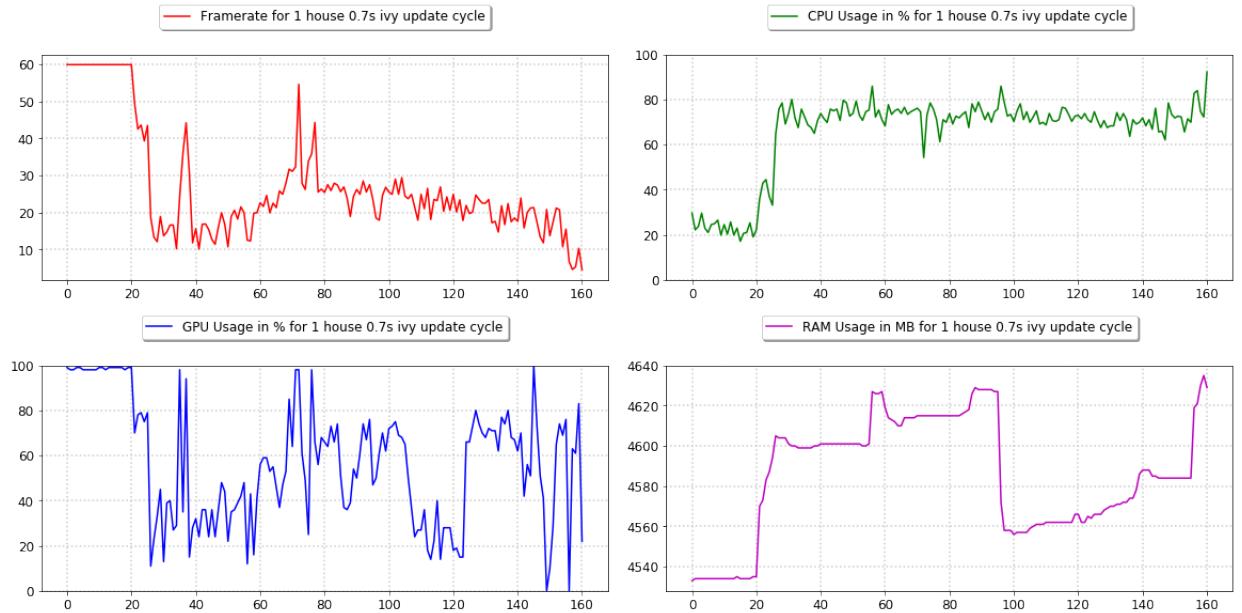


Figura 10: Evoluția performanței sistemului când aplicația rulează cu o singură clădire și iederă care crește încet (la 0,7 secunde)

memoria în transe.

După testare, se constată că aplicația poate funcționa și pe calculatoare low-end, dar necesită un timp de rulare mai mare pentru a genera fragmentelor și vegetației. Aplicația oferă un efect aproximativ realist, exemplu în Figura 17. Doar clădirile sunt afectate de distrugere, elementele decorative din scenă sunt afectate doar la nivel de shader. Iedera poate crește pe orice obiect din scenă atât timp cât acel obiect are coliziunile activate.

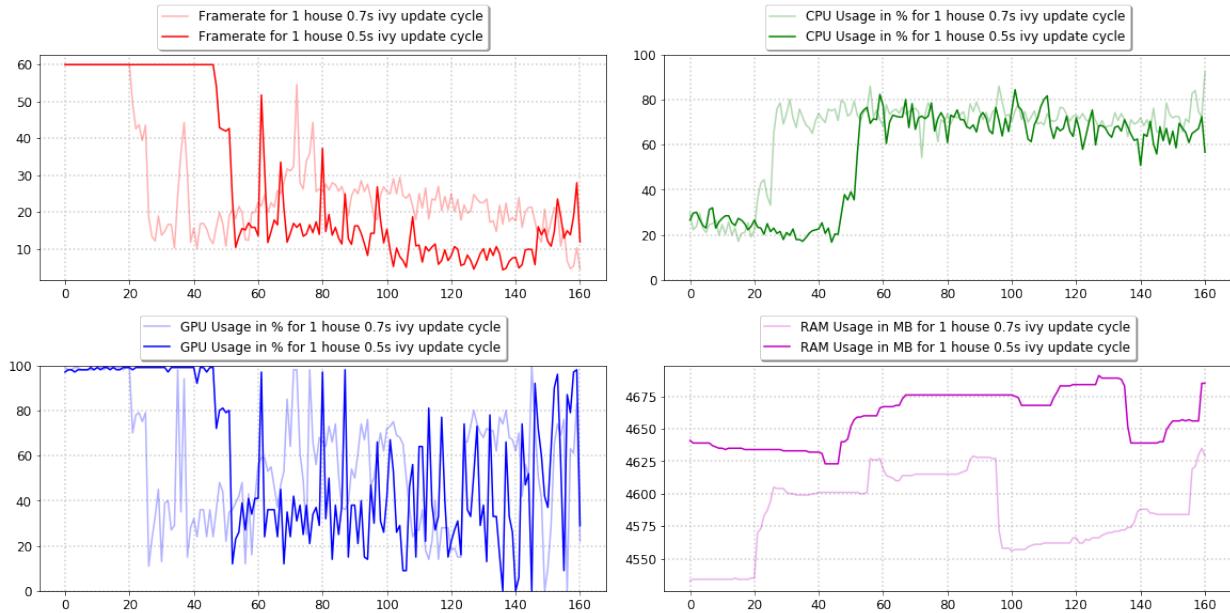


Figura 11: Comparație între rularea cu un interval de 0,7 secunde vs 0,5 secunde pentru creșterea iederei într-o scenă cu o singură clădire

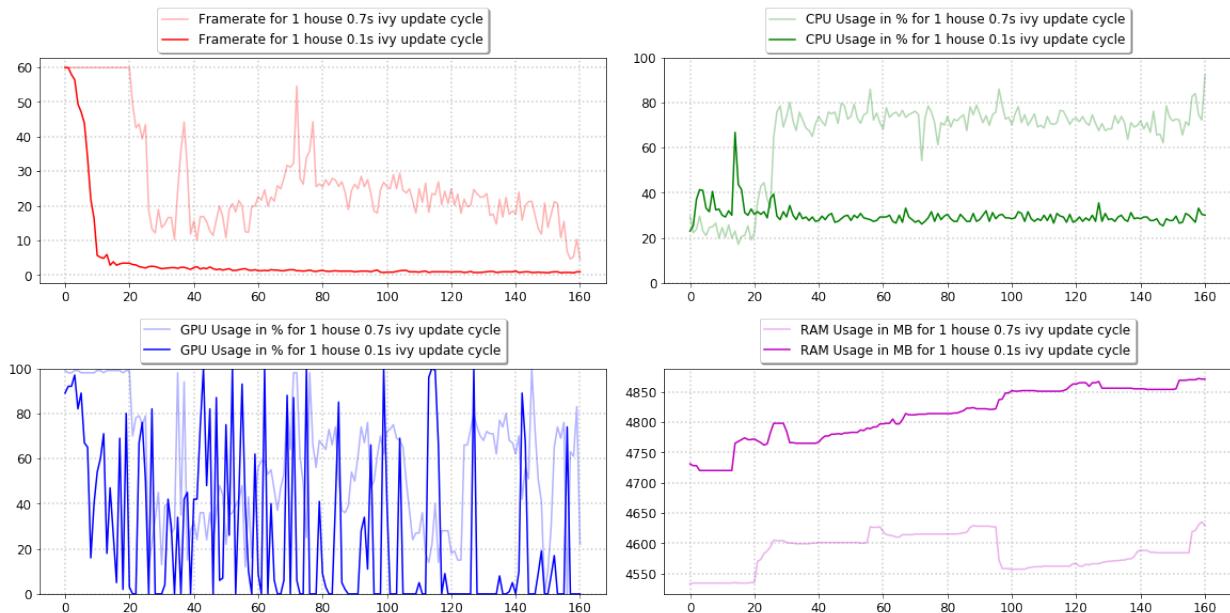


Figura 12: Comparație între rularea cu un interval de 0,7 secunde vs 0,1 secunde pentru creșterea iederei într-o scenă cu o singură clădire

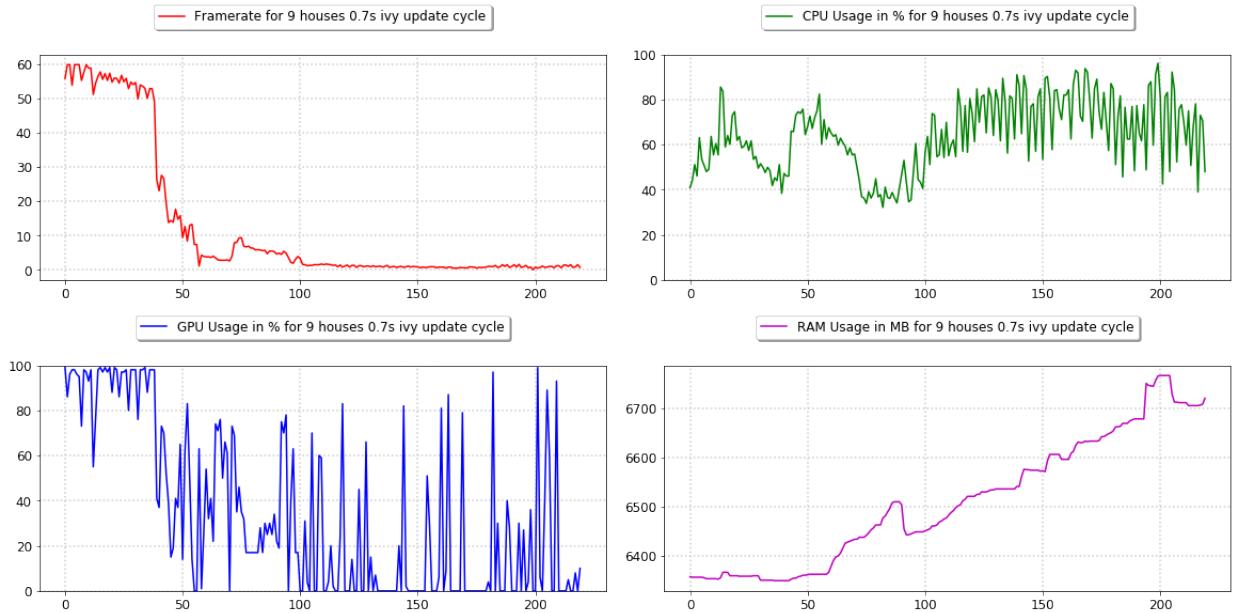


Figura 13: Evoluția performanței sistemului când aplicația rulează cu 9 clădiri și iederă care crește încet (la 0,7 secunde)

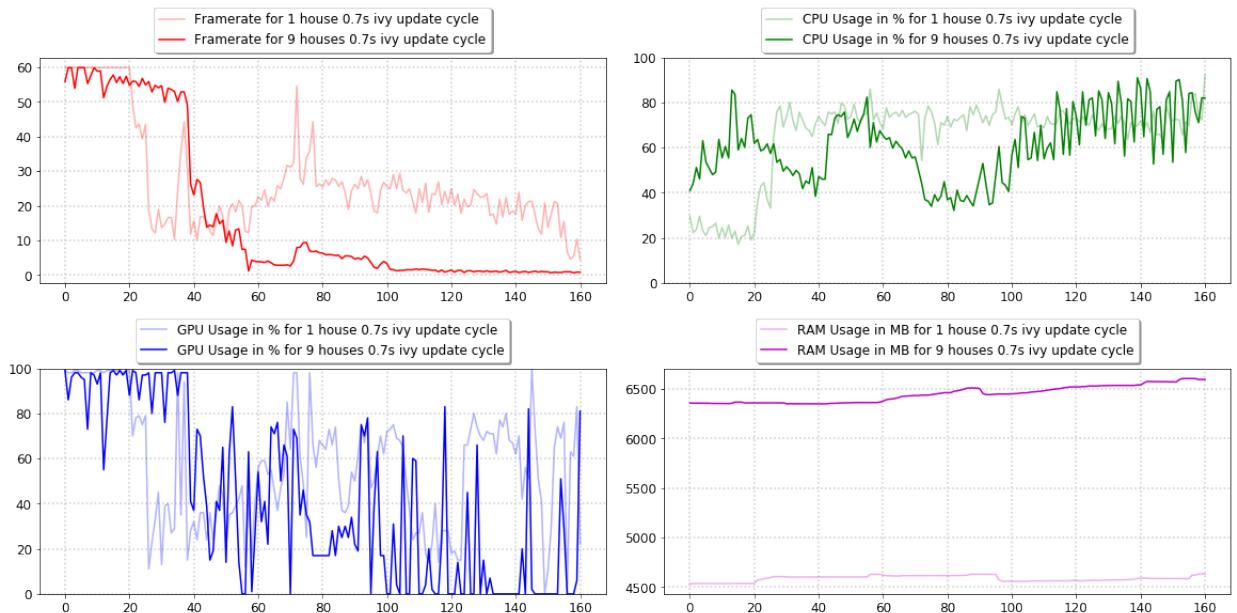


Figura 14: Comparație între rularea cu un interval de 0,7 secunde pentru creșterea iederei într-o scenă cu o singură clădire vs o scenă cu 9 clădiri

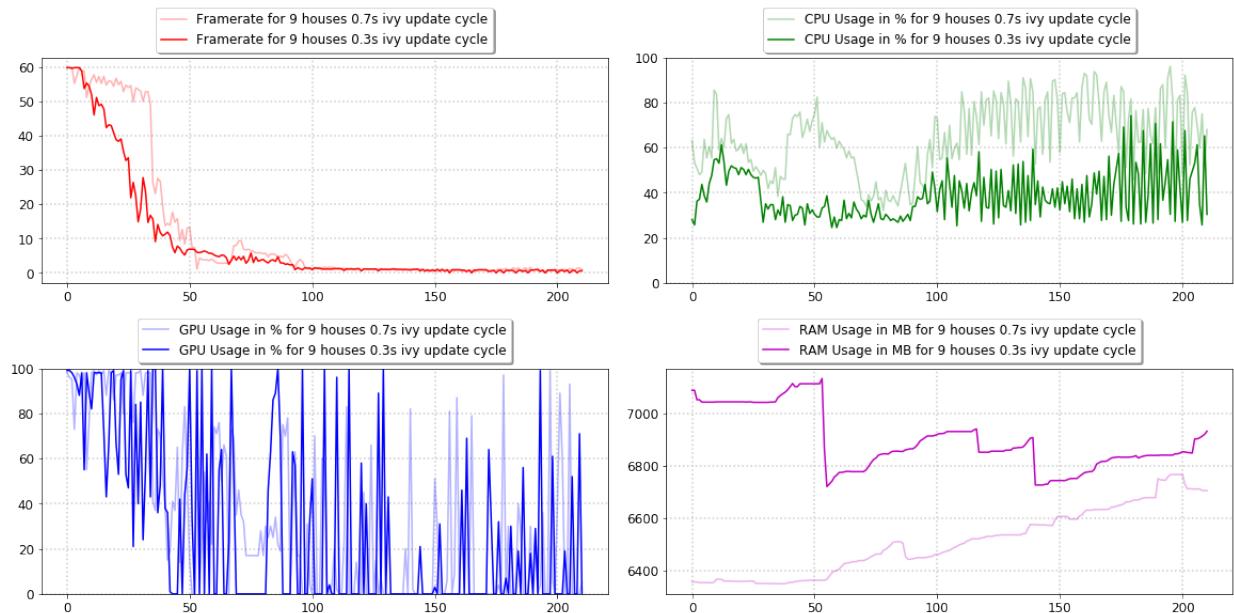


Figura 15: Comparație între rularea cu un interval de 0,7 secunde vs 0,3 secunde pentru creșterea iederei într-o scenă cu 9 clădiri

7 CONCLUZII

În acest proiect s-a plecat de la ideea de a crea o aplicație care să degradeze o scenă și să o facă să arate dintr-un scenariu post-apocaliptic. Proiectul s-a împărțit în 2 părți, una pentru distrugerea clădirilor, cealaltă pentru creșterea vegetației.

În partea de distrugere a clădirilor, am găsit 2 algoritmi de fragmentare a unui obiect 3D în timp real. Primul algoritm se bazează pe faptul că fețele unui obiect 3D sunt triunghiuri. Fiecare triunghi al feței este transformat într-un obiect separat și detasat de obiectul original [2]. Algoritmul poate fi extins, fiecare triunghi este, la rândul sau, separat în alte 3 triunghiuri până la un anumit prag. Acest algoritm este rapid, dar nu creează efecte realiste.

Celălalt algoritm de distrugere este mai complex. Obiectul 3D se împarte în volume încadratoare convexe, iar fiecare astfel de volum se împarte folosind diagrame Voronoi [1] 3D [3]. Efectul de distrugere este mai realist, dar algoritmul este mai încet datorită împărțirilor care necesită intersecții de volum.

În final acestei etape, am decis să creez un algoritm care să fie la mijlocul celor 2. Să iau triunghiurile obiectului 3D, să le împart folosind diagrame Voronoi [1], să le transform în prisme și să le detasez de obiectul original în momentul exploziei. Comparativ cu algoritmul de fragmentare din Blender, algoritmul are un timp de rulare mai bun pentru obiecte cu număr redus de fețe.

Pe fațadele clădirilor va crește iederă folosind algoritmul lui Thomas Luft [4] care ține cont de mai mulți factori precum direcția luminii din scenă, coliziuni și gravitație.

Proiectul a fost implementat folosind motorul Unity în limbajul de programare C# și limbajul de shading HLSL. Aplicația rulează destul de greu pentru un calculator low end chiar și cu o scenă minimalistă (o singură clădire care poate fi distrusă și vegetație care crește încet). Rezultatul este acceptabil, se poate observa în figurile 16 și 17 cum a crescut iedera pe clădiri, cum s-au distrus anumite părți din clădiri și cum a crescut mușchi pe fațadele caselor.

În momentul de față, creșterea iederei se face secvențial într-un singur cadru. Aplicația poate fi îmbunătățită prin împărțirea acestei creșteri în mai multe cadre (la fel cum sunt împărțite fragmentarea și explozia unui obiect 3D). Acest lucru va elimina momentele când aplicația îngheată când în scenă există vegetație multă care trebuie să crească.



Figura 16: Casă peste care a crescut iederă



Figura 17: Scenă post-apocaliptică creată folosind aplicația

BIBLIOGRAFIE

- [1] Eric Weisstein. Voronoi Diagram. *MathWorld*. Accesat 23 iunie 2020. <https://mathworld.wolfram.com/VoronoiDiagram.html>
- [2] Cherno (13 iulie 2015). Re: Script to break mesh into smaller pieces? Accesat 16 iunie 2020. <http://answers.unity.com/answers/1006344/view.html>
- [3] Anton Grönberg (18 iunie 2017). Real-time Mesh Destruction System for a Video Game (Dissertation). Accesat 13 iunie 2020. <http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-64533>
- [4] Thomas Luft (2007). Ivy Generator. Acesat 15 iunie 2020. <http://www.ivy-generator.com/>
- [5] Johan Knutzen (martie 2009). Generating Climbing Plants Using L-Systems. Accesat 15 iunie 2020. <http://www.cse.chalmers.se/~uffe/xjobb/climbingplants.pdf>

ANEXE



Figura 18: Clădire cu un perete fragmentat



Figura 19: Clădire aproape distrusă

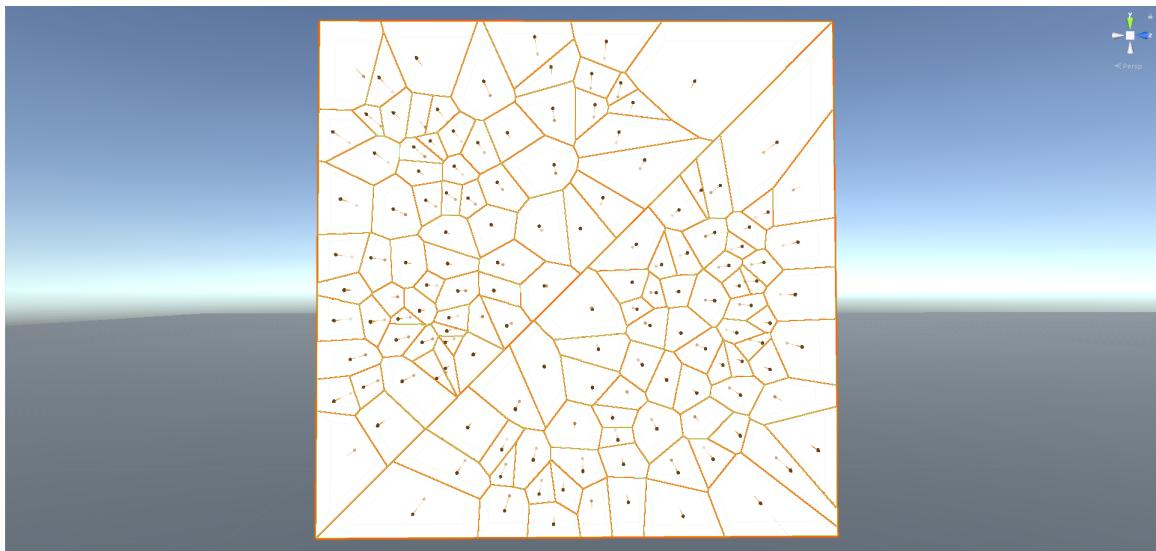


Figura 20: 2 triunghiuri fragmentate folosind diagrame Voronoi [1]

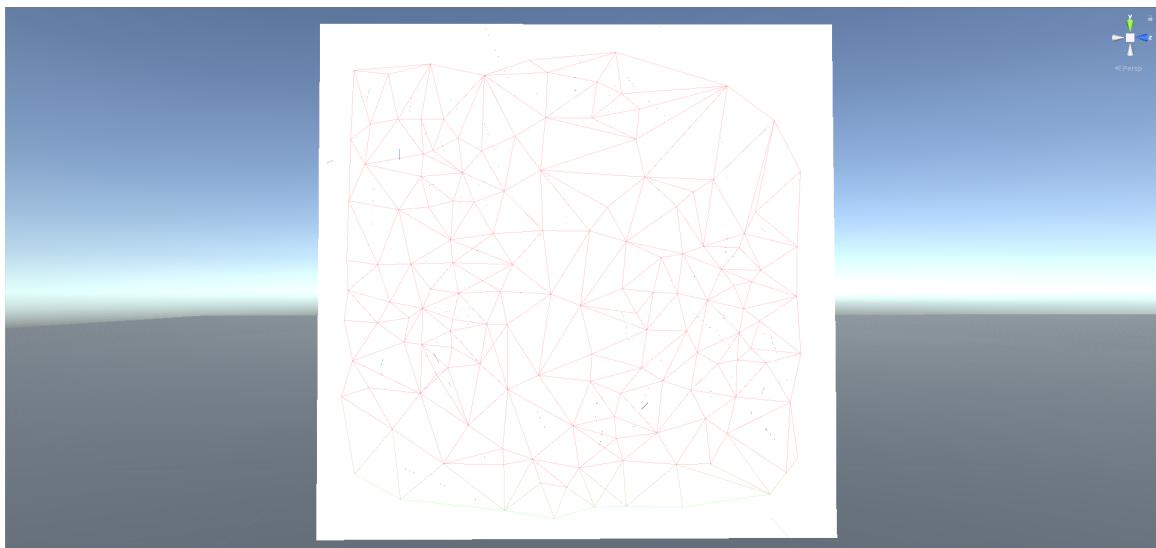


Figura 21: Graf neorientat de fragmente