

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Пояснювальна записка до дипломного проєкту

магістра

(освітній ступінь)

на тему «Метод процедурного синтезу тривимірних ландшафтів у геймдизайні»

ХАІ.603.667п2.121.1663110.200

Виконав: студент 6 курсу групи № 667п2
Спеціальність 121 – Інженерія програмного
забезпечення

(код та найменування)

Освітня програма Хмарні обчислення та
Інтернет речей

(найменування)

Ільніков А.О.

(прізвище й ініціали студента)

Керівник Шостак І.В.

(прізвище та ініціали)

Рецензент Іващенко Г.С.

(прізвище та ініціали)

Харків – 2020

Міністерство світи і науки України
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу
(повне найменування)

Кафедра інженерії програмного забезпечення
(повне найменування)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – інженерія програмного забезпечення
(код та найменування)

Освітня програма хмарні обчислення та Інтернет речей
(найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

І. Б. Туркін

(підпис)

(ініціали та прізвище)

“ ”

2020 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Ільнікову Антону Олександровичу

(прізвище, ім'я, по батькові)

1. Тема дипломного проекту Метод процедурного синтезу тривимірних ландшафтів у геймдизайні

керівник дипломного проекту Шостак Ігор Володимирович, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “ ” 2020 року №

2. Термін подання студентом роботи

3. Вихідні дані до роботи: способи та засоби синтезу тривимірних ландшафтів у геймдизайні

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
критичний аналіз способів та засобів синтезу тривимірних ландшафтів у геймдизайні;

розроблення методу процедурного синтезу тривимірних ландшафтів у геймдизайні;

розробка програмного додатку на основі розробленого методу;

проведення порівняльного аналізу розробленого ПЗ з аналогами

5. Перелік графічного матеріалу

РПЗ – стор. 80, рисунків – 14 шт., таблиць – 2 шт., презентація – 18 слайдів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Шостак І.В., проф. каф. 603		
2	Шостак І.В., проф. каф. 603		
3	Шостак І.В., проф. каф. 603		

8. Нормоконтроль _____ В.А. Постернакова « ____ » _____ 2020 р.
(підпис) (ініціали та прізвище)

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	Отримання і затвердження теми диплому	03.09.2019	
2	Аналіз предметної області	04.09.2019	
3	Постановка задачі	20.11.2019	
4	Проведення теоретичних досліджень	22.11.2019	
5	Розробка прототипу ПЗ	02.09.2020	
6	Підготовка пояснювальної записки	22.10.2020	
7	Оформлення пояснювальної записки до дипломного проекту	10.11.2020	
8	Передзахист дипломного проекту	27.11.2020	
9	Захист дипломного проекту	18.12.2020	

Студент

_____ (підпис)

Ільніков А.О.

(прізвище та ініціали)

Керівник роботи

_____ (підпис)

Шостак І.В.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту містить 80 стор., 14 рис., 1 додаток, 33 джерела.

Об'єкт дослідження – процес синтезу віртуальних пейзажів у геймдизайні.

Предмет дослідження – моделі та методи синтезу тривимірних ландшафтів у геймдизайні.

Метою дослідження є підвищення ефективності синтезу тривимірних ландшафтів у геймдизайні шляхом використання процедурної генерації ландшафтів.

Для досягнення поставленої мети необхідно вирішити ряд завдань: провести критичний аналіз способів та засобів синтезу тривимірних ландшафтів у геймдизайні; розробити метод процедурного синтезу тривимірних ландшафтів у геймдизайні; розробити програмний додаток на основі розробленого методу; провести порівняльний аналіз розробленого ПЗ з аналогами.

Методи досліджень. В роботі використовуються методи комп'ютерного моделювання, статистичні та емпіричні методи.

Наукова новизна. Удосконалено метод процедурного синтезу тривимірних ландшафтів у геймдизайні, який на відміну від існуючих використовує геодезичні карти як джерело інформації про ландшафт, що дозволяє підвищити рівень реалістичності графічного моделювання.

Практична значимість отриманих результатів роботи полягає у можливості застосування отриманих результатів для ефективного використання геодезичних карт для процедурної генерації ландшафтів.

**ПРОЦЕДУРНА ГЕНЕРАЦІЯ, КОМП'ЮТЕРНА ГРАФІКА, ОБРОБКА
ЗОБРАЖЕНЬ, ГЕНЕРАЦІЯ ЛАНДШАФТІВ**

ABSTRACT

Explanatory note to the master's thesis 80 pp., 14 fig., 1 app., 33 sources.

The object of study - the process of synthesis of virtual landscapes in game design.

The subject of research - models and methods of synthesis of three-dimensional landscapes in game design.

The aim of the study is to increase the efficiency of the synthesis of three-dimensional landscapes in game design by using procedural generation of landscapes.

To achieve this goal it is necessary to solve a number of tasks: to conduct a critical analysis of methods and means of synthesis of three-dimensional landscapes in game design; to develop a method of procedural synthesis of three-dimensional landscapes in game design; develop a software application based on the developed method; to conduct a comparative analysis of the developed software with analogues.

Research methods. The work uses methods of computer modeling, statistical and empirical methods.

Scientific novelty. The method of procedural synthesis of three-dimensional landscapes in game design is improved, which, unlike the existing ones, uses geodetic maps as a source of information about the landscape, which allows to increase the level of realism of graphic modeling.

The practical significance of the obtained results of the work is the possibility of applying the obtained results for the effective use of geodetic maps for the procedural generation of landscapes

PROCEDURAL GENERATION, COMPUTER GRAPHICS, IMAGE PROCESSING, LANDSCAPE GENERATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ	11
1.1 Задача процедурної генерації ландшафтів	11
1.2 Джерела даних про ландшафти	12
1.3 Приклади використання процедурної генерації ландшафтів	13
1.4 Внутрішнє представлення ландшафту	14
1.4.1 Полігонна сітка.....	14
1.4.2 Триангульована нерегулярна сітка.....	15
1.4.3 Звичайні карти та поля висот	15
1.4.4 Потокова сітка	16
1.4.5 Вокселі.....	17
1.5 Текстурування згенерованого ландшафту.....	18
1.6 Навколишнє середовище	18
1.7. Методи, які використовуються в галузі генерації рельєфу	18
1.8 Постановка мети і завдань дослідження.....	20
1.9 Висновки по розділу 1	21
2 МЕТОД ПРОЦЕДУРНОГО СИНТЕЗУ ТРИВИМІРНИХ ЛАНДШАФТІВ У ГЕЙМДИЗАЙНІ	32
2.1 Вихідні дані методу.....	32
2.1.1 Інформація про ландшафт	33
2.2 Отримання загальної карти висот.....	34

2.3 Обробка зображень	37
2.4. Додавання малих деталей.....	49
2.4. Генерація шуму	49
2.7 Висновки по розділу 2	51
3 РОЗРОБЛЕННЯ ПРОТОТИПУ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ПРОЦЕДУРНОГО СИНТЕЗУ ТРИВИМІРНИХ ЛАНДШАФТІВ.....	52
3.1 Аналіз геодезичних карт.....	52
3.2. Уточнення карти висот	56
3.4 Проведення текстуровання.....	59
3.5 Перевірка ефективності розробленого ПП	67
3.5 Висновки по розділу 3	74
ВИСНОВКИ.....	75
ПЕРЕЛІК ПОСИЛАНЬ	76
ДОДАТОК А	80

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ,
СКОРОЧЕНЬ І ТЕРМІНІВ**

ПЗ – програмне забезпечення.

ПП – програмний продукт.

ВСТУП

Актуальність. Процедурне створення контенту - це процес створення медіа ресурсів за допомогою комп'ютерних алгоритмів. Створення та візуалізація віртуальних пейзажів є актуальним завданням у різних сферах, таких як розроблення навчальних та моделюючих середовищ, відеоігор, створення матеріалів для фільмів та анімації, а також середовища для моделювання різних природних процесів. З огляду на зростаючу популярність пристроїв віртуальної реальності, тема стає ще актуальнішою. Це призводить до необхідності розробки нових, швидких і ефективних методів генерації ландшафтів. Більшість алгоритмів сучасного покоління створюють ландшафти, які потребують подальшого ручного перегляду дизайнерами або потребують автоматичного завершення складними алгоритмами ерозії, накладання кліматичних та погодних особливостей. Такий підхід добре підходить для програм, де потрібно створити єдину досить невелику площу або коли ландшафти налаштовуються вручну, але не підходять для генерації великих територій місцевості в повністю автоматичному режимі.

Об'єкт дослідження – процес синтезу віртуальних пейзажів у геймдизайні.

Предмет дослідження – моделі та методи синтезу тривимірних ландшафтів у геймдизайні.

Метою дослідження є підвищення ефективності синтезу тривимірних ландшафтів у геймдизайні шляхом використання процедурної генерації ландшафтів.

Для досягнення поставленої мети необхідно вирішити ряд завдань:

- провести критичний аналіз способів та засобів синтезу тривимірних ландшафтів у геймдизайні;
- розробити метод процедурного синтезу тривимірних ландшафтів у геймдизайні;

- розробити програмний додаток на основі розробленого методу;
- провести порівняльний аналіз розробленого ПЗ з аналогами.

Методи досліджень. В роботі використовуються методи комп'ютерного моделювання, статистичні та емпіричні методи.

Наукова новизна. Удосконалено метод процедурного синтезу тривимірних ландшафтів у геймдизайні, який на відміну від існуючих використовує геодезичні карти як джерело інформації про ландшафт, що дозволяє підвищити рівень реалістичності графічного моделювання.

Практична значимість отриманих результатів роботи полягає у можливості застосування отриманих результатів для ефективного використання геодезичних карт для процедурної генерації ландшафтів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Задача процедурної генерації ландшафтів

Створення та візуалізація віртуальних пейзажів є актуальним завданням у різних сферах, таких, як розроблення навчальних та моделюючих середовищ, відеоігор, створення матеріалів для фільмів та анімації, а також середовища для моделювання різних природних процесів. З огляду на зростаючу популярність пристроїв віртуальної реальності, тема стає ще актуальнішою. Це призводить до необхідності розробки нових швидких і ефективних методів генерації ландшафтів. Більшість алгоритмів сучасного покоління створюють ландшафти, які потребують подальшого ручного перегляду дизайнерами або потребують автоматичного завершення складними алгоритмами ерозії, клімату та погодних штамів. Такий підхід добре підходить для програм, де потрібно створити єдину досить невелику площу або коли ландшафти налаштовуються вручну, але не підходять для генерації великих територій місцевості в повністю автоматичному режимі.

Генерація процедурного вмісту має мати три важливі особливості. По-перше, вона повинна бути швидкою, а це означає, що вона повинна використовувати лише частку обчислювальної потужності комп'ютерних систем з поточним поколінням. По-друге, вона повинна бути, як випадковою, так і структурованою таким чином, щоб створювати різноманітний і реалістичний вміст. По-третє, вона повинна бути керованою і інтуїтивною [1].

Пейзаж слугує двом важливим цілям у візуалізації сцени. По-перше, він є основним, окремим будівельним блоком на якому розміщуємо всі інші об'єкти на сцені. Розміщення всіх інших об'єктів на ньому також допомагає нам швидше розпізнавати напрям «вгору» та орієнтацію у віртуальному просторі. По-друге, пейзажі також використовуються для значного збільшення загального реалізму

всієї сцени. Якщо наведені віртуальні об'єкти розміщені на поверхні ландшафту, то загальне відчуття сцени стане набагато достовірнішим та реалістичним. Саме тому поверхня повинна містити деталі, які не можуть бути повністю відтворені лише застосовуючи прості текстури. Використання ландшафту, додає відсутні деталі на сцену та значно підвищує загальний реалізм [2].

1.2 Джерела даних про ландшафти

Перш, ніж розпочати рендеринг ландшафту, обов'язково необхідно мати дані про цей ландшафт. Так, як ландшафт представлений у комп'ютері так само, як будь-який інший об'єкт, його цифрові дані можна отримати за допомогою одного з трьох методів — шляхом сканування, моделювання або створення об'єкта.

Сканування місцевості:

- При скануванні реальних об'єктів, камера або аналогічний пристрій використовується для сканування поверхні об'єкта. В особливих випадках внутрішня частина об'єкта також сканується, якщо це можливо (і потрібно). Те ж саме стосується і ландшафту — єдиною відмінністю є розмір камери. Земля регулярно буває сфотографована з аерографів та супутників, в результаті чого фотографії з високою роздільною здатністю поверхні Землі є загальнодоступними. Однією з переваг цього підходу є те, що створювана ним місцевість виглядає більш реалістичною, ніж місцевість, створена будь-яким іншим методом — при спробі наслідувати природу найпростішим рішенням є копіювання вже існуючого ландшафту.

Моделювання ландшафту:

- Інший варіант отримання достовірних даних про місцевість — це моделювання всього бажаного рельєфу у відповідному програмному забезпеченні 3D-моделювання (наприклад, Blender, Cinema 4D, Maya або 3DS Max). Таким чином, користувач повністю контролює кінцевий вигляд місцевості, що є перевагою, якщо користувач створює навмисно артистичний світ. В іншому

випадку, це є недоліком, тому що моделювання реальної місцевості, як правило, є дуже складним завданням.

Генерація місцевості:

- Цей підхід полягає у створенні алгоритму, який генерує дані про певну поверхню рельєфу. Формування місцевості є найбільш корисним в ситуаціях, коли велику роль грає розмір доступної пам'яті для збереження ландшафту. Алгоритм може генерувати теоретично нескінченний ландшафт місцевості в реальному часі практично не використовуючи диск (єдиний виняток є диск підкачки). З іншого боку, все залежить від обраного алгоритму, його налаштувань і складності — деякі алгоритми дуже швидко випускають свої результати, але їх результати не надто реалістичні, а інші працюють довго та видають дуже реалістичні результати.

Комбінація попередніх підходів:

- Комбінація підходу отримання даних про ландшафт, а потім заповнення його деталями. Зазвичай це, з одного боку, ручна підгонка ландшафту, а потім — використання певного ітеративного алгоритму для додавання більшої деталізованості — або навпаки, генерація всієї місцевості та ручне додавання деталей.

1.3 Приклади використання процедурної генерації ландшафтів

Розглянемо деякі приклади застосування процедурної генерації ландшафтів.

1 Тренажери для підготовки військовослужбовців. Процедурна генерація використовується для генерації тренажерів що дозволяють динамічно візуалізувати необхідні ландшафти. Вони використовуються для симуляції наземних операцій, польотів авіації тощо.

2 ПЗ для створення ландшафтів для програм навігації. Постійне збільшення продуктивності персональних комп'ютерів зробило можливим використання віртуальних додатків, подібних до NASA World Wind⁶ та Google Earth⁷. Оскільки одним із ключових завдань, що вирішується за допомогою

таких програм є навігація, багато їх функцій поступово впроваджуються в навігації GPS. Так, однією з основних функцій, які вже впроваджуються в останніх поколіннях пристроїв, є навігація у тривимірній місцевості, яка замінює 2D-карти. Це покращує орієнтацію у просторі та підвищує чіткість навігації для більшості людей.

3 Кіно та телебачення, відеоігри. На сьогодні при створенні кінематографічних творів широко використовуються програмні засоби для генерації ландшафтів. Прикладами такого ПЗ є World Machine 1, VistaPro2, Bryce3, Vue4 та Terragen5. Зокрема, програмні пакети Vue і Terragen використовувалися таких фільмах, як Time Machine, Day after tomorrow та багатьох інших.

Отже, віртуальні пейзажі у сучасному інформаційному світі використовуються у великій кількості медіа-продуктів: у тренажерах та інших програмних застосунках, що потребують реалістичної візуалізації, у фільмах та відеоіграх, в анімації, у новинах на телебаченні та в Інтернет-застосуваннях.

1.4 Внутрішнє представлення ландшафту

Зберігання формалізованої інформації про згенерований ландшафт може надати декілька наступних переваг:

- швидка передача даних, що забезпечує швидкий доступ;
- найнижчі накладні витрати на пам'ять;
- проста модифікація існуючих даних. Існує багато можливих способів зберігання інформації про ландшафти на диску та / або в ОЗП.

1.4.1 Полігонна сітка

Оскільки поверхня рельєфу складається з вершин і граней, стає можливо зберігати її у вигляді звичайної багатокутної сітки. Головною перевагою цього підходу є його універсальність — кожен згенерований ландшафт можна представити і зберегти у вигляді загальної багатокутної сітки. З іншого боку,

його універсальність також є одним із недоліків. Полігонні сітки, як правило, дуже добре підходять до статичних поверхонь, але дуже непрактичні для динамічно змінюваної місцевості. Ще один недолік полягає у тому, що велика частина алгоритму в значній мірі залежить від використовуваного формату та експортера. Також ще однією проблемою є те, що для ситуацій коли необхідно оглянути частину мешу, ми все одно повинні завантажити його повністю, що негативно впливатиме на об'єм оперативної пам'яті [3].

1.4.2 Триангульована нерегулярна сітка.

Концепція триангульованої нерегулярної сітки (Triangulated irregular network - TIN), є досить простою, оскільки це всього лиш спрощена полігональна сітка. Формально це сукупність сусідніх непересічних трикутників, вершини яких розташовані адаптивно над моделлю вершин. Це лежить в основі головної переваги TIN, яка полягає в змінній величині вершин, які будуть використовуватися для зберігання даних про місцевість. Коли справа доходить до гір, кількість вершин збільшується, а TIN може зберігати будь-яку необхідну деталь, тоді мережеві трикутники будуть дуже малими. Однак, дуже багато вершин оптимізуються, коли мова йде про більш плоскі поверхні, і один трикутник може представляти досить велику площу землі. Основною проблемою, пов'язаною з TIN, є процес їх створення. TIN можуть бути або вручну змодельовані (що може бути незручним) або отримані в результаті процесу триангуляції існуючої сітки. Початкова модель введення завантажується (або генерується), кількість вершин оптимізується відповідним алгоритмом, якщо це необхідно, і триангулюється [4].

1.4.3 Звичайні карти та поля висот

У роботі з великою кількістю даних про місцевість зручно максимально їх спростити. Звичайна карта висот це двовимірний прямокутний масив параметрів

висоти, індекси яких є точками сітки (значення осей X і Z), а значення є значенням висоти (значення на вісь Y) в цій точці.

Серед основних переваг регулярних висотних площ є:

1Простота використання - дуже легко змінити геометрію місцевості, незалежно чи на місцевому, чи глобальному рівні;

2Менше споживання пам'яті: все що потрібно зберегти - це фактичні значення висоти та відстань між двома точками на осі. Після цього всі координати X - та Z -вершин можна безперешкодно обчислити під час виконання програми.

На жаль, звичайні карти висот не можуть відобразити печери та завитки, які послідовно генеруються - кожна точка показує лиш одне значення висоти [5].

Звичайні поля висот, дуже схожі на карти висот. Поля висот - це двовимірне растрове зображення, кожний піксель якого являє собою індивідуальне значення висоти в точці сітки. Вищі значення кольору зазвичай представляють більшу висоту, і навпаки.

Одна з двох проблем, що виникає, полягає в обмеженості діапазону значень пікселів на зображенні. Зображення чорно-білого кольору, яке використовує 8 біт на піксель, може мати не більше 256 різних значень.

Для більшості випадків це незручно, оскільки це спричиняє нереалістичні, помітні переходи між окремими рівнями висоти.

Іншою проблемою є нездатність зберігати будь-які додаткові дані на зображенні, особливо реальну відстань між двома сусідніми пікселями.

Незважаючи на ці недоліки, звичайні поля і карти висот є надзвичайно поширеними.

1.4.4 Потокова сітка

Якщо є необхідність відтворити великі ландшафти, то може бути, що багатокутна сітка буде занадто великою для збереження в пам'яті повністю. Одне з можливих рішень полягає в тому, щоб розділити сітку на кластери і

завантажувати лише необхідні кластери в оперативну пам'ять. Інший варіант полягає в тому, щоб використовувати потокову сітку, яка є «логічною послідовністю індексованих вершин та трикутників з інформацією, коли вони є необхідними і коли необхідність в них відпадає». Іншими словами, це сітка де сусідні вершини та трикутники, логічно організовані та погруповані ближче один до одного, що б надати змогу не зберігати весь меш в пам'яті. Завдяки реорганізації сітки, можна динамічно завантажувати та вивантажувати потрібні частини моделі під час виконання без необхідності завантаження всього файлу [6].

Технологія потокової сітки використовується у багатьох сучасних додатках, які вимагають візуалізації великих, детальних світів, наприклад, Google Планета Земля або більшість відеоігор, з відкритим світом.

1.4.5 Вокселі

Воксель це аналог пікселя в дискретному тривимірному просторі. Це найменший елемент, який забарвлений одним кольором. Формою вокселя зазвичай є куб. Використання вокселів для об'ємного зображення дозволяє нам обчислювати не тільки поверхню рельєфу, але і підземелля. Вони є, особливо, корисними та пізнавальними у наукових програмах та дослідженнях, де потрібно моделювати фізичні закони. Також варто зауважити, що вокселі є надзвичайно вимогливими до швидкої дії комп'ютера в порівнянні з сітками, як з точки зору обсягу пам'яті, так і швидкості обробки — що є ще однією причиною, чому вони зазвичай не використовуються. Проте, все більше і більше розробників експериментують з вокселями, і вони знаходять своє використання в кількох додатках за останні кілька років (особливо широко у галузі відеоігор) [7].

1.5 Текстурування згенерованого ландшафту

Іншим фактором, який значно сприяє реалістичності поверхні місцевості є застосування текстури поверхні, що може бути зроблено двома різними способами [8]:

- у випадку моделювання існуючого ландшафту є можливість використати зображення супутника цього ландшафту;
- в іншому випадку, можливо звести відповідність текстури до місцевості. Крім того, можливо використовувати кілька різних текстур, щоб диференціювати їх серед біотопів і змішувати їх на кордонах.

1.6 Навколишнє середовище

Одночасно, з рендерингом ландшафту, є і інші елементи, що допомагають покращити остаточний вигляд сцени [9]:

- Skybox та хмари — якщо додаємо фон та завернемо сцену в skybox, максимально посилюється реалістичне відчуття сцени в цілому. Більше того, як правило, це додасть відчуття висоти та розміру ландшафту.
- Освітлення — правильне налаштування освітлення дуже важливе у тому випадку, якщо необхідно досягти природного почуття сцени. Крім того, якщо ландшафт відкидає тіні на себе та на інші об'єкти то просторове сприйняття значно покращується.
- • Поверхня води. Також можемо встановити певну висоту, як рівень моря та зробити витіювату місцевість з озерами або океаном з кількома поверхнями острівної води, що призведе до збільшення різноманіття сцен.

1.7. Методи, які використовуються в галузі генерації рельєфу

Розглянемо Броунівську поверхню.

Броунівська поверхня — це двовимірна поверхня, заснована на фрактальній теорії, розробленій Бенотом Б. Мандельбротом (1975р.). Це розширення поняття одновимірного броунівського руху до двох вимірів.

Нехай X — точка, яка рухається у горизонтальній осі з постійною швидкістю. Ця точка вертикально відхилюється випадковими імпульсами, що є нормально розподіленими з однаковими інтервалами. Одномірний броунівський рух (далі позначений як W_m) є сукупною функцією цих випадкових імпульсів $W(t)$ з нормальним розподілом $N(0,1)$.

Двомірне поле W_m , яке іноді називається фракційною поверхнею Брєнґі, є однорідним розширенням БМ-контур без одного розміру.

Реальна поверхня Землі найбільш точно нагадує W_m , створений при $D = 2.3$, оскільки середній фрактальний розмір поверхні Землі дорівнює приблизно 2.3. Проте, навіть, поверхні, створені за допомогою D , трохи менше або більше, ніж 2.3, можуть бути досить знайомі. Це відбувається тому, що фрактальний вимір є нерівним в реальному світі, і його величина відрізняється залежно від того, де ми робимо замір. Для Землі коефіцієнт D лежить в інтервалі 2,1-2,5 [10].

Розглянемо алгоритм переміщення середнього положення. Алгоритм переміщення середньої позиції (також відомий як алмазний квадрат) — це алгоритм генерації рельєфу, введений в 1982 році Фурньєром, Фусселем та Карпентерім, основною ціллю якого є більш глибоке та тонке налаштування броунівського руху. Одномірна варіація алгоритму:

1 Встановити кутові значення f_1 і f_2 на значення згенеровані звичайним розподілом.

2 Обчислити значення висоти центральної точки

$$f_{min} = \frac{f_1 + f_2}{2}$$

де f_{min} - висота центральної точки,

f_1 і f_2 — дискретний інтервал з регулярно розташованими точками.

3 Повторно повторювати кроки 2, 3 по всьому інтервалу, поки вони містять більше однієї точки.

Дана версія алгоритму обмежена певною кількістю точок, а отже, і ступенем деталізації. Тим не менш, його можна реалізувати таким чином, що додаткові ітерації додатково обчислюються при збільшенні на кривій [11].

Розглянемо Метод випадкових помилок, який є одним із найпростіших методів генерації рельєфу. Цей метод спрямований на моделювання сил природи, таких, як розділення тектонічних плит та ерозії берегової лінії.

Основний принцип полягає у тому, що в основі алгоритму лежить випадкове «псування» полігональної сітки, яка інтерпретується, як карта висот.

Це ітераційний метод, який починається з дискретної двовимірної прямокутної сітки Ω та значення помилки δ .

У першій частині, всі значення з встановлюються з первинною висотою разом із значенням похибки δ , яка встановлена для початкового значення. Після цього повторюються наступні кроки $n \in \mathbb{N}$ разів:

- 1 Дві точки A та B , де $A \neq B$ і які обидві лежать в сітці Ω , обрані псевдовипадково.
- 2 Точки A та B чітко визначають лінію l , яка містить обидві точки.
- 3 Значення висоти всіх точок в сітці Ω , що лежать на одній стороні від прямою змінюються на $+\delta$. Значення висот в інших точках змінюється на $-\delta$.
- 4 Зменшити значення помилки δ за заданою формулою, але таким чином, щоб він залишався більшим за 0.

Алгоритм закінчує свою роботу після певного, наперед визначеного, числа ітерацій. Звичайно, він також може бути реалізований таким чином, щоб обчислювати наступні ітерації, поки результат візуально не буде задовільним і після цього бути зупиненим вручну [12].

1.8 Постановка мети і завдань дослідження

Метою дослідження є підвищення ефективності синтезу тривимірних ландшафтів у геймдизайні шляхом використання процедурної генерації ландшафтів.

Для досягнення поставленої мети необхідно вирішити ряд завдань:

- провести критичний аналіз способів та засобів синтезу тривимірних ландшафтів у геймдизайні;
- розробити метод процедурного синтезу тривимірних ландшафтів у геймдизайні;
- розробити програмний додаток на основі розробленого методу;
- провести порівняльний аналіз розробленого ПЗ з аналогами.

1.9 Висновки по розділу 1

В першому розділі проведено огляд та аналіз способів та засобів синтезу тривимірних ландшафтів у геймдизайні. Описані джерела вихідних даних для синтезу тривимірних ландшафтів. Зроблено огляд методів автоматичної генерації ландшафтів. Обрано та описано процедурну генерацію ландшафтів, як автоматичного методу їх побудови. Зроблено постановку мети й завдань дослідження.

2 МЕТОД ПРОЦЕДУРНОГО СИНТЕЗУ ТРИВИМІРНИХ ЛАНДШАФТІВ У ГЕЙМДИЗАЙНІ

2.1 Вихідні дані методу

Для генерації ландшафту обов'язково необхідні дані про ландшафт, їх можна отримати 4 способами:

- сканування місцевості - при скануванні реальних об'єктів, камера або аналогічний пристрій використовується для сканування поверхні об'єкта. В особливих випадках внутрішня частина об'єкта також сканується, якщо це можливо (і потрібно). Те ж саме стосується і ландшафту — єдиною відмінністю є розмір камери. Земля періодично буває сфотографована з аерографів та супутників, в результаті чого фотографії з високою роздільною здатністю поверхні Землі є загальнодоступними. Однією з переваг цього підходу є те, що створювана ним місцевість виглядає більш реалістичною, ніж місцевість, утворена будь-яким іншим методом — при спробі наслідувати природу найпростішим рішенням є копіювання вже існуючого ландшафту [13];

- моделювання ландшафту - інший варіант отримання даних про місцевість — це моделювання всього бажаного рельєфу у відповідному програмному забезпеченні 3D-моделювання (наприклад, Blender, Cinema 4D, Maya або 3DS Max). Таким чином, користувач повністю контролює кінцевий вигляд місцевості, що є перевагою, якщо користувач створює навмисно артистичний світ. В іншому випадку, це є недоліком, тому що моделювання реальної місцевості, як правило, є дуже важким завданням;

- генерація місцевості: цей підхід полягає в створенні алгоритму, який генерує дані про певну поверхню рельєфу. Формування місцевості є найбільш корисним в ситуаціях, коли велику роль грає розмір доступної пам'яті для збереження ландшафту. Алгоритм може генерувати теоретично нескінченний

ландшафт місцевості в реальному часі практично не використовуючи диск (єдиний виняток є диск підкачки). З іншого боку, все залежить від обраного алгоритму, його налаштувань та складності — деякі алгоритми дуже швидко генерують результат, але їх результати не дуже реалістичні, а інші працюють тривалий час, проте, видають дуже реалістичні результати [14];

– комбінація попередніх підходів: комбінація підходу отримання даних про ландшафт, а згодом наповнення його деталями. Зазвичай це, з одного боку, ручна підгонка ландшафту, а потім — використання певного ітеративного алгоритму для додавання більшої деталізованості — або навпаки, генерація всієї місцевості та ручне додавання деталей.

Метод генерації повинен високий ступінь впливу користувача на процес генерації та надавати естетично привабливий, реалістичний результат. Саме тому, способом отримання даних про ландшафт було обрано комбінацію зчитування інформації про ландшафт та генерації даних про ландшафт. Поєднавши ці два методи зможемо досягти високої якості згенерованих даних, підтримуючи вплив користувача на результат. Таким чином, результат не буде випадковим, користувач матиме змогу генерувати дані для вже існуючого ландшафту.

2.1.1 Інформація про ландшафт

Джерелом інформації про ландшафт були вибрані геодезичні карти. Вони є досить розповсюдженими, безперешкодно можна знайти необхідну карту майже про кожен досліджений регіон землі. Вони є достатньо формалізованими, на відміну від панорамних фотографій, або супутникових топографічних знімків. З них ми можемо отримати інформацію про масштаб ландшафту, побачити переходи висот та знайти абсолютну висоту над рівнем моря. Також на деяких картах може міститися інформація про біологічні регіони місцевості, як-то річки, озера, лісосмути, тощо [15]. На рис. 2.1 зображено приклад геодезичної карти.

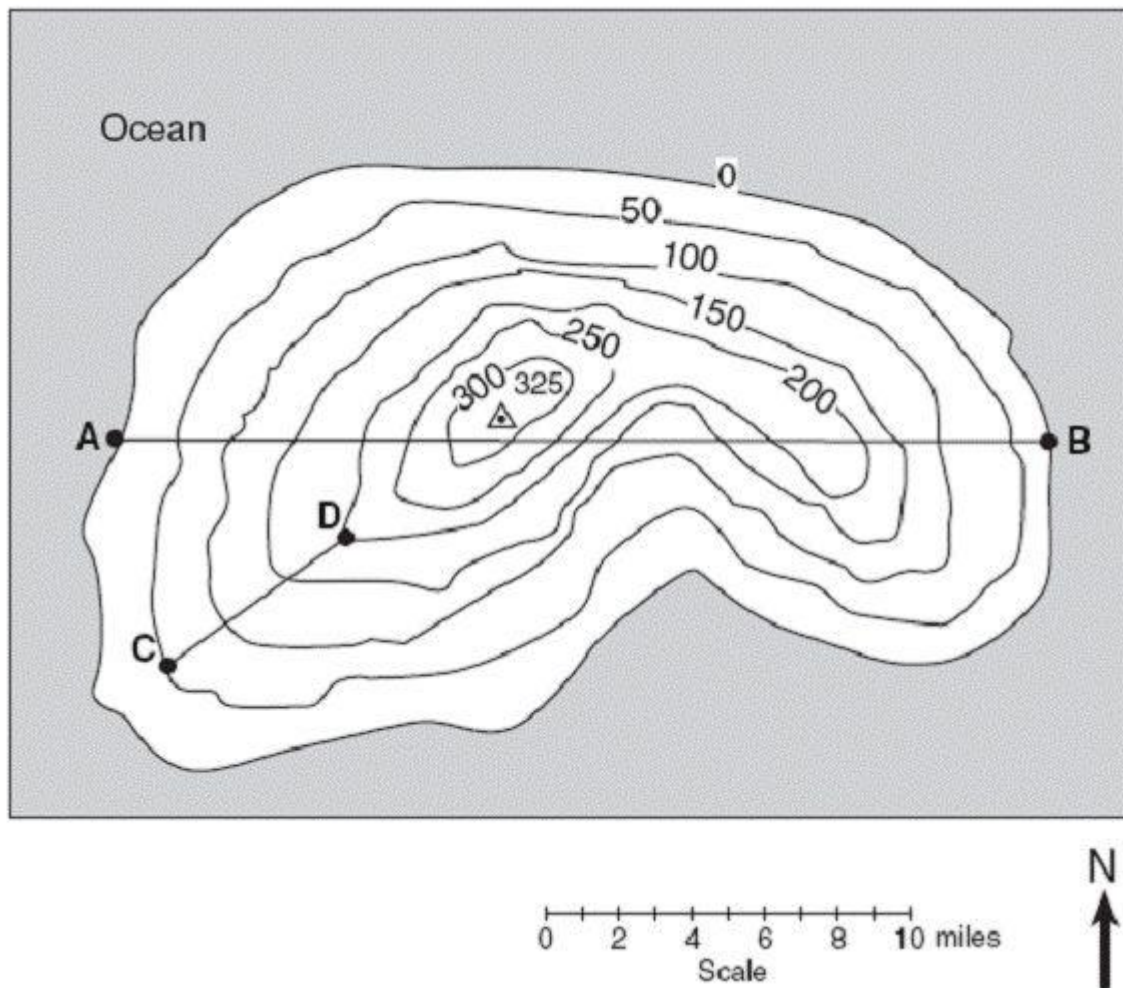


Рисунок 2.1 - Приклад найпростішої геодезичної карти

2.2 Отримання загальної карти висот

У сфері обробки цифрових зображень деякі методи вимагають розділення зображення на регіони, що представляють об'єкти, та регіони що являють собою фон.

Одним з найпоширеніших методів є визначення порога [16]. Основна концепція порогового методу полягає в тому, щоб класифікувати групу пікселів, які мають значення переднього плану, вищі від порогових значень, та іншу групу пікселів, які мають значення, рівні або нижчі від порогових значень. В першому

випадку група цих пікселів буде являти собою цільове зображення, в другому випадку — фон. У цій роботі порогове значення буде обрано користувачем.

При генерації карти висот з геодезичних карт алгоритм буде аналізувати лінії геодезичної карти, що відповідають змінам висот та генерувати відповідну карту висот. Згенерована карта буде відповідати топографічній карті що задає користувач, таким чином надаючи можливість візуалізації топографічних мап.

Злиття статистичного регіону (SRM) — це нова методика сегментації кольорових зображень, що базується на зростанні та злитті регіонів [17]. Метод розглядає сегментацію моделей, як проблему розділення зображення на регіони, в якій зображення розглядається, як спостережуваний екземпляр невідомого теоретичного зображення, статистичні (справжні) області яких повинні бути реконструйовані. Переваги цього методу включають його простоту, обчислювальну ефективність та чудову продуктивність без використання роздроблення або зміни кольору.

Нехай I буде спостережуваним зображенням, що містить l пікселів, кожен з яких складається з значень колірних каналів R, G, B , що належать до множини $\{0, 1, \dots, g-1\}$ (де $g = 256$ для 8-розрядних зображень RGB). I є спостереженням з вихідного зображення I^* , в якому пікселі відмінно представлені сімейством розподілів, з яких відбирається кожен із спостережуваних кольорових каналів. Оптимальні статистичні області в I^* поділяють властивість однорідності, тобто, всередині будь-якої статистичної області i з урахуванням будь-якого колірних каналів, статистичні пікселі мають однакове очікування, тоді, як очікування суміжних статистичних областей будуть відрізнятися, принаймні в одному колірному каналі.

I отримано з I^* , відбираючи кожен піксель для спостережуваних значень RGB. Порядок злиття регіону впливає з інваріантності A , що означає, що при проведенні будь-якого тесту між двома частинами в реальному регіоні, подальших тестів не потрібно. Нехай SI являє собою набір, який містить всі пари сусідніх пікселів у зображенні на основі 4-з'єднання, p і p_0 — пікселі у зображенні I та $R(p)$ — для поточної області, до якої належить піксель p .

Виконується тест злиття $P(R(p), R(p_0))$ для будь-якої пари пікселів (p, p_0) , для якої $R(p) \neq R(p_0)$, і злиття $R(p)$ і $R(p_0)$, якщо він повертає істину [18].

Аналогічно, фаза злиття може бути виконана у лінійному часі за допомогою ефірного алгоритму об'єднання. Оскільки представлена вище модель генерації зображень передбачає, що спостережувані колірні варіації в реальних областях повинні бути значно меншими, ніж між регіонами, один спосіб апроксимації A є обчислення локальних градієнтів між пікселями, а потім обчислити їх максимальну зміну на канал у $f(\cdot)$, тобто $f(p, p_0) = \max_{c \in \{R, G, B\}} |f_c(p) - f_c(p_0)|$. Найпростішим вибором для $f(\cdot)$ є використання значень піксельного каналу безпосередньо:

$$f(p, p') = |p_a - p'_a|, \quad (2.1)$$

де p_a і p'_a - значення піксельного каналу/

Після того, як було проведено аналіз ліній отримали набір пікселів, що можна однозначно ідентифікувати, як лінію. Після цього для кожної такої лінії виконується модифікований алгоритм flood fill.

Flood fill — це алгоритм, який визначає область, з'єднану з даним вузлом у багатовимірному масиві. Він використовується в інструменті наповнення «відро» графічних програм для заповнення пов'язаних, кольорових областей іншим кольором.

Алгоритм flood fill має три початкові параметри: початкова точка, цільовий колір та колір заміни. Алгоритм шукає всі точки масиву, які з'єднані із початковою точкою та мають цільовий колір після чого змінюють їх на колір заміни. Існує багато способів, за допомогою яких алгоритм заповнення залишку може бути структурований, але всі вони використовують структуру даних черги або стеку явно або неявно.

Алгоритм можна пришвидшити шляхом заповнення ліній. Замість того, щоб додавати кожну потенційну майбутню піксельну координату до стеку, вона

перевіряє сусідні рядки (попередні та наступні), щоб знайти суміжні сегменти, які можуть бути заповнені в майбутньому проході; координати (як початок, так і кінець) сегменту лінії додаються до стеку. У більшості випадків цей алгоритм сканування є, щонайменше, на порядок швидше, ніж для кожного пікселя.

Ефективність: кожен піксель перевіряється один раз [19].

2.3 Обробка зображень

Сегментоване зображення часто містить регіони, які є частиною фону. Щоб усунути ці регіони, потрібно визначити колір фону. Можна зробити чотири патчі розміром 20×20 пікселів з кутів зображення та обчислити середнє значення R, G, B пікселів. Цей середній колір приймається, як оцінюваний колір фон. Тут виключаються світлі кольори регіонів, тобто регіони, середні кольори яких мають відстань менш, ніж 60 до кольору фону. Крім того, регіони, які торкаються кадру зображення, та ті, що мають прямокутні межі, відкидаються. Початковий результат виявлення кордону отримується шляхом видалення ізольованих областей, а потім злиття решти регіонів [20].

Розпізнані регіони представлені у вигляді структури що зберігає позиції всіх точок регіону, всіх граничних точок (точки що належать безпосередньо лінії) та опірної точки що є найлівішою точкою лінії.

В той час, як дані про місцевість можуть бути представлені кількома способами, найпоширенішою структурою для представлення рельєфу є поле висоти. У математичному плані поле висоти є скалярною функцією двох змінних, такими, що кожна пара координат (x, y) відповідає значенню висоти h .

На практиці поле висоти зазвичай реалізується, як двомірна, прямокутна сітка з значеннями висоти і еквівалентна графічному зображенню. Поля висот мають обмеження: вони не можуть представляти структури, в яких кілька поверхонь мають однакові координати (x, y) (такі, як печери та звисання гір), але є достатніми для більшості застосувань і можуть бути високо оптимізовані для рендеринга та генерації ландшафтів.

На цьому етапі на вході маємо — це 2D «карту» полігональних «регіонів», де вказується приблизний розмір, форма та розташування різних типів місцевості (наприклад, велика еліптична область гір, оточена з усіх боків пагорбами). Користувач матиме можливість створювати ці регіони вручну, використовуючи геодезичні карти. У будь-якому випадку, оскільки лінійні межі цієї грубої штучної форми можуть бути помітними, як артефакти у сформованій місцевості, вони повинні бути розбиті на більш природні, нерівні границі. Виконуємо цю модифікацію краю шляхом підрозділу кожного краю на послідовність точок і застосування GA для отримання прийнятної граничної форми.

Точки в початковому наближенні границь висот створюються наступним чином:

1 Для певного сегмента вхідної границі нехай P_0 — вихідна точка, а P_n — кінцева точка;

2 Будемо генерувати серію p проміжних точок P_i , які будуть з'єднані для створення нового набору лінійних сегментів, що з'єднують P_0 з P_n , рівномірно розташовані за їх координатами. Створення сегменту першого рядка який з'єднує P_0 з P_1 : вибрати довільний кут θ , що лежить в межах заданого діапазону кутів, і розмістити P_1 уздовж цього кута при фіксованому x -значенні зсуву;

3 Тепер створимо лінійний сегмент, який з'єднує P_i з P_{i+1} , генеруючи довільний кут θ таким чином, що результуючий кут, утворений з P_{i-1} , P_i і P_{i+1} , падає нижче встановленого користувачем порога. Цей поріг, визначений користувачем, дає міру локальної гладкості кордону (малі значення вказують на менші варіації i , отже, більш гладкі межі);

4 Повторювати крок 3 ($n - 1$) раз. Зверніть увагу, що P_n , може, не знаходитися в тій же позиції, в якій була вихідна кінцева точка;

5 Оскільки вихідні кінцеві точки границі не будуть, в основному, відповідати тим, що відповідали початковій формі, застосовують обертання та масштабування, щоб встановити згенеровану межу в потрібне місце.

2.4. Додавання малих деталей

Отримана на попередньому кроці карта висот при візуалізації буде відповідати заданій геодезичній карті/карті висот, проте через малу кількість деталей вона не буде виглядати реалістично. В реальності ландшафт має велику кількість ерозійних слідів - нерівностей, ям, ярів, пагорбів і т. д.

Для симуляції цього ефекту пропонується використати модифікований шум Перліна. На рис. 2.2 зображений приклад шуму Перліна.



Рисунок 2.2 - Приклад двовимірного шуму Перліна

Шум Перліна в данній роботі реалізується як тривимірна функція, але може бути визначений довільною кількістю вимірів. Реалізація складається з трьох кроків:

- визначення сітки,
- обчислення скалярного добутку градієнтних векторів,
- інтерполяція між цими значеннями.

Існують також інші варіації алгоритму [22].

2.4. Генерація шуму

Визначаємо тримірну сітку. Кожній координаті сітки присвоїти одиничний вектор такої самої розмірності. У випадку одновимірної сітки кожній координаті буде присвоєно значення між +1 та -1, для двовимірної сітки кожній координаті буде присвоєно випадковий вектор з одиничного кола, і так далі для більшої кількості вимірів.

Визначення випадкових градієнтів для одного чи двох вимірів є тривіальним. Для більшої кількості вимірів пропонується використовувати наближення Монте Карло, де випадкові координати вибираються з одиничного куба, а точки за межами одиничної сфери відкидаються. Процес продовжується, поки не отримаємо необхідну кількість випадкових градієнтів. Далі отримані вектори нормалізують.

З метою зменшення витрат на обчислення нових градієнтів для кожної координати пропонується використовувати хеш-таблиці і таблиці пошуку для обмеження кількості попередньо обчислювальних градієнтних векторів. Хеш таблиці заповнюються випадковими числами. Використання хешу також дозволяє включити випадкові сіди, де необхідно кілька разів застосувати шум Перліна [23].

Другий крок - це визначення, в яку комірку сітки потрапляє окрема точка. Для кожного вузла сітки визначаємо вектор відстані між координатами вузла і точкою. Після цього обчислюємо скалярні добутки визначених векторів та градієнтних векторів кожного вузла комірки. Складність алгоритму $O(2n)$.

Інтерполяція. Фінальний крок - інтерполяція обчислених значень скалярних добутків що знаходяться в кожному вузлі. Інтерполяція виконується з використанням функції, що має нульову першу похідну (і, можливо, другу похідну також) на обох кінцевих точках. Лінійна функція, для кінцевих точок на 0 та 1, зі значеннями a_0 та a_1 , може бути такою:

$$f(x) = a_0(1 - x) + a_1 * x,$$

де a_0, a_1 - задані параметри функції

Найчастіше використовуються нормалізовані функції шуму, що дають значення в діапазоні $[-1,1]$. В даній роботі був також використаний нормалізований шум. Щоб результати шуму Перліна залишалися у цьому проміжку, інтерпольовані значення мають коригуватись за допомогою масштабуючого фактора.

2.7 Висновки по розділу 2

В другому розділі детально описано кожен з етапів удосконаленого методу процедурного синтезу тривимірних ландшафтів у геймдизайні, який на відміну від існуючих використовує геодезичні карти як джерело інформації про ландшафт, що дозволяє підвищити рівень реалістичності графічного моделювання. Наведено схеми та підібрано математичні засоби для реалізації кожного етапу пропонованого методу.

3 РОЗРОБЛЕННЯ ПРОТОТИПУ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ПРОЦЕДУРНОГО СИНТЕЗУ ТРИВИМІРНИХ ЛАНДШАФТІВ

Процес розроблення прототипу програмного додатку для процедурного синтезу тривимірних ландшафтів можна розбити на декілька окремих частин:

- отримання загальної карти висот;
- додавання ерозійних шумів;
- текстурювання (додавання об'єктів дерев та трави).

3.1 Аналіз геодезичних карт

Для ідентифікації ліній що відповідають перепадам висот вся картинка переводиться в чорно білий колір. Вибирається поріг що відповідає білому кольору на картинці, він не завжди може бути (1,1,1), це залежить від характеру зображення. Після того як вибраний поріг білого кольору перебирається, кожен піксель картини за наступним алгоритмом (см. рис. 3.1):

Крок 1. Після того як був знайдений перший не білий піксель він реєструється як лінія.

Крок 2. Починається перебір пікселів в околі даного пікселя. Якщо були знайдені не білі пікселі, які не належать до даної лінії - вони визначаються як нові пікселі лінії. Після цього кроку для всіх пікселів лінії знову виконується крок 2. Вихід з рекурсії виконується коли в околі пікселя не було знайдено не білих точок, що не належать лінії.

Крок 3. Продовжується перебір всіх пікселів, пікселі, які належать зареєстрованим лініям ігноруються. Якщо знайдений піксель не є білим та не належить жодній з зареєстрованих ліній то для нього виконується крок 2.

Крок 4. Після того як були пройдені всі пікселі ми отримуємо масиви з пікселями що належать лініям.

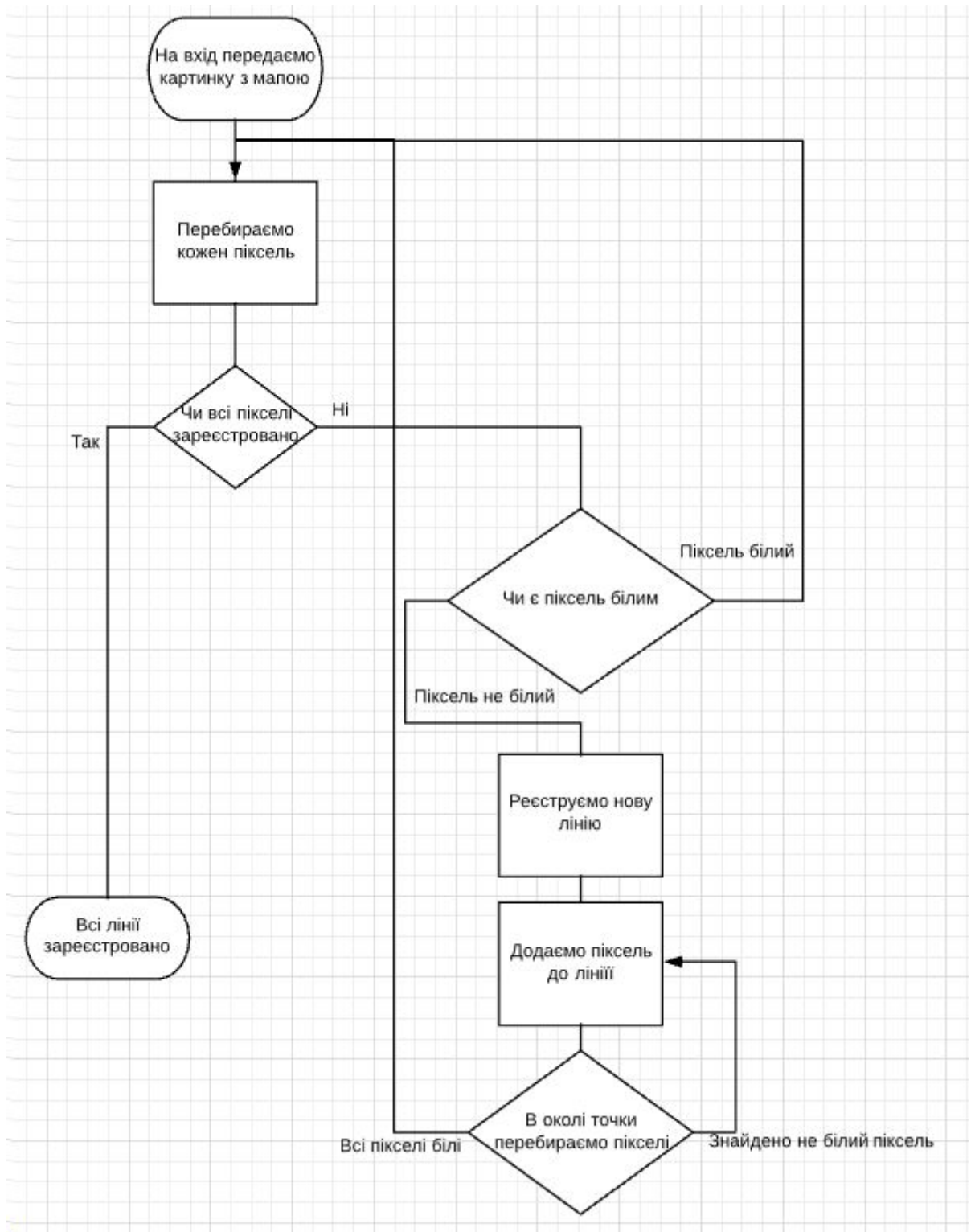


Рисунок 3.1 - Алгоритм визначення перепаду висот

Після того як усі лінії було зареєстровано для визначення регіонів буде використовуватися алгоритм flood fill.

Для того, щоб визначити початкову точку для старту алгоритму використовується опорна точка. Під час аналізу ліній перед тим як додати нову точку до лінії перевіряється її координата x . Якщо вона менше ніж в опорної точки - точка стає новою опорною. Проте, сама опорна точка не може надати точну позицію точки, що лежить всередині регіону. Так як точка є най лівішою - для вирішення проблеми можна було б взяти точку, що знаходиться справа, проте цей підхід може спрацювати помилково. Якщо товщина лінії більше одного пікселя, цей спосіб не спрацює. Також він буде некоректний для деяких крайових випадків, коли опорна точка є також найвищою\найменшою.

Для гарантованого вибору точки, що лежить всередині регіону необхідно порахувати середню позицію деякого набору сусідів опорної точки. Дана точка і буде стартовою позицією алгоритму заповнення регіону. Вона гарантовано буде розміщена всередині регіону на деякій відстані від границі лінії. Кількість сусідніх точок які необхідно враховувати може відрізнятися в залежності від товщини ліній та характеру зображення.

Алгоритм виконує заповнення ліній. Замість того, щоб додавати кожен потенційну піксельну координату до черги, вона перевіряє сусідні ряди (попередні та наступні), щоб знайти суміжні сегменти, які можуть бути заповнені в майбутньому проході; координати (як початок, так і кінець) сегменту лінії виштовхуються на стек. Тобто кожного разу, коли перевіряємо точку, також добавляємо у чергу пікселі, які знаходяться над та під обраним пікселем. У більшості випадків цей алгоритм сканування є, щонайменше, на порядок швидше, ніж для кожного пікселя. На рис. 3.2 зображено алгоритм.

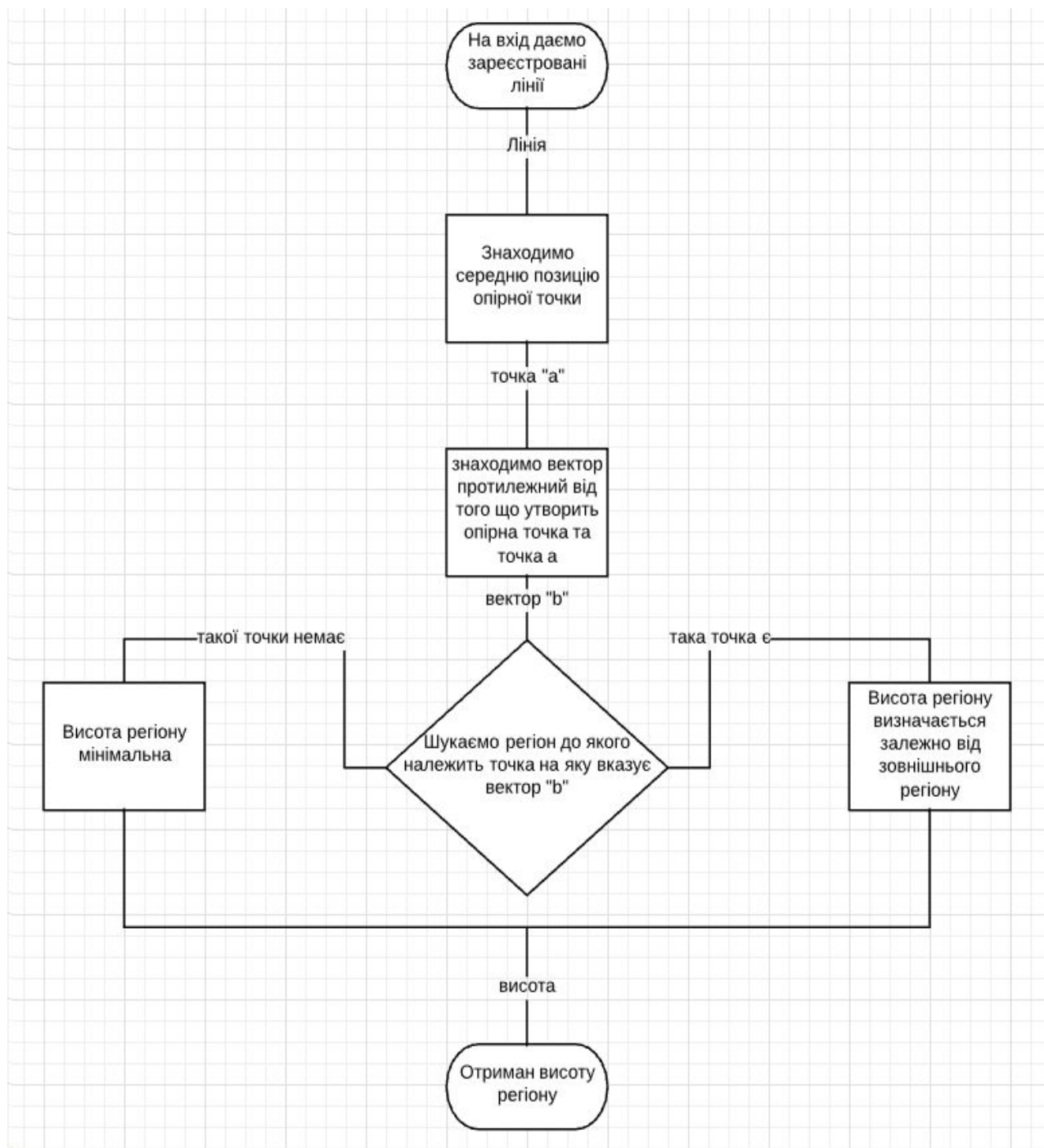


Рисунок 3.2 - Алгоритм визначення висоти регіону

Ефективність: кожен піксель перевіряється один раз. При використанні геодезичних карт карта біомів буде повністю визначатися висотою точки. Наприклад, для точок, що знаходяться значно вище, ніж середня висота точок буде використано біом "гора".

3.2. Уточнення карти висот

При генерації карти висот зазначеними вище способами будуть дуже помітними переходи між висотами геодезичних карт, або між заданими висотами різних біомів. Реальні ландшафти натомість дуже рідко мають різкі переходи між різними висотами, зазвичай одна висота плавно перетікає в іншу. Нам необхідно додати пост обробку карти висот, яка б забезпечила відповідні плавні переходи.

Методом згладжування карти висот був обраний метод розмивання Гауса, що буде застосовуватися ітеративно. Користувач матиме змогу задати обрану кількість ітерацій, що буде згладжувати отриману карту нормалей. Змінюючи кількість ітерацій можна змінювати різкість переходу між різними висотами. Так для отримання ефекту крутих гір доцільно буде використовувати невелику кількість ітерацій. На рис. 5 зображено приклад використання розмивання Гауса.



Рисунок 3.3 - Використання розмивання Гауса, 10 ітерацій

Таким чином, адаптуємо грубу карту висот до використання алгоритмом генерації ландшафту.

Розмивання Гауса це тип фільтру розмивання зображення, що використовує функцію Гауса (яка також зустрічається у нормальному розподілі в області статистики) для розрахунку зміни значення кожного пікселя у

зображенні. Піксель буде вираховуватися з рівняння одновимірної функції Гауса:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

де σ стандартне відхилення розподілу Гауса.

Для двовимірного випадку, вираз складається з двох таких функцій, кожна з яких буде ставитися у відповідність кожній осі:

$$G(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

де σ є стандартне відхилення розподілу Гауса.

Коли метод застосовується у двох вимірах, отримується поверхня, контури якої є концентричними колами розподілу Гауса з центральної точки.

Значення з цього розподілу використовуються для створення матриці згортки. Для кожного нового значення пікселя визначається середнє значення сусідів в околі пікселя. Значення поточного оригінального пікселя має більшу вагу (найвище значення розподілу Гауса), а сусідні пікселі отримують все меншу вагу в залежності від того наскільки далеко вони знаходяться від поточного оригінального пікселя. Це надає ефект розмитості, яка зберігає кордони та краї краще, ніж інші, аналогічні фільтри розмиття [21]. На рис. 6 зображено алгоритм обрахунку висоти точки мешу.

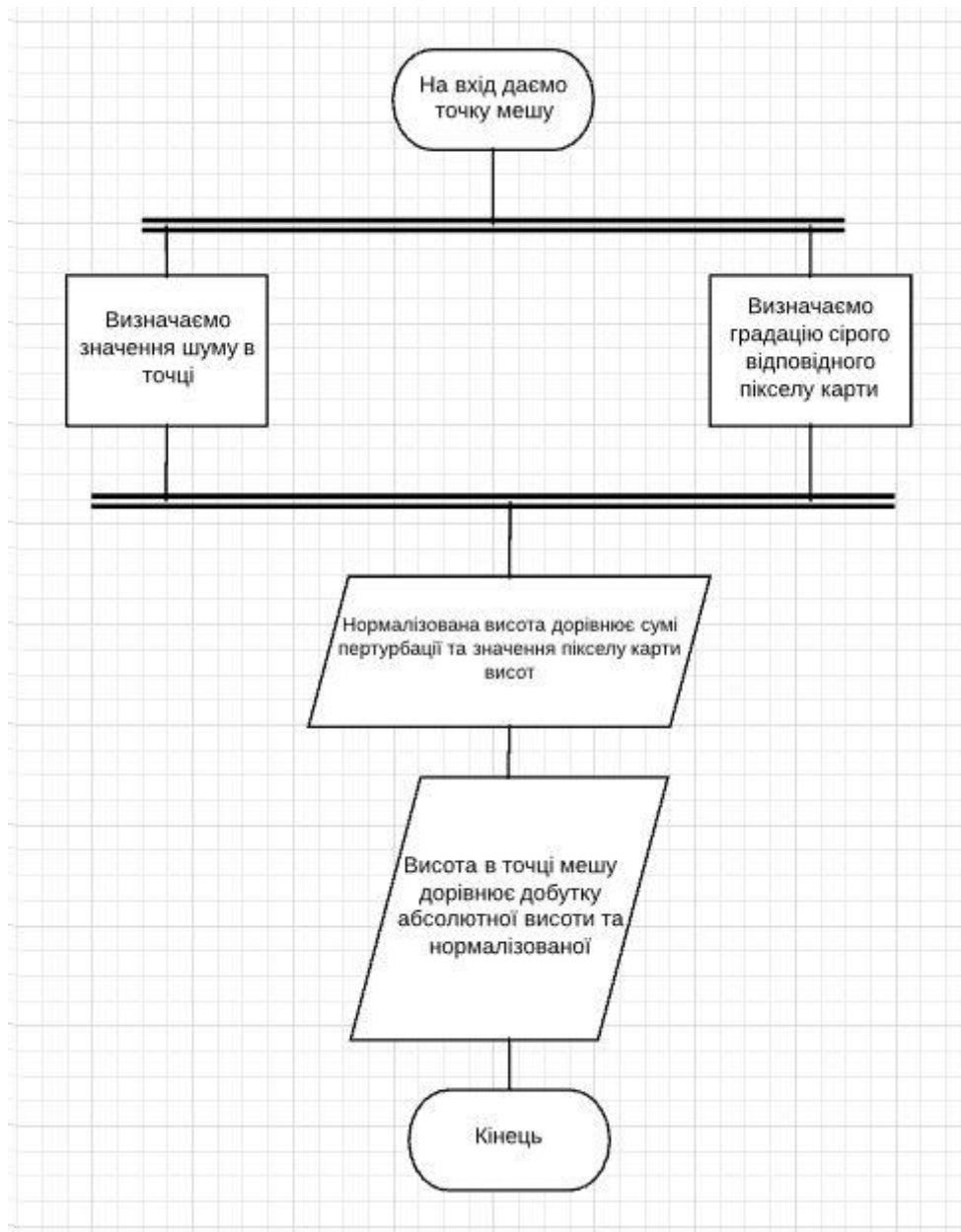


Рисунок 3.4 - Алгоритм визначення висоти точки мешу

3.3 Додавання ерозійних шумів

Деталі не повинні суттєво впливати на отриману карту висот, адже таким чином можливе викривлення бажаного результату при генерації ландшафту з геодезичної карти.

Генеруємо декілька шумів Перліна розміром, що відповідає бажаному розміру карти висот. Кожен з них буде відрізнятися амплітудою та частотою.

При чому чим більше амплітуда тим менше частота та навпаки. При низькій амплітуді та високій частоті отриманий шум буде нагадувати маленькі ями та пагорби, тріщини і т. д.. При високій амплітуді та високій частоті отримаємо шум, що буде відповідати глибоким ярам, високим пагорбам, які не зустрічатимуться часто. Результуючу карту ерозійних шумів складемо підсумувавши всі згенеровані шуми. Таким чином отримано карту, яка має в собі велику кількість деталей, проте які, не вносять суттєвих змін до висот.

Кожному біому відповідають підібрані коефіцієнти кожної карти шумів, щоб результуюча карта ерозійних шумів залежала від біому. Таким чином можна окреслити характерні особливості кожного регіону: наприклад, для гір та кам'янистих регіонів відповідатиме велика кількість неглибоких тріщин та нерівностей, у той же час як для полів більш характерними будуть більш глибокі яри та пагорби.

Для ландшафту була визначена певна абсолютна висота, що відповідає 1 на нормалізованій карті висот. Отже для отримання висоти точки буде використовуватися формула:

$$h_{x,y} = (P(x, y, HeightMap[x, y]) + HeightMap[x, y]) * absHeight$$

де $h_{x,y}$ - висота в точці x, y ;

P - трьохвимірний функція що підраховує сумарне значення шумів в координаті;

$HeightMap$ - карта висот;

$absHeight$ - абсолютна висота ландшафту.

3.4 Проведення текстуровання

Користувач матиме змогу визначити які дерева/рослини/текстури необхідно використати для генерації ландшафту, поставивши кожному біому у

відповідність характерні для нього об'єкти. Також він буде мати можливість визначити на геодезичній карті до якого біому відноситься той чи інший регіон. Якщо регіони не зазначені - вони будуть присвоєні автоматично виходячи з висоти.

Для кожного біому буде свій характерний набір текстур та об'єктів. Таким чином гори та поля будуть візуально відрізнятися.

Для вирішення проблеми візуального розподілу різних біомів (наприклад, чітка границя між текстурами каменю та землі) використовується нормалізована функція Гаусса, що є функцією виду:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

де c - ширина регіону,

b - позиція центру,

a – амплітуда.

Графік функції Гауса є характерною симетричною кривою у формі дзвону, що швидко спадає на нескінченності. Параметр a є висотою піку кривої, b є позицією центру, і c контролює ширину «дзвону» [24].

На рис. 3.5 зображена функція Гауса з довільними параметрами.

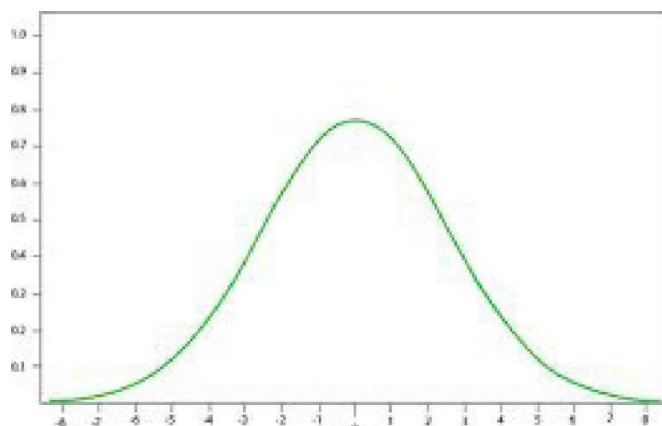


Рисунок 3.5 - Графік функції Гауса з довільними параметрами

Так як функція є нормалізованою то параметр a беремо рівним одиниці. В піку функція дорівнюватиме 1, на краях визначеного регіону ширину регіону c та позицію центру b визначатимемо з заданих висот текстур. Для кожної текстури вказаний регіон висоти $h_{min}...h_{max}$ де вона може використовуватися. В такому випадку параметри прийматимуть значення:

$$b = h_{min} + (h_{max} - h_{min})/2;$$

$$c = h_{max} - h_{min};$$

$$a = 1,$$

де $h_{min}...h_{max}$ - задані регіони текстури,

c - ширина регіону,

b - позиція центру,

a – амплітуда.

Функція Гаусса з такими параметрами для довільної висоти повертатиме:

$$\begin{cases} 0 & h < h_{min} \text{ або } h > h_{max} \\ 0 \dots 1 & h \geq h_{min} \text{ та } h \leq h_{max} \\ 1 & h = c \end{cases}$$

де $h_{min}...h_{max}$ - задані регіони текстури,

c - ширина регіону,

b - позиція центру,

a – амплітуда.

При такому розподілі точка буде мати колір текстури якщо висота цієї точки c . Якщо висота цієї точки лежить в рамках дозволеного регіону для текстури, але не є піком то альфа канал кольору текстури в точці буде <1 .

Для того, щоб забезпечити плавний перехід між текстурами - кожна текстура що відповідає регіону має мінімальну та максимальну границю в піках сусідніх регіонів.

Результуючий колір точки буде сумішшю кольору всіх текстур.

Тривіальне рішення простого присвоєння текстури відповідному біому буде виглядати дуже нереалістично. Адже дуже рідко можна зустріти поле, земля в якому є абсолютно однаковою на всій площині. Тому при текстуруванні буде застосовуватися шум Перліна низької октавності. Прив'язавши силу текстури в точці до значення шуму в цій точці матимемо можливість отримати текстурування регіону декількома текстурами, що реалістично змішуються та переходять одна в одну.

Тому текстура в заданій точці буде сумішшю текстур, які відповідають даному біому. При чому їхнє співвідношення буде прив'язане до сили шуму в цій точці.

Наприклад, можна узяти, що альфа канал однієї текстури в точці дорівнюватиме значенню шуму Гаусса, а іншої 1 - значення шуму Гаусса в цій точці. Таким чином гарантовано, що сума альфа каналів текстур завжди буде рівна 1.

Для підсилення ефекту ерозійних шумів використаємо спеціальні текстури що відповідатимуть незначним переходам висот всередині регіону. Так якщо посеред зеленого луку є різкий та протяжний яр - його стінки було б краще протекстуризувати не зеленою травою (що є характерною для поля/степу), а наприклад текстурою ґрунту/піску.

При додаванні об'єктів рослин/дерев алгоритм буде керуватися характеристиками біому. Так для гір може бути визначена невелика кількість трави, мінімальна кількість дерев. Для симуляції лісу має бути створена карта густини дерев, для досягнення ефекту плавного переходу лісу в інші біоми.

Ліс має велику щільність дерев в середині регіону, проте наближаючись до краю він має нижчу густину. Якби ми чітко обмежували допустиму область для

рослин, ми б отримали різкий видимий перехід, що візуально би виділявся та не виглядав би реалістично. Для того, щоб запобігти цьому ми використали нормальний розподіл Гауса для визначення шансу створення об'єкта рослинності в певній точці. Це функція є нормально розподілена по всьому регіону спадаючи до країв. Якщо ми візьмемо значення цієї функції як шанс створення об'єкту в точці, то ми отримаємо ефект плавного зменшення щільності наближеного до країв біому.

Також для підвищення реалістичності шанс створення об'єкту в точці залежить також від шуму Перліна в цій точці. Це надасть нам луки заповнені травою з рідкими прогалинами, що відповідає реальним ландшафтам. На рис. 3.6 зображено цей алгоритм.

Архітектура розробленої системи:

1 GLineParser - клас, що відповідає за збір та класифікацію геодезичних ліній. Він відповідальний за створення масиву об'єктів GLine. На рис. 3.7 зображено його структуру.

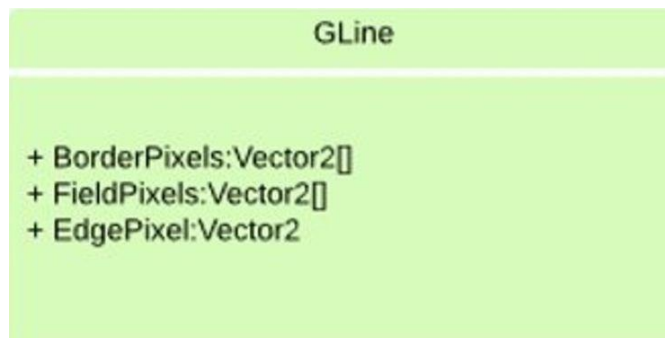


Рисунок 3.7 - Структура GLine

1 BorderPixels - він формує масив та визначає краю опорну точку EdgePixel.

2 MapInterpreter - клас, що відповідає за визначення регіонів, що описуються лініями та визначення їх взаємоположення. Він формує зображення нормалізованої карти висот.

3 TerrainGenerator - клас, який відповідальний за створення відповідного мешу. Він визначає кривизну мешу в кожній точці, керує пертурбаціями та зберігає згенерований кривий меш.

4 SplatAsigner - клас, що відповідає за визначення розмітки текстур на меші. Також він створює об'єкти рослинності.

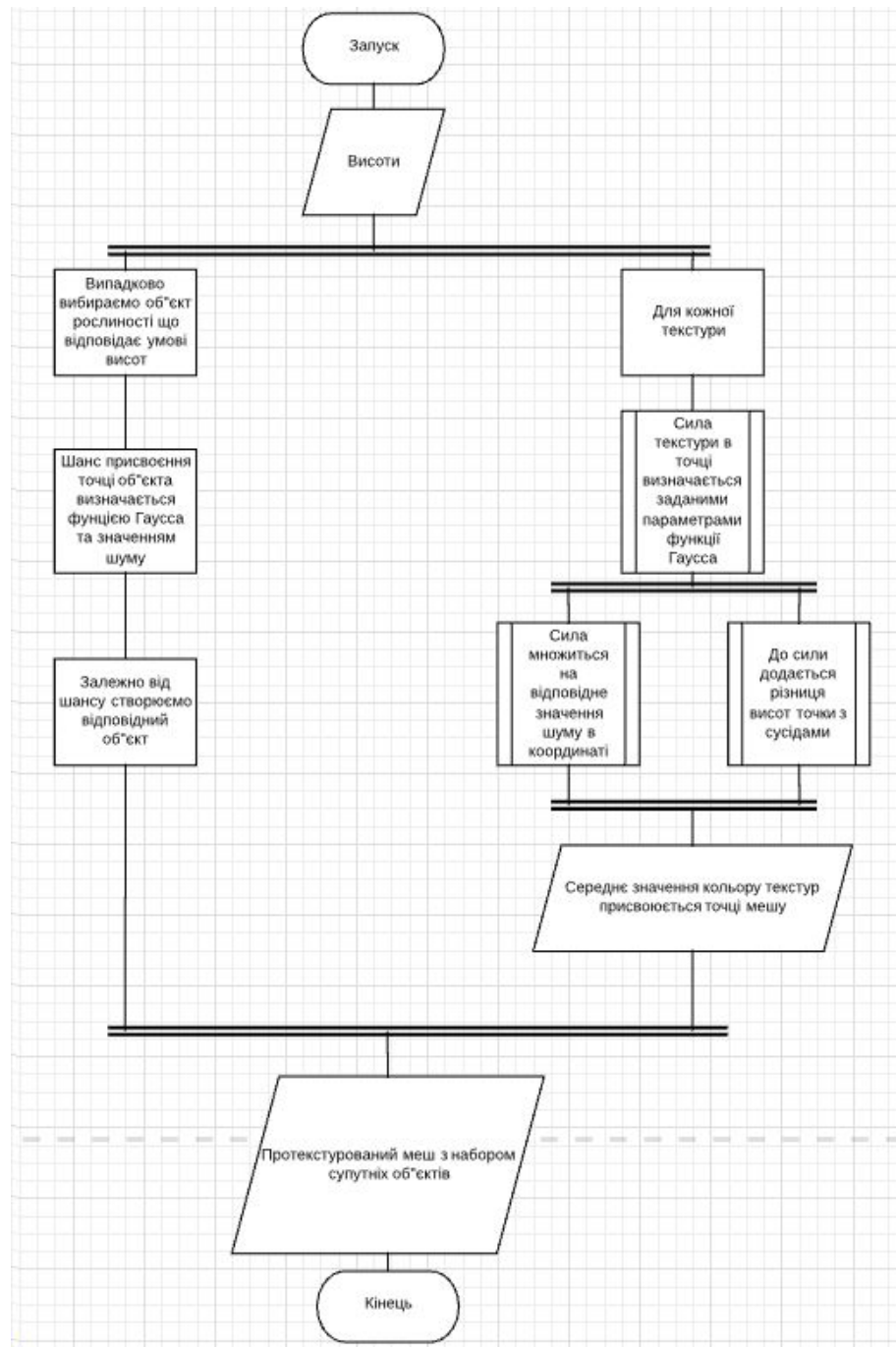


Рисунок 3.6 - Алгоритм створення текстурної розмітки мешу, створення об'єктів рослин

На рис. 3.8 зображено архітектуру розробленої системи, а на рис. 3.9 зображений загальний алгоритм.

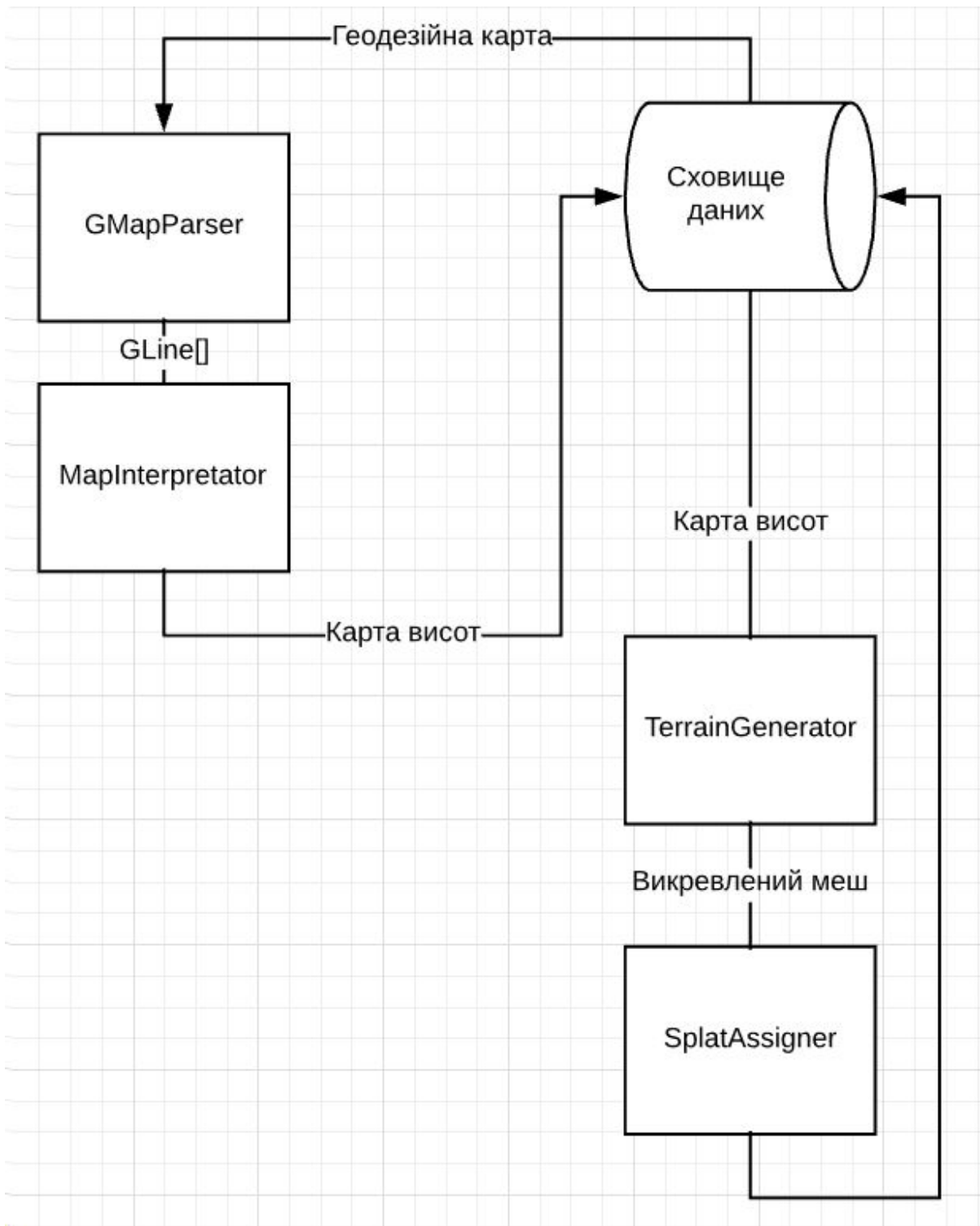


Рисунок 3.8 - Архітектура розробленої системи

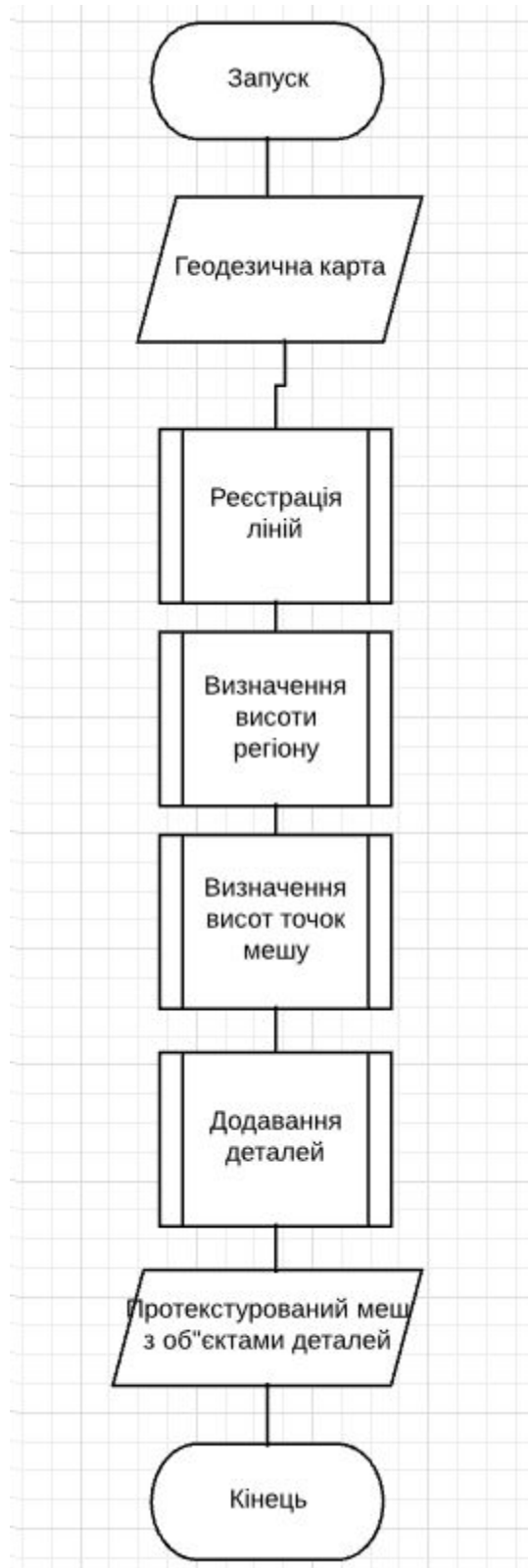


Рисунок 3.9 – Загальна схема методу процедурного синтезу тривимірних ландшафтів у геймдизайні

3.5 Перевірка ефективності розробленого ПП

У ході даної роботи було розроблено програмне забезпечення, яке дозволяє генерувати ландшафти. Однією з найбільших переваг розробленого продукту є можливість генерації ландшафту за заданим користувачем шаблоном. Розроблений продукт також генерує реалістичний результат та відрізняється великою кількістю деталей.

Виконаємо порівняльний аналіз ПЗ, розробленого на основі запропонованого методу з аналогічними ПП.

Бібліотека FastNoise - це бібліотека з генерацією шумів із відкритим кодом, з великою кількістю різних алгоритмів шуму, результат якої зображений на рис. 12. Ця бібліотека була розроблена для використання в реальному часі, тому вона була оптимізована для швидкості без втрати якості шуму.

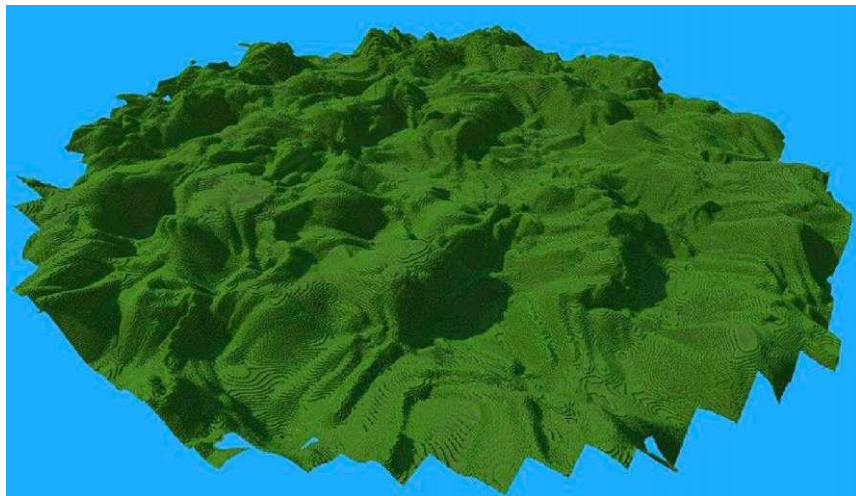


Рисунок 3.10 - FastNoise

Особливості бібліотеки FastNoise:

- Можливість застосування шуму Періна у 2D, 3D варіантах.
- Можливість застосування шуму Simplex у 2D, 3D, 4D варіантах.
- Можливість застосування кубічного шуму у 2D, 3D варіантах

- Можливість застосування градієнту Perturb у 2D, 3D варіантах
- Наявність фрактальних варіантів для всього перерахованого вище
- Можливість застосування стільникового (Регіони Вороного) шуму у 2D, 3D варіантах
- Можливість застосування білого шуму у 2D, 3D, 4D варіантах
- Підтримка типів double або float

Ця бібліотека є широко відомою та добре оптимізованою. Проте, основним недоліком, який вирішує наш метод, є відсутність можливості визначення форми рельєфу користувачем. Користувач може не прямо, але впливати на результат генерації: визначати параметри шумів, визначати випадкове зерно. Проте, цей вплив є дуже обмеженим — він не дозволяє користувачеві напряму визначати результат. Для того щоб зібрати бажаний рельєф необхідно довго перебирати можливі налаштування. Задачу створення моделі чітко визначеного ландшафту дана бібліотека вирішити не зможе.

Ще однією перевагою розробленого способу в порівнянні з FastNoise є те, що спосіб генерує меш, а не набір воксельних об'єктів. Крім того, що меш має набагато більш реалістичніший вигляд, цей спосіб є більш оптимізованим у випадку невеликого розширення мешу.

На відміну від FastNoise даний розроблений метод передбачає можливість експорту карти до картинки формату png. До нього включене поняття біому, що зможе додати багато різноманітності. Завдяки біомам ландшафти виглядають набагато реалістичніше та візуально привабливіше, ніж абсолютно зелена/жовта поверхня що генерується цією бібліотекою. 2. Landscape Generator 3.

Landscape Generator 3 система, основана на створенні біомів, забезпечує більшу різноманітність у генерованій місцевості та дозволяє створювати різні середовища в одному ландшафті. Результат роботи цієї системи зображений на рис. 3.11.

Перелічимо деякі особливості Landscape Generator 3:

- Генерація та картографування ландшафтів на основі біома;

- Збереження та завантаження ландшафтних профілів;
- Повне налаштування параметрів генератора;
- Можливість застосування безкінечних текстурних шарів;
- Застосування нескінченної кількості біомів;
- Велика деталізація та щільність розміщення об'єктів;
- Простий у користуванні користувальницький інтерфейс;
- Розміщення об'єктів масової деталізації;
- Експорт карти до файлів.png.



Рисунок 3.11 - Landscape Generator 3

Більшість цих особливостей так чи інакше реалізовані в нашому способі генерації. Проте, замість генерації ландшафтів на основі біому пропонується генерувати біоми на основі висот ландшафту, таким чином надаючи користувачеві більше можливостей для визначення результату генерації. Можливість визначати біоми дає користувачеві вплив на результат генерації. Таким чином користувач може визначити положення лісів/гір на згенерованому ландшафті. Проте, цей спосіб буде унеможлиблювати створення чітко визначених рельєфів ландшафту. Запропонований нами спосіб робить можливим

визначення не лише біомів та їх характеристик, але й отримання бажаних форм згенерованих ландшафтів.

Єдиною перевагою Landscape Generator 3 в порівнянні з нашим способом генерації є можливість створення безкінечних об'єктів ландшафтів. Для генерації ландшафту за нашим способом необхідно визначена користувачем форма ландшафту.

Є можливим визначити будь-яку велику карту ландшафтів, проте — процедурно генерувати її на безкінечному просторі не вийде. Це не є критичним недоліком, так як розроблена система повинна використовуватися у випадках, коли необхідно візуалізувати уже розроблений ландшафт, що відкидає необхідність у випадковій генерації.

Procedural Terrain Generator ця бібліотека дозволяє створювати та змінювати місцевість. Ці функції виконуються через спеціальний редактор або під час виконання через виклики API. Результат роботи цієї системи зображений на рис. 3.12.

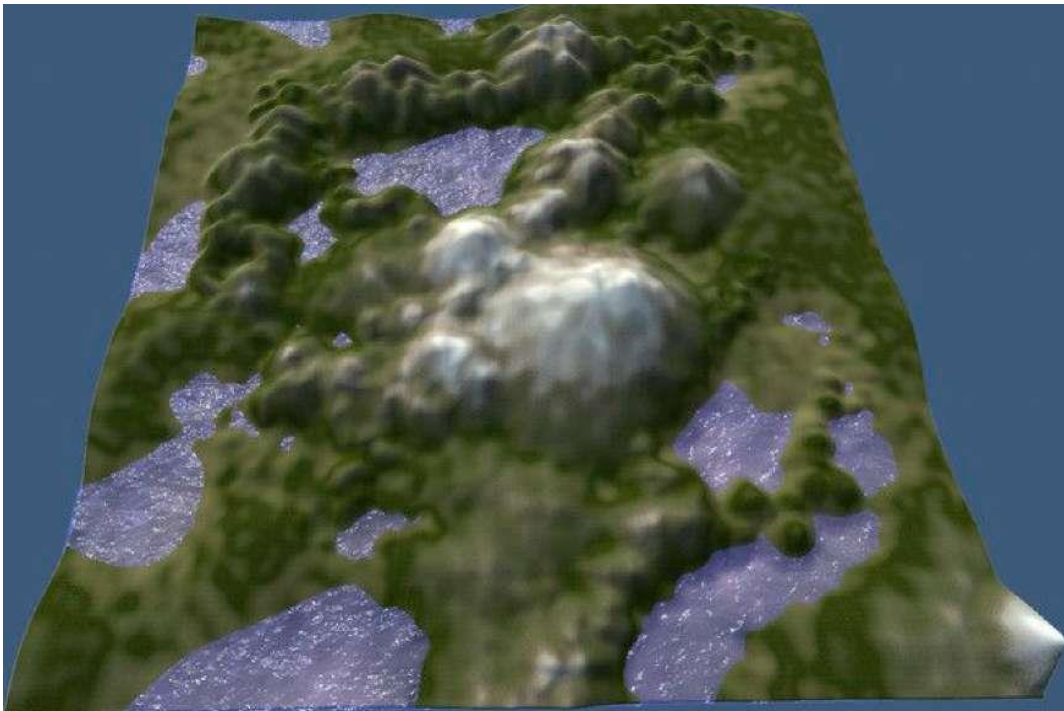


Рисунок 3.12 - Procedural Terrain Generator

Використання компонента в редакторі Unity здійснюється за допомогою спеціального редактора, який входить до складу цього пакета. Це дозволяє дизайнерові застосовувати модифікації рельєфу до, під час та після інших налаштувань за допомогою інструмента Terrain. Налаштування інструментів місцевості:

1. Довжина й ширина повинні бути однаковими.
2. Висота може бути налаштована за бажанням. Роздільна здатність повинна бути не менше 512.
3. Під час виконання компонентом може бути доступ через відкритий API.

Зміст пакета: Географічне об'єднання та спеціальний редактор Демо-сцена

Приклад сценарного менеджера із кодом, що демонструє використання API-інтерфейсу під час виконання є Procedural Terrain Generator.

Procedural Terrain Generator має зручний інтерфейс та є зручно інтегрованим у редактор Unity. Проте, сам алгоритм генерації полягає у візуалізації карти висот згенерованій шумом Перліна. Користувач може задавати октавність, амплітуду, частоту шуму щоби генерувати більш гористі або пологі ландшафти. Однак, користувач не матиме змоги визначити бажану форму ландшафту. З одного боку, це компенсується можливістю підгонки ландшафту вручну, але такий спосіб вимагатиме багато часу із сторони художника. Шум Перліна в більшості випадків не виглядає реалістично, тому згенеровані ландшафти виглядають штучно та не естетично.

Основною перевагою розробленого в ході даної роботи методу генерації є можливість тонкого налаштування результату генерації. Під час порівняння з іншими рішеннями в цій сфері можна підкреслити, що жодне з розглянутих рішень не надає можливості визначити рельєф згенерованого ландшафту. Розроблений метод видає більш детальний та реалістичний результат, ніж розглянуті аналоги. Можливість генерації мапи ландшафту у форматі png також не є характерною для всіх розглянутих продуктів.

Для кожного з біомів доступне тонке налаштування:

- Характер ерозійних шумів
- Текстури
- Рослинність

Таким чином, користувач може дуже тонко підлаштувати результат генерації під себе. Для використання ландшафту в маркетингових матеріалах, або, наприклад, у виробництві фільмів є надзвичайно важливою можливістю заздалегідь визначати зовнішній вигляд генерованого об'єкту.

Процес отримання даних із геодезичних карт вимагає перебору кожного пікселя картинки, він має складність $O(p)$. Тому для пришвидшення швидкодії програми доцільно використовувати геодезичні карти невеликого розширення. Так, як ці карти використовуються для надання загальних обрисів ландшафту, покращення розширення мапи не призведе до поліпшення результату, натомість зменшення розширення збільшить швидкодію.

Оскільки критерії якості візуалізації є суб'єктивними, то до оцінки якості роботи способу процедурної генерації ландшафтів було залучено експертів. Експертна оцінка результатів застосування розробленого ПЗ наведена в табл. 3.1.

Таблиця 3.1 - Експертна оцінка

Назва	Реалістичність	Якість дрібних деталей	Візуальна привабливість
Розроблений ПЗ	7	8	8
Procedural Terrain Generator	3	2	5
Landscape Generator 3	6	8	8
Fast Noise	3	1	4

Були проведені тести по генерації ландшафту з такими параметрами:

- 1) Розмір геодезичної мапи: 512x512
- 2) Розмір мешу ландшафту: 1028x1028

Як результат, проведені тести швидкодії показали, що перенесення карти висот та додавання малих випадкових деталей ландшафту виконується за ~40 секунд, а проведені тести швидкодії показали, що текстурування та додавання об'єктів рослинності до ландшафту виконується за ~30 секунд.

Також, на швидкодію впливає обране представлення мешу — Unity terrain object. Дане представлення є стандартом у роботі рушія Unity, та має багато зручних функцій. Але, одним із недоліків є велика затримка при зміні об'єкта. У наших тестах вона сягала ~30 сек.

Однією із можливостей оптимізації є використання звичайного мешу, проте для більшості користувачів, це буде значним недоліком, адже з об'єктами Unity terrain дуже зручно працювати.

Зовнішній вигляд отриманого ландшафту залежить від обраних користувачем параметрів. Використання текстур високого розширення, різноманітні та якісні моделі рослинності дадуть змогу користувачеві створити ландшафт необхідної якості.

Таблиця 3. 2 - Порівняння програмних засобів за об'єктивними критеріями

Назва	Швидкодія (с)	Використання ОП	Розмір файлу
Розроблений ПЗ	100	-2ГБ	3 МБ
Procedural Terrain Generator	70	-1ГБ	1 МБ
Landscape Generator 3	110	-2ГБ	15МБ
FastNoise	160	-1ГБ	-

3.5 Висновки по розділу 3

В третьому розділі описано процес розроблення прототипу програмного додатку для процедурного синтезу тривимірних ландшафтів на основі пропонованого методу. Зроблено проектування ПЗ, наведено його архітектуру.

Ефективність розробленого ПЗ перевірено за допомогою порівняння результатів генерації ландшафтів з аналогічними ПП. Аналіз показав, що швидкодія алгоритму відповідає середній швидкодії існуючих алгоритмів. Також експертна оцінка показала що розроблений алгоритм видає реалістичний та детальний результат.

ВИСНОВКИ

В ході виконання дипломного проєкту було проведено огляд та аналіз способів та засобів синтезу тривимірних ландшафтів у геймдизайні. Описані джерела вихідних даних для синтезу тривимірних ландшафтів. Зроблено огляд методів автоматичної генерації ландшафтів. Обрано та описано процедурну генерацію ландшафтів, як автоматичного методу їх побудови. Зроблено постановку мети й завдань дослідження.

Розроблено удосконалений метод, який на відміну від існуючих дозволяє отримати візуально реалістичний результат. Детально описано кожен з етапів удосконаленого методу процедурного синтезу тривимірних ландшафтів у геймдизайні, який на відміну від існуючих використовує геодезичні карти як джерело інформації про ландшафт, що дозволяє підвищити рівень реалістичності графічного моделювання. Наведено схеми та підібрано математичні засоби для реалізації кожного етапу пропонованого методу

В роботі також було виконано програмну розробку. Тестування розробленого ПЗ показало, що швидкодія алгоритму відповідає середній швидкодії існуючих алгоритмів. Також експертна оцінка показала що розроблений алгоритм видає реалістичний та детальний результат.

ПЕРЕЛІК ПОСИЛАНЬ

1. Moss, Richard uses of procedural generation that all developers should study. January 1, 2016. - 1- 2p.
2. Massive Software About Massive. Retrieved 12 June 2016. - 1- 2p.
3. F. Kenton Musgrave, Craig E. Kolb and Robert S. Mace: The Synthesis and Ren-dering of Eroded Fractal Terrains. Computer Graphics, Volume 23, Number 3. July 1989,- 41-50p.
4. Alan Fournier, Don Fussell and Loren Car-penter: Computer Rendering of Stochastic Models. Communications of the ACM, Vol-ume 25, Issue 6 June 1982. - 41-50p.
5. Benes, B., Androgbatch, R. Layered data represen-tation for visual simulation of terrain erosion. In Proc. of the Spring Conf. on Comp. Graphics, 2001. - 80-85p.
6. Gain, J., Marais, P., Andstrasser, W.. Terrain sketch-ing. In Proc. of Interactive 3D graphics and games, 2009. - 31-38p.
7. Plessinger P. Voxel terrain engine Archived 2013-11-13 at the Wayback Machine", introduction. In a coder's mind, 2005. - 22p.
8. Ebert, D., Musgrave, K., Peachy, D., Perlin, K., Androwely, S. 1998. Texturing and Modeling: A Procedural Approach. Academic Press Professional 2005. - 7p.
9. Ebert, D., Musgrave, K., Peachy, D., Perlin, K., Androwely, S.. Texturing & Modeling, a Procedural Approach, 3rd ed. Morgan Kaufmann/Elsevier, Amsterdam. 2003. - 74p.

10. Stachniak, Stuerzlinger. An Algorithm for Automated Fractal Terrain Deformation. Computer Graphics and Artificial Intelligence. Ed. D Plemenos. ISBN 291425607-8, 64-76, May 2005. - 13p.
11. J. Jilesen et al., Three-dimensional midpoint displacement algorithm for the generation of fractal porous media, Journal Computers & Geosciences. Tarrytown, NY, US: September 2012. - 46p.
12. Jason Sanders and Edward Kandrot, "CUDA by Example", Addison-Wesley, 2012. - IOp.
13. Berry, J. K. Moving mapping to analysis of mapped data. GeoWorld(December), 2004. - 18-19p.
14. Gain, J., Marias, P., Andraster, W.. Terrain sketching. In Proc. of Interactive 3D graphics and games, 2009. - 31-38p.
15. A. Lenzhen, K. Rafi, J. Tao, The shadow of a Thurston geodesic to the curve graph, Journal of Topology, 2015. - 1085-1118p.
16. A. Al-Hussain and A. El-Zaart. Moment-preserving thresholding using gamma distribution. Volume 6, pages V6-323. IEEE, 2010. - 47p.
17. Nock and Frank Nielsen. Statistical Region Merging Richard 2004. - 12p.
18. E. Sharon, A. Brandt, and R. Basri, "Fast Multiscale Image Segmentation," Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, pp. 2000. - 70-77p.
19. Displays by C. Bond. An Efficient and Versatile Flood Fill Algorithm for Raster Scan, 2011. - 10-20p.

20. Field, D. J. and N. Brady . Visual sensitivity, blur and the sources of variability in the amplitude spectra of natural scenes. *Vision Research* 1997. - 3367-3383p.
21. Mark S. Nixon and Alberto S. Aguado. *Feature Extraction and Image Processing*. Academic Press, 2008. - 88p.
22. Perlin, Ken. "An Image Synthesizer". *SIGGRAPH Comput. Graph.* 19 July 1985.- 287-296p.
23. Hartj. C.: Perlin noise pixel shaders. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* 2001. - 87-94p.
24. Parent, A., M. Morin, and P. Lavigne. "Propagation of super-Gaussian field distributions." *Optical and quantum electronics* 24.9 1992. - 54p.
25. Freeman, R. Edward; Moutchnik, Alexander. "Stakeholder management and CSR: questions and answers". *UmweltWirtschaftsForum*. 2010. - 5-9p.
26. Freeman, R.E. and Reed, D.L., Stockholders and stakeholders: A new perspective on corporate governance. *California management review*, 25(3), 1983. - 88-106p.
27. Post, James (2002). "Redefining the Corporation: Stakeholder Management and Organizational Wealth". Stanford University Press. Retrieved 2009. - 51p.
28. DeGeorge, R.T. *Business Ethics*. Pearson Education, Inc. 2010. 192p.
29. Organisation for Economic Co-operation and Development. "Income". *OECD Better Life Index*. 4p.
30. Linden, Fabian. "A Marketer's Guide to Discretionary Income (abstract)". US Department of Education. Retrieved 2007-12-27. - 48p.

31. Peter Harris . Income tax in common law jurisdictions: from the origins to 1820, Volume 1.2006.-34p.
32. FASB, 2001. Improving Business Reporting: Insights into Enhancing Voluntary Disclosures. Retrieved on April 20, 2012 - 3p.
33. Williams, Jan R.; Susan F. Haka; Mark S. Bettner; Joseph V. Carcello 2008. Financial & Managerial Accounting. 2008 . p. 40

ДОДАТОК А

Лістинг фрагментів коду

```

using System; using System.Collections;
using System.Collections.Generic;
using Assets, scripts;
using strange.extensions.mediation, impl;
using strange.extensions.signal.impl;
using TreeOptimizer;
using UnityEngine;
using UnityEngine.UI;
using Random = UnityEngine.Random;
public interface ITerrainGenerator
{
void StartGeneration(); void LoadTerrain();
}
public class TerrainGeneratorMediator : Mediator
{
[Inject] public ITerrainModel TerrainModel { get; set; } [Inject] public
TerrainGeneratedSignal Signal { get; set; } [Inject] public LoadTerrainSignal LoadTerrainSignal {
get; set; }
[Inject] public LoadRequestSignal LoadRequestSignal { get; set; }

[Inject] public ITerrainView Terrain View { get; set; }
public override void OnRegister() {
base.OnRegister();
Signal.AddListener(OnTerrainGenerated);
LoadRequestSignal.AddListener(OnLoadRequest); }
private void OnLoadRequest(Terrain[,] terrains) { LoadTerrainSignal.Dispatch(terrains); }
private void OnTerrainGenerated(TerrainGenerator.GeneratedTerrainsData data) {
TerrainModel.SaveHeightsFloats(data.HeightFloats);
float width = data.TerrainsfO, OJ.terrainData.size.x; float height = data.TerrainsfO,
OJ.terrainData.size.z;

```

```

Terrain View.Init(data.Terrains, width, height, data);
for (var i = 0; i < data.splatMapData.GetLength(0); i++)
for (var j = 0; j < data.splatMapData.GetLength(1); j++) {
//TerrainModel.SaveSplatMapDetailsFloats(data.splatMapData[i, j].detailLayers, i, j);
TerrainModel.SaveSplatMapTexturesFloats(data.splatMapData[i, j].splatmapData, i, j);
TerrainModel.SaveTerrainData(data.TerrainDatas[i, j], i, j);
} }
public class TerrainGenerator : View, ITerrainGenerator
{
[Inject] public LoadRequestSignal LoadRequestSignal { get; set; }
[Inject] public TerrainGeneratedSignal TerrainGeneratedSignal { get; set; }
private const int OffsetForWaterBiom = -20;
private const float MultiplayerForRockBiom = 1,5f;
private const float RockHeight = 0.5f;
private const float ForestHeight = 0.2f;
private const float FieldHeight = 0.18f;
private const float WaterHeight = 0.15f;
private const float MaxHeightPerturbation = 0;//0.03f;
private const int SmoothIterations = 1;
private const int SmoothIterationsSecond = 1;
private const int TerrainHeight = 100;
private const float NoiseBigFrequency = 0.0025f; private const int NoiseBigOctaves = 3;
private const float NoiseBigLanctuary = 4f; private const float NoiseBigPersistence = 0.6f;
private const float NoiseSmallFrequency = 0.9f; private const int NoiseSmallOctaves = 7;

private const float NoiseSmallLanctuary private const float NoiseSmallPersistence private
const float NoiseSmallAmplitude =
[SerializeField] private GameObject water; [SerializeField] private readonly float bilboarStart
= 500;
[Header("Unity terrain optimizing settings")] [SerializeField] private readonly float
detailDinstance = 250f;
[SerializeField] private readonly float fadeDistance = 50; private readonly float height = 1,5f;
private readonly float heitTexturSclale = 1,4f;

```

```

//private readonly float heitTexturSclale = 1,5f; [SerializeField] private readonly int
maxMeshTrees = 1000;

private readonly int terrainResoltion = 1028; private readonly int terrainWidth = 1028;
[SerializeField] private readonly float treeDistance = 100; private int amountOfSplatMapsToAssign;
' = 2.5f; : = 0.15f; = 0.15f;

[SerializeField] private Texture2D ColorTexture2D;
public TextureCreator.ColorToBinomTypef] ColorsBinomTypes;
private DetailPrototypef] detailPrototypes;

[SerializeField] private GameObjectf] detailsPrefabs;
[Header("Details settings")] [SerializeField] private Texture2D[] detailsTextures;
[SerializeField] private Color grassTintColor;
private int heightCounter, heightMax, numberOfTerrains;
[SerializeField] private RawImage debugMap;
[SerializeField] private Texture2D heighTexture2D;
private float[,] heights;
private int heightTextureResolution;
private float hmHeight;
private float hmWidth;
[SerializeField] private Texture2D[] normalsForTerrainTexxuters;
[SerializeField] private Texture2D[] prefabsTexture2Ds;
private int prevX, prevY;
[SerializeField] private Transform rockPrefab;
private Dictionary<Vector2f, BiomSite> sites;
private Terrain terrain;
private GeneratedTerrainsData terrainData;
[SerializeField] private Terrainf,] terrains;
public Action terrainsReady;

[SerializeField] private Texture2D[] texturesForTerrain;
[SerializeField] private Transform treePrefab;
private TreePrototypef] treePrototypes;
[SerializeField] private GameObjectf] treesPrefabs;

```

```

private BiomVoronnoy voronoi; private Bioms bioms;
private float debuggTime;
public float[,] HeightFloats { get; private set;}
#region Load Terrain
public void LoadTerrain() {
GenerateTerrains();
ApplyPrototypesToTerrain();
PaintTerrain();
LoadRequestSignal.Dispatch(terrains); }
#endregion
private void OnGUI() {
float proggres = heightCounter / (float)heightMax;

string heightproggres = "Height progress: " + proggres;
if (heightMax != 1 && Mathf.FloorToInt(proggres) != 1)
GULLabel(new Rect(10,10, 500, 20), heightproggres); }
public class SplatMapData
{
public int[] coordinates; public float[,] splatmapData;
public SplatMapData(float[,] splatmapData) {this.splatmapData = splatmapData; } }
public struct GeneratedTerrainsData {
public Vector3 EndRiverLocal;
public Vector2 EndRiverTerrainIndex;
public float[,] HeightFloats;
public SplatMapData[,] SplatMapData;
public Vector3 StartRiverLocal;
public Vector2 StartRiverTerrainIndex;
public TerrainData[,] TerrainDatas;
public Terrain[,] Terrains;
public TreeGrid[,] TreeGrids;
public Bioms Bioms; }

//}

```

```

#region Common
private void ApplyPrototypesToTerrain() {
    treePrototypes = new TreePrototype[treesPrefabs.Length];
    //detailPrototypes = new DetailPrototype[detailsTextures.Length + detailsPrefabs.Length];
detailPrototypes = new DetailPrototype[detailsTextures.Length];
    for (var i = 0; i < treePrototypes.Length; i++) treePrototypes[i] = new TreePrototype {prefab
= treesPrefabs[i]};
    for (var i = 0; i < detailsTextures.Length; i++)
        detailPrototypes[i] = new DetailPrototype
        {
            prototypeTexture = detailsTextures[i],
            healthyColor = grassTintColor
        };
    for (var i = 0; i < detailPrototypes.Length; i++)
        // detailPrototypes[i] = new DetailPrototype {prototype = detailsPrefabs[i], prototypeTexture
= prefabsTexture2Ds[i], healthyColor = grassTintColor};
    terrain.terrainData.detailPrototypes = detailPrototypes;

    var _terrainDataSample = new TerrainData();
    heightTextureResolution = terrainResolution;//Mathf.FloorToInt(heighTexture2D.width /
heightTextureScale);
    _terrainDataSample.heightmapResolution = terrainResolution;
    _terrainDataSample.size = new Vector3(terrainWidth, terrainWidth, terrainWidth);
    numberOfTerrains = /*terrainResolution / _terrainDataSample.heightmapWidth +*/ 1;
    terrains = new Terrain[numberOfTerrains, numberOfTerrains];
    for (var i = 0; i < numberOfTerrains; i++)
        for (var j = 0; j < numberOfTerrains; j++) {
            var _terrainData = new TerrainData {heightmapResolution = terrainResolution, size = new
Vector3(terrainWidth, TerrainHeight, terrainWidth)};
            _terrainData.SetDetailResolution(terrainResolution, 8);
            var _terrain = Terrain.CreateTerrainGameObject(_terrainData);
            _terrain.transform.SetParent(transform);
            _terrain.GetComponent<Terrain>().heightmapPixelError = 1;

```



```

        _terrain.transform.position = new
Vector3(_terrain.GetComponent<Terrain>().terrainData.size.x * i, 0,
_terrain.GetComponent<Terrain>().terrainData.size.x * j);
        _terrain.gameObject.AddComponent<TerrainToolkit>();

        terrainsfi, j] = _terrain.GetComponent<Terrain>(); }
    }
#endregion
#region Generate terrain
public void StartGeneration() {
    debugTime = Time.timeSinceLevelLoad; GenerateTerrains();
    heightMax = 1; heightCounter = 1;
    StartCoroutine(GenerateHeightsToTerrains());
    ApplyPrototypesToTerrain(); }
private void PaintTerrain() {
    var waterPos = bioms.GetBiomCenters(BiomType.Water);
    foreach (var vector2 in waterPos) {

        GameObject.Instantiate(water, new Vector3(vector2.x, 188.36f, vector2.y),
Quaternion.identity); //todo height of
        terrain
    }
    textures = new SplatPrototypetexturesForTerrain.Length];
    for (var i = 0; i < texturesForTerrain.Length; i++) {
        texturesfi] = new SplatPrototype(); texturesfi] .texture = texturesForTerrainfi]; texturesfi]
.normalMap = normalsForTerrainTexxutersfi]; texturesfi] .tileOffset = new Vector2(0, 0);
        texturesfi] .tileSize = new Vector2(15,15); }
    for (var i = 0; i < terrains.GetLength(0); i++)
    for (var j = 0; j < terrains.GetLength(1); j++) {
        terrain = terrainsfi, j];
        terrain. terrainData.splatPrototypes = textures; terrain.terrainData.treePrototypes =
treePrototypes; terrain.terrainData.detailPrototypes = detailPrototypes; terrain.detailObjectDistance
= detailDinstance; terrain.treeBillboardDistance = bilboarStart; terrain.treeDistance = treeDistance;

```

```

terrain.terrainData.alphamapResolution = terrainResolution; terrain.treeCrossFadeLength =
fadeDistance; terrain.detailObjectDistance = 700; terrain. treeMaximumFullLODCount =
maxMeshTrees;

    var splatMapAssigner = terrain.gameObject.AddComponent<SplatAssigner>();
    var t = terrain.gameObject.AddComponent<TerrainView>();
    t. SetVoronnoi(voronoi.Bioms);
    splatMapAssigner.X = i;
    splatMapAssigner. Y = j;
    splatMapAssigner.Rock = rockPrefab;
    splatMapAssigner.Tree = treePrefab;
    splatMapAssigner.callbackAction = OnFinishGenerating;
    amountOfSplatMapsToAssign++;
    terrainsReady += splatMapAssigner. StartGeneration; }
}

private void OnFinishGenerating(TreeGrid treeGrid, float[, ] splatmapData, TerrainData
terrainDataPers, int[]
coordinates) {
    amountOfSplatMapsToAssign--;
    terrainData.splatMapData[coordinates[0], coordinates[1]] = new
SplatMapData(splatmapData); terrainData.TerrainDatas[coordinates[0], coordinates[1]] =
terrainDataPers; terrainData.TreeGrids[coordinates[0], coordinates[1]] = treeGrid; terrainData.Bioms
= voronoi.Bioms;

    if (amountOfSplatMapsToAssign == 0)
    TerrainGeneratedSignal.Dispatch(terrainData);
}

private void SmoothTerrains() {
    float hmWidth = terrainsfO, OJ.terrainData.heightmapWidth; float hmHeight = terrainsfO, 0]
.terrainData.heightmapHeight;
    for (var i = 0; i < heightTextureResolution; i++)
    for (var j = 0; j < heightTextureResolution; j++) {
        var xIndex = (int)(Mathf.FloorToInt(i) / hmWidth); var yIndex = (int)(Mathf.FloorToInt(j) /
hmHeight); terrain = terrainsfxIndex, yIndex];
        terrain.GetComponent<TerrainToolkit>().SmoothTerrain(3, 0.5f); }
}

```

```

    }
    private float[,] SmoothTerrain(Terrain terrain, float[,] heightMap, int shiftFromStart,
    int iterations) {
        var terData = terrain.terrainData; int Tw = terData.heightmapWidth;
        int Th = terData.heightmapHeight;
        TerrainToolkit.GeneratorProgressDelegate generatorProgressDelegate =
terrain.GetComponent<TerrainToolkit>().dummyGeneratorProgress;
        return terrain.GetComponent<TerrainToolkit>().smooth(heightMap, new Vector2(Tw, Th),
        generatorProgressDelegate, shiftFromStart, iterations); }
    private Enumerator GenerateHeightsToTerrains() {
        hmWidth = terrainsfO.terrainData.heightmapWidth; hmHeight =terrains[0,
0].terrainData.heightmapHeight; terrain = terrainsfO, 0];

        ColorTexture2D.Resize(Mathf.CeilToInt(hmWidth), Mathf.RoundToInt(hmHeight));
        heights = terrain.terrainData.GetHeights(0, 0, Mathf.FloorToInt(hmWidth),
Mathf.FloorToInt(hmHeight));
        heightMax = heightTextureResolution * heightTextureResolution;
        HeightFloats = new float[numberOfTerrains, numberOfTerrains] [,];
        float riverTopHeight = 0, riverBotHeight = 1000;
        float voronnoiScale = 10;
        var topRiverIndex = new Vector2();
        var botRiverIndex = new Vector2();
        var topRiverPos = new Vector3();
        var botRiverPos = new Vector3();
        var bounds = new Rectf(0, 0, hmWidth, hmHeight);
        var points = new List<Vector2f> {new Vector2f(Random.Range(0, hmWidth),
Random.Range(0, hmHeight))}; //todo random range??
        voronoi = new BiomVoronnoy(points, bounds, voronnoiScale, terrainResoltion);
        voronoi.resolution = terrainResoltion;
        voronoi.SetupBiomModule();
        bioms = voronoi.Bioms;
        sites = voronoi.SitesIndexedByLocation;
        for (var i = for (var j =

```

```

0; i < hmWidth; i+= 1) 0; j < hmHeight; j+= 1)
var textureScale = heighTexture2D. width / hmWidth;

// OperatePoint(i, j);
//OperatePoint(i+1 j+1);
//OperatePoint(i+3 j+3);
//OperatePoint(i+2j+2);
//OperatePoint(i j+1);
//OperatePoint(ij+2);
//OperatePoint(ij+3);
//OperatePoint(i+1 j);
//OperatePoint(i+2j);
//OperatePoint(i+3 j);
#endregion
#region oldAlgorithm
var xIndex = (int)(Mathf.FloorToInt(i) / hmWidth); var yIndex = (int)(Mathf.FloorToInt(j) /
hmHeight);
heights = HeightFloatsfxIndex, yIndex] ?? new float[Mathf.FloorToInt(hmWidth),
Mathf.FloorToInt(hmHeight)];
float pixelVal = heighTexture2D.GetPixel(Mathf.FloorToInt(i *
textureScale),Mathf.FloorToInt(j * textureScale)).grayscale;
float heightVal = pixelVal * height;// + Mathf.Pow(pixelVal, 3) * 2; //float heightVal =
pixelVal * height + Mathf.Exp(Power(pixelVal, 50));
riverTopHeight = heightVal;
topRiverPos = new Vector3(j - Mathf.FloorToInt(hmHeight * yIndex), height, i -
Mathf.FloorToInt(hmWidth * xIndex));
topRiverIndex = new Vector2(xIndex, yIndex);
//}
else {
if (heightVal < riverBotHeight) {
riverBotHeight = heightVal;
botRiverPos = new Vector3(j - Mathf.FloorToInt(hmHeight * yIndex), height, i -
Mathf.FloorToInt(hmWidth * xIndex));

```

```

    botRiverIndex = new Vector2(xIndex, yIndex); }
    */
    heights[j - Mathf.FloorToInt(hmHeight * yIndex), i - Mathf.FloorToInt(hmWidth * xIndex)]
= heightVal; HeightFloats[xIndex, yIndex] = heights;

    #endregion // ReSharper disable once CompareOfFloatsByEqualityOperator
    if (Math.Abs(Math.Truncate((double)heightCounter / 1000000) - heightCounter /
1000000f) == 0)
        yield return new WaitForEndOfFrame(); }
    heights
        =
        HeightFloats[Mathf.FloorToInt(topRiverIndex.x),
Mathf.FloorToInt(topRiverIndex.y)];
    heights[Mathf.FloorToInt(topRiverPos.x), Mathf.FloorToInt(topRiverPos.z)] = 250;
HeightFloats[Mathf.FloorToInt(topRiverIndex.x), Mathf.FloorToInt(topRiverIndex.y)] = heights;
    heights
        =
        HeightFloats[Mathf.FloorToInt(botRiverIndex.x),
Mathf.FloorToInt(botRiverIndex.y)];
        heights[Mathf.FloorToInt(botRiverPos.x),
Mathf.FloorToInt(botRiverPos.z)] = 250;
    HeightFloats[Mathf.FloorToInt(botRiverIndex.x), Mathf.FloorToInt(botRiverIndex.y)] =
heights;
    for (var i = 0; i < terrains.GetLength(0); i++)
    for (var j = 0; j < terrains.GetLength(1); j++) {
        terrain = terrainsfi, j]; terrain.heightmapMaximumLOD = 0;
        terrain.terrainData.SetHeights(0, 0, SMOothTerrain(terrain, HeightFloatsfi, j], 0,
SmoothIterations)); }
    // SetBorderlnessToTerrains();

    AddPertubration();
    for (var i = 0; i < terrains.GetLength(0); i++)
    for (var j = 0; j < terrains.GetLength(1); j++) {
        terrain = terrainsfi, j]; terrain.heightmapMaximumLOD = 0;
        terrain.terrainData.SetHeights(0, 0, SMOothTerrain(terrain, HeightFloatsfi, j], 0,
SmoothIterationsSecond)); }
    for (var i = 0; i < terrains.GetLength(0); i++)
    for (var j = 0; j < terrains.GetLength(1); j++) {

```

```

terrain = terrainsfi, j]; terrain.heightmapMaximumLOD = 0;
terrain.terrainData.SetHeights(0, 0, HeightFloatsfi, j]); }

terrainData = new GeneratedTerrainsData {HeightFloats = HeightFloats, Terrains = terrains,
TerrainDatalNumberofTerrains, питЪеЮГГеггав], splatMapData = new
SplatMapDatafnumberofTerrains, питЪеЮГГеггав], TreeGrids = new
TreeGridfnumberofTerrains, numberofTerrains], EndRiverLocal = botRiverPos,
EndRiverTerrainIndex = botRiverIndex, StartRiverLocal = topRiverPos, StartRiverTerrainIndex =
topRiverIndex};

Debug.LogError("DSD");
PaintTerrain();
ColorTexture2D.Apply();
debugMap.texture = ColorTexture2D;

debuggTime = Time.timeSinceLevelLoad - debuggTime;
Debug.Log("Time to complete hehight generation " + debuggTime);
terrainsReady.Invoke(); }

private void SetBorderlnessToTerrains() {
for (var i = 0; i < terrains.GetLength(0); i++)
for (var j = 0; j < terrains.GetLength(1); j++) {
//Terrain bot = null, top = null, left = null, right = null;
float[,] botHeight = null, leftHeight = null, topHeight = null, rightHeight = null;
if (i+ 1 < terrains. GetLength(0)) botHeight = HeightFloatsfi + 1, j]; if (i - 1 >= 0)
topHeight = HeightFloatsfi - 1, j]; if (j + 1 < terrains. GetLength(1)) rightHeight =
HeightFloatsfi, j + 1]; if(j-1>= 0) leftHeight = HeightFloatsfi, j - 1];
heights = HeightFloatsfi, j]; for (var k = 0; k < hmWidth; k++)
for (var l = 0; l < hmHeight; l++) {
if (rightHeight != null)

heights[Mathf.FloorToInt(hmWidth) - 1,1] = /*Mathf.Lerp(*/rightHeightfO,1]; //,
if (leftHeight != null)
heightsfO,1] = /* Mathf.Lerp(* leftHeight[Mathf.FloorToInt(hmWidth) -1,1];
if (botHeight != null)

```

```

    heightsfk, Mathf.FloorToInt(hmHeight) - 1] = /*Mathf.Lerp(* / botHeight[k, 0]; //, if
(topHeight != null)
    heightsfk, 0] = /*Mathf.Lerp(* / topHeightfk, Mathf.FloorToInt(hmHeight) - 1];
    //, heightsfk, 0], 0.5f); }
    HeightFloatsfi, j] = heights; }
    }
    private void OperatePoint(int i, int j)
    {
        var xIndex = (int)(Mathf.FloorToInt(i) / hmWidth); var yIndex = (int)(Mathf.FloorToInt(j) /
hmHeight);
        heights = HeightFloatsfxIndex, yIndex] ?? new float[Mathf.FloorToInt(hmWidth),
Mathf.FloorToInt(hmHeight)];
        var biomAtPoint = voronoi.RandomBiomTypeAtPoint(new Vector2f(i, j));
        Vector2 center = bioms.GetClosestBiomCenter(biomAtPoint, new Vector2(i, j));
        float heightVal = GetHeightForBiom(i, j, biomAtPoint, Vector2.Distance(center, new
Vector2(i, j)), bioms. GetBiomSize(biomAtPoint, center));
        // heightVal = heightVal * height + Power(heightVal, 3) * 2;
        heightsfi - Mathf.FloorToInt(hmHeight * yIndex), i - Mathf.FloorToInt(hmWidth * xIndex)]
= heightVal; HeightFloatsfxIndex, yIndex] = heights;
        Color color = Color, white;
        foreach (var colorToBinomType in ColorsBinomTypes) {
            if (colorToBinomType.biomType == biomAtPoint)
                color = colorToBinomType.color; }
        ColorTexture2D.SetPixel(i, j, color); }
        private void AddPertubration() {
            for (var i = 0; i < hmWidth; i += 1)
                for (var j = 0; j < hmHeight; j += 1) {
                    //var xIndex = (int)(Mathf.FloorToInt(i) / hmWidth); //var yIndex = (int)(Mathf.FloorToInt(j)
/ hmHeight);
                    // var biomAtPoint = voronoi.RandomBiomTypeAtPoint(new Vector2f(xIndex, yIndex));
                    heights = HeightFloatsfi, 0];
                    float heightVal = heightsfi, i];
                    heightVal += GetPertubratedHeight(i, j, heightVal);

```

```

HeightFloatsfi), 0] = heights; }
}
/* private void AddPertubration() {
for (var i = 0; i < hmWidth; i += 1)
for (var j = 0; j < hmHeight; j += 1) {
var xIndex = (int)(Mathf.FloorToInt(i) / hmWidth); var yIndex = (int)(Mathf.FloorToInt(j) /
hmHeight);
// var biomAtPoint = voronoi.RandomBiomTypeAtPoint(new Vector2f(xIndex, yIndex));
heights = HeightFloatsfxIndex, yIndex] ?? new float[Mathf.FloorToInt(hmWidth),
Mathf.FloorToInt(hmHeight)]; float heightVal = heightsfj - Mathf.FloorToInt(hmHeight * yIndex), i
- Mathf.FloorToInt(hmWidth * xIndex)]; heightVal += GetPertubratedHeight(i, j);
heightsfj - Mathf.FloorToInt(hmHeight * yIndex), i - Mathf.FloorToInt(hmWidth * xIndex)]
= heightVal;
HeightFloatsfxIndex, yIndex] = heights; }
} */
private float GetHeightForBiom(int i, int j, BiomType biom, float distanceToBiomCenter,
float biomSize) {
float pixelVal = 0;
case BiomType.Undefined:
Debug.LogErrorFormat("ReicivedUndefinedBiom at' {0}, {1}'", i, j); break;
case BiomType. Water:
// pixelVal += OffsetF orWaterBiom;
var ditanceInfluence = (biomSize / biomSize - distanceToBiomCenter / biomSize);
pixelVal = Mathf.Lerp(FieldHeight, WaterHeight,
ditanceInfluence*ditanceInfluence);//Mathf.Clamp(FieldHeight - ditanceInfluence , WaterHeight,
FieldHeight);
//pixelVal = Mathf.Clamp(WaterHeight +
(distanceToBiomCenter/voronoi.GetBiomSize(biom))*0.04f, FieldHeight, ForestHeight);
break;
case BiomType.Forest:
pixelVal = Mathf.Clamp(ForestHeight - distanceToBiomCenter / 500, FieldHeight,
ForestHeight);

```



```

var ss = pixelVal;
break;
case BiomType.Rock:
    pixelVal = Mathf.Clamp(RockHeight - distanceToBiomCenter / 1500, ForestHeight +
MaxHeightPertubration, RockHeight); //TODO MAGICK NUMBERS
    break;
case BiomType.Field:
    pixelVal = /*(pixelVal-0.5f) * 2 * MaxHeightPertubration +*/ +FieldHeight; break;
default:

throw new ArgumentOutOfRangeException("biom", biom, null); }
return pixelVal; }
private float GetHeightForBiom( BiomType biom) {
float pixelVal = 0;
switch (biom) {
case BiomType.Water: pixelVal = WaterHeight; break;
case BiomType.Forest: pixelVal +=ForestHeight; break;
case BiomType.Rock: pixelVal = RockHeight; break;
case BiomType.Field: pixelVal = FieldHeight; break; default:
throw new ArgumentOutOfRangeException("biom", biom, null); }

private float GetPertubratedHeight(int i, int j, float height) {
var noiseMethod = Noise.NoiseMethods[(int)Noise.NoiseMethodType.Perlin][2];
var pert = Noise.Sum(noiseMethod, new Vector3(i, height, j), NoiseBigFrequency,
NoiseBigOctaves, NoiseBigLanctuarity, NoiseBigPersistence);
pert += Noise.Sum(noiseMethod, new Vector3(j, height, i), NoiseSmallFrequency,
NoiseSmallOctaves, NoiseSmallLanctuarity, NoiseSmallPersistence) * NoiseSmallAmplitude;
return pert * O.If; }
#endregion using System;
using JetBrains.Annotations; using System.Collections; using System.Collections.Generic;
using System.Linq; using UnityEngine;
using Random = UnityEngine.Random;
public class MapParser : MonoBehaviour

```

```

{
[SerializeField] private Texture2D map; [SerializeField] private Transform debugTarget;
private const float PixelTreshold = 0.25f;
private const float DistanceToAdmitLineBelongs = 2; private const int
AmountOfNeighborsToSelectStartPoint = 10; public List<GLine> lines;
private MapInterpretator mapInterpretator = new MapInterpretator(); #region DEBUG
private bool drawing = false; #endregion
[ContextMenu("Parse map")]
public void ParseMapDebug() {
var pixels = Parse(map);
lines = GenerateLines(pixels);
Debug.LogFormat("Found '{0}' lines", lines.Count);
for (var i = 0; i < lines.Count; i++) {
var line = lines[i];
Debug.LogFormat("Line '{0}' contains '{1}' pixels", i + 1, line.BoardPixelsCount); }
filterMode
var texture = new Texture2D(map.width, map.height, TextureFormat.ARGB32, false) {
= FilterMode.Point
for (var i = 0; i < fillColorArray.Length; ++i) {
fillColorArray [i] = Color.white; }
texture.SetPixels(fillColorArray); texture.Apply();
for (var i = 0; i < lines.Count; i++) {
var line = lines[i];
foreach (var p in line.BoardPixels)
{
texture.SetPixel(Mathf.RoundToInt(p.Position.x),          Mathf.RoundToInt(p.Position.y),
line.LineColor);
}
}
texture.Apply();
debugTarget.GetComponent<Renderer>().material.mainTexture = texture;
Debug.LogFormat("Starting filling field pixels");
var markedColors = Parse(texture);

```

```

for (var i = 0; i < lines.Count; i++) {
    Debug.LogFormat("progres: " + i / lines.Count, "%"); var line = linesfi];
    FillLineFieldPixels(line, markedColors);
    Debug.Log("line " + i + " has " + line.FieldPixels.Count + " field pixels"); }
for (var i = 0; i < lines.Count; i++) {
    var line = linesfi];
    foreach (var p in line.FieldPixels)
    {
        texture.SetPixel(Mathf.RoundToInt(p.Position.x),          Mathf.RoundToInt(p.Position.y),
line.LineCollor);
    }
}
    texture.SetPixel(lines[0].EdgePixel.Position.x,  lines[0].EdgePixel.Position.y,  Color.red);
texture.SetPixel(lines[0].FLOODPixel.Position.x,  lines[0].FLOODPixel.Position.y,  Color.red);
texture.Apply();
    debugTarget.GetComponent<Renderer>().material.mainTexture = texture;

    mapInterpretator.GenerateHeightMap(lines, map. width,map.height,debugTarget); }
void Start() {
    StartCoroutine(ParseMap()); }
[ExecuteInEditMode]
IEnumerator ParseMap() {
    Debug.Log("Starting parsing map"); yield return new WaitForEndOfFrame(); var t =
Time.realtimeSinceStartup; var pixels = Parse(map); var time = Time.realtimeSinceStartup -1;
    Debug.LogWarning("Parsing map time: " + time);
    Debug.Log("Starting generating lines");
    yield return new WaitF orEndOfF rame();
    t = Time.realtimeSinceStartup;
    lines = GenerateLines(pixels);
    time = Time.realtimeSinceStartup -1;
    Debug.LogFormat("Found '{0}' lines", lines.Count);
    Debug.LogWarning("Parsing lines time: " + time);
    yield return new WaitForSeconds(0.If);

```

```

var line = linesfi];
Debug.LogFormat("Line '{0}' contains '{1}' pixels", i+ 1, line.BoardPixelsCount); }
var texture = new Texture2D(map.width, map.height, TextureFormat.ARGB32, false) {
filterMode = FilterMode.Point
};
for (var i = 0; i < lines.Count; i++) {
var line = linesfi];
foreach (var p in line.BoardPixels)
{
texture.SetPixel(Mathf.RoundToInt(p.Position.x),          Mathf.RoundToInt(p.Position.y),
line.LineCollor);
}
}
yield return new WaitForSeconds(0.5f); Debug.LogFormat("Starting filling field pixels");
var markedColors = Parse(texture);
yield return new WaitForSeconds(0.5f);
Debug.LogFormat("progres: " + i / lines.Count, "%");
var line = linesfi];
while (drawing) {
yield return new WaitF orEndOff rame(); }
texture.Apply();
debugTarget.GetComponent<Renderer>().material.mainTexture          =          texture;
StartCoroutine(debug_FillLineWithPixels(line,          markedColors)); //FilLineFieldPixels(line,
markedColors);
texture.SetPixel(linesfO].EdgePixel.Position.x,  lines[0].EdgePixel.Position.y, Color.red);
texture.SetPixel(lines[0].FLOODPixel.Position.x,  lines[0].FLOODPixel.Position.y, Color.red);
texture.Apply();
debugTarget.GetComponent<Renderer>().material.mainTexture = texture; }
texture.Apply();
debugTarget.GetComponent<Renderer>().material.mainTexture = texture;
while (drawing) {
yield return new WaitF orEndOff rame());

```

```

var line = linesfi];
foreach (var p in line.FieldPixels)
{
    texture.SetPixel(Mathf.RoundToInt(p.Position.x),          Mathf.RoundToInt(p.Position.y),
line.LineCollor);
}
}
texture.Apply();
debugTarget.GetComponent<Renderer>().material.mainTexture = texture; yield return new
WaitForSeconds(0.5f);
mapInterpretator.GenerateHeightMap(lines, map.width, map.height, debugTarget); }
private Enumerator debug_FillLineWithPixels(GLine line, Optimized2Array<Pixel> pixels)
{
    int i = 1; drawing = true;
    var pos = line.EdgePixel.Position; List<Pixel> neighborPixels = new List<Pixel>();
    for (i = 1; i < AmountOfNeighborsToSelectStartPoint; i++)
        neighborPixels.AddRange(GetNeighborPixels(pos, i, pixels)); }
    //var linePixels = neighborPixels.Where(p => p. Value == line.LineCollor).ToList(); var
linePixels = new List<Pixel>();
    foreach (var pixel in neighborPixels) {
        if (line.BoardPixels.Contains(pixel))
            linePixels Add(pixel); }
    var averagePos = new Vector2Int(0, 0);
    foreach (var linePixel in linePixels) {
        averagePos += linePixel.Position;
    }
    if (linePixels.Count() == 0) {
        averagePos = pos; }
    else {
        averagePos /= linePixels. CountQ;

```

```

while
(!ColorHelper.IsColorWhite(pixels[averagePos.x,averagePos.y],Value,PixelTreshould)) {
    averagePos += Vector2Int. right;
    if (pixelsfaveragePos.x, averagePos.y] == null)
        yield break; }
    line.FLOODPixel = new Pixel(averagePos, Color.red); yield return new
WaitForSeconds(0.If);
    Debug.Log("Found anchor point, begin to floodfill"); Debug.Log("Anchor point is " +
averagePos.ToString()); //FloodFill(averagePos, pixels, line);
    StartCoroutine(debug_FloodFill(averagePos, pixels, line)); }
    private Enumerator debug_FloodFill(Vector2Int startPosition, Optimized2Array<Pixel>
pixels, GLine line)
    {
        HashSet<Vector2Int> q = new HashSet<Vector2Int>(); HashSet<Vector2Int> qt = new
HashSet<Vector2Int>();
        q.Add( startPosition);
        line.AddFieldPixel(new Pixel(new Vector2Int(startPosition.x, startPosition.y),
line.LineCollor));

        while (q.Count > 0)
        {
            yield return new WaitF orEndOfF rame();
            foreach (var pixel in q)
            {
                w = pixel; e = pixel;
                while (ColorHelper.IsColorWhite(pixels[w.x -1, w.y],Value, PixelTreshould)) {
                    w += Vector2Int.left;
                    if(!pixels.InBounds(w + Vector2Int.left)) break;
                }
                // while (pixelsfe.x, e.y] == Color.white)
                while ((pixels.InBounds(e + Vector2Int.right)) && ColorHelper.IsColorWhite(pixels[e.x + 1,
e.y],Value,
PixelTreshould))

```

```

    {
    e += Vector2Int. right;
    if (!pixels.InBounds(e + Vector2Int.right))
    break; }
    if (ColorHelper.IsColorWhite(pixels[w.x + i, w.y],Value, PixelTreshould)) {
    pixelsfw.x + i, w.y]. Value = line.LineCollor;
    // var t = (Texture2D) debugTarget.GetComponent<Renderer>().material.mainTexture; //
t.SetPixel(w.x + i, w.y, Random.ColorHSV(0.6f,1)); //1. Apply ();
    //debugTarget.GetComponent<Renderer>().material.mainTexture = t; //yield return new
WaitF orEndOfF rame();
    line.AddFieldPixel(new Pixel(new Vector2Int(w.x + i, w.y), line.LineCollor));
    pixelsfw.x + i, w.y]. Value = line.LineCollor;
    }
    if (ColorHelper.IsColorWhite(pixels[w.x+ i, w.y + 1].Value, PixelTreshould)) qt.Add(new
Vector2Int(w.x + i, w.y + 1));
    if (ColorHelper.IsColorWhite(pixels[w.x+ i, w.y - 1].Value, PixelTreshould)) qt.Add(new
Vector2Int(w.x + i, w.y - 1));
    }
    for (int i = 0; i <= e.x - pixel.x; i++) {
    if (ColorHelper.IsColorWhite(pixels[e.x - i, e.y],Value, PixelTreshould)) {
    pixelsfe.x - i, e.y],Value = line.LineCollor;
    // var t = (Texture2D) debugTarget.GetComponent<Renderer>().material.mainTexture; //
t.SetPixel(e.x - i, e.y, Random.ColorHSV()); //1. Apply ();
    // debugTarget.GetComponent<Renderer>().material.mainTexture = t; //yield return new
WaitF orEndOfF rame();
    line.AddFieldPixel(new Pixel(new Vector2Int(e.x - i, e.y), line.LineCollor));
    pixelsfe.x - i, e.y],Value = line.LineCollor;
    }
    if (ColorHelper.IsColorWhite(pixels[e.x - i, e.y + 1].Value, PixelTreshould)) qt.Add(new
Vector2Int(e.x - i, e.y + 1));
    if (ColorHelper.IsColorWhite(pixels[e.x - i, e.y - 1].Value, PixelTreshould)) qt.Add(new
Vector2Int(e.x - i, e.y - 1));
    } }

```

```

q.Clear();
q.UnionWith(qt); qt.Clear();
var tex = (Texture2D) debug Target. GetComponent<Renderer>().material.mainTexture; var
c = Random.ColorHSV();
foreach (var i in q) {

tex.Apply();
debugTarget.GetComponent<Renderer>().material.mainTexture = tex; }
Debug.Log("Floodfil of line has finished, found " + line.FieldPixels.Count + "field pixels");
drawing = false; }
private void FilLineFieldPixels(GLLine line, Optimized2Array<Pixel> pixels) {
int i = 1;
var pos = line.EdgePixel.Position; List<Pixel> neighborPixels = new List<Pixel>();
for (i = 1; i < AmountOfNeighborsToSelectStartPoint; i++) {
neighborPixels.AddRange(GetNeighborPixels(pos, i, pixels)); }
//var linePixels = neighborPixels. Where(p => p. Value == line.LineCollor).ToList(); var
linePixels = new List<Pixel>();
foreach (var pixel in neighborPixels)
linePixels.Add(pixel); }
var averagePos = new Vector2Int(0, 0);
foreach (var linePixel in linePixels) {
averagePos += linePixel.Position;
}
averagePos /= linePixels. CountQ;
line.FLOODPixel = new Pixel(averagePos, line.LineCollor);
FloodFill(averagePos, pixels, line); }
private List<GLLine> GenerateLines(Optimized2Array<Pixel> pixels) {
List<GLLine> lines = new List<GLLine>();
for (int i = 0; i < pixels.GetLength(0); i++) {
for (int j = 0; j < pixels.GetLength(1); j++) {

if (pixels[i, j].Gray scaleNormalized > PixelTreshould) {
bool existingLinePixel = false;

```



```

foreach (var line in lines) {
    if (line.BoardPixels.Contains(pixels[ij]))
    {
        existingLinePixel = true;
        break; }
    }
    if (! existingLinePixel) {
        var gline = ParseGLineFromPixel(new Vector2Int(i, j), pixels);
        lines.Add(gline); }
    } } }
return lines; }

HashSet<Pixel> GetNeighborPixels(Vector2Int position, int offset,
Optimized2Array<Pixel> pixels) {
    HashSet<Pixel> neighborPixels = new HashSet<Pixel>();

    var p10 = GetPixelFromArray(position.x + offset, position.y, pixels);
    var p11 = GetPixelFromArray(position.x + offset, position.y + offset, pixels);
    var p01 = GetPixelFromArray(position.x, position.y + offset, pixels);
    var p10 = GetPixelFromArray(position.x - offset, position.y, pixels);
    var p11 = GetPixelFromArray(position.x - offset, position.y + offset, pixels);
    var p1m1 = GetPixelFromArray(position.x + offset, position.y - offset, pixels);
    var p0m1 = GetPixelFromArray(position.x, position.y - offset, pixels);
    var pmlm1 = GetPixelFromArray(position.x - offset, position.y - offset, pixels);
    if (p10 != null) neighborPixels.Add(p10); if (p11 != null) neighborPixels.Add(p11);
    if (p01 != null) neighborPixels.Add(p01);
    if (p10 != null) neighborPixels.Add(p10);
    if (p11 != null) neighborPixels.Add(p11);
    if (p1m1 != null) neighborPixels.Add(p1m1);
    if (p0m1 != null)
    if (pmlm1 != null) neighborPixels.Add(pmlm1);
    return neighborPixels; }

[CanBeNull]
Pixel GetPixelFromArray(int i, int j, Optimized2Array<Pixel> pixels) {

```

```

return pixels.InBounds(iJ) ? pixels[i,j] : null;
}
GLine ParseGLineFromPixel(Vector2Int pos, Optimized2Array<Pixel> pixels) {
List<Vector2Int> posList = newList<Vector2Int>() {pos};
GLine line = new GLine(new float[pixels.GetLength(0), pixels.GetLength(1)]);
bool foundNeighborPixel = true;
HashSet<Pixel> neighborPixels = new HashSet<Pixel>();
int curOffset = 1;
var p = pixels[pos.x,pos.y];
line.AddBorderPixel(p);
while (foundNeighborPixel) {

for (var i = 0; i < posList.Count; i++) {
var curPos = posList[i];
neighborPixels = GetNeighborPixels( curPos, curOffset, pixels);
foreach (var pixel in neighborPixels) {
if ((pixel.GrayscaleNormalized < PixelTreshould) || !line.AddBorderPixel(pixel)) continue;
//Debug .Log(pixel. Value);
foundNeighborPixel = true; //todo contitnue here curOffset = 1;
posList.Add(pixel.Position);
}
if (! foundNeighborPixel && curOffset < DistanceToAdmitLineBelongs) {
curOffset++; posList.Remove(curPos); foundNeighborPixel = true;
} } }
return line;
}

private Optimized2Array<Pixel> Parse(Texture2D texture) {
for (int j = 0; j < texture.width; j++) {
texture.SetPixel(j, 0, Color.black);
texture.SetPixel(j, texture.height - 1, Color.black);
}
for (int j = 0; j < texture.height; j++) {

```

```

texture.SetPixel(0,j, Color.black);
texture.SetPixel(texture.width - 1,j, Color.black);
}
texture.Apply();
debugTarget.GetComponent<Renderer>().material.mainTexture = texture;
Optimized2Array<Pixel> pixels = new Optimized2Array<Pixel>(texture.height,
texture.width);
for (int i = 0; i < texture.height; i++) {
for (int j = 0; j < texture.width; j++) {
var pixel = texture.GetPixel(i, j);
//pixelsfi, j] = Mathf.Pow((1 - pixel.grayscale)*5,2);
pixelsfi, j] = new Pixel(new Vector2Int(i, j), pixel);
}
return pixels; }
/*
* Flood-fill (node, target-color, replacement-color):
If target-color is equal to replacement-color, return.
If color of node is not equal to target-color, return.
Set Q to the empty queue.
Add node to Q.
For each element N of Q:
Set w and e equal to N.
Move w to the west until the color of the node to the west of w no longer matches target-color.
Move e to the east until the color of the node to the east of e no longer matches target-color.
For each node n between w and e:
Set the color of n to replacement-color.
If the color of the node to the north of n is target-color, add that node to Q.
If the color of the node to the south of n is target-color, add that node to Q.
Continue looping until Q is exhausted.
Return. */
private void FloodFill(Vector2Int startPosition, Optimized2Array<Pixel> pixels, GLine line)
{
List<Vector2Int> q = new List<Vector2Int>();

```

```

List<Vector2Int> qt = new List<Vector2Int>(); q.Add( startPosition);
line.AddFieldPixel(new Pixel(new Vector2Int(startPosition.x, startPosition.y),
line.LineCollor)); Vector2Int w, e;
while (q.Count > 0)
{
foreach (var pixel in q)
{
w = pixel; e = pixel;
while (pixelsfw.x, w.y].Value == Color.white) {
w += Vector2Int.left;
}
while (pixelsfe.x, e.y].Value == Color.white) {
e += Vector2Int. right;
}
for (int i = 0; i <= pixel.x - w.x; i++) {
if (pixelsfw.x + i, w.y]. Value == Color, white) {
pixelsfw.x + i, w.y]. Value = line.LineCollor;
line.AddFieldPixel(new Pixel(new Vector2Int(w.x + i, w.y), line.LineCollor));}
if (pixelsfw.x + i, w.y + 1]. Value == Color, white) qt.Add(new Vector2Int(w.x + i, w.y + 1));
if (pixelsfw.x + i, w.y - 1]. Value == Color.white) qt.Add(new Vector2Int(w.x + i, w.y - 1));
}
for (int i = 0; i <= e.x - pixel.x; i++) {
if (pixelsfe.x - i, e.y]. Value == Color.white)
{
pixelsfe.x - i, e.y].Value = line.LineCollor;
line.AddFieldPixel(new Pixel(new Vector2Int(e.x - i, e.y), line.LineCollor));
}
if (pixelsfe.x - i, e.y + 1]. Value == Color.white) qt.Add(new Vector2Int(e.x - i, e.y + 1));
if (pixelsfe.x - i, e.y - 1]. Value == Color.white) qt.Add(new Vector2Int(e.x - i, e.y - 1));
} }
q.ClearQ;

qt.Clear(); }}

```