VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROCEDURÁLNÍ GENEROVÁNÍ KRAJINY V UNITY

PROCEDURAL GENERATION OF TERRAIN IN UNITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DENNIS VYMER**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ MILET**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Vymer Dennis**
Program: Informační technologie
Název: **Procedurální generování krajiny v Unity**
Procedural Generation of Terrain in Unity
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte engine Unity a metody procedurálního generování.
2. Navrhněte plugin pro generování otexturovaných modelů terénu. Zaměřte se na vizuální kvalitu.
3. Implementujte navržený plugin.
4. Zhodnoťte dosažené výsledky a vytvořte demonstrační video.

Literatura:

- dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a prototyp pluginu.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Milet Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Tato bakalářská práce se zabývá procedurálním generováním krajin. Cílem je navrhnout a implementovat aplikaci, která dokáže vygenerovat krajinu na základě uživatelských vstupů. Práce zahrnuje generování výškové mapy a vodních ploch, aplikaci textur, polohování vegetace a simulaci eroze. Důraz je kladen na uživatelské rozhraní při tvorbě výškové mapy, jelikož výšková mapa ovlivní všechny následující kroky.

Abstract

This bachalor thesis is about procedural terrain generation. The goal is to design and implement an aplication that can generate a landscape based on user input. The work includes generation of an elevation map and water areas, the application of textures, vegetation placement and the simulation of erosion. Emphasis is placed on the user interface when creatin an elevation map, as the elevation map affects all subsequent steps.

Klíčová slova

3D, procedurální generování, procedurální generování krajin, node editor, Unity, C#

Keywords

3D, procedural generation, procedural landscape generation, node editor, Unity, C#

Citace

VYMER, Dennis. *Procedurální generování krajiny v Unity.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

Procedurální generování krajiny v Unity

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Mileta. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Dennis Vymer
27. května 2020

Poděkování

Na tomto místě chci poděkovat vedoucímu mé bakalářské práce, panu Ing. Tomášovi Miletovi za cenné rady, čas, ochotu a dostatek nápadů při konzultacích.

Obsah

Kapitola 1

Úvod

Tato bakalářská práce se zabývá procedurálním generováním krajin. Popisuje, jakým způsobem je navrhnuta a implementována aplikace, která vygeneruje krajinu na základě uživatelských vstupů. Vygenerovaná krajina zahrnuje výškovou mapu převedenou na otexturovaný 3D objekt, simulaci eroze, vegetaci a vodní plochy. Důraz je kladen na uživatelský vstup při generování výškové mapy, jelikož tato ovlivňuje následné úpravy, a na možnost manuálních uživatelských úprav libovolných detailů po vygenerování krajiny.

Spolu se zlepšováním výkonnosti výpočetních strojů, jako jsou počítače a mobilní telefony, roste také složitost, rozmanitost a propracovanost elektronických her. S rychle se zvětšujícím počtem hráčů elektronických her se tvoří celé nové průmyslové odvětví. Toto odvětví již nezahrnuje pouze hry, ale také sledování herních zápasů (e-sporty), vznik herních influencerů a streamerů.

Velkým milníkem se stal 28. červenec roku 2019. V tento den se konalo mistrovství světa ve hře Fortnite. Mezi výherce bylo rozděleno 30 milionů dolarů a živá sledovanost tohoto mistrovství přesahovala 2 miliony lidí, přičemž následné sledování záznamů bylo několikanásobné [14]. Toto značí vzestup popularity e-sportů. Dále například Wimbledon Gentleman's Finale, které se konalo v podobný čas, zaznamenalo 3.3 milionů živých diváků [7], ale mnohem menší následnou sledovanost ze záznamů.

Tento rychlý nárůst popularity vedl ke změnám ve způsobu vývoje elektronických her. Dříve si každá ze společností, která vyvíjela elektronické hry, zároveň vyvíjela i vlastní herní engine[1]. Se vznikem samostatných herních enginů, jako je například Unity, Unreal Engine a Godot, se mohou na trhu uplatnit i menší vývojové skupiny či dokonce jedinci. Předpřipravených herních enginů ale začínají využívat i velké společnosti. Mezi nejznámější hry vyvinuté pomocí samostatných herních enginů patří tyto:

Unity

- Ori and the Blind Forest - 2D logická akční adventura,
- Wasteland 2 - role play hra v post-apokalyptickém světě,
- Rust - videohra o přežití, pouze pro více hráčů 1.1,
- Hearthstone - sběratelská karetní hra publikovaná společností Blizzard Entertainment.

[1] Program, který slouží k vývoji her. Slouží jako API, které v sobě obsahuje různé předem připravené herní objekty, jako jsou například kamery, fyzikální systém, apod.

Unreal Engine 4

- ARK: Survival Evolved - videohra s hlavním cílem přežít v akčním světě 1.1,
- Final Fantasy VII Remake - akční role play hra,
- Fortnite - online battle royale hra pouze pro více hráčů.

(a) Snímek ze hry Rust.

(b) Snímek ze hry Ark: Survival Evolved.

Obrázek 1.1: Snímky krajin z úspěšných her vytvořených pomocí herních enginů.

1.1 Procedurální generování

Procedurální generování je používáno v počítačové grafice v různých podobách. Lze jej použít pro generování různých materiálů jako je dřevo či hlína, dále také pro generování oblačnosti, terénu, vodních struktur či celých měst.

Procedurální techniky mají dlouhou historii využívání, ale skutečný průlom nastal až v roce 1985, kdy profesor Ken Perlin představil svoji verzi nekonečného šumu, později po něm pojmenovaného jako Perlinův šum [12]. Tento šum se stal základním stavebním kamenem procedurálního generování. Popularita a složitost procedurálního přístupu poté začala jen stoupat s výkonnějšími procesory a grafickými kartami.

Při procedurálním generování je možno využít mnoha různých postupů. Lze využít například L-systémů, neuronových sítí, 3D snímků krajin a mnoha dalších.

Nejpopulárnější postupy jsou založeny na fraktálních algoritmech. Tento přístup nese několik výhod oproti jiným způsobům, mezi hlavní se řadí znatelné snížení paměťových nároků, jednoduchá modifikace a možnost unikátnosti. S tímto přístupem jsou ale spojeny i určité nevýhody, zejména problematická tvorba realisticky vypadajících terénů a procesorová kapacita.

V praxi se často využívají přístupy kombinované. V hrách s velkými světy se mnohdy procedurálně vygeneruje krajina, kterou následně skupina umělců upravuje tak, aby se takto vygenerovaný svět blížil jejich představám.

Vytváření procedurálních krajin je důležitou součástí nejen herního průmyslu, je také využíváno v průmyslu filmovém či při různých fyzikálních simulacích.

Kapitola 2

Procedurální generování

Procedurální generování spočívá v algoritmickém generování, probíhá tedy bez manuálního zásahu ve fázi modelování objektu. Například při generaci terénu místo manuálního tvarování výškové mapy a texturování využijeme variaci matematických funkcí.

2.1 Vlastnosti procedurálního generování

Mezi základní vlastnosti procedurálního generování se řadí abstrakce, která nám umožňuje značné úspory paměti. Namísto ukládání celých objektů a jejich vlastností můžeme tedy objekt znázornit v algoritmické podobě.

Nevýhodou využití abstrakce je však výpočetní náročnost algoritmu, protože jsou uloženy jen matematické reprezentace objektu a nikoliv vlastní data.

2.2 Pseudonáhodná čísla

Při procedurálním generování je často využívána pseudonáhodnost. Pokud by se využilo ryze náhodného generování, mohlo by to vést k více různým výsledkům se stejnými parametry, což by znamenalo nekonzistenci programu, která je samozřejmě nežádoucí. Proto se při pseudonáhodném generování čísel využívá „semínka“ (anglicky „seed“), které se předloží pseudonáhodnému generátoru, aby generoval náhodná čísla vždy ve stejném pořadí.

Lineární kongruentní generátor

Mezi nejjednodušší a nejpoužívanější generátory pseudonáhodných čísel se řadí lineární kongruentní generátor. Základní tvar rovnice je následující:

$$x_{i+1} = (a \cdot x_i + b) \ mod \ m \tag{2.1}$$

kde a, b a m jsou zvolené konstanty. Počáteční hodnota x_0 je semínkem. Výsledná hodnota se vždy nachází v intervalu $0 < x_i < m$ a po m vygenerovaných hodnot se čísla začnou znovu opakovat [4].

Nevýhodou je omezená perioda pseudonáhodně generovaných hodnot, předvídatelnost výsledných hodnot a rozdělení výsledných hodnot.

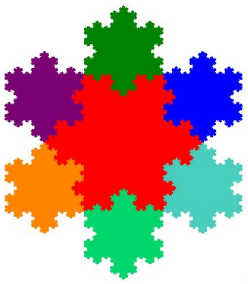
Příklad lineárního kongruentního generátoru s použitím semínka je velmi jednoduchý k implementaci, jak lze vidět v útržku 2.1.

```
public sealed class PseudoRandomNumberGenerator
{
    private readonly unsigned long int i_x;

    public PseudoRandomNumberGenerator(int seed = 1)
    {
        this.i_x = seed;
    }

    public double NextNumber(){
        i_x = i_x * 69069L + 1;
        return i_x / ((double) ULONG_MAX + 1);
    }

    public double NextNumber(double minValue, double maxValue){
        return minValue + (NextNumber() * (maxValue - minValue));
    }
}
```

Výpis 2.1: Implementace kongruentního pseudonáhodného generátoru v jazyce C♯. Třída vrací pseudonáhodná čísla s periodou 2^{32}. Možnost využít stejného počátečního bodu.

2.3 Fraktál

Fraktál vzniká periodickým opakováním zvoleného geometrického tvaru, jako například známý Koch Snowflake, který lze vidět na obrázku 2.1. Hlavní vlastností fraktálů je jejich sebepodobnost nezávislá na měřítku [13].



Obrázek 2.1: **Koch Snowflake** je příklad fraktálu. Prostřední část je složena z šesti úhelníků (červená oblast). Tyto úhelníky se opakují šestkrát ve zmenšené podobě na krajích původního úhelníku a tak tvoří sobě podobný útvar ve zvětšené podobě. Obrázek převzat z [13].

Při procedurálním generování lze využít těchto vlastností z trošku jiného hlediska, a to tak, že na sebe budeme skládat jednoduché funkce, čímž nám vznikají na sebe navazující funkce složitější. Výhoda spočívá zejména v jednoduchosti výpočtu.

2.4 Šumové a fraktálové algoritmy

Při generování krajiny nelze použít jen pseudonáhodné generátory čísel, protože jejich výsledné hodnoty mohou obsahovat velké náhlé skoky. Tyto skoky by ve vygenerované krajině nevypadaly příliš reálně. Je zapotřebí využít složitějších funkcí, které generují plynule se měnící pseudonáhodné hodnoty.

2.4.1 Perlinův šum

Již dříve zmíněný Perlinův šum znamenal průlom v počítačové grafice, obzvláště v odvětví procedurálního generování. Tento druh šumu je využíván pro různé efekty jako jsou například plameny ohně, kouř a vodní hladiny. Tento algoritmus je nejčastěji implementován jako dvou, tří nebo čtyř rozměrná funkce (může být definován jako n-rozměrná funkce).

Implementace se typicky dělí na 3 kroky:

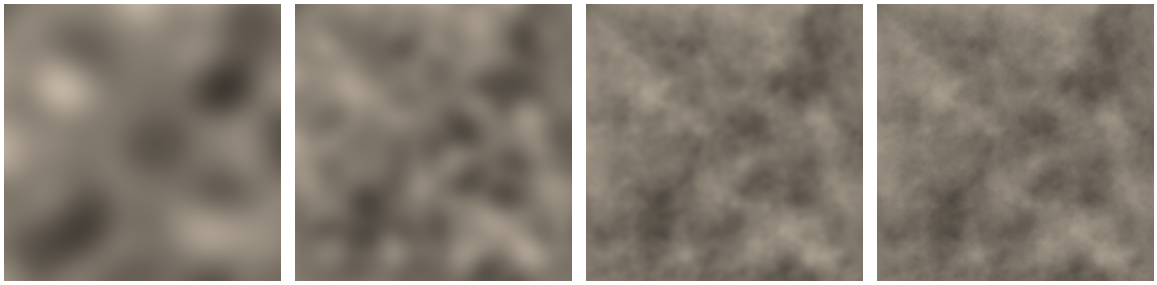
- definice mřížky s náhodnými gradientovými vektory,
- počítání DOT produktu mezi směrovými vektory a
- interpolace mezi těmito hodnotami, k interpolaci se využívá funkce $f(t) = -3t^2 - 2t^3$.

Tato implementace s sebou nesla jeden velký nedostatek, a to nenulovou druhou derivaci v bodech $t = 0$ a $t = 1$. Tato nenulovost vytváří nespojitost funkce, která poté může vést k nepředvídatelnému chování. Tato chyba byla odstraněna použitím nové interpolační funkce, $f(t) = -6t^5 - 10t^4 + 15t^3$. Po úpravě interpolační funkce a provedení dalších drobných úprav, které vedly ke zrychlení výpočtu, byl v roce 2002 vydán Vylepšený Perlinův šum (anglicky „Improved Perlin Noise“, také známo jako „Simplex noise“) [12].

V praxi se většinou využívá multi-fraktálového Perlinova šumu skládajícího se z Perlinova šumu s různými frekvencemi (počtem opakování za jednotku) a amplitudami (intenzitou), příklad algoritmu 1. Typicky se s větší frekvencí využívá nižší amplitudy, rozdílné výsledky lze vidět na obrázku 2.2. Tyto fraktály se běžně nazývají oktávy.

Algorithm 1: Generation of Perlin noise value for input (X, Y) coordinates with multiple octaves. Decreasing amplitude with increasing frequency.

Result: Perlin noise value consisting of multiple octaves.
octaves ← number of wanted octaves;
persistance ← value, which is less than one;
lacunarity ← value, which is greater than one;
frequency ← 1.0;
amplitude ← 1.0;
result ← 0.0;
while *number of octaves is greater than zero* **do**
  value ← $PerlinNoise(X \cdot frequency, Y \cdot frequency)$;
  result ← $result + value \cdot amplitude$;
  amplitude ← $amplitude \cdot persistance$;
  frequency ← $frequency \cdot lacunarity$;
  octaves ← octaves − 1
end



(a) 1 oktáva (b) 2 oktávy (c) 4 oktávy (d) 8 oktáv

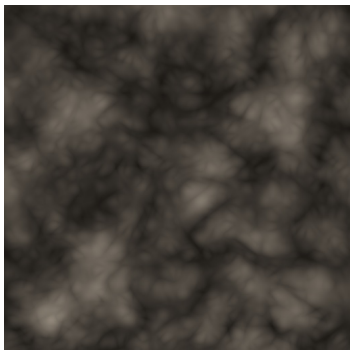
Obrázek 2.2: Příklady Perlinova šumu s různými oktávami, stejnou persistencí a přiblížením.

2.4.2 Rigid šum

Základem generování je využití již existujícího způsobu generování šumu, například multifraktálního Perlinova šumu. Rozdílem je, že výstup každé oktávy je modifikovaný funkcí, která vrací vždy kladnou hodnotu, například můžeme brát jen absolutní hodnoty z oktáv, jak je znázorněno v rovnici 2.2.

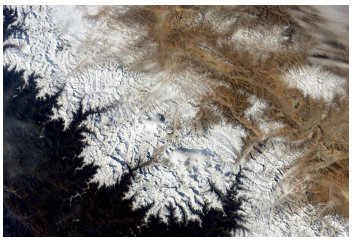
$$RigidNoise[x, y] = AbsoluteValue(PerlinNoise[x, y])) \tag{2.2}$$

Tímto postupem opět vzniká unikátní šum, který na první pohled reprezentuje říční delty či horské struktury, což lze porovnat na obrázcích 2.3.



(a) Rigid šum, 4 oktávy.

(b) Delta řeky. Převzato z [1].



(c) Himalájské pohoří. Převzato z [3].

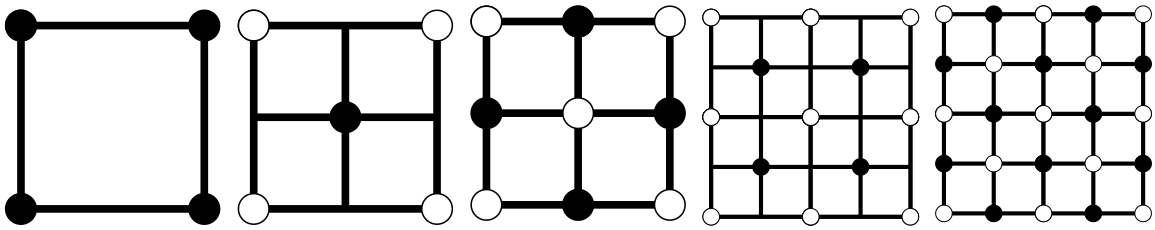
Obrázek 2.3: Porovnání vygenerovaného Rigid šumu s přírodními jevy.

2.4.3 Algoritmus Diamant-čtverec

Řadí se mezi populární, jednoduché a rychlé fraktálové algoritmy pro generování výškových map. Výstup algoritmu je dvourozměrné pole se stranou o velikosti mocniny čísla dva plus jedna. Algoritmus nejprve nastaví výšky rohových hran na pseudonáhodné hodnoty a následně opakuje diamantový krok a čtvercový krok, lze vidět na obrázku 2.4, dokud nejsou všechny hodnoty výsledného pole nastaveny [8].

Opakující se kroky jsou následující:

- **Diamantový krok** - Z bodů aktivního čtverce se vypočítá aritmetický průměr, přičte se k němu pseudonáhodná hodnota a bod uprostřed aktivního čtverce se nastaví na takto vypočtenou hodnotu.
- **Čtvercový krok** - Vyplní zbylé hodnoty mezi Diamantovými kroky, opět sečtením aritmetického průměru okolních hodnot s pseudonáhodnou hodnotou.



(a) Inicializace krajních bodů. (b) Diamantový krok. (c) Čtvercový krok. (d) Následné iterace diamantových kroků. (e) Následné iterace čtvercových kroků.

Obrázek 2.4: Vizualizace algoritmu Diamant-čtverec. Prvním krokem je inicializace krajních hodnot výsledného pole. Poté se střídá diamantový krok s čtvercovým krokem, dokud nejsou vypočteny všechny hodnoty výsledného pole.

Výhoda využití tohoto algoritmu oproti využití multi-fraktálových šumových algoritmů spočívá v tom, že se každý bod počítá pouze jednou. Při výpočtu například osmi oktáv u Perlinova šumu se počítá pro každý bod 8 různých hodnot, které se poté sčítají. Diamant-čtverec algoritmus nám umožní rekurzivně vypočítat hodnotu pro každý bod při jednom průchodu.

2.5 Přírodní elementy

Nepostradatelnou součástí jakékoliv krajiny jsou také přírodní elementy, jako je voda, její vliv a vegetace.

2.5.1 Vodní eroze

Vodní eroze simuluje změny v terénu způsobené tekoucí vodou, která rozpouští malé množství materiálu a transportuje jej na jiné místo. Při simulaci vodní eroze se kvůli její rychlosti musí klást důraz na nejdůležitější části přírodního jevu. Jednotlivé kapky vody z deště za miliony let zanechají v krajinách jedinečné struktury. Tyto se v simulaci jen velmi zjednodušeně napodobují, aby bylo možné dosáhnout určité spokojenosti s vzhledem výsledné krajiny. Erozi lze rozdělit do následujících základních kroků [2]:

1. Vznik nové vody, například kapky deště.
2. Eroze terénu pod vzniklou vodou a odebrání části sedimentu.
3. Transport sedimentu.
4. Vypařování vody a průběžná depozice sedimentu.

2.5.2 Vodní plochy

Součástí kompletní krajiny jsou i vodní plochy. Tyto lze simulovat či generovat různými způsoby, záleží na jaký typ vodní plochy se chceme zaměřit. Nicméně se dá obecně stanovit, že je u simulace vodních ploch nutné zohlednit průhlednost na základě hloubky, různé zbarvení vody, vlnění, směr toku a distorze objektů pod vodní hladinou.

2.5.3 Vegetace

Při generování vegetace se mohou brát v potaz různé vlivy a je zapotřebí zohlednit různé technické a biologické parametry [9]:

Vlivy při generování vegetace

- Druh rostliny: například smrk či buk.
- Vzdálenost mezi rostlinami: minimální vzdálenost mezi dvěma rostlinami, často je vázána také na velikost daného druhu rostliny.
- Skupina rostlin: různé druhy rostlin mohou patřit do stejné skupiny rostlin, protože například rostou ve stejné nadmořské výšce.
- Pokrytí rostlin: množství zástupu druhu rostliny v daném prostředí či krajině.

Dále jsou k umístění rostlin potřeba následující vstupní technické údaje:

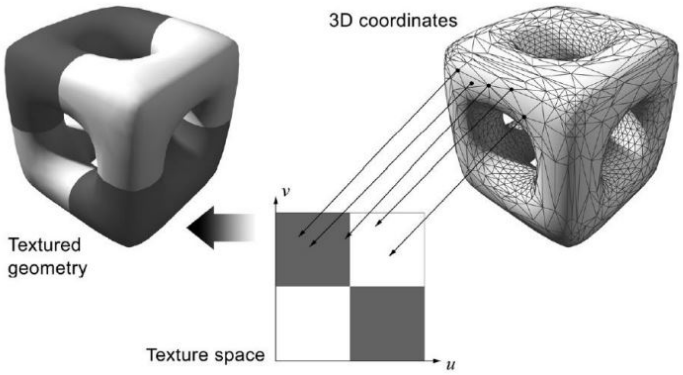
Technické údaje potřebné k rozmístění

- Vygenerovaná mapa: je zapotřebí pro výpočet vlastností vegetace a výpočet distribuční mapy.
- Vegetační data: vstupní výše uvedené vlastnosti jednotlivých druhů rostlin.
- Modely rostlin: jeden či více modelů pro daný druh rostliny.
- Texturové mapy: používány k dekoraci modelů rostlin.

Za pomocí výše zmíněných vlivů a údajů lze zjednodušeně polohovat vegetaci do krajiny. Není tedy nutné brát v potaz množení rostlin, simulovat růst rostlin či simulovat různé přírodní vlivy.

2.5.4 Textury

Textury přináší vhodný způsob jak zvýšit vizuální kvalitu krajiny za nízkou výpočetní cenu. Aplikování textury na vygenerovaný objekt lze provádět mnoha způsoby, zvolený přístup záleží na objektu, který má být texturován.



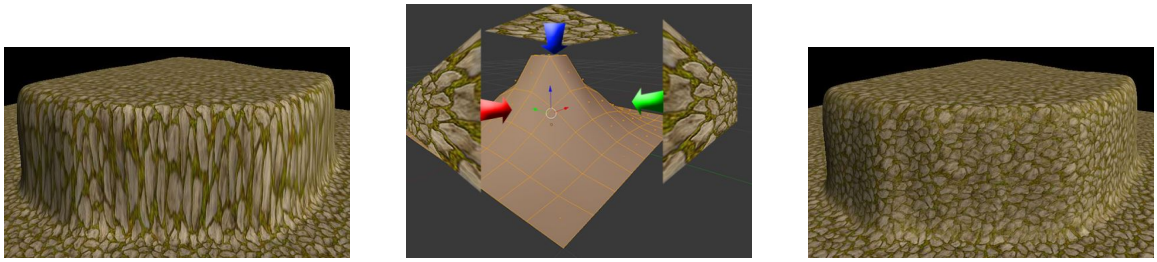
Obrázek 2.5: Vizualizace přímého mapovaní textur podle souřadnic. Převzato z [11].

Přímé mapování textury

Při mapování textur přímo na objekt se využívá souřadnicového systému, který určuje pozice textury na objektu 2.5. Souřadnicové systémy mohou být použity různé, obecně se nazývají „texture coordinates“.
Často lze využít sekce stejné textury na více bodů objektu a tudíž není nutné tyto sekce textury opakovat, mohou se pak vícekrát využít stejné texturové souřadnice k mapování. Tento přístup také umožňuje využití funkce k mapování textury na objekt. To lze například využít při mapování textury na šachové pole.

Triplanar texture mapping

Je-li potřeba nanést texturu na povrch, který není rovný, mohou vzniknout nechtěné vedlejší efekty, jako je roztáhnutí či deformace textury. Tento přístup řeší problém tím, že se namapuje textura ze všech tří stran, což lze vidět na obrázku 2.6.



(a) Příklad roztáhnutí textury. (b) Vizualizace postupu. (c) Výsledek mapovaní.

Obrázek 2.6: Vizualizace Triplanar texture mappingu. Převzato z [10].

Splat mapping

Splat mapping je metoda kombinování více textur, kde se na nejvyšší úrovni využívá tzv. alfamapy, do které se zapisuje pro každý bod procentuální zastoupení vybraných textur. Toto nám umožní ž„malovat” více textur přes sebe, což ulehčí problém plynulých přechodů mezi různými texturami, znázorněno na obrázku 2.7.

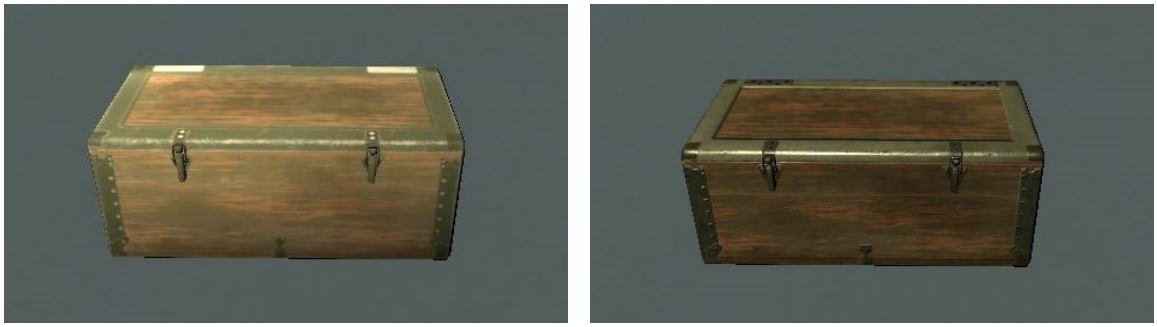
Při aplikaci dvourozměrných textur se nejčastěji používá trojrozměrná alfa mapa k zapisování procentuálního zastoupení textur, alfa mapa má tedy typicky o jeden rozměr více, než mají aplikované textury.

(a) První textura. (b) Druhá textura. (c) Výsledná textura složená z první a druhé.

Obrázek 2.7: Vizualizace plynulého přechodu mezi texturami za použití splat mappingu. Výsledná textura obsahuje pro každou texturovou souřadnici normalizované zastoupení první a druhé textury. V levém horním rohu výsledné textury by tedy zastoupení první textury bylo rovno jedné a druhé textury rovno nule. V pravém dolním rohu by zastoupení textur bylo opačné.

Normal mapping

Normal maps (do češtiny lze přeložit jako mapy s normálovými vektory, ale spíše se používá anglická verze) jsou vhodné pro přidávání detailů do textury. Tato metoda je velmi užitečná, protože nemusíme přidávat detaily přímo do meshe vygenerovaného objektu, ale pouze pozměníme normálové vektory daného objektu a docílíme tím značné vizuální změny, jak lze vidět na obrázku 2.8.



(a) Příklad objektu bez použití mapy s normálovými vektory. (b) Příklad objektu za použití mapy s normálovými vektory.

Obrázek 2.8: Vizualizace stejného objektu, s mapou s normálovými vektory a bez.

Kapitola 3

Návrh řešení

Tato kapitola ze zabývá upřesněním cílů této práce a návrhem řešení. Dále popisuje také architekturu aplikace, která nepřímo popisuje tok práce s aplikací.

3.1 Upřesnění cílů

Procedurální generování krajiny je velice rozsáhlý pojem. Existuje mnoho velmi rozdílných krajin, které lze generovat. Krajinu v přírodě ovlivňuje mnoho faktorů, tyto mohou být buď ryze přírodní, které většinou tvarují geologické vlastnosti, anebo ovlivněné lidským faktorem, tyto mají hlavní vliv na biosféru.

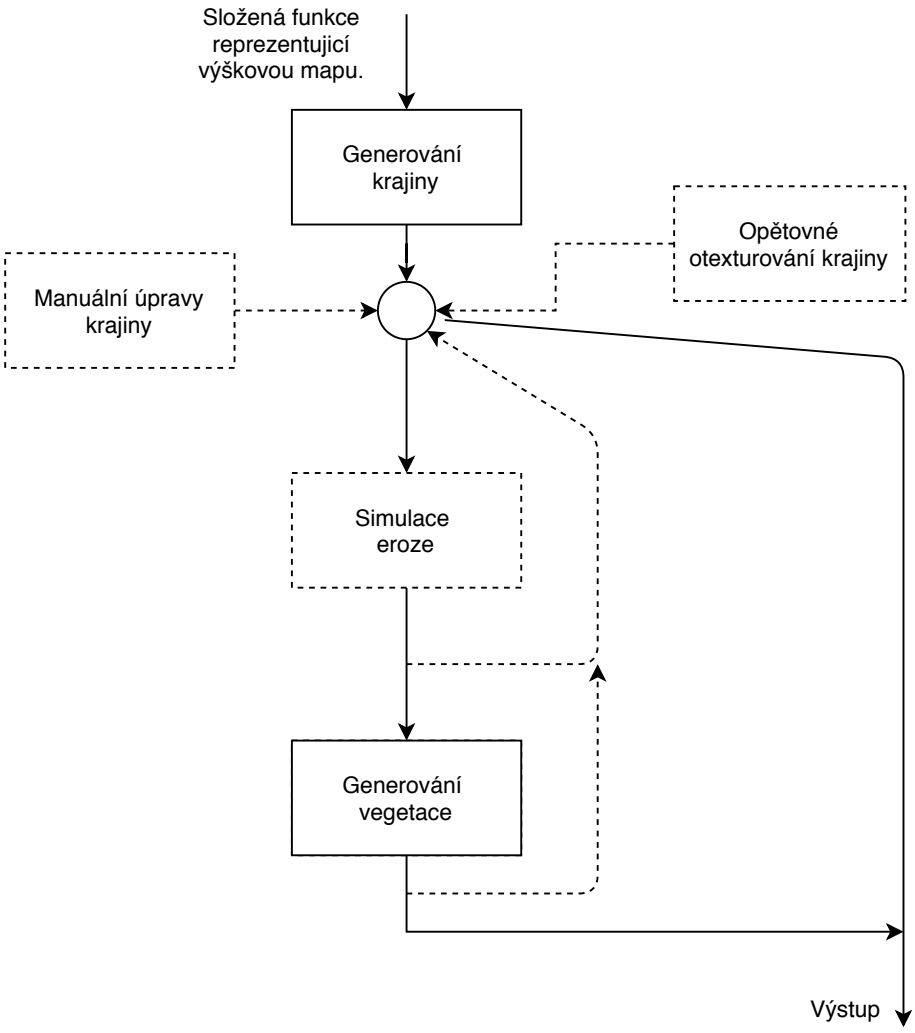
Pokud by procedurální generátor krajiny měl být schopen generovat a simulovat všechny tyto faktory a umět generovat všechny různé druhy krajin, které na světě můžeme nalézt, stal by se z toho obrovský, příliš složitý projekt. Je tedy vhodné generátor specializovat.

Generátor krajin, který tato bakalářská práce popisuje, klade důraz na generování středně velkých ostrovních útvarů, příklad lze vidět na obrázku 3.1. Tyto útvary bývají obklopeny mořem či oceánem, který je využíván k doplnění scenérie. Není tedy kladen hlavní důraz na vodní plochy, ty spíše naznačují nemožnost úniku z ostrova. Důležité je, aby byla umožněna rozmanitost biomů, aby byla k dispozici různá místa k prozkoumání. Také musí být umožněna manuální úprava scenérie, aby bylo možné doplnit uživatelsky specifické požadavky.

Obrázek 3.1: Příklad reálného ostrova, který slouží jako inspirace pro tuto práci. Převzato z [6].

3.2 Architektura

Celý proces generování krajiny lze rozdělit do čtyř částí: generování výškové mapy, otexturování vygenerované krajiny, simulace vodní eroze a generování vegetace. Všechny fáze jsou závislé na vygenerovaném terénu. Na obrázku 3.2 lze vidět propojení jednotlivých fází, jejich vstupy a výstupy.



Obrázek 3.2: Obrázek popisuje fáze generování krajiny. Popisuje vstupy, výstupy a možnost manuální úpravy v každé fázi generování. Vstupem generátoru výškové mapy je uživatelem sestavená složená funkce, která je převedena na 3D objekt. Následný objekt lze poté již libovolně upravovat dál. Uživatel tedy může simulovat erozi, manuálně upravit krajinu, opět nanést textury, nebo vygenerovat vegetaci. Tyto kroky lze opakovat v libovolném pořadí a libovolném množství.

Architektura aplikace je navrhnuta tak, aby uživateli umožnila co největší kontrolu nad prováděnými procesy a průběžnou možnost úpravy. Je tedy po jakékoliv fázi možné zasáhnout do již vygenerovaných částí a ty upravit dle libosti.

3.2.1 Práce s výškovou mapou

Jelikož je možné použít velké množství zdrojů výškové mapy a tyto libovolně slučovat či transformovat, musí se dodržet jednotná práce s výškovou mapou. Výšková mapa je reprezentována dvourozměrným float polem. Zvolí-li se čtvercová reprezentace výškové mapy, usnadní to následnou práci.

Rozměr výškové mapy je proměnlivý, což znamená, že se musí algoritmy přizpůsobit velikosti pole. Pro generování Perlinova šumu či Rigid šumu to není příliš velký problém, ten nastává až při generaci výškové mapy pomocí algoritmu Diamant-čtverec. Tento algoritmus vždy generuje dvourozměrné pole o velikosti mocniny čísla dva plus jedna. Kdyby výšková mapa musela mít rozměr, který by odpovídal tomuto požadavku, tak by se velikosti výškové mapy příliš rychle zvyšovaly do bodu, kde by již výpočetní doba mapy nebyla akceptovatelná. Tento problém lze vyřešit tím, že se vždy najde první číslo mocniny čísla dva, které je větší anebo rovno velikosti výškové mapy. Poté se vygeneruje výšková mapa algoritmem Diamant-čtverec a hodnoty, které přesahují požadovanou velikost pole, se zahodí. Toto řešení není příliš náročné, jelikož je tento algoritmus velice rychlý.

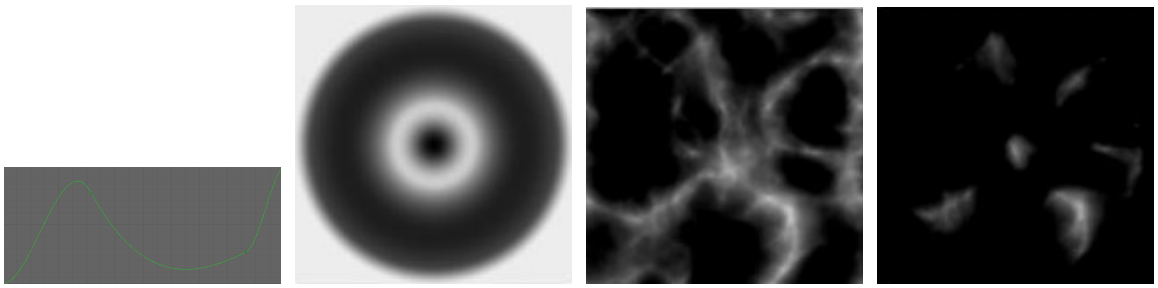
Výšková mapa má fixní velikost, se kterou všechny třídy počítají. Dále se také bude výšková mapa držet v rozsahu $< 0, 1 >$. Tento rozsah usnadní pozdější generování výsledné krajiny, protože pole již bude normalizované. Problémy mohou vznikat při transformacích výškové mapy, například při sčítání dvou výškových map či odečítání, kde hodnoty mohou lehce přesáhnout daný rozsah. Toto lze jednoduše vyřešit ořezáním hodnot výsledné výškové mapy.

3.2.2 Generování výškové mapy

Vytvoření výškové mapy je navrhnuto tak, že si uživatel může pomocí uživatelského rozhraní vytvořit vlastní složenou funkci z dílčích operací a podle ní poté vygenerovat výškovou mapu. Jako dílčí operace lze využít buď výše zmíněné fraktálové a šumové algoritmy, předem definované transformace, nebo definovat vlastní uživatelské operace.

Fall-off mapa

Fall-off mapa je nutnou součástí při generování ostrovních struktur. Složenou funkci lze poupravit tak, že uživatel definuje křivku, podle které bude upraven každý bod. Ten je upraven tak, že se určí vzdálenost bodu od středu, tato vzdálenost se normalizuje a poté se vyhodnotí podle uživatelem definované křivky.



(a) Grafická deklarace vstupní funkce. (b) Fall-off mapa vygenerovaná ze vstupní funkce. (c) Složená funkce před aplikací fall-off mapy. (d) Složená funkce po odečtení fall-off mapy.

Obrázek 3.3: Vizualizace případu použití fall-off mapy.

3.2.3 Vodní hladina

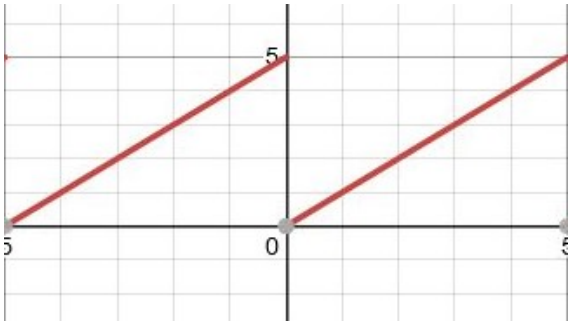
Hlavní vodní plochou při generování ostrovních struktur je moře či oceán, tedy velká spojitá vodní hladina. Tato hladina hraje ve hrách důležitou roli, protože uživateli dává najevo, že z ostrova nelze odejít. Ve většině her bývá tato vodní masa spíše poklidná a není na ní kladen hlavní důraz.

Pohyb vodní hladiny a vlnění

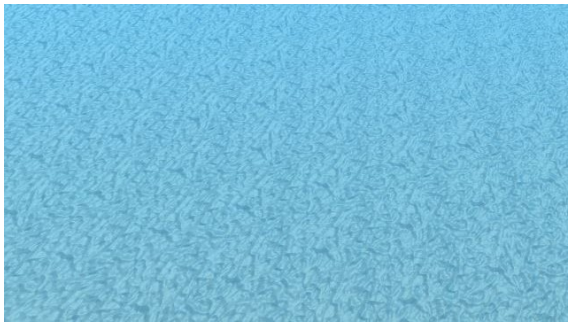
Vodní hladina, jako jakýkoliv pevný objekt, je jen otexturovaná plocha. Rozdílem je, že tato plocha se hýbe a bývá průhledná. Pohyb vodní hladiny, tedy textury nanesené na plochu, lze docílit animací UV souřadnic [1]. Tuto animaci lze simulovat přičítáním časové hodnoty k UV souřadnicím. Takový krok nám umožní plynulý tok textury v jeden směr, což nesimuluje příliš tok vody. Je vhodné toto rozšířit o distorzi, která může být buď náhodná, nebo může být použita textura. Použití textury značně ulehčuje tvorbu reálného toku, problém vzniká při dlouhém pozorování stejného místa na vodní hladině, protože se po stejném časovém okamžiku opakuje.

Při simulaci vlnění není možné posouvat a překreslovat vrcholy meshe, protože by to bylo výpočetně nereálné. Jelikož tento projekt generuje středně velké ostrovy, rozměr cca 9km x 9km, musí se 3D efekt vlnění pouze simulovat. K simulaci 3D efektu vlnění lze využít měnění normálových vektorů. Další bod je směr vlnění, k tomuto směru je použita textura obsahující směrové vektory. Pomocí těchto vektorů lze poté posouvat UV souřadnice. Tyto směry lze aplikovat jen po určitou dobu a poté se musí začít od začátku, protože by výsledně aplikovaná textura jinak byla k nepoznání. Výsledný jev opakující se textury lze vidět na obrázku 3.4. Kvůli zamezení tohoto jevu je použit pilový vzor (anglicky „sawtooth pattern“), rovnice pilového vzoru 3.1. Tento vzor umožní reset texturových souřadnic od počátku. Dále je vhodné tento přechod rozmazat pomocí Perlinova šumu, aby efekt v čase resetu nebyl příliš znatelný.

$$x(t) = t \mod amplitude \tag{3.1}$$



(a) Pilový vzor vytvořený pomocí výše zmíněné rovnice 3.1, kde amplituda je rovna pěti.



(b) Výsledné vlnění, region o velikosti 1km x 1km. Lze vidět opakující se texturu.

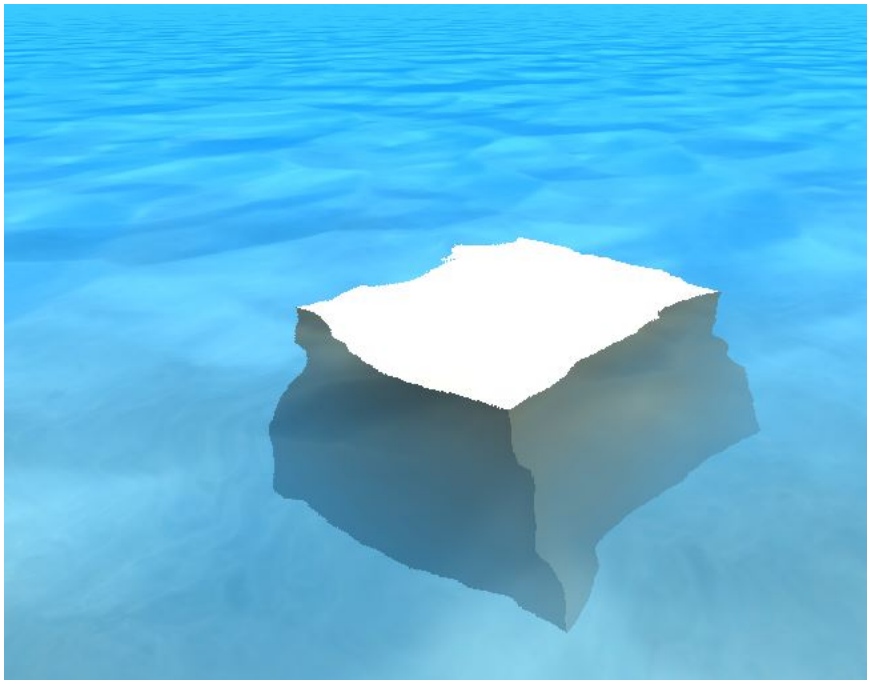
Obrázek 3.4: Snímek vlnění vody vytvořený pomocí výše zmíněných postupů.

[1] UV souřadnice jsou to stejné jako souřadnice texturové. Používají se k reprezentaci 3D objektu ve 2D, což zjednodušuje mapování textur na 3D objekt.

Distorze objektů pod vodní hladinou

Když foton vnikne do vody, v důsledku odlišné optické hustoty vody se pohybuje jinou rychlostí. Tento jev se projevuje zejména tehdy, když je ve viditelné vzdálenosti pod vodní hladinou nějaký objekt. Objekt poté vypadá jako by byl ohnutý, znázorněno na obrázku 3.5.

K docílení tohoto jevu je nejvhodnější použít aktuální hodnotu normálového vektoru, stačí přičíst normálový vektor jako offset pro UV souřadnice. Normálový vektor udává směr a velikost vlnění, což souhlasí s aproximací směru distorze objektu pod vodní hladinou. Dále je potřeba získat objekty, které jsou vykreslované pod vodní hladinou. Tyto objekty lze nechat předem vykreslit do textury a poté je jednoduše pomocí UV souřadnic indexovat, či pomocí normálových vektorů posunout indexované UV souřadnice.

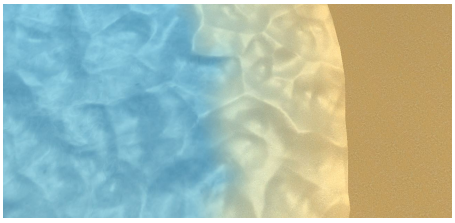


Obrázek 3.5: Snímek zobrazující posunutý objekt pod vodní hladinou.

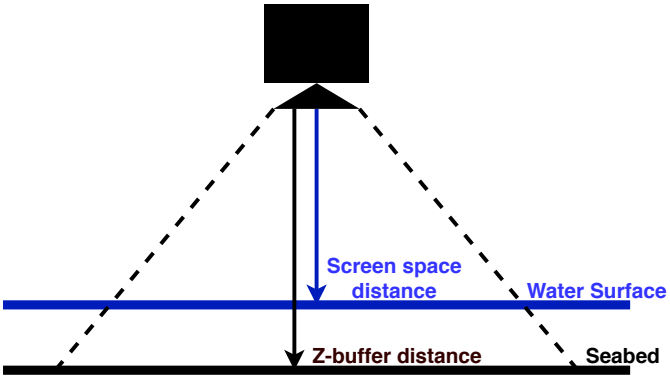
Průhlednost vodní hladiny

Vodní hladina je vždy průhledná, ale jen do určité hloubky. S rostoucí hloubkou se průhlednost vody snižuje. Hloubku vodní hladiny lze vypočítat pomocí aktuálně vykresleného obrazu z kamery. Z aktivní kamery lze získat linearizovanou Z-buffer[2] hodnotu aktuální souřadnice, v shaderu nazývanou jako fragment. Tato hodnota bude ignorovat vodní hladinu, protože vodní hladina se vykresluje jako transparentní vrstva, tedy Z-buffer ji ignoruje. Získanou relativní hodnotu lze převést do řádných souřadnic. Dále lze získat vzdálenost aktuálního fragmentu od kamery (tzv. „screen space coordinate“). Pomocí těchto hodnot je možné vypočítat hloubku pod vodní hladinou, znázorněno na obrázku 3.6. Poté lze již lineárně interpolovat intenzitu zabarvení fragmentu vodní hladiny podle hloubky s tím, že můžeme definovat maximální hloubku viditelnosti.

[2]Pole hodnot v rozmezí $< 0, 1 >$, které určuje relativní vzdálenost od kamery.



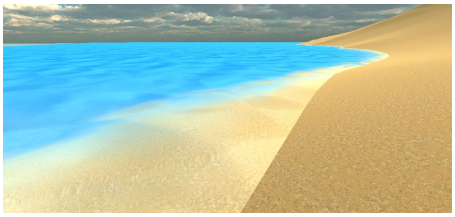
(a) Kolmý pohled na vodní hladinu.



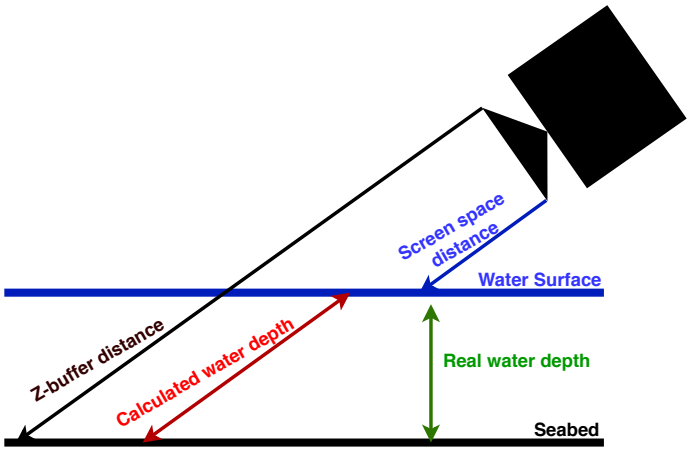
(b) Grafika popisující získávaní hloubky pod vodní hladinou.

Obrázek 3.6: Snímky zobrazující postup získávání hloubky pod vodní hladinou. Pomocí Z-buffer hodnoty aktivní kamery lze získat vzdálenost hladiny od kamery. Dále v shaderu, který je aplikovaný na vodní hladině, lze získat vzdálenost aktuálního fragmentu od aktivní kamery. Poté rozdíl těchto hodnot udává hloubku pod vodní hladinou.

Tento postup s sebou však nese jednu nevýhodu. Dívá-li se uživatel na vodní hladinu kolmo z vrchu dolů, vše funguje dle představ. Je-li pozorována vodní hladina ze strany, tedy pod úhlem jiným, než kolmým, tak Z-buffer hodnota fragmentu již nebude odpovídat kolmé hloubce pod vodní hladinou a může dojít k jistému zkreslení. Toto zkreslení je ale v případě této aplikace pouze nepatrné 3.7.



(a) Pohled na vodní hladinu, který zkresluje hloubku pod vodní hladinou.



(b) Grafika znázorňující důvod vzniklého artefaktu.

Obrázek 3.7: Snímky zobrazují vzniklé zkreslení hloubky pod vodní hladinou. Na prvním obrázku lze vidět, že vypočtená hloubka neodpovídá opravdové hloubce pod vodní hladinou, protože se nebere v úvahu úhel pozorování.

3.2.4 Simulace vodní eroze

Výše zmíněný přírodní jev vodní kapky je převeden do algoritmické podoby následovně:

Algorithm 2: Simplified droplet algorithm using the most important parts of the actuall process.

```
Result: Droplet Simulation
Set maximum lifetime of droplet;
while lifetime is greater than zero do
    Droplet acquires sediment from its current position according to the angle of
     that point;
    Droplet finds the lowest neighbour in its Moore neighbourhood;
    if found neighbour is lower than current position then
        Move droplet to the lowest neighbour;
        Leave an amount of the acquired sediment according to the angle of the
         point;
        Reduce lifetime by one;
    else
        break;
    end
end
```

Tento algoritmus je dostatečně zjednodušený, takže umožní dostatečné množství iterací simulace vodní eroze, které jsou zapotřebí pro středně velké krajiny.

3.2.5 Generování vegetace

Generování stromů je rozděleno do podkategorií podle druhu rostliny. Pro každý druh lze vybrat na jakém druhu pevniny (v případě programu textury) roste, kolikrát se na mapě může maximálně nalézt a vymezit sklon terénu, na kterém může růst.

Travnaté plochy není nutné dělit dle druhu pevniny, zcela dostačující je jeden druh travnatého porostu. Travnaté plochy v krajině spíše zaplňují prázdná místa a slouží jako podpůrný element. Uživatel tedy má možnost definovat množství zatravnění, toto zatravnění je poté generováno pomocí Perlinova šumu, který umožní vetší spojité travnaté plochy.

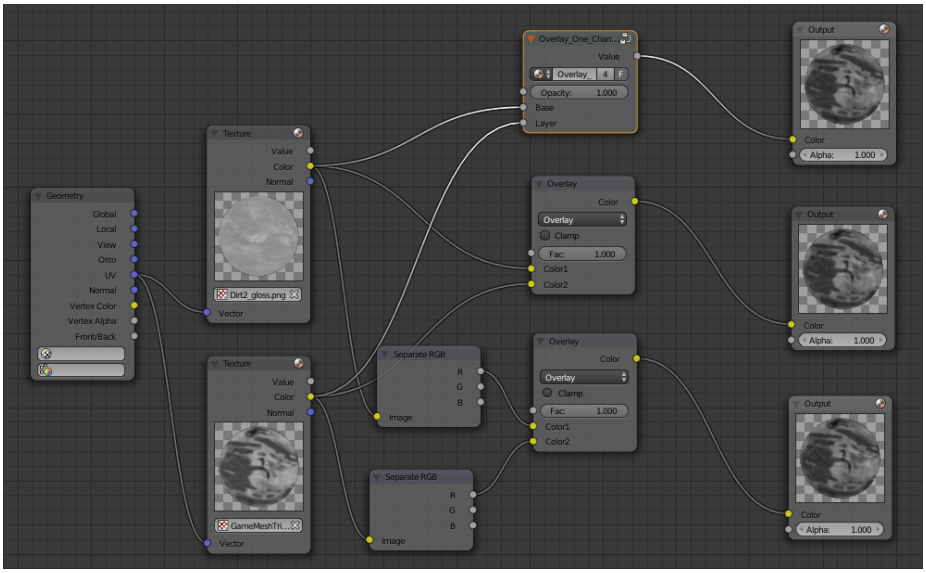
3.3 Uživatelské rozhraní

Součástí řešení projektu je i tvorba uživatelského rozhraní. Tento projekt uživatelské rozhraní dělí do výše zmíněných kategorií. K přírodním elementům je implementováno jednodušší uživatelské rozhraní, aby nebyl koncový uživatel přehlcen informacemi. Tato rozhraní umožňují měnit jen předem definované parametry.

3.3.1 Node editor

Pro uživatelsky přívětivé generování výškové mapy byl navrhnut node editor. Návrh node editoru je inspirován programem Blender[3], příklad lze vidět na obrázku 3.8.

[3]Blender je open source software pro modelování a vykreslování trojrozměrné počítačové grafiky, více naleznete na https://www.blender.org/.



Obrázek 3.8: Snímek existujícího node editoru z programu Blender. Tento node editor slouží jako inspirace.

Navržený node editor rozděluje uzly do 3 kategorií na uzly zobrazovací, zdrojové a transformační. Dále umožňuje uživateli definici vlastních uzlů.

Zobrazovací uzel

Zobrazovací uzly lze napojit kamkoliv a zobrazí uživateli momentální stav, lze je tedy použít k zobrazení mezikroků. Tento uzel předgeneruje zmenšenou výškovou mapu a nastíní uživateli momentální či výsledný stav operací pomocí vykreslené textury.

Zdrojové uzly

Zdrojové uzly slouží jako vstupní uzly. Nemohou být na nic napojeny. Generují výstupní výškovou mapu, která poté může být různě upravená transformačními uzly. Zdrojové uzly jsou typicky šumové a fraktálové algoritmy.

Transformační uzly

Uzly transformační jsou používány ke spojování či upravování uzlů zdrojových. Jsou implementovány jak jednoduché operace jako je násobení, dělení, sčítání a odčítání, tak i složitější operace jako Gaussovské rozostření či transformační matice.

Vlastní definice uzlů

Implementace zohledňuje i uživatelský tvůrčí duch a je možné zvolit jako vstupní či transformační uzel i uživatelem implementovaný kód. Toho se jednoduše docílí tím, že uživatel vytvoří vlastní třídu a implementuje buď rozhraní ICustomSourceNode pro zdrojový uzel nebo rozhraní ICustomTransformNode pro transformační uzel a poté jej jen do grafu přidá.

Kapitola 4

Implementace řešení

Jedním z hlavních úskalí byla plynulá návaznost jednotlivých krajinových podobjektů, které společně tvoří jeden velký celek. Vygenerovaná výšková mapa se musí při generování objektů rozdělit, ale při simulaci eroze či generování vegetace opět použít jako celek a poté aplikovat změněnou výškovou mapu na jednotlivé podobjekty. Tento proces také musí být proveden v akceptovatelném čase. Kvůli tomu je naposled vygenerovaná výšková mapa ukládaná jako celek a v podkrocích je vždy určeno, které krajinové podobjekty mají být změněny.

4.1 Unity

Unity je multiplatformní herní engine s podporou 2D, 3D i virtuální reality. Má vestavěný fyzikální systém PhysX od společnosti NVIDIA[1]. Také umožňuje vývoj směrovat dle výkonnosti platformy. Unity lze rozšířit o různé balíčky pomocí Unity Asset Store. Tento obchod poskytuje jak grafické tak podpůrné aplikace pro vývoj aplikací v Unity. Každý uživatel má možnost publikovat vlastní rozšíření Unity na tomto virtuálním obchodním prostředí.

4.1.1 Unity GameObject

Unity objekty se skládají z komponent, ať už to je *mesh*, aby byl objekt vidět, anebo více podrobný *collider*, aby objekt mohl kolidovat s jiným. Programátor má také možnost komponenty libovolně ovládat, přidávat a odebírat přímo ze skriptu. Toto umožňuje bázová třída $MonoBehavior$. Tato třída otevře její zděděné třídě přístup k mnoha funkcím, kterými lze přímo ovlivnit objekt ve scéně. Například lze využít funkce $Start()$, aby objekt právě na začátku hry provedl uživatelem definovanou akci, nebo funkce $Update()$, která je volána každý jeden snímek. Takto je možné objekt například plynule posouvat v prostoru.

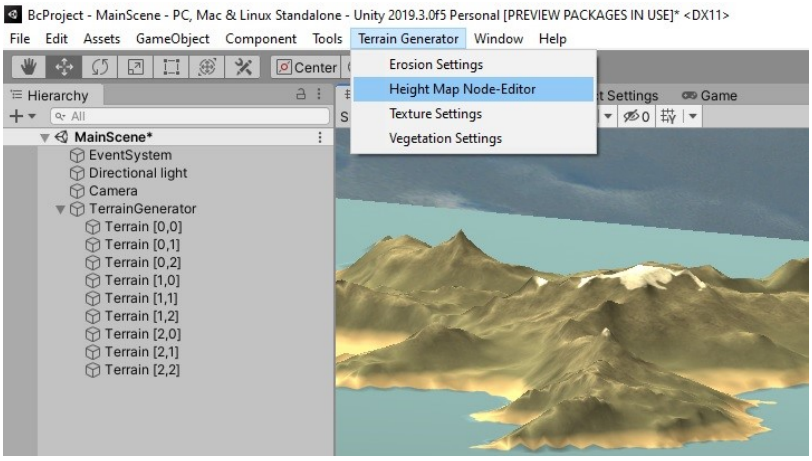
Tento projekt generuje objekty po spuštění generování. Dané objekty jako je krajina či vegetace se vždy vytváří pod stejnými jmény, tyto jména jsou poté využívána k vyhledávání a identifikaci objektů. Tento přístup je zvolen proto, že po úpravách skriptů Unity automaticky provede rekompilaci, čímž se ztratí reference. Přístup vyhledávání objektů podle jmen je přívětivější. Po generování krajiny a vegetace v hierarchii Game Objektů automaticky přibudou nové objekty, které jako jejich děti dále nesou vygenerované objekty. Objekty jsou pojmenovány tak, aby reprezentovaly danou vygenerovanou část krajiny.

[1] Nvidia Corporation je hlavním výrobcem grafických procesních jednotek.

4.1.2 GUI skriptování

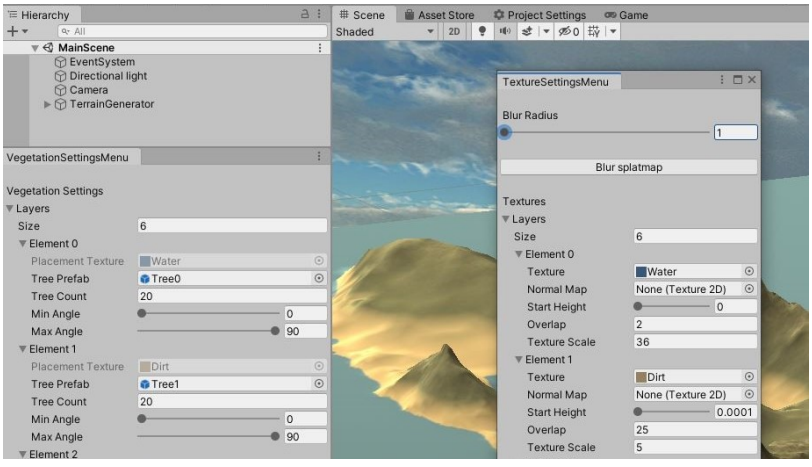
Unity poskytuje vlastní komponenty pro skriptování uživatelského rozhraní. Toto umožňuje vyvíjení nativních pluginů[2]. Umožní vytváření a přidávání nastavení do jakékoliv kategorie v již existujícím standardním uživatelském rozhraní. Podmínkou pro využití rozšíření již existujícího uživatelského rozhraní je umístění všech skriptů používajících Unity GUI skriptování do složky */Editor*. Tato složka oznamuje překladači, že se jedná o GUI komponenty a že se mají přeložit přednostně. Celý Unity GUI skriptovací systém má velkou nevýhodu, a to tu, že se řadí mezi legacy komponenty, což znamená, že jej Unity již aktivně nevyvíjí.

Při vývoji grafického rozhraní byl plugin integrován přímo do Unity rozhraní. To znamená, že po importování tohoto pluginu do Unity projektu se v horní liště v Unity editoru objeví nová položka 4.1. Pomocí této položky lze poté ovládat vytvořený plugin.



Obrázek 4.1: Vzniklý plugin je součástí horní lišty v Unity.

Další výhodou nativního skriptování je, že vytvořené rozhraní je vytvořeno ze stejných grafických elementů, jako je celý editor. Grafické rozhraní tedy zapadá do editoru a je možné jej přetahovat a připínat na stejná místa jako standardní okna Unity editoru 4.2.



Obrázek 4.2: Připnuté a nepřipnuté uživatelské rozhraní.

[2]Plugin je software, který nepracuje samostatně, ale rozšiřuje funkčnost již existující aplikace.

4.1.3 Zobrazení objektů

K zobrazení objektů jsou použity dvě různé metody. Při zobrazení vodních ploch je využit *mesh* objekt a pro zobrazení krajiny je využit *UnityTerrainobjekt*.

Vytvoření mesh objektu

Standardní způsob zobrazení terénu je pomocí triangulace. Tento způsob je založen na převedení výškové mapy na mesh objekt. Při jeho aplikaci se z jednotlivých bodů výškové mapy vytvářejí vertexy. Z těchto vertexů se poté tvoří pravotočivé trojúhelníky, pro které lze jednoduše spočítat normálový vektor. Tento postup lze urychlit generováním dvojic trojúhelníků najednou, tedy vytváříme vždy aktuální čtverec sestaven ze dvou trojúhelníků v jednom kroku algoritmu, implementovaný algoritmus 4.1.

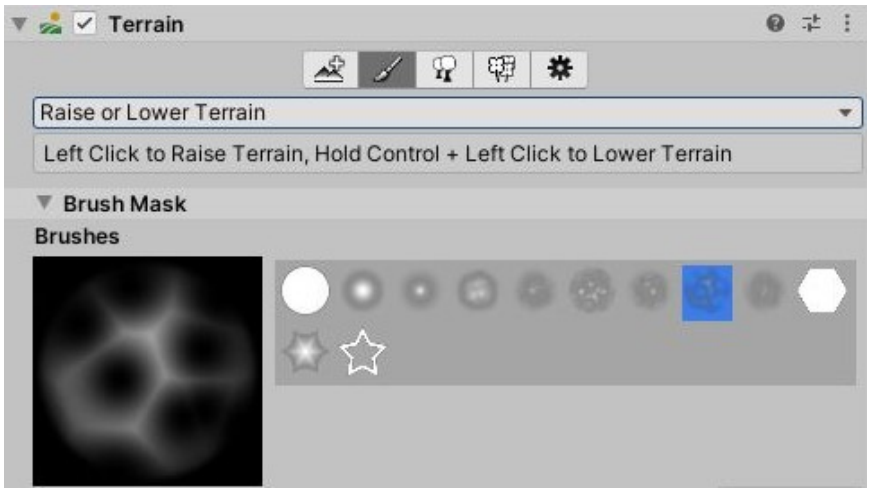
```
public static Mesh SetupMesh (float[,] heightMap) {
        Mesh mesh = new Mesh();
        xSize = heightMap.GetDimension(0);
        ySize = heightMap.GetDimension(1);

        vertices = new Vector3[xSize * ySize];
        int[] triangles = new int[xSize * ySize * 6];

        int idx = 0;
        for (x = 0; x < xSize; ++x) {
                for (int y = 0; y < ySize; ++y) {
                        vertices[idx++] = new Vector3(x, y, heightMap[x, y]);
                }
        }

        int triangleIndex = 0, vertexIndex = 0;
        for (x = 0; x < xSize; ++x) {
                for (int y = 0; y < ySize; ++y) {
                        triangles[triangleIndex] = vertexIndex;

                        triangles[triangleIndex + 3] =
                            triangles[triangleIndex + 2] = vertexIndex + 1;

                        triangles[triangleIndex + 4] =
                            triangles[triangleIndex + 1] = vertexIndex + xSize + 1;

                        triangles[triangleIndex + 5] = vertexIndex + xSize + 2;

                        triangleIndex += 6;
                        vertexIndex++;
                }
                vertexIndex++;
        }
        mesh.vertices = vertices;
        mesh.triangles = triangles;
}
```

Výpis 4.1: Implementace generátoru mesh objektu pomocí triangulace z výškové mapy v jazyce C♯. Vytváří vždy dva trojúhelníky v jednom kroku.

S tímto jsou spojeny značné výhody v rychlosti zobrazování, ale také značné nevýhody, a to zejména v rychlosti generování. Po vygenerování mesh objektu je nemožné objekt dále upravovat a musí se popřípadě celý vygenerovat znovu. Tento algoritmus je použit při generování mesh objektu vodní hladiny.

Vytvoření Unity Terrain objektu

Pro zobrazení terénu v Unity je možné využít také Unity Terrain objektu, více viz [15]. Tento objekt je vysoce optimalizovaný a poskytuje uživateli možnost interakce a libovolných úprav po vygenerování. K úpravám výškové mapy, textur a jiných vlastností již vygenerované krajiny lze využít integrovaných nástrojů.



Obrázek 4.3: Unity komponenta, pomocí které lze upravovat Unity Terrain objekt.

Při simulaci vodní eroze se pomocí uložených referencí na Unity Terrain objekty z daných objektů vydoluje výšková mapa a provede výše zmíněný algoritmus, který simuluje erozi. Tento postup byl zvolen, protože je ponechána možnost úpravy krajiny uživatelem. Tento postup je sice výpočetně pomalejší, ale kvůli volnosti manuální úpravy krajiny před simulací vodní eroze je takové zpomalení zcela akceptovatelné.

Spojování vytvořených terénů

V hrách s velkým otevřeným světem je nemožné vykreslovat vždy celý herní svět. Načtené objekty v operační paměti by zabíraly příliš mnoho místa a vykreslovací algoritmus by většinu objektů stejně nevykreslil.

Z toho důvodu bývají herní světy rozděleny do sekcí, které se nahrávají a vymazávají z operační paměti za běhu podle aktuální pozice hráče. Při procedurálním generování rozsáhlejších map se tato vlastnost musí zohlednit a vygenerovaná krajina se musí rozdělit do jednotlivých sekcí, aby byl výsledný produkt reálně použitelný.

Ke spojení vytvořených Unity Terrain objektů se využívá uložených referencí ve 2D poli. Takto se dá určit, který terén hraničí na jaké světové straně s jiným. Toto propojení ulehčí pozdější možnost manuálních změn, jelikož po spojení objektů lze aplikovat změny na vygenerovanou krajinou jako celek.

4.1.4 Vegetace

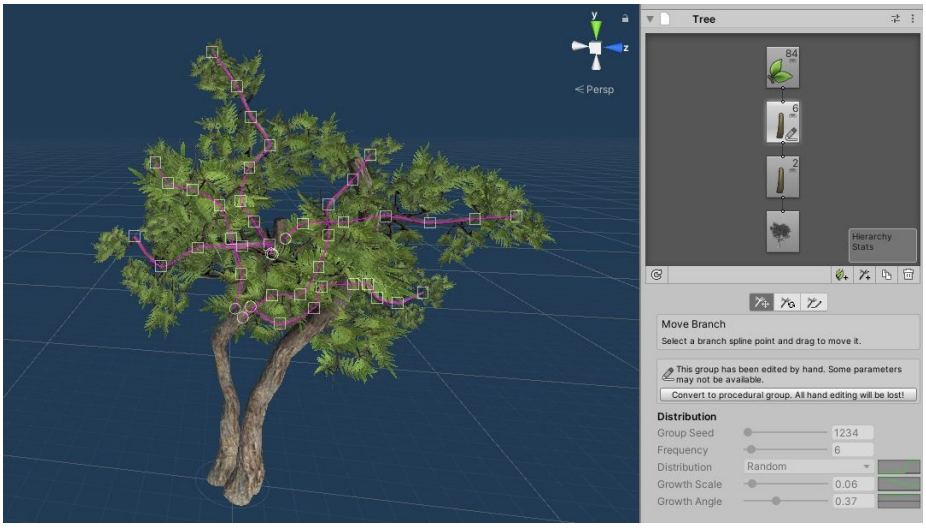
Unity má již předpřipravené nástroje pro tvoření vegetace. V těchto nástrojích lze pomocí L-systémů[3] definovat různé parametry pro daný typ vegetace. Vegetace vytvořená pomocí nativních nástrojů obnáší velké výhody, a to zejména v optimalizaci. Toto umožní lehkou aplikaci level of detail (také LOD) systému[4].

Součástí pluginu jsou také demonstrační modely. Tyto modely jsou rozmístěny podle nastavení uživatele, tedy pro každou texturu, která je aplikovaná, si může uživatel zvolit jaké objekty si na ní přeje pokládat. K rozmístění objektů je poté použit Perlinův šum, který umožní i vytváření menších lesů či shluků keřů.

Stromy

Při vytváření stromových objektů lze v Unity využít předem vytvořených L-systémů k definici počtu větví, listů, rozvětvení a také lze definovat různé materiály pro dané části. Tyto stromy poté budou mít podobný, ale přesto rozdílný vzhled, což napomáhá výslednému pocitu reálnosti celkové krajiny. Při vykreslování stromů za použití LOD systému se v dáli mohou 3D stromové objekty proměnit na otexturované 2D plochy.

Součástí projektu je pět demonstračních stromů. Tyto stromy jsou vytvořeny tak, aby pro každý biom byl alespoň jeden strom k dispozici. Stromy jsou vytvořeny pomocí Unity L-systémů.



Obrázek 4.4: Strom vytvořený pomocí Unity Tree editoru.

Travnaté plochy a jiné detaily

Krajina může také obsahovat 3D travnaté plochy, ne jen otexturované plochy a malé 3D detaily, jako jsou například kameny a keře. Tyto si uživatel opět může vybrat podle vybrané textury. Součástí projektu jsou i demonstrační keře a kameny. Vygenerovanou krajinu se stromy i detaily lze vidět na obrázku 4.5.

[3] L-systém je formální gramatika vyvinutá pro modelování růstů rostlin.

[4] Level of detail systém vygeneruje jeden objekt s více úrovněmi detailu. Tyto úrovně jsou vypínány a zapínány podle vzdálenosti od bodu aktivní kamery. Tedy je-li aktivní kamera blízko danému objektu, je objekt vykreslen s vysokou úrovní detailu a naopak.

Obrázek 4.5: Snímek krajiny s vygenerovanou vegetací a detaily.

4.2 Texturování krajiny

Jako způsob mapování textur byl zvolen splat mapping, protože se interně používá při manuální aplikaci textur na Unity Terrain objekt. Takto je zajištěna kompatibilita a možnost pozdější úpravy textur uživatelem.

K aplikaci textur běžně dochází podle výškové mapy. Specifikují se počáteční a maximální výšky jednotlivých textur a bod po bodu se textury aplikují. Způsob aplikace můžeme rozšířit o rozmazání přechodu mezi texturami. Tento přechod buď může být hladký, jak lze vidět výše 2.7. Také se může dále plynulý přechod rozšířit pomocí šumů. Můžeme například použít Perlinův šum k rozmazání hranic přechodu mezi dvěma texturami. To je vhodné použít obzvláště při přechodu mezi sněhovými plochami na vrcholcích hor. Sněžná čára[5] v přírodě nikdy není v jedné a té samé nadmořské výšce. Vždy se pohybuje v nějakém rozmezí a je ovlivněna mnoha faktory. Po rozšíření pevné hranice sněhu o použití Perlinova šumu lze vidět značný rozdíl v reálnosti vygenerované krajiny, znázorněno na obrázku 4.6.

(a) Pevně stanovená sněžná čára.

(b) Pohyblivá sněžná čára za pomocí Perlinova šumu.

(c) Porovnání s reálnou sněžnou čarou. Převzato z [5].

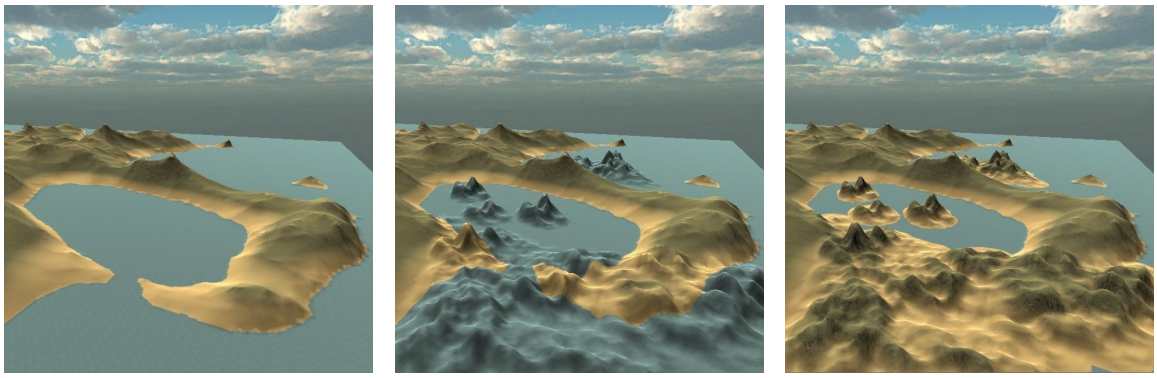
Obrázek 4.6: Použití Perlinova šumu k rozmazání pevného přechodu mezi texturami.

[5]Sněžná čára (anglicky „Snow Line“) je oblast pomezí sněhem zakryté krajiny se sněhem nezakrytou krajinou. Tato mez je proměnlivá, například podle ročního období.

Další možností rozšíření způsobu aplikace textur je podle náklonu daného bodu na krajině. Toto rozšíření umožní proložení skalnatých prvků mezi níže položené strmé úseky a naopak proložení travnatých ploch mezi výše položené ploché úseky.

Z výše zmíněných faktorů lze sestavit rovnice, pomocí kterých se určí zastoupení dané textury v daném bodě. Zastoupení každé aplikované textury je vypočítáno jednou rovnicí a celé zastoupení nakonec normalizované, aby součet aplikovaných textur v každém bodě byl roven jedné.

Zohledněna je také možnost opětovného aplikování textur na vygenerovaný terén, což je obzvláště vhodné po manuálních změnách výškové mapy uživatelem, znázorněno na obrázku 4.7.



(a) Krajina před změnami.

(b) Krajina po manuální změně výškové mapy.

(c) Krajina po opětné aplikaci textur.

Obrázek 4.7: Opětné aplikování textur po manuální úpravě krajiny.

4.3 Implementace node editoru pomocí návrhového vzoru composite

Aby mohl být využit návrhový vzor composite byla implementována bázová abstraktní třída *Node*, která obsahuje abstraktní metody Calculate a Save, implementace znázorněna v úryvku 4.2.

```
public abstract class Node {
    protected List<Node> inNodes; //possibility of multiple input nodes
    public abstract float[,] Calculate(Mode mode); //must be implemented
    //generic method used by all nodes
    protected static int GetGenerationSize(Mode mode){ ... }
}
//example of a class implementing the abstract Node class
public class MultiplicationNode : Node {
    public override float[,] Calculate(Mode mode){
        float[,] subGraphA = inNodes[0].Calculate(mode);
        float[,] subGraphB = inNodes[1].Calculate(mode);
        return Multiply2dArrays(subGraphA, subGraphB);
    }
}
```

Výpis 4.2: Implementace bázové třídy node a třídy implementující třídu node v jazyce C♯.

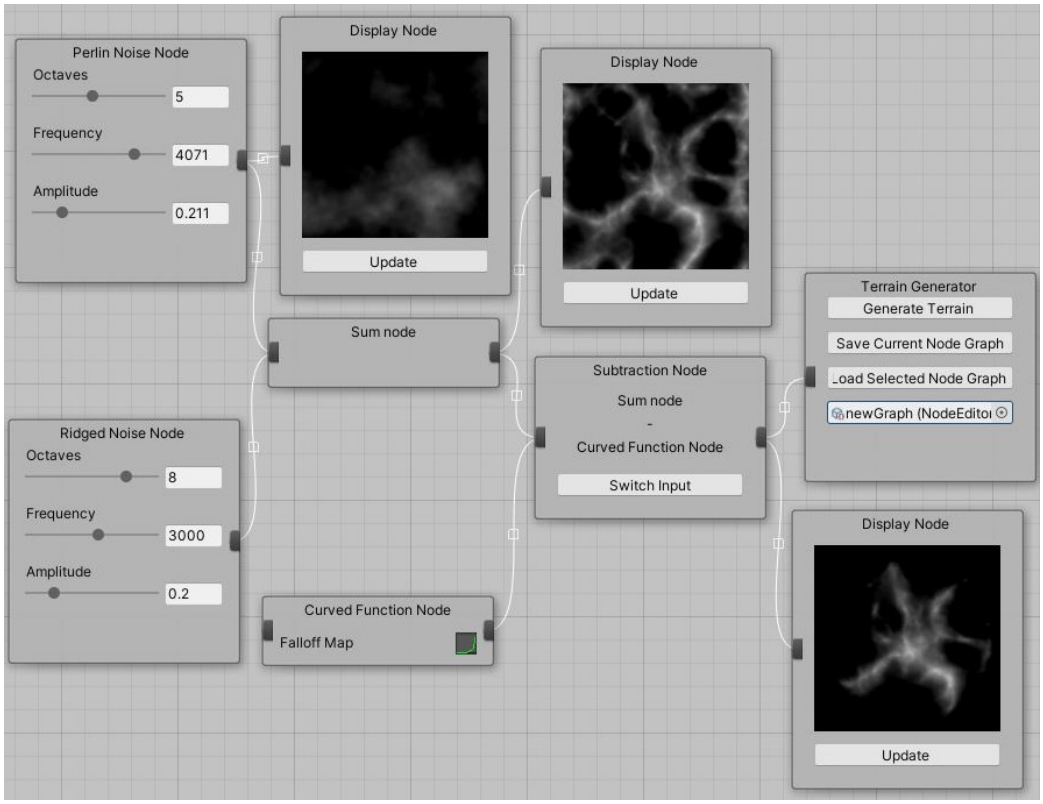
Metoda Calculate přijímá parametr, který určuje, zda má generovat zmenšené výsledné pole, které je využité k zobrazení mezikroků, anebo reálné, které je využité ke generování krajiny. Zmenšené výsledné pole je generováno tak, že se přeskakuje každá n-tá souřadnice.

Dále jsou uzly rozděleny na zdrojové a transformační. Zdrojové uzly nemají vstupní uzly a generují podle jejich nastavení jen výsledné pole. Transformační uzly přijímají dle druhu jeden či více nodů na vstup a transformují je, což je znázorněno na obrázku 4.8.

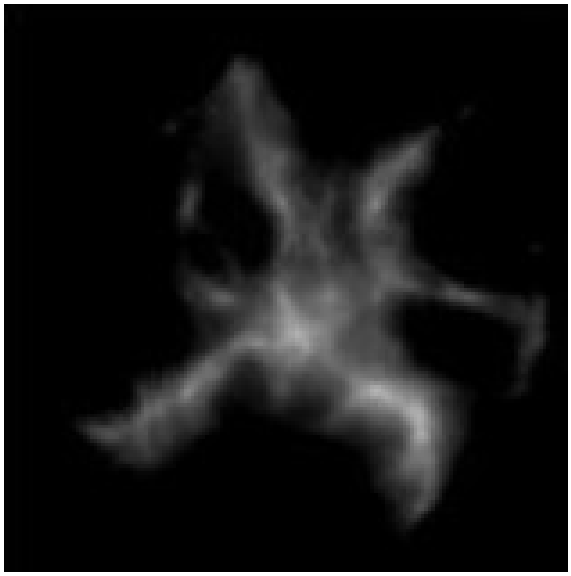
K implementaci je možné přistupovat dvěma různými způsoby, a to použitím rozhraní nebo bázové třídy. Při implementaci byl zvolen přístup pomocí abstraktní bázové třídy nad rozhraním, protože se vyplatila přímá implementace pomocných metod v bázové třídě.

Specifickou třídou jsou uzly, které volají funkci v jejich vstupním uzlu. Do této kategorie se řadí zobrazovací uzel („Display Node“) a uzel, který generuje krajinu („Terrain Generator“). Zobrazovací uzel volá funkci Calculate s parametrem, který udává, že jde o generování výškové mapy ve zmenšené podobě. Tato zmenšená podoba je poté převedena na texturu a vykreslena přímo v editoru. Vykreslená textura přímo odpovídá vygenerované krajině, lze vidět na obrázku 4.9.

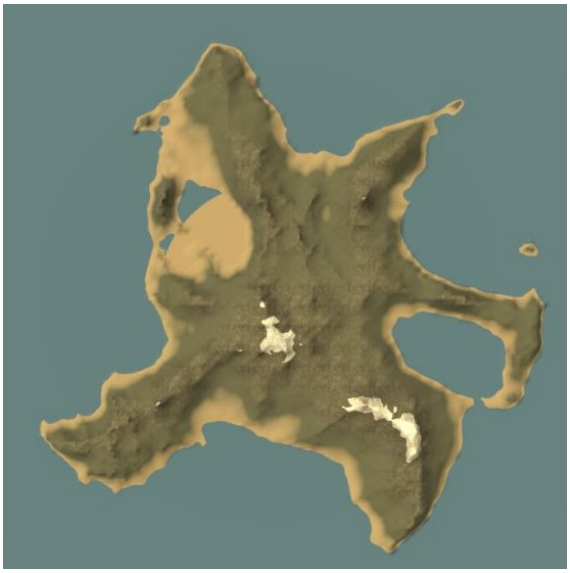
Při vytváření zmenšené podoby výškové mapy nejde jen zmenšit velikost vygenerovaného pole. Zdrojové uzly při generování pole berou v potaz jen každý n-tý prvek a vygenerují hodnotu pro něj. Tak lze docílit stejného vzhledu výškové mapy ve zmenšené podobě.



Obrázek 4.8: Snímek vytvořeného node editoru. Lze vidět uživatelem sestavenou složenou funkci. Tato funkce obsahuje tři zdrojové uzly, „Perlin Noise Node“, „Ridged Noise Node“, „Curved Function Node“, a dva transformační uzly, „Sum Node“ a „Subtraction Node“. Dále jsou napojeny různé zobrazovací uzly. Výsledná složená funkce je napojena do uzlu, který již generuje výslednou krajinu.



(a) Výsledný zobrazovací uzel.



(b) Pohled na výslednou vygenerovanou krajinu shora.

Obrázek 4.9: Porovnání náhledu výškové mapy s vygenerovanou krajinou.

4.4 Ukládání uživatelských nastavení

Součástí použitelné aplikace je perzistence uživatelských nastavení či dat. Tato jsou serializována pomocí Unity třídy Scriptable Object vždy po změně uživatelem. Při prvním spuštění plug-inu jsou základní uživatelská nastavení předgenerována, na tyto tovární nastavení se lze kdykoliv vrátit.

U výše zmíněného node editoru lze také uložit více uživatelských grafů a ty poté dynamicky načítat. To umožní uložení více různých krajin s malou paměťovou náročností. Tyto grafy lze případně vygenerovat i za běhu (například ve hře při vstoupení do portálu lze vygenerovat nová mapa za běhu).

4.5 Kompatibilita s Unity

Kvůli využití výše zmíněného Unity Terrain objektu lze vygenerovanou mapu libovolně upravovat nativními Unity nástroji.

Po manuální úpravě terénu již nanesené textury nebudou odpovídat terénu, proto je přidána možnost opětovného vygenerování textur pro manuálně upravený terén.

4.6 Použité technologie

Aplikace je naprogramovaná v jazyce C# a HLSL v herním engine Unity (Unity 2019.3.0f5). Ke generování Perlinova šumu a Ridged šumu je použita externí knihovna.

Kapitola 5

Závěr

Jako součást této závěrečné práce byl implementován procedurální generátor 3D krajiny v programu Unity. Tato aplikace dává uživateli volnou ruku při generování krajiny. Má možnost buď využít předem definované dílčí funkce k sestavení vlastní složené funkce, nebo dokonce může definovat a vložit vlastní algoritmus jako součást složené funkce.

Zadání se splnit podařilo. Výsledná krajina se drží reálných aspektů krajiny specifikovaných v kapitole 3.2. Ta po vygenerování může mít proměnlivou velikost, může být použita různá výšková mapa, kterou si uživatel může graficky sestavit pomocí již předem připravených funkcí, nebo definovat vlastní funkce.

Výslednou aplikaci lze využít k vygenerování krajiny, kterou může uživatel následně dále upravovat a doplnit vlastní umělecké nápady či drobné detaily. Tato vlastnost dělá z vytvořeného generátoru robustní a použitelnou aplikaci i u projektů, ve kterých je kladen důraz právě na malé detaily.

Ve srovnání s jinými průmyslově používanými aplikacemi má tato aplikace výhodu v jednoduchosti používání. Nebylo možné docílit stejné úrovně kvality průmyslově vytvořeného produktu, ale to zčásti kompenzuje možnost následné manuální úpravy.

Možná vylepšení

Momentální vyhodnocování složené funkce pro generování výškové mapy a simulace vodní eroze probíhá na CPU, pro urychlení by bylo vhodné přepsat algoritmus do formy compute shaderů[1]. Toto urychlení by umožnilo mnohonásobné zvýšení detailů a větší složitost algoritmů.

Dalším možným vylepšením by bylo přidání možnosti generování řek, toto by s sebou neslo i zcela novou implementaci vody. Dále by se k procedurálnímu generování řek dalo přidat i rozdělení krajiny do biomů podle vlhkosti dané oblasti.

[1] Tento typ shaderu je používán pro akceleraci výpočtů na GPU.

Literatura

[1] Allen, J. *New Land on the Louisiana Coast* [online]. NASA Earth Observatory, 25. října 2014 [cit. 2020-30-04]. Dostupné z: https://earthobservatory.nasa.gov/images/85261/new-land-on-the-louisiana-coast.

[2] Bedřich Beneš, R. F. *Visual Simulation of Hydraulic Erosion* [online]. Campus Ciudad de Mexico, 2002 [cit. 2020-30-04]. Dostupné z: http://wscg.zcu.cz/WSCG2002/Papers_2002/F23.pdf.

[3] Cowing, K. *Three Views of Mt. Everest and Himalayas From Orbit* [online]. OnOrbit, 30. prosince 2014 [cit. 2020-30-04]. Dostupné z: http://spaceref.com/onorbit/three-views-of-mt-everest-and-himalayas-from-orbit.html.

[4] Deng, L.-Y. a Bowman, D. Developments in pseudo-random number generators: Pseudo-random number generators. *Wiley Interdisciplinary Reviews: Computational Statistics*. Srpen 2017, sv. 9, s. e1404. DOI: 10.1002/wics.1404.

[5] Gupta, P. *Life under an active volcano in Ecuador* [online]. aljazeera, 2016 [cit. 2020-10-05]. Dostupné z: https://www.aljazeera.com/indepth/features/2015/11/life-active-volcano-ecuador-151118101849350.html.

[6] Hope, D. *Madeirawonders* [online]. Madeira Wonders, 2013 [cit. 2020-17-05]. Dostupné z: Howtogetamaderiamap.

[7] Lager, R. *Ratings Roundup: Wimbledon Gentlemen's Final Hits 3.33 Million Viewers; 2019 MLB All-Star Game Numbers Drop* [online]. Sports Video Group, červenec 2019 [cit. 2020-29-04]. Dostupné z: https://www.sportsvideo.org/2019/07/17/ratings-roundup-wimbledon-gentlemens-final-hits-3-33-million-viewers-2019-mlb-all-star-game-numbers-drop/.

[8] Olsen, J. *Realtime Procedural Terrain Generation* [online]. Citace.com, 31. října 2004 [cit. 2020-30-04]. Dostupné z: http://web.mit.edu/cesium/Public/terrain.pdf.

[9] Onrust, B., Bidarra, R., Rooseboom, R. a Koppel, J. van de. Ecologically Sound Procedural Generation of Natural Environments. *International Journal of Computer Games Technology*. Hindawi. Leden 2017. DOI: 10.1155/2017/7057141. Dostupné z: https://doi.org/10.1155/2017/7057141.

[10] Owens, B. *Use Tri-Planar Texture Mapping for Better Terrain* [online]. tutsplus, 2014 [cit. 2020-30-04]. Dostupné z: https://gamedevelopment.tutsplus.com/articles/use-tri-planar-texture-mapping-for-better-terrain--gamedev-13821.

[11] Papaioannou, G. *ComputerGraphics-Texturing* [online]. eclass.aueb.gr, 2015 [cit. 2020-30-04]. Dostupné z: https://eclass.aueb.gr/modules/document/file.php/INF142/%CE%A0%CE%B1%CF%81%CE%B1%CE%B4%CF%8C%CF%83%CE%B5%CE%B9%CF%82/ComputerGraphics-Texturing.pdf.

[12] Perlin, K. An Image Synthesizer. *SIGGRAPH Comput. Graph.* New York, NY, USA: Association for Computing Machinery. Červenec 1985, sv. 19, č. 3. DOI: 10.1145/325165.325247. ISSN 0097-8930. Dostupné z: https://doi.org/10.1145/325165.325247.

[13] Riddle, L. *Koch Snowflake* [online]. Agnes Scott College, 10. ledna 2020 [cit. 2020-29-04]. Dostupné z: http://larryriddle.agnesscott.org/ifs/ksnow/ksnow.htm.

[14] Spankler, T. *Fortnite World Cup Finals 2019 Draws Over 2 Million Live Viewers* [online]. Variety, červenec 2019 [cit. 2020-29-04]. Dostupné z: https://variety.com/2019/digital/news/fortnite-world-cup-finals-2019-live-viewers-championship-1203282771/.

[15] Technologies, U. *Unity User Manual* [online]. Unity Technologies, 2020 [cit. 2020-30-04]. Dostupné z: https://docs.unity3d.com/Manual/index.html.