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

BLENDER ADD-ON PRO SIMULACI VODNÍ EROZE

BLENDER ADD-ON FOR SIMULATION OF WATER EROSION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ VLČEK**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

139866

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: **Vlček Ondřej**
Program: Informační technologie
Specializace: Informační technologie

Název: **Blender add-on pro simulaci vodní eroze**
Kategorie: Počítačová grafika
Akademický rok: 2022/23

Zadání:

1. Naučte se základy práce s 3D modelovacím programem Blender (verze 3.2 a výše).
2. Seznamte se se skriptovacím Blender Python API.
3. Nastudujte metody pro řešení simulace vodní eroze na 3D objektech.
4. Nastudujte možná a existující řešení dané problematiky.
5. Navrhněte uživatelské rozhraní pro výsledný add-on.
6. Add-on implementujte a demonstrujte jeho použití na různých typech dat.
7. Zdokumentujte a zveřejněte výsledný add-on.
8. Vytvořte video reprezentující výsledky vaší práce.

Literatura:

- Blender API documentation
- Beneš, Bedřich, et al. "Hydraulic erosion." *Computer Animation and Virtual Worlds* 17.2 (2006): 99-108.
- Mei, Xing, Philippe Decaudin, and Bao-Gang Hu. "Fast hydraulic erosion simulation and visualization on GPU." *15th Pacific Conference on Computer Graphics and Applications (PG'07)*. IEEE, 2007.
- Anh, Nguyen Hoang, Alexei Sourin, and Parimal Aswani. "Physically based hydraulic erosion simulation on graphics processing unit." *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. 2007.
- Tychonievich, Luther A., and M. D. Jones. "Delaunay deformable mesh for the weathering and erosion of 3d terrain." *The Visual Computer* 26.12 (2010): 1485-1495.
- Kider, Joseph T., Jianbo Shi, and Stephen H. Lane. "Simulation of 3D Model, Shape, and Appearance Aging by Physical, Chemical, Biological, Environmental, and Weathering Effects." (2012).

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 5, experimenty vedoucí k bodu 6.

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

Vedoucí práce: **Chlubna Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 31.10.2022

Abstrakt

Cílem práce je rozšíření programu Blender o erozi terénu. Představený zásuvný modul implementuje vodní erozi založenou na texturách s využitím grafické karty. Oproti současným řešením práce využívá paralelního zpracování, prezentuje alternativní pracovní postup pro úpravu terénu a vytváří dodatečné informace využitelné při vykreslování modelu.

Abstract

The aim of this thesis is adding terrain erosion functionality to Blender. The solution presented here implements hydraulic erosion based on textures and the use of the graphics card. Compared to other existing solutions this add-on utilizes parallel processing, offers an alternate workflow for erosion and creates additional information usable in further rendering of the model.

Klíčová slova

Blender, addon, vodní eroze, eroze terénu

Keywords

Blender, addon, hydraulic erosion, terrain erosion

Citace

VLČEK, Ondřej. *Blender add-on pro simulaci vodní eroze*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Chlubna

Blender add-on pro simulaci vodní eroze

Prohlášení

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

.....

Ondřej Vlček

8. května 2023

Obsah

Kapitola 1

Úvod

S vyššími nároky na rozsah a realismus digitálních světů nastává potřeba modelovat tím detailnější a složitější krajiny. Pro vytvoření hrubého základu terénu již existují metody jak procedurální, tak ruční, a to v libovolném modelovacím programu. Dosažení uvěřitelných rysů ale vyžaduje exponenciálně více práce, a to například simulací přírodních jevů, jež je tvoří.

Cílem projektu je obohatit program Blender skrze zásuvný modul o simulaci vodní eroze. Řešení je cíleno na alternativní postup práce oproti současným rozšířením, tedy čistě pomocí textur, a na urychlení algoritmů s využitím grafické karty. Představený algoritmus také vytváří přídavná data dále využitelná ve stylizaci a je oproti již existujícím rozšířením schopen transportu barev.

Inspirací pro projekt byl právě pokus vytvořit ručně rozsáhlé pohoří pro využití ve hře. Postup a dosažitelný výsledek zde ale ovlivňovala limitovaná výpočetní síla. Daná zkušenost vedla k dále prezentovanému přístupu, jež byl zaměřen na odlehčení práce s modelem po erozi a snížení výpočetních nároků.

Následující kapitola 2 je zaměřena na seznámení se současnými algoritmy vodní eroze a existujícími řešeními, jež je možné ve spojení s aplikací Blender použít. Také se zaobírá důležitými technologiemi využitými v projektu. V kapitole 3 je pak prezentován návrh uživatelského rozhraní a poskytnuté funkcionality, včetně přehledu samotné simulace.

Kapitola 4 následně zachází do detailu implementace veškeré funkcionality rozšíření. Měření a zjištěné limitace projektu jsou probrány v kapitole 5. Závěrečná kapitola 6 také přestavuje souhrn plánu budoucí práce.

Kapitola 2

Existující algoritmy a použité technologie

Centrálním prvkem vytvářeného modulu je samotný algoritmus simulace eroze, je tedy nutné první představit současné metody řešení daného problému, jejich varianty a vlastnosti. Dále jsou popsány existující aplikace, či přímo zásuvné moduly do programu Blender, jež je možné použít. Kapitola je zakončena rozborem technologií a nástrojů, jež byly použity v řešení této práce, a které budou dále odkazovány v popisu návrhu a implementace.

2.1 Vodní a termální eroze

Vodní eroze je geologický proces, při kterém dochází k odnosu, transportu a usazování hornin a půdního materiálu vlivem vodních toků. Tento proces vzniká v důsledku působení sil vody, které narušují pevnost a stabilitu svahů, erozují půdní vrstvy a odkládají materiál v podobě sedimentů. Příklad vodní eroze je ilustrován v obrázku 2.1[1].

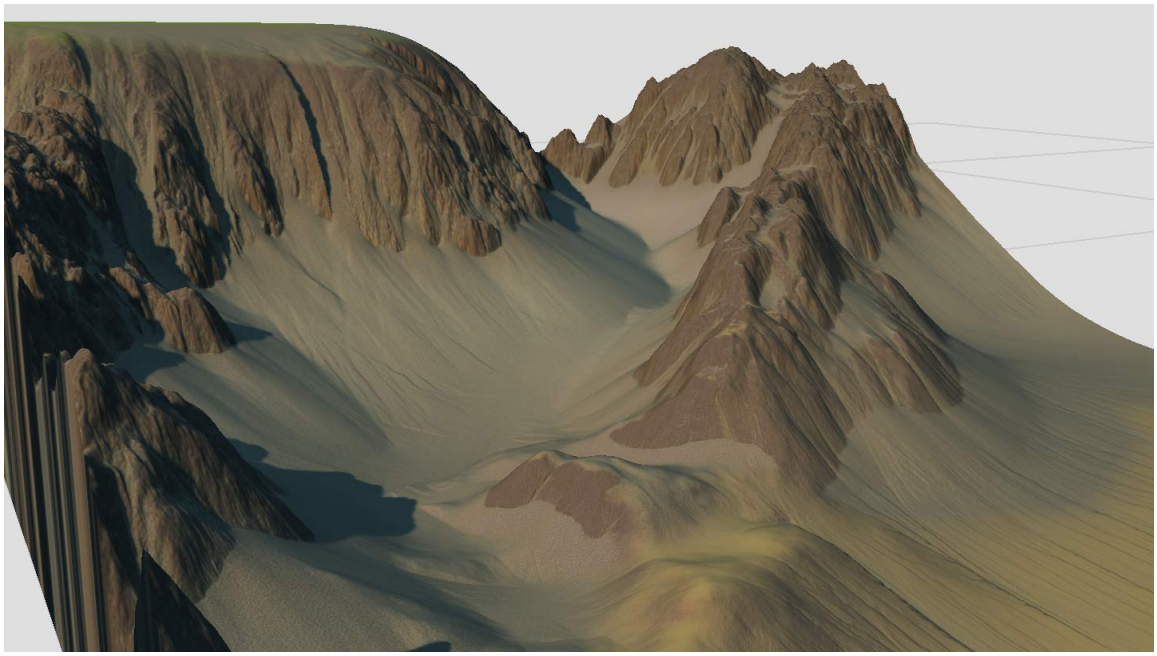
Vodní eroze se dělí na dva základní typy - povrchovou a hlubinnou erozi. Povrchová eroze se vyskytuje na povrchu půdy, kde jsou erozní procesy působeny přímo na vrstvu půdy. Hlubinná eroze vzniká, když voda pronikne do podloží a narušuje ho vlivem tlaku a pohybu vody. Algoritmy, kterými se kapitola dále zabývá, se soustředí především na povrchovou vodní erozi [3].

Termální eroze je další geologický proces, při kterém dochází k odnosu, transportu a usazování hornin a půdního materiálu vlivem tepelných vlivů, zejména cyklické změny teploty. Tento proces vzniká v důsledku rozdílné expanze a kontrakce hornin vlivem změn teploty.

Vrstva uvolněná teplotními cykly není pevná, částice jsou tedy schopny pohybu. Přesun uvolněného materiálu je zde uvažovován pouze díky gravitaci, v přírodě dochází ale také k přesunu pomocí vody či větru. Povrch vytvořený termální erozí se odvíjí od takzvaného sypného úhlu, což je maximální úhel vzhledem k vodorovné ploše, jež je materiál schopen vytvořit. Ilustrace simulace termální eroze je uvedena v obrázku 2.2[2].

[1] Ilustrace byla získána ze stráněk agric.wa.gov.au.
[2] Zdrojem obrázku je blog world-machine.com.

Obrázek 2.1: Ilustrace vodní eroze. Je vidět koryto vytvořené vodním tokem brázdy na jeho stěnách.

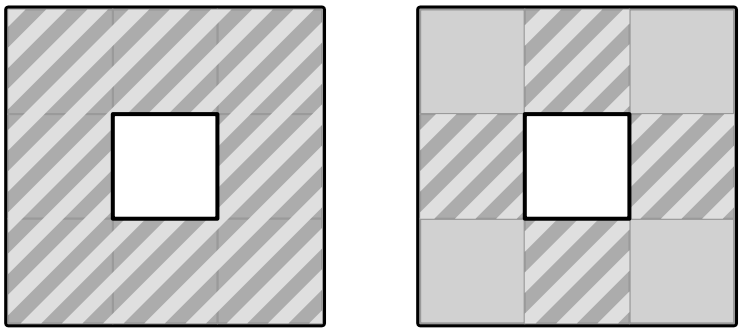


Obrázek 2.2: Ilustrace simulace termální eroze. Svah uvolněného materiálu nabývá maximálně sypného úhlu.

2.2 Celulární automaty

První, ale stále používanou, metodou je simulace vodní eroze pomocí celulárních automatů [8]. Celulárním automatem je zde označován časově i prostorově diskrétní model pracující nad dvoudimenzionálním polem buněk, nejčastěji takzvanou výškovou mapou. Jedná se o pravoúhlou mřížku, kde v každé buňce je uložena výška bodu, kterému buňka odpovídá. Výšek pro každou buňku, nebo dalších hodnot, jako je množství či rychlost vody, může být ale v závislosti na algoritmu definováno více.

Samotná eroze pak probíhá změnou stavu všech buněk v iteracích algoritmu. Pro výpočet změn je využíváno okolí vyhodnocované buňky, a to nejčastěji buď Moorovo okolí, zde dále označovano jako osmiokolí, nebo Von Neumannovo okolí, dále značeno jako čtyřokolí. Jejich rozdíl je ilustrován v obrázku 2.3.



(a) Moorovo okolí (b) Von Neumannovo okolí

Obrázek 2.3: Ilustrace okolí buňky. Šrafovaně jsou značeny buňky patřící do okolí a bíle je zakreslena zkoumaná buňka.

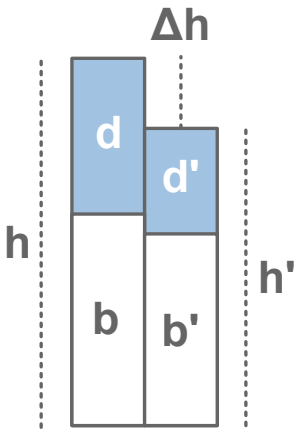
2.3 Simulace pomocí rozpouštění

Jeden z možných algoritmů eroze využívá faktu, že pomalu tekoucí voda se vsakuje do zeminy a vytváří pohyblivou vrstvu materiálu. Daná vrstva se nazývá regolit a chová se jako hustá kapalina. Jejím přesunem je možné dále simulovat ukládání v nižších oblastech terénu [11].

Dále uvedený model je nejčastěji aplikovatelný pro dna jezer či pomalých řek. Svým vyhlazovacím účinkem se značně liší od simulací založených na síle, jež jsou rozebrány v další kapitole 2.4. Zde uvedená metoda byla představena v práci Beneš *et al.* a rozděluje proces na následující kroky [2]:

1. Přísun vody
2. Eroze materiálu a zachycení do vody
3. Transport vody i materiálu
4. Uložení sedimentu na nové pozici

Každá buňka v simulaci má definované tři hodnoty, výšku terénu b, výšku sloupce vody d a množství nesené látky s v daném bodě. Vzdálenost buněk je předpokládána jako jednotková, stejně jako kroky v čase. Jejich interpretace je zobrazena v ilustraci 2.4.



Obrázek 2.4: Ilustrace zobrazuje uspořádání výšek terénu b, vody d a celkové výšky h ve sloupcích.

Prvním krokem simulace je přísun vody v čase t, jež odpovídá zvýšení sloupců vody:

$$d_{t+1} = d_t + K_r \tag{2.1}$$

Zde je K_r konstanta, jež má význam stejně silného deště po celé oblasti simulace. Pro realističtější simulaci deště lze konstanta nahradit funkcí závislou na pozici. Větší množství vody by mělo být generováno ve středu simulace a případné variace lze dosáhnout fraktalními funkcemi [2]. Pozemní zdroje vody jako jsou řeky či jezera lze reprezentovat podobným způsobem.

Limitace množství vody v modelu je dále zajištěna vypařováním, jež v modelu pomocí konstanty vypařování K_e a původního množství vody d_0 vychází z následující rovnice:

$$d_t = d_0 e^{-K_e t} \tag{2.2}$$

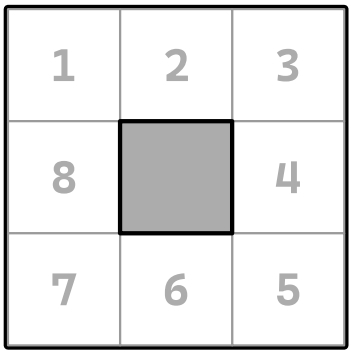
Daný exponenciální model nedovoluje celkovému vyschnutí, tedy dosáhnutí nulové výšky, jež může být žádoucí. Za tímto účelem autor definuje prahovou hodnotu T a upravený výpočet uveden v rovnici 2.3.

$$d_t = \begin{cases} 0, & \text{pokud } d_0 e^{-K_e t} < T \\ d_0 e^{-K_e t} - T, & \text{jinak} \end{cases} \tag{2.3}$$

Pokud výška sloupce nedosahuje prahové hodnoty, pak je nastavena na nulu. Pro změny v jednotlivých iteracích při konstantních krocích lze pak hodnotu e^{-K_e} předpočítat a jednoduše aplikovat násobením. Dalším krokem je depozice a eroze materiálu. Klíčovou hodnotou pro rozhodování mezi danými procesy tvoří úroveň nasycení, jež je definována následovně:

$$S_t = K_s d_t \tag{2.4}$$

kde K_s je koeficient nasycení, který udává množství materiálu, jež může nést jedna jednotka vody. Je-li nesené množství látky s_t vyšší než maximální nasycení S_t, pak dochází k ukládání materiálu. V opačném případě je materiál odebrán z terénu a přidán k rozpuštěnému množství. V obou případech platí:



Obrázek 2.5: Příklad indexace sloupců v osmiokolí

$$b_{t+1} = b_t + (s_t - S_t) \tag{2.5}$$

Rychlost ukládání i eroze materiálu lze upravit pomocí koeficientů. Dané hodnoty mohou být zásvislé na pozici v mapě, čímž lze simulovat různé materiály. Posledním krokem je přesun kapaliny a rozpuštěného materiálu. Za tímto účelem je definována celkovou výšku buňky h jako součet výšek terénu a sloupce vody:

$$h = b + d \tag{2.6}$$

K přesunu materiálu dochází v osmiokolí kolem zkoumané buňky. Pro každou z nich je zjištěn rozdíl výšek $\Delta h = h - h'$, kde h' je výška sousední buňky. K přesunu dochází pouze do buněk, jež nejsou výše, tedy $\Delta h \leq 0$.

Množství přesunuté každé sousední buňce by mělo odpovídat rozdílům výšek vzhledem ke zkoumanému bodu. Výšky sousedů zde budou zapisovány jako h_i, kde $i \in \{0, 1, ...7\}$, na přesném pořadí zde ale nezáleží. V ilustraci 2.5 je uveden příklad pořadí indexace buněk.

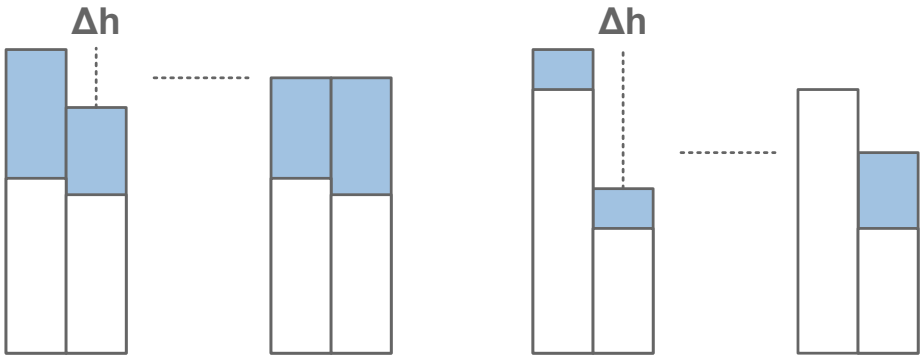
Nejdříve je získán součet výšek sousedů sum, jež jsou níže, nebo na stejné úrovni. Výpočet je uveden v rovnici 2.7.

$$sum = \sum_{i=0}^{7} \begin{cases} \Delta h_i, & \text{pokud } \Delta h_i >= 0 \\ 0, & \text{jinak} \end{cases} \tag{2.7}$$

Pro součet i následující výpočet je také možné použít rozdíly výšek sousedů a vyhodnocované buňky namísto celkových výšek. Řešení by pak silněji preferovalo nejnižší sousedy. Daný součet je dále využit k proporcionálnímu rozdělení vody mezi sousedy v rovnici 2.8.

$$\Delta w_i = \Delta w \frac{\Delta h_i}{sum} \tag{2.8}$$

kde Δw_i je přísun vody pro souseda o indexu i a Δw je množství vody, jež středová buňka může poskytnout. Autor v práci neuvádí přesný výpočet daného množství, lze ale například uvažovat případ pro každé dvě buňky dle ilustrace 2.6.
Dále jsou sousední výšky sloupců označeny pomocí h' a w'. Vyrovnanou výšku hladiny středové buňky $\hat{w}$ pak lze vypočítat jako:



(a) Vyrovnání sloupců vody (b) Přesun celého sloupce vody

Obrázek 2.6: Případy přesunu vody

$$\hat{w} = \frac{h + w + h' + w'}{2} - h \tag{2.9}$$

Je-li menší než nula, pak je veškerá voda přesunuta níže. Jednotlivé přesuny lze sečíst a případně nastavit na celý sloupec vody w_t, pokud jej součet přesahuje. Tímto krokem je završena celá iterace algoritmu a opět může začít další přísunem vody.

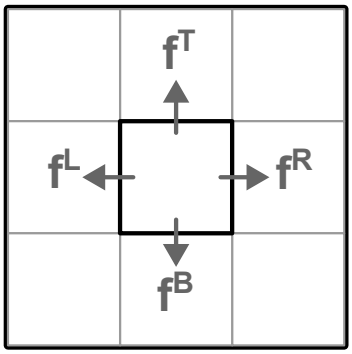
Prezentovaný model představuje řešení o nízkých paměťových nárocích, jež lze použít v kombinaci s dalšími metodami. Odlišným řešením a adaptací na použití s paralelním zpracováním se zabývá následující kapitola.

2.4 Simulace pomocí sil a rour

Předchozí algoritmus se zaměřuje na pomalu tekoucí či stojatou vodu, pro přesnější popis vlivu řek či deště je ale třeba zavést několik rozšíření. Zde uvedený model byl představen v práci Mei *et al.* [7] a jeho hlavním poznatkem je efekt síly, tedy rychlosti a množství, tekoucí vody na terén. Pro účely paralelizace výpočtu a využití grafické karty model také zavádí několikaprůchodový algoritmus a takzvané virtuální roury (anglicky *pipes*) pro výměnu hodnot mezi buňkami.

K předchozím hodnotám buněk, tedy výšce terénu b, sloupci vody d a množství sedimentu s, zavádí model dvourozměrný rychlostní vektor $\vec{v}$ a výtok f (anglicky uváděno jako *outflow flux*), který reprezentuje rychlost vody předávané sousedním buňkám. Model bude pracovat s čtyřokolím, výtok se tedy skládá ze čtyř hodnot f^L, f^R, f^T a f^B, jež odpovídají levému, pravému, hornímu a dolnímu spojení. Roury modelu ilustruje obrázek 2.7.
Postup simulace je dále třeba rozšířit o práci s rychlostním polem. Kroky, jež jsou jednotlivě dokončeny zcela před počátkem dalšího, jsou rozděleny následovně:

1. Přísun vody
2. Aktualizace toků a pole rychlosti
3. Výpočet eroze a uložení sedimentu na základě rychlostí
4. Sediment pozůstalý ve vodě je transportován dle rychlosti
5. Vypařování vody



Obrázek 2.7: Propojení rour prostřední buňky s okolím

2.4.1 Přísun vody

Pro výpočet nového sloupce vody vyžaduje model několik výpočtů. Mezikroky zde budou značeny d_A a d_B. Proměnnými x a y jsou značeny souřadnice zkoumané buňky. Pro první úpravu sloupce přidáním vody pak platí:

$$d_A(x,y) = d_t(x,y) + r_t(x,y) \tag{2.10}$$

kde Δt je časový krok simulace a r_t je funkce určující příchozí množství vody. Podobně jako v předchozím modelu může reprezentovat různé koncentrace deště, pozice vodních ploch nebo i záviset na čase.

2.4.2 Výtoky

Po dokončení přísunu vody je dalším krokem simulace výpočet výtoků. Na rychlost vody má v modelu vliv jak gravitace a výškové rozdíly, tak nastavitelné parametry roury mezi buňkami. Pro spoj je uvažován obsah průřezu roury A a pro reprezentaci velikosti modelu je zavedena délka mezi buňkami l.

Délky mohou být různé pro různé osy, je-li například obdélníkový terén vyhodnocován na čtvercovém poli. Hodnotou průřezu lze limitovat rychlost vody ve spoji, může být tedy konstantní či závislá na čase dle potřeb modelu. V práci Šťáva *et al.* je například využita hodnota l^2. Pro mezivýpočet výtoku levého spojení v dalším kroce $\hat{f}$ pak platí [11]:

$$\hat{f}^L(x,y) = \max\left(0, f_t^L(x,y) + \Delta t \cdot a \ \cdot \frac{g \cdot \Delta h^L(x,y)}{l}\right) \tag{2.11}$$

kde g je tíhové zrychlení a $\Delta h^L(x,y)$ je rozdíl výšek mezi zkoumanou a její levou buňkou, tedy $(x-1,y)$.

Výška buněk je opět počítána jako součet terénu a sloupce vody, tedy $d_t + b_t$. Výtok bude nabývat vždy kladných hodnot, neboť přesun vody opačným směrem bude uložen ve zdrojové sousední buňce. Daný vzorec byl odvozen z práce O'Brien *et al.* [9] bez uvažování externího tlaku na kapalinu. Ostatní strany mají analogické rovnice. Pro proporcionální rozdělení toků dále je zaveden faktor K následovně:

$$K = \min\left(1, \frac{d_A}{\Delta t} \cdot \frac{l_X l_Y}{\hat{f}^L + \hat{f}^R + \hat{f}^T + \hat{f}^B}\right) \tag{2.12}$$

kde l_X a l_Y jsou délky buněk na odpovídajících osách (a tedy i vzdálenosti buněk) a d_A je mezihodnota sloupce vody. Výsledné výtoky jsou násobkem mezivýpočtu a získané hodnoty K, tedy:

$$f^L_{t+\Delta t}(x,y) = K \cdot \hat{f}^L_{t+\Delta t}(x,y) \tag{2.13}$$

a podobně pro ostatní směry.

2.4.3 Úprava sloupce vody

Dalším krokem algoritmu je zjištění objemů k přesunutí a odvození druhého mezivýsledku sloupce vody. Rozdíl objemu vody je pomocí výtoků vyjádřen následovně:

$$\Delta V(x,y) = \Delta t \cdot \left(\sum f_{in} - \sum f_{out}\right) \tag{2.14}$$

kde f_{in} jsou výtoky v sousedních buňkách směrem ke zkoumanému bodu a f_{out} jsou výtoky zkoumané buňky. Suma vstupů je tedy:

$$\sum f_{in} = f^R_{t+\Delta t}(x-1,y) + \cdots + f^B_{t+\Delta t}(x,y+1) \tag{2.15}$$

Výstupní tok je součtem výtoků zkoumané buňky:

$$\sum f_{out} = \sum_{i=L,R,T,B} f^i_{t+\Delta t}(x,y) \tag{2.16}$$

Uvažujeme-li obsah buňky jako $l_X \cdot l_Y$, pak změna sloupce a druhý mezivýpočet d_B lze z rozdílu objemu vyjádřit jako:

$$d_B(x,y) = d_A(x,y) + \frac{\Delta V(x,y)}{l_X l_Y} \tag{2.17}$$

2.4.4 Úprava pole rychlosti

Pro určení rychlosti vody je nejdříve uvažováno množství protékající zkoumaným bodem určitým směrem, zde značené ΔW, jež je rozděleno na jednotlivé osy. Pro směr osy X je pak průtok vody (v objemu za jednotku času) vyjádřen následovně:

$$\Delta W_X = \frac{1}{2} \cdot \left(f^R(x-1,y) - f^L(x,y) + f^R(x,y) - f^L(x+1,y)\right) \tag{2.18}$$

Kladné hodnoty ΔW tedy vyjadřují dominantní tok směrem vpravo na ose X. Pomocí průměrné výšky sloupce vody $\overline{d}$ a délky buňky v druhé ose l_Y je možné určit obsah průřezu tekoucí vody a komponentu rychlosti u v čase $t + \Delta t$ lze pak získat podílem:

$$u_{t+\Delta t} = \frac{\Delta W_X}{l_Y \cdot \overline{d}} \tag{2.19}$$

Při uvážení rovnoměrného přesunu vody mezi buňkami za časový krok lze průměrnou výšku sloupce vody zapsat $\overline{d} = \frac{1}{2} \cdot (d_A + d_B)$, neboť d_A byla výška před výměnou. Obdobným postupem je pak možné vyjádřit komponentu v pro osu Y.

2.4.5 Eroze a ukládání sedimentu

Uváděný model využívá empirickou metodu pro výpočet kapacity vody danou rovnicí [6][3]:

$$C(x,y) = K_c \cdot \sin(\alpha(x,y)) \cdot |\vec{v}(x,y)| \tag{2.20}$$

kde α vyjadřuje náklon povrchu vzhledem k vodorovné ploše, $\vec{v}$ rychlost vody a K_c konstantu kapacity. Pro ploché terény, kde rychlost sice může nabývat vyšších hodnot, ale náklon se blíží nule, také autor uvádí možnost zavedení prahové hodnoty náklonu. Namísto $\alpha(x,y)$ by pak bylo použito:

$$\max(T_\alpha, \alpha(x,y)) \tag{2.21}$$

kde T_α je uživatelem zvolené minimum. Nahrazení by zajišťovalo alespoň nějakou úpravu terénu. Po získání kapacity lze hodnotu porovnat se současně neseným sedimentem, reprezentovaným proměnnou s_t, s mezivýsledkem s_A. Platí-li, že $C > s_t$, pak dochází k rozpuštění části terénu do vody:

$$\begin{aligned} b_{t+\Delta t} &= b_t - K_s(C - s_t) \\ s_A &= s_t + K_s(C - s_t) \end{aligned} \tag{2.22}$$

kde K_s je konstanta eroze. Pokud naopak platí $C \leq s_t$, potom dochází k sedimentaci:

$$\begin{aligned} b_{t+\Delta t} &= b_t + K_d(C - s_t) \\ s_A &= s_t - K_d(C - s_t) \end{aligned} \tag{2.23}$$

kde K_d je konstanta ukládání sedimentu.

2.4.6 Přesun vody, sedimentu a vypařování

Transport materiálu v modelu vychází z rovnice advekce pro nestlačitelnou kapalinu, daného rovnicí:

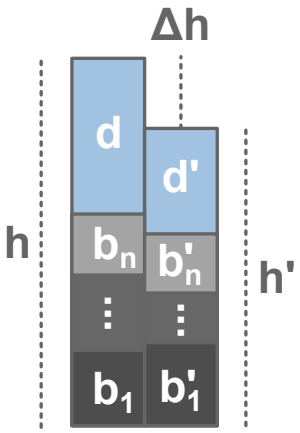
$$\frac{\partial s}{\partial t} = -(\vec{v} \cdot \nabla s) \tag{2.24}$$

Numerických řešení rovnice je více. V modelu Mei *et al.* je pro výsledné množství sedimentu $s_{t+\Delta t}$ použit vzorec [7]:

$$s_{t+\Delta t}(x,y) = s_A(x - u \cdot \Delta t, y - v \cdot \Delta t) \tag{2.25}$$

kde u a v jsou komponenty X a Y rychlosti vody. Namísto dopředného přesunu sedimentu tedy vyhledává zpětně v čase. Neleží-li takto vytvořený bod přímo na mřížce, pak model využívá interpolace z okolních buněk. Alternativním řešením diferenciální rovnice je uvedeno například v práci Šťáva *et al.*, jež využívá metody zvané *Semi-Lagrangian MacCormack*, jež má vyšší stupeň přesnosti a je stále stabilní [10]. Posledním krokem modelu je vypařování

[3] Rovnice je uvedena na stránce 760 jako $S \cdot q \cdot \overline{u}$, kde S je konstanta, q je tangens úhlu povrchu a $\overline{u}$ je průměrná rychlost vody. Změna na sinus úhlu byla provedena v Mei *et al.* pro zjednodušení výpočtu.



Obrázek 2.8: Sloupce materiálu zobrazující několik vrstev terénu b

vody. Pro mezivýsledek sloupce vody d_B je výsledná hodnota získána dle rovnice 2.26, kde K_e je konstanta vypařování.

$$d_{t+\Delta t}(x,y) = d_B(x,y) \cdot (1 - K_e \cdot \Delta t) \tag{2.26}$$

2.5 Vícevrstvé modely

Výše zmíněné metody se zabývají pouze homogenním materiálem. Ve skutečném světě se ale terén skládá z více vrstev, jež jsou různě ovlivněny procesem eroze. Rozšířením algoritmu z předchozí kapitoly se zabývá práce Šťáva et al. [11], jež přidává pro každou buňku nová pole. Sloupec buňky pak vypadá jako v ilustraci 2.8.

Každá vrstva materiálu má tedy vlastní tloušťku pro každou buňku uloženou v polích. Tloušťka pro vrstvu k je zde značena jako b_k. Celkovou výšku buňky lze pro sloupec vody w získat součtem se všemi vrstvami:

$$h(x,y) = w(x,y) + \sum_k b_k(x,y) \tag{2.27}$$

Odlišnou sílu eroze v závislosti na vrstvách model reprezentuje odlišnou kapacitou sedimentu. Předchozí vzorec pro kapacitu 2.20 upravuje následovně:

$$T_k(x,y) = C_k \cdot |v(x,y)| \cdot \sin(x,y) \tag{2.28}$$

kde konstanta C_k je definována pro vrstvu k a $\sin(x,y)$ je sinus úhlu povrchu na daných souřadnicích. Dále je třeba zakomponovat vrstvy do samotné úpravy terénu, tedy eroze a depozice.

K ukládání materiálu dochází v případě, že množství rozpuštěného sedimentu přesahuje kapacitu nejvyšší vrstvy, tedy $s_t(x,y) > T_{top}(x,y)$. Výpočet uloženého materiálu se řídí stejným vzorcem jako je vzorec 2.23, dochází ale k depozici pouze do nejvyšší vrstvy, jež by měla reprezentovat vždy sediment nebo nejlehčí materiál ve sloupci.

Přesahuje-li kapacita rozpuštěný sediment, pak dochází k erozi vrstev povrchu. Zde je ale třeba dbát na vyšší povrch, jež byl již rozpuštěn. Práce Šťáva et al. zavádí upravenou kapacitu sedimentu $\hat{T}_k$ pro zkoumanou úroveň:

$$\hat{T}_k(x,y) = T_k(x,y) - \sum_{i=k+1}^{top} b_i \tag{2.29}$$

kde k je index zkoumané úrovně a *top* je index té nejvyšší. Výsledná hodnota tedy vyjadřuje maximální hloubku, kam se dostane voda a tedy kterou rozpustí, po proniknutí všemi vyššími vrstvami. Sloupec buňky je pak procházen od nejvyšší vrstvy a je postupně porovnávána upravená kapacita s rozpuštěným sedimentem s.

Proces je ilustrován v algoritmu 1, jež odstraní část vrstvy a přidá ji do neseného sedimentu. Pokud už na další erozi nemá voda kapacitu, pak je cyklus zastaven.

Algorithm 1: Postupná eroze vrstev

```
for k ← top to 0 do
    if T̂_k(x, y) < s(x, y) then
        Erode()
    else
        break
```

2.6 Simulace termální eroze

Pomocí celulárních automatů lze rovněž simulovat termální erozi. Model pro tento proces je použitelný i pro přemísťování uloženého sedimentu. Práce autora J. Balázse se zabývá právě simulací termální eroze na grafické kartě [5].

Daný algoritmus termální eroze využívá systému rour stejně jako výše představený model vodní eroze 2.4. Prvním krokem algoritmu je zjištění maximálního množství materiálu, které je buňka schopná předat sousedům. Pro výpočet je zaveden rozdíl výšek H mezi zkoumanou buňkou a nejnižším sousedem. Platí tedy:

$$H = \max\{b - b_i; i = 1, \ldots, 8\} \tag{2.30}$$

Množství, které lze následně přesunout je dáno rovnicí 2.31. Rovnice 2.32 ilustruje výpočet v případě, že je třeba použít lokální tvrdost materiálu R. Oba výpočty využívají plochu buňky a. Dále lze upravit přesunuté množství pomocí konstanty C_t.

$$\Delta S = \frac{a \cdot H}{2} \tag{2.31}$$

$$\Delta S = \frac{a \cdot H \cdot \Delta t \cdot C_t \cdot R_t(x,y)}{2} \tag{2.32}$$

Další částí výpočtu je zjištění množství materiálu, které v jednotlivých rourách bude přesunuto. Za tímto účelem autor zavádí množinu A, která obsahuje sousedy buňky, jež jsou níže a jejichž úhel je vyšší než sypný úhel materiálu. Podmínka je zobrazena v rovnici 2.33.

$$\forall n \in A : b > b_i \wedge \frac{b - b_n}{l} > R(n) \cdot C_a + C_i \tag{2.33}$$

V rovnici 2.33 je b výška zkoumané buňky, b_n výška souseda a l je vzdálenost mezi danými buňkami. Sypný úhel materiálu ovládají konstanty C_a a C_i. Pro lokální tvrdost povrchu zahrnuje rovnice funkci R.

Podobně jako u vodní eroze jak pak třeba rozdělit proporcionálně množství materiálu, které bude odebráno. Vypočtené množství materiálu je násobeno podílem výšek ovlivněných sousedů a jejich sumy.

$$\Delta S_i = \Delta S \frac{b_i}{\sum_{b \in A} b} \tag{2.34}$$

Druhou fází termální eroze je úprava výšek povrchu. Stejně jako v modelu vodní eroze v kapitole 2.4 je výsledný rozdíl materiálu získán součtem množství procházejícím rourami.

2.7 Simulace částicemi

Celulární automaty nejsou jediný způsob využití výškových map pro účely eroze. Namísto abstrakce protékajícího množství vody se výše zmíněné mechanizmy dají aplikovat na jednotlivé částice pohybující se v daném poli. Autor H.T. Bayer ve své práci zavádí právě model vodní eroze simulující jednotlivé kapky [1].

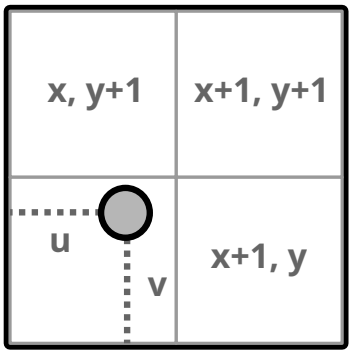
Pro reprezentaci každé částice jsou definovány dvourozměrné vektory pozice p a směru pohybu dir. Dále je zavedena rychlost pohybu v, neboť směr je zaveden jako normalizovaný vektor, tedy vždy bude platit $|dir| = 1$. Každá kapka také může mít různý objem vody w a podobně jako u celulárních automatů je definováno množství neseného sedimentu s.

Principem algoritmu je pohyb částice směrem největšího sklonu terénu a skládá se z následujících kroků:

1. Rozmístění částic na povrchu
2. Výpočet nového směru a pozice
3. Eroze nebo depozice
4. Určení nové rychlosti
5. Vypařování

Kroky 2 až 5 se opakují dokud částice neopustí simulovanou oblast nebo není zachycena v některém lokálním minimu povrchu, tedy dosáhne nulové rychlosti. Pro omezení délky simulace je také možné nastavit maximální počet iterací.

Pro pohyb částice je nejdříve nutné získat náklon povrchu v daném bodě. Výpočet nového směru využívá gradienty výškové mapy g, jejíž hodnotu je možné získat různými způsoby. Jako bod, pro který je gradient počítán, lze například považovat střed buňky, kde se částice nachází. Pro přesnější určení přímo v pozici kapky autor využívá nejbližších čtyřech buněk a bilineární interpolace.



Obrázek 2.9: Ilustrace okolí kapky při výpočtu gradienty. Posun kapky uvnitř buňky je zapsán jako proměnné u a v. Okolní buňky jsou označeny jejich souřadnicemi.

Leží-li částice na ose mezi celočíselnými souřadnicemi x, $x+1$, y a $y+1$ a je-li výška buňky značena $h_{x,y}$, pak gradientu lze vyjádřit dle rovnice 2.35. Posun unvitř daného okolí je značen proměnnými u a v pro osy X a Y. Posun je zobrazen v ilustraci 2.9.

$$g = \begin{bmatrix} (h_{x+1,y} - h_{x,y}) \cdot (1-v) + (h_{x+1,y+1} - h_{x,y+1}) \cdot v \\ (h_{x,y+1} - h_{x,y}) \cdot (1-u) + (h_{x+1,y+1} - h_{x+1,y}) \cdot u \end{bmatrix} \tag{2.35}$$

Výpočet nového směru pak pomocí konstanty setrvačnosti c_i lze vyjádřit vzorcem:

$$dir_A = dir_{t-1} \cdot c_i - g \cdot (1 - c_i)\, dir_t = \frac{dir_A}{|dir_A|} \tag{2.36}$$

Výsledný směr dir_t je tedy získán normalizací mezivýsledku dir_A. Nastane-li situace, kdy by byl vektor směru nulový, pak lze zvolit náhodný směr. Důvodem rozdělení rychlosti na směr a velikost je prevence vytváření artefaktů na povrchu při vyšších rychlostech. Díky jednotkové velikosti směru jej lze přímo přičíst k pozici kapky bez rizika přeskočení buňky:

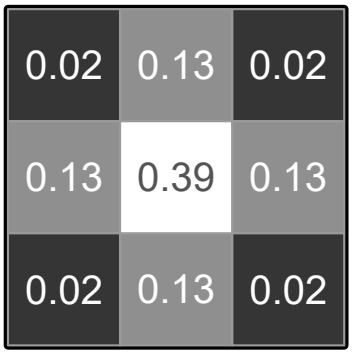
$$p_t = p_{t-1} + dir_t \tag{2.37}$$

Situace, při které by částice strávila mnoho kroků uvnitř jedné buňky, je naopak zanedbána. Iterace algoritmu tedy nepředstavují konstatní časové kroky. Pro účely výpočtu eroze je následně vyjádřen rozdíl výšek mezi novou a starou pozicí:

$$\Delta h = h_t - h_{t-1} \tag{2.38}$$

Je-li $\Delta h > 0$, pak se částice pohybovala směrem vzhůru a v algoritmu bude docházet k depozici materiálu. Místo depozice je stará poloha částice, aby se zaplnila předpokládaná prohlubeň, ze které se kapka dostala. Je-li množství neseného sedimentu dostatečné k zaplnění prohlubně, pak zde uloží potřebný sediment a zbytek si ponechá. Jinak je uložen veškerý nesený sediment. Záporný rozdíl výšek vede k přepočítání kapacity kapky. Pro volitelnou konstantu c_c je kapacita sedimentu definována následovně:

$$C = c_c \cdot \max(-\Delta h, c_{min}) \cdot v_t \cdot w_t \tag{2.39}$$



Obrázek 2.10: Váhy eroze pro okolní buňky. Poloměr eroze je nastaven na 1,5.

Autor také definuje konstantu c_{min} pro nastavení minimálního rozdílu výšek. Vyšší hodnoty vedou k silnější erozi i ve velmi plochých oblastech terénu, což může být žádoucí. Je-li výsledná kapacita menší než nesený sediment, pak je přebytek uložen do staré polohy částice. Množství depozice lze také upravovat volitelnou konstantou:

$$c_d \cdot (s_t - C) \tag{2.40}$$

V případě, že částice má přebytečnou kapacitu, pak dochází k erozi. Zde je kladen důraz na nevytváření děr do povrchu. Množství materiálu, jež je odebráno ze staré pozice částice, se řídí vzorcem:

$$\min\left(c_e \cdot (C - s_t), -\Delta h\right) \tag{2.41}$$

kde c_e je konstanta eroze. Daný postup zajišťuje, že částice neodebere větší výšku terénu, než kterou sama sestoupila. V opačném případě by docházelo k nestabilitě a oscilaci povrchu. Pro rozdělení vlivu jak eroze, tak depozice, je možné znovu využít interpolaci.

Neboť částice může zacházet pouze se svým nejbližším okolím, může docházet k vytváření rysů povrchu, jež jsou užší, než je žádoucí. Pro rozšíření vlivu částice autor zavádí váhy eroze pro okolní buňky rovnicí 2.42. V dané rovnici figuruje P_i jako pozice okolní buňky o indexu i a konstanta c_r, jež definuje poloměr eroze. Váhy tedy vytváří kužel kolem částice. Možné váhy ilustruje obrázek 2.10.

$$w_i = \frac{\max\left(0, c_r - (|P_i - p_t|)\right)}{\sum_{k=0}^{n} \max\left(0, c_r - (|P_k - p_t|)\right)} \tag{2.42}$$

Při zpomalení kapky je důvodem nejčastěji velmi lokální změna terénu. Depozice je tedy stále prováděna pouze pro nejbližší okolí, neboť širší ovlivnění by nebylo žádoucí. Další možností odstranění příliš tenkých rysů je míchání s rozmazanou verzí současné mapy ilustrované v algoritmu 2.43. Zde figuruje konstanta c_b, jež ovlivňuje přimíchané množství.

$$map = c_b \cdot map_{blurred} + (1 - c_b) \cdot map_{original} \tag{2.43}$$

Následujícím krokem je získání nové rychlosti částice. Hodnota je vyvozena z uraženého rozdílu výšek Δh a konstanty pro gravitaci c_g geometrickým průměrem:

$$v_t = \sqrt{v_{t-1}^2 + \Delta h \cdot c_g} \tag{2.44}$$

Podobně jako pro ostatní modely je množství vody násobeno faktorem vypařování c_e:

$$w_t = w_{t-1} \cdot (1 - c_e) \tag{2.45}$$

2.8 Využité technologie

V této kapitole jsou rozebrány nástroje a technologie, na kterých je vytvořené rozšíření založeno a které dále budou odkazovány v implementaci v kapitole 4.

2.8.1 Blender

Blender je bezplatný open-source sofware s širokou nabídkou nástrojů pro 3D tvorbu jako modelování, animace, vizuální efekty nebo i fyzikální simulace. Aplikace je vytvářena neziskovou nadací Blender, jež se snaží poskytovat přístupné a kvalitní 3D nástroje pro veřejnost a také nabídnout soběstačné prostředí pro 3D tvorbu. Nově také Blender nabízí prostředí pro 2D animaci a vektorovou grafiku.

Aplikace je multiplatformní, přístupná jak pro Windows, tak pro macOS nebo Linux. Zaměřuje se na interoperabilitu s jinými aplikacemi a herními enginy, jako například Source, Unity, nebo Unreal Engine. Podporuje jak importování, tak exportování široké škály formátů včetně průmyslových standardů jako FBX a OBJ.

Kromě samotné 3D tvorby Blender nabízí kvalitní a fyzicky přesný vykreslovací engine Cycles, jež využívá raytracing. Také podporuje hardwarové urychlení grafických karet i procesorů a technologie odstranění šumu OpenImageDenoise a OptiX. Pro rychlé vykreslování a animace ještě poskytuje engine zvaný EEVEE, který používá kombinace tradičních vykreslovacích metod. Další enginy, jako například LuxCore, Octane nebo Corona, lze také přidat a integrovat do aplikace skrze rozšíření.

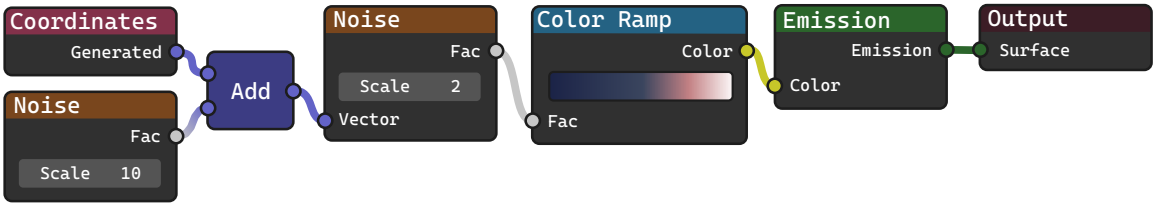
Díky snadné rozšiřitelnosti a flexibilnímu uživatelskému rozhraní je aplikace užitečná jak pro amatérské tak profesionální uživatele. Široké využití nachází ve filmové i herní tvorbě, nebo například v architektonické vizualizaci. Skrze zásvuné moduly je možné použít Blender v odvětvích jako například 3D tisku, nebo datové vizualizaci. Také má aktivní komunitu, jež dále rozšiřuje funkcionalitu.

Blender API

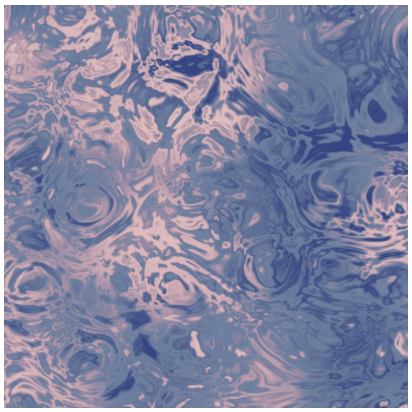
Prvky uživatelského rozhraní, veškeré operace i data všech prvků v aplikaci Blender jsou přístupná skrze aplikační rozhraní v jazyce Python. Vše, čeho může v aplikaci uživatel dosáhnout, lze rovněž provést skrze skripty. V kombinaci s nabízeným flexibilním způsobem tvorby GUI aplikace Blender poskytuje snadné prostředí pro tvorbu rozšíření.

K datům a funkcím aplikace lze přistoupit uvnitř skriptů skrze Python knihovny Blenderu. Modulů přístupných rozšířením je mnoho a pokrývají celou škálu funkcionality, včetně práce se zvukem, grafickou kartou nebo fonty. Mezi základní součásti knihovny patří následující moduly:

- `context` — poskytuje současný stav scény v aplikaci včetně vybraných prvků.



(a) Graf vykresleného materiálu. Využívá procedurální texturu `Noise` pro posun mapování druhé textury. Výsledek je převeden na barvy pomocí uzlu `Color Ramp` a vykreslen uzlem `Emission`.



(b) Vykreslený objekt

Obrázek 2.11: Příklad Blender materiálu s deformovanou procedurální texturou

- `data` — obsahuje seznamy všech vytvořených entit pod správou aplikace.
- `types` — definuje třídy základních datových typů, jako je objekt nebo obrázek.

Aby se předešlo únikům paměti, tak správa zdrojů je zcela ovládaná aplikací Blender. Tvorba objektů, obrázků a obecně alokace většího množství zdrojů je možná pouze skrze funkce poskytované knihovnou aplikace. Vrácené objekty, se kterými následně skript pracuje, mohou být následně zneplatněny, například při změně aktivního objektu, je-li uložen do proměnné. S prvky scény a uživatelského rozhraní se tedy pracuje pomocí řetězcových jmen, jež je jednoznačně identifikují.

Uživatelské skripty a rozšíření jsou spouštěny pomocí interpretu jazyka Python, jež je vlastní součástí instalace aplikace. S tímto interpretem je nainstalováno již několik knihoven, jako například `numpy`. Jsou-li potřeba externí knihovny pro skript, musí být získány skrze nástroje daného interpretu.

Materiály

V aplikaci Blender lze definovat vzhled objektů pomocí materiálů. Každý objekt jich může mít několik, kde každý může být aplikován na vybrané vrcholy či polygony. Také mohou být sdíleny mezi objekty. Uživatel v aplikaci Blender materiály definuje pomocí editoru grafů.

Vlastnosti povrchu jsou v grafu materiálu stavěny spojováním různých typů uzlů. Každý má definované vstupy a výstupy, jež jsou barveny dle jejich datového typu. Při spojení uzlů

je datový typ řádně kontrolován, aplikace ale dovoluje i automatické převody mezi některými typy. Příkladem je převod vektoru na barvu či naopak.

Nabízených uzlů je mnoho a pokrývají širokou škálu vstupů a operací. Základním prvkem pro vykreslování jsou takzvané shader uzly, které přímo definují typ povrchu, jež zahrnuje například difůzní, metalické a další. Rozsáhlou podporu také poskytuje barevným operacím, jako jsou různé typy míchání barev, úpravy HSL hodnot nebo invertování. S barvami také souvisí mnohé procedurální textury přístupné v materiálech a nástroje pro jejich mapování. Příklad struktury materiálu, převodů a výsledku lze vidět v ilustraci 2.11.

Rovněž lze provádět libovolné výpočty užitím skalárních, vektorových i Booleovských matematických operací. Je-li výpočet příliš složitý i pro dané uzly, pak lze využít skriptovacího prvku. Zde může uživatel vložit libovolný program napsaný v jazyce Open Shading Language, zkracovaného jako OSL. Vykreslování je ale limitováno na použití procesoru.

Další uzel, jež bude v tomto projektu zmiňován, vytváří efekt zvaný *bump* mapování. Užitím textury napojené na daný uzel lze docílit iluze deformace povrchu. Výstup materiálu navíc zahrnuje možnost skutečného posunu vrcholů objektu, v případě že je použito vykreslování Cycles. Daný proces se nazývá *displacement* mapování.

Důležité jsou také vstupní uzly, které poskytují informace nejen o samotném objektu, ale také o jeho geometrii, povrchu nebo například o vlastnostech vykreslovaného paprsku. Tyto prvky je možné použít ke komplexnímu mapování textur na objekty, nebo změně směru výše zmiňovaných deformací.

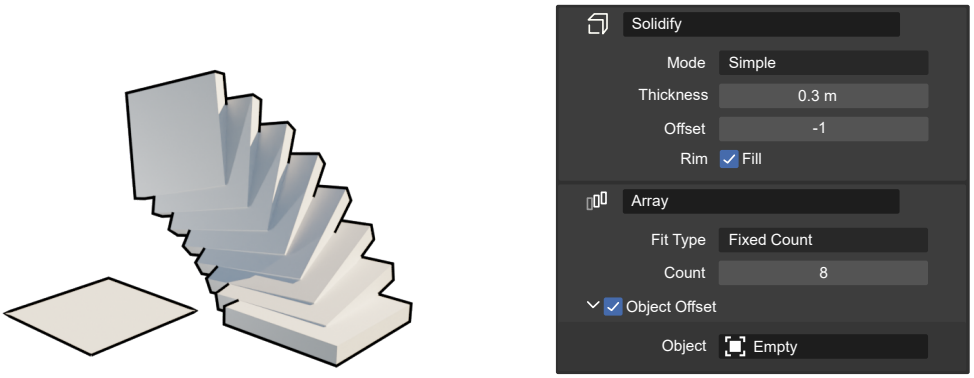
Uzly uvnitř materiálu lze vytvářet, spojovat i pozicovat zcela skrze Blender API, je tedy možné procedurálně a automaticky přidávat i měnit materiály dle potřeb rozšíření.

Modifikátory

Pro nedestruktivní úpravu objektů, neboli bez permanentní změny jejich skutečné geometrie, nabízí aplikace Blender prvky zvané modifikátory. S jejich pomocí lze snadno duplikovat, deformovat nebo i animovat 3D modely. Úpravy samotné geometrie se pak okamžitě projeví ve výsledném povrchu.

Mezi nejčastější modifikátory patří `Subdivision Surface`, jež štěpí polygony povrchu, čímž přidává modelu větší detail. Dále je možné použít `Array`, jež vytvoří řadu duplikátů modelu, buď se specifikovaným posunem, nebo dle pomocného objektu. Lze tak snadno vytvářet komplexní útvary, nebo libovolné n-úhelníky. Příklad tohoto modifikátoru lze vidět v ilustraci 2.12. S tímto prvkem také souvisí `Mirror`, jež zrcadlí objekt dle dané osy či objektu.

Další úpravou využitou v tomto projektu je `Displace` modifikátor, jež deformuje povrch dle poskytnuté textury. Posun, rotaci a škálování dané textury na povrch lze provádět využitím pomocného objektu. Tato nastavení lze rovněž ovládat skrze Blender API, včetně vytváření nových úprav, nebo přeskupování již existujících.



(a) Transformace čtverce na komplexní útvar (b) Použité modifikátory

Obrázek 2.12: Příklad modifikátorů s použitím *Freestyle* vykreslovací funkce pro zviditelnění. Levé ploše je přidána tloušťka a posun objektu je definován neviditelným objektem `Empty`.

Shape Keys

Další metodou nedestruktivní úpravy je použití funkcionality zvané *shape keys* pro daný model. Jedná se o způsob přídání variace nebo oprav modelu pomocí posunu jednotlivých vrcholů. Ty jsou definovány vůči vrstvě základního modelu, zvané báze. Také nabízí možnost násobit tento posun, je tedy možné snížit nebo zvýšit vliv těchto úpravy. Příklad použití Shape Keys ilustruje obrázek 2.13.

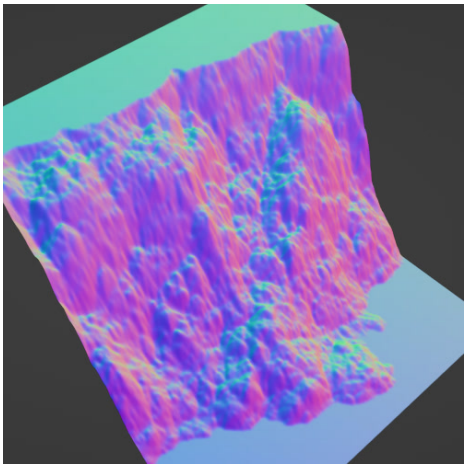
Vytvářet vrstvy lze přímo z modifikátorů a vrstev lze takto udělat několik. Jejich vlivy je tak následně možné kombinovat. Rozdíl mezi modifikátory a shape keys je ten, že modifikátory upravují libovolnou vstupní geometrii, zatímco shape keys definují pro každý vrchol specifickou pozici. Modifikátory, které mění počet vrcholů tedy nejde jako shape keys aplikovat.

2.8.2 OpenGL

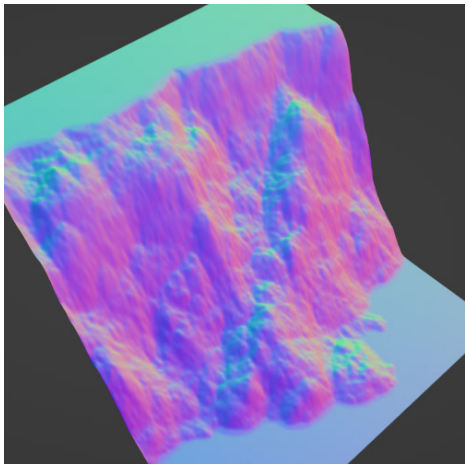
OpenGL je víceplatformní grafické API s podporou jak pro 2D, tak 3D vykreslování. Využití nachází v široké škále aplikací, od jednoduchých nástrojů či her, tak v CAD návrhu, datové vizualizaci nebo virtuální realitě. OpenGL dovoluje programátorovi využít grafickou kartu a hardwarové urychlení výpočtů a poskytuje vysoce upravitelný systém a nástroje.

Řetězec kroků vykreslování, anglicky zvaný *rendering pipeline*, se dělí na specifické fáze, jež se s postupným vývojem OpenGL dále rozšiřují a obohacují. Obrázek 2.14 danou strukturu ilustruje. Základní fáze, jež budou dále zmiňovány, jsou následující:

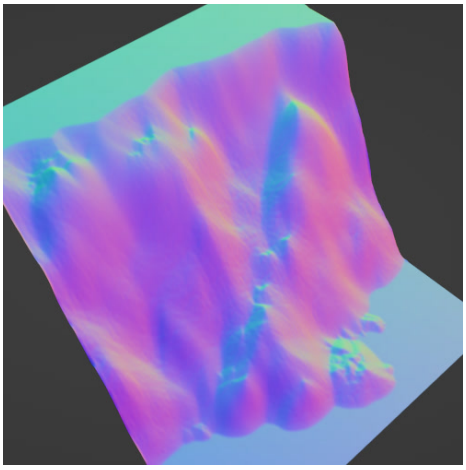
1. Aplikační
2. Geometrická
3. Rasterizace
4. Fragmentová
5. Výstupní



(a) Bázový model



(b) Poloviční vliv úpravy



(c) Plný vliv

Obrázek 2.13: Příklad a srovnání použití Shape Keys pro termální erozi. Zbarveno dle normál povrchu pro lepší viditelnost.

Aplikační fáze zahrnuje práci prováděnou na CPU. Zde skrze knihovnu OpenGL dochází k nastavení scény a objektů. Voláním z aplikační fáze dochází k zahájení vykreslení. Zobrazované prvky se skládají z vrcholů, jež nejprve procházejí geometrickou fází.

V geometrické fázi je možné modely transformovat pomocí matic, či libovolných matematických operací. Kombinací různých matic pak lze například dosáhnout deformace povrchu, pozicování objektu ve scéně nebo perspektivního zobrazení. OpenGL také umožňuje povrch modelu dále modifikovat tvorbou nových vrcholů, nebo štěpením vstupních prvků (proces zvaný *teselace*).

Po dokončení transformace modelu dochází na fázi rasterizace. V této části vykreslování dojde k převodu grafických primitiv na jednotlivé fragmenty a vyřazení částí mimo viditelnou oblast. Data, jež vrcholy vstupní geometrie nesou, se také interpolují mezi jednotlivými fragmenty. Například určením texturových souřadnic pro jednotlivé vrcholy je pak umožněno snadné vykreslování obrázků v prostoru.

Vytvořené fragmenty a interpolovaná data využívá následující fragmentová fáze. Ta zpracovává rasterizované pixely na jejich finální barvu. Součástí tedy může být osvětlování, stínování nebo mapování textur. Před dokončením vykreslení mohou být ještě výsledné fragmenty testovány vůči již vykreslenému obsahu, nebo s ním mohou být smíchány. Součástí tohoto vyhodnocení bývá často vyřazení fragmentů na základě hloubky, čímž je možné vykreslovat překrývající se objekty.

OpenGL nabízí mnoho způsobů jak jednotlivé fáze dále přizpůsobit, včetně různých vykreslovacích modelů, mapování nebo světelných efektů. Také je možné psát vlastní shadery v jazyce OpenGL Shading Language, neboli GLSL.

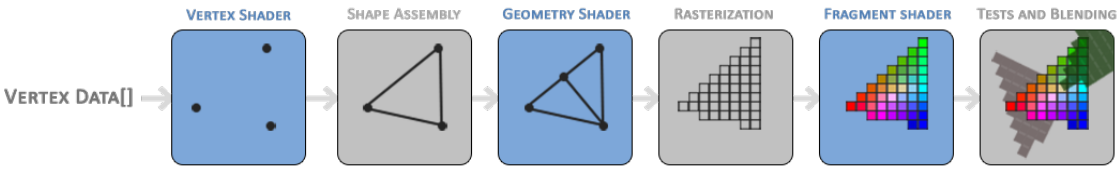
Nezávislou fází v OpenGL je takzvaný výpočetní shader. Daný fáze slouží pro obecné výpočty na grafické kartě, na rozdíl od výše zmíněného přístupu zaměřeného čistě na vykreslování. Daný program je spuštěn paralelně s více vlákny, kde každé z nich pracuje nad blokem dat. Fáze se nachází po výpočtu geometrie, ale před výpočtem fragmentů. Výstup může být navázán na další fáze, nebo může být vrácen zpět procesoru.

Pro vykreslování je třeba s pomocí OpenGL vytvořit několik objektů. Pole dat je nejdříve nutné převádět na takzvané buffer objekty. Pro vykreslení modelu se bude jednat o pole vrcholů a popřípadě jejich indexů, jež slouží k deduplikaci dat. Tyto objekty se následně kombinují ve `Vertex Array Object`, jež je zkracován jako VAO. Do daného objektu je zahrnutý i shader program. V tomto projektu také bude použito vykreslování do textury, jež využívá takzvaný `Frame Buffer Object`, čili FBO. Ten může zahrnovat i několik různých textur.

Pro účely implementace je také nutné zmínit, že OpenGL vykresluje pouze vrcholy, jež se nachází v krychli o souřadnicích $[-1, 1]$. Tento prostor se nazývá *Normalized Device Coordinates* a zkracuje se jako NDC. Pro jakékoliv zobrazení, ať už perspektivní nebo kolmé, musí být veškeré modely transformovány do tohoto rozmezí.

2.8.3 GLSL

OpenGL Shading Language, neboli GLSL, je vysokoúrovňový programovací jazyk zaměření na grafické zpracování v OpenGL. Byl navržen k psaní vykreslovacích programů spustitelných na grafické kartě bez znalosti assembly jazyka ARB, jež GLSL předcházel.



Obrázek 2.14: OpenGL proces vykreslování. Zdrojem je LearnOpenGL.com

Syntaxe a datové typy GLSL vychází z jazyka C, navíc ale přímo poskytuje matematické funkce, vektory, matice a další. Také je možné v programu použít zabudované proměnné, jež obsahují další informace o vrcholech či fragmentech, jež jsou právě zpracovávány.

Pomocí klíčových slov *in* a *out* lze definovat vlastní vstupy a výstupy jednotlivých programů a předávání vlastních dat mezi fázemi vykreslování. Naopak externí parametry lze dodat programu skrze takzvané *uniform* proměnné, jež zahrnují i textury nebo pole. Pro textury je možné pomocí speciálních funkcí provádět i zápis, čehož tato práce silně využívá.

2.8.4 ModernGL

ModernGL je Python knihovna, jež zaobaluje OpenGL. Jejím cílem je poskytnout rozhraní a syntaxi blížší jazyku Python (anglicky je přístup označován slovem *Pythonic*). Tímto přístupem se liší od knihoven jako jsou PyOpenGL nebo OpenTK (pro jazyk C#), jež se snaží o přesné zachování zaobalených funkcí.

Rozhraní je navíc usnadněno seskupením navazujících operací ve společná volání. Například texturu je možné vytvořit a nastavit pomocí argumentů jediné funkce, na rozdíl od jednotlivých kroků vytvoření, navázání a nastavení, jež je typický pro OpenGL. Ke všem nastavením a nástrojům OpenGL má ale s knihovnou uživatel stále přístup.

Mezi další výhody knihovny patří automatická správa OpenGL zdrojů skrze objekty v Pythonu, což redukuje riziko úniků paměti a dalších chyb, a vylepšení zpracování chyb a diagnostiky aplikací. Velká část knihovny je navíc již zkompilována v jazyce C++, což vede k urychlení programu a redukuje velikost knihovny. Důležitou funkcionalitou knihovny pro tuto práci je schopnost se napojit na již **existující OpenGL kontext**, jako například uvnitř aplikace Blender.

2.9 Existující rozšíření

V současnosti jsou již rozšíření, jež se erozí povrchu zabývají. Díky tomu, že jsou spuštěny přímo v aplikaci Blender, je možné, aby pracovali přímo nad tvořenými objekty a lze je využít ve spojení s další funkcionalitou aplikace Blender. Alternativní řešení skrze externí programy jsou probrány v další kapitole 2.10. Tato část uvede vlastnosti dvou vybraných příkladů.

2.9.1 A.N.T. Landscape

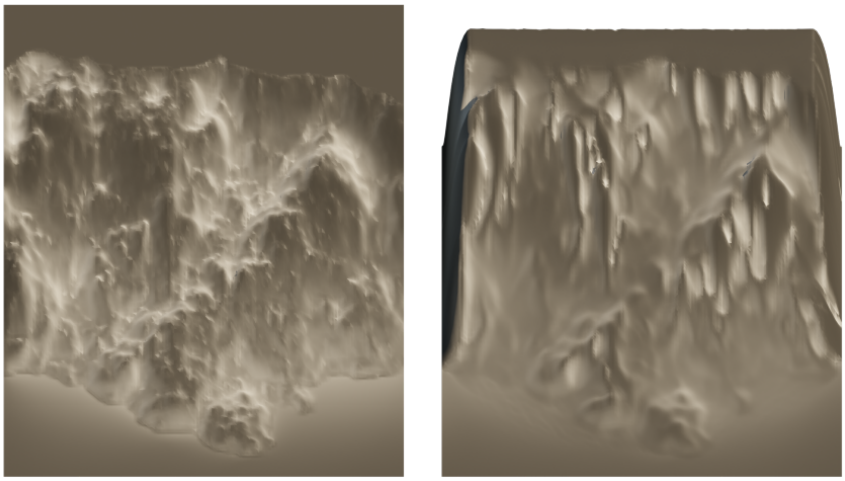
Součástí instalace aplikace Blender je již od základu rozšíření A.N.T. Landscape, jež obsahuje funkci pro erozi povrchu. Jeho hlavním zaměřením je tvorba terénů na základě kombinací procedurálních textur, lze ale také vytvářet planety a jednoduché vodní plochy[4].

Daný zásuvný modul vytváří objekty, jež ukládají vlastní parametry. Mezi ně patří například rozlišení objektu, použitá procedurální funkce, oblast, kde se aplikuje, typ stratifikace a další. Díky uložení těchto nastavení je lze následně měnit, tím tedy i terén samotný.

Nad rovinnými terény vytvořenými daným rozšířením je také poskytována simulace eroze. Jako příklad výsledku je uvedena ilustrace 2.15. Eroze se v rozšíření skládá ze zvoleného počtu iterací tří různých algoritmů, kde každý má dále vlastní počet opakování.

Prvním je difůze materiálu do okolních buněk s nastavitelnou silou, kde velký vliv vede k uhlazení povrchu. Dále nabízí termální erozi s maximálním úhlem povrchu a silou vlivu jako parametry. Poslední komponentou je vodní eroze, jež jako argumenty poskytuje například množství a variaci deště, sílu eroze a depozice, nebo sedimentovou kapacitu vody.

Oproti navrhovanému projektu je možné dané rozšíření použít pouze nad terény, jež samo vygenerovalo. Dalším rozdílem je rychlost eroze. Rozšíření A.N.T. Landscape běží na procesoru, pro větší rozlišení terénu tedy proces může výrazně trvat.



Obrázek 2.15: Kombinovaná vodní a termální eroze povrchu vytvořeného rozšířením A.N.T. Landscape

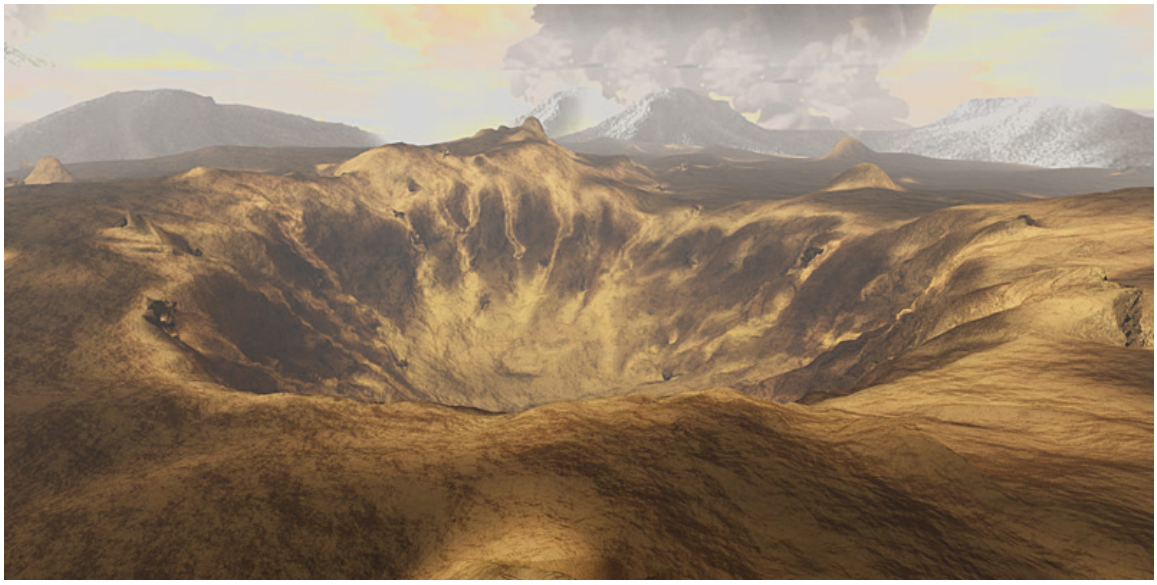
[4]Popis rozšíření byl získán z dokumentace rozšíření na stránkach Blender.org a experimentováním s programem.

2.9.2 Erosion Add-on

Pro erozi je také možné použít placené rozšíření Erosion Add-on přístupné na stránkách Blender Market. Tento zásuvný modul poskytuje vlastní algoritmus jak 2D, tak prostorové eroze s podporou převisů[5].

Erozi lze provádět nad libovolným objektem a její průběh je možné přizpůsobit pomocí křivek profilů terénu. Kromě eroze také nabízí mapy několika typů toků. Výsledek eroze je také možné snadno kombinovat s částicovým systémem aplikace Blender. Plná verze rovněž nabízí tvorbu modelů řek a jezer. Příklad krajiny vytvořené rozšířením ilustruje obrázek 2.16.

Navrhovaný projekt nabízí oproti danému rozšíření urychlení na grafické kartě, algoritmus termální eroze a přímou erozi obrázků. Dané rozšíření také pracuje pouze přímo s vrcholy oproti texturovému přístupu navrhovaného projektu.



Obrázek 2.16: Terén vygenerovaný rozšířením Erosion Add-on.

2.10 Externí aplikace

Mimo rozšíření, jež jsou přímo spustitelná v aplikaci, existuje i řada samostatných programů pro úpravu terénu. Díky vysoké flexibilitě exportu i importu aplikace Blender lze dané nástroje použít pro vývoj modelů, i přes relativně složitější postup oproti zásuvným modulům. Jako reprezentativní příklady jsou zde uvedeny aplikace Wilbur, pracující na základě textur, a Terragen, jež je schopna i vlastního vykreslování.

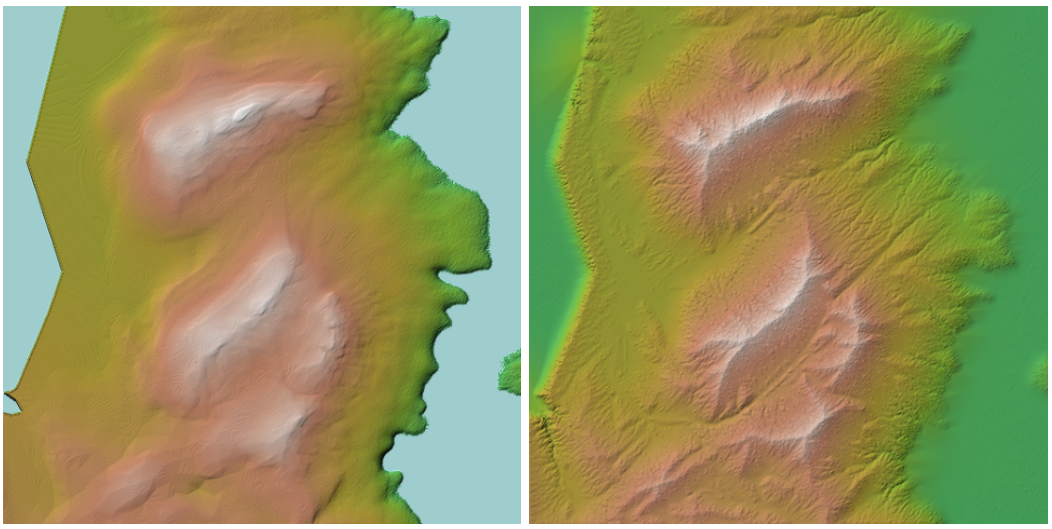
[5]Popis a ilustrace získány ze stránek rozšíření na blendermarket.com

2.10.1 Wilbur

Wilbur je aplikace pro tvorbu terénu s dlouholetou historií. Poskytovaná funkcionalita se zaměřuje na práci s výškovými mapami. Nabízená je jak generace terénů pomocí různých procedurálních funkcí, tak jejich následná úprava řadou metod[6].

Důležitými úpravami v kontextu této práce jsou funkce vodní eroze, jež jsou v programu dvojího typu. Model zvaný *Percipiton* nabízí pouze jeden parametr pro sílu, dá se ale nastavit typ okolí buněk v modelu, opakování mapy na okrajích i vliv moře v určité výšce. Jednotlivé iterace lze mezi sebou nastavitelně míchat a k výsledku přidávat šum. Výstupem je detailní a tenké zvrásnění povrchu.

Druhý model eroze je zvaný *Incision* a oproti předchozímu algoritmu tvoří rozsáhlejší rysy povrchu. Jako parametry využívá množství toku a sílu rozmazání povrchu před a po dané operaci. Příklad eroze terénu je ilustrován v obrázku 2.17.



(a) Původní terén (b) Terén po erozi

Obrázek 2.17: Aplikace obou metod eroze programem Wilbur

Program lze získat zdarma a je stále vyvíjen. Mapy vytvořené v této aplikaci lze uložit v mnoha formátech a dvou bitových hloubkách jako obrázky, nebo přímo vyexportovat jako model.

2.10.2 Terragen

Terragen je placená aplikace pro vytváření terénů. Jejím cílem je kromě generování i plné vykreslování realistických světů. Současná verze Terragen 4 disponuje řadou metod tvorby od velikosti několika kamínků až po celé planety a na terén lze aplikovat prakticky neomezený fraktální detail. Pracovní postup aplikace využívá grafů pro specifikaci terénů, materiálů i způsobu vykreslování. Uživatel tedy aplikuje změny vytvářením a spojováním uzlů podobně jako v aplikaci Blender[7].

[6] Program Wilbur byl získán ze stránky fracterra.com. Popis funkcí byl získán experimentováním s programem.

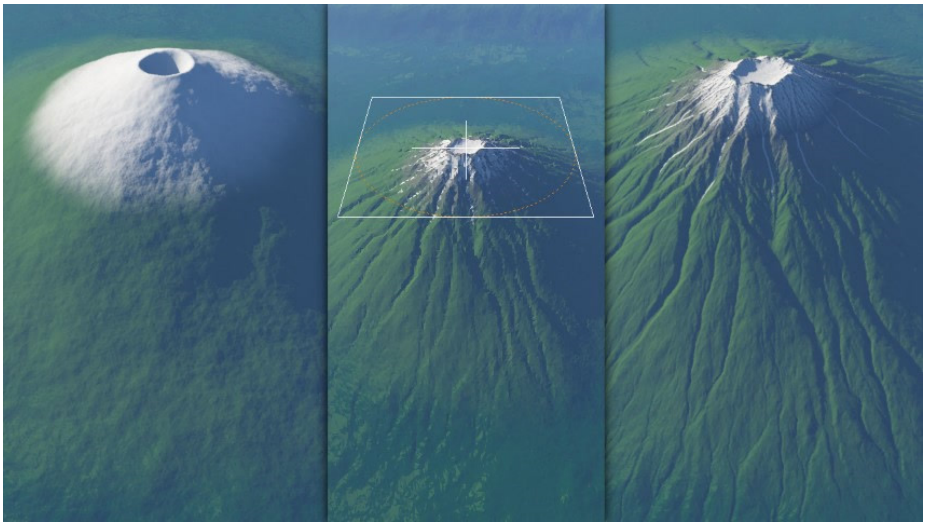
[7] Popis funkcí aplikace Terragen byl získán z přehledu na oficiálních stránkách.

Jeden ze zabudovaných uzlů grafu terénu slouží pro erozi výškové mapy. Jako parametry podporuje délku toku, rozděluje sílu eroze a depozice a zahrnuje i sesuv půdy pro maximální úhel terénu. Výstupem uzlu může být vytvořená výšková mapa, nebo rozdíl oproti původnímu povrchu[8]. Příklad eroze povrchu tímto uzlem je ilustrován v obrázku 2.18.

(a) Původní terén (b) Terén po erozi

Obrázek 2.18: Příklad výsledků funkce eroze. Zdrojem obrázků jsou wiki stránky aplikace

Aplikace Terragen také nabízí rozšíření funkcionality přes zásuvné moduly. Jedním z nich je *Classic Erosion*, jež oproti zabudovanému řešení přidává parametr škály terénu, experimentální erozi řekami a nastavení oblasti pomocí objektů, kde se bude eroze provádět[9]. Uvedené rozšíření je placené, stejně jako Terragen ale nabízí rovněž omezenou verzi zdarma. Pro použití s aplikací Blender lze výškové mapy z programu vyexportovat. Příklad eroze terénu je ukázán v ilustraci 2.19.



Obrázek 2.19: Příklad výsledků rozšíření. Zdrojem jsou stránky autora.

[8]Přehled parametrů lze najít na wiki stránkách aplikace.
[9]Přehled možností rozšíření *Classic Erosion* lze najít na stránkách autora: `https://daniilkamperov.com/`

Kapitola 3

Návrh řešení

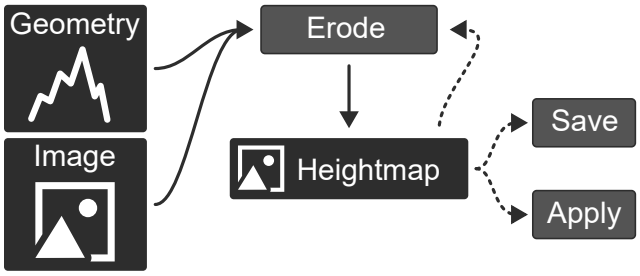
V této kapitole je představen zamýšlený pracovní postup při použití aplikace, jež bere v potaz požadavky výše zmíněných algoritmů a vlastnosti i limitace dostupných prostředků aplikace Blender. Dále se kapitola zabývá uživatelským rozhraním, jež bylo na daný postup orientováno s cílem dynamického zobrazení. V poslední řadě jsou zde prezentovány požadavky na funkcionalitu, jež je třeba implementovat. Výsledný projekt byl pojmenován *Hydra*, dle čehož jsou odvozeny uživatelské prvky, adresář aplikace i prefixy funkcí a tříd zmíněných později v části implementace.

3.1 Zacházení s rozšířením

Cílem projektu je rozšíření aplikace Blender schopné simulace vodní a termální eroze na libovolné vstupní geometrii, reprezenetované objektem ve 3D scéně, nebo na libovolném obrázku výškové mapy. Uživatel má také možnost erozi aplikovat vícenásobně.

Principem rozšíření je převod buď obrázku, nebo zvoleného objektu, na výškovou mapu, která je následně zvětrána. Výslednou texturu pak rozšíření zobrazí nebo nanese zpět na objekt. Navrhovaný pracovní postup eroze, jež je ilustrován v obrázku 3.1, se skládá z následujících kroků:

1. Automatické vygenerování výškové mapy
2. Vodní nebo termální eroze
3. Automatické zobrazení výsledku
4. Aplikování nebo exportování výsledku
5. Uvolnění dat



Obrázek 3.1: Navrhovaný pracovní postup vodní a termální eroze

Při prvním spuštění eroze je vygenerována výšková mapa zvolené velikosti. Daná velikost je pak pro objekt **neměnná** a udává rozměry eroze. Jak pro erozi obrázků, tak geometrie, je navržen stejný systém výškových map, jež se dělí na dvě různé textury:

1. Zdrojová mapa — textura, ze které bude vycházet eroze.
2. Současná mapa — výsledek simulace eroze.

Zdrojová mapa figuruje jako záchytný bod spjatý s objektem. Je-li následně volána několikrát funkce eroze, pak vždy vychází z dané textury, čímž má uživatel možnost zkoušet různé parametry. Chce-li uživatel dále pracovat již s výsledkem eroze, pak je schopný nastavit současnou mapu jako zdrojovou. Tímto způsobem lze také následně aplikovat termální erozi a naopak.

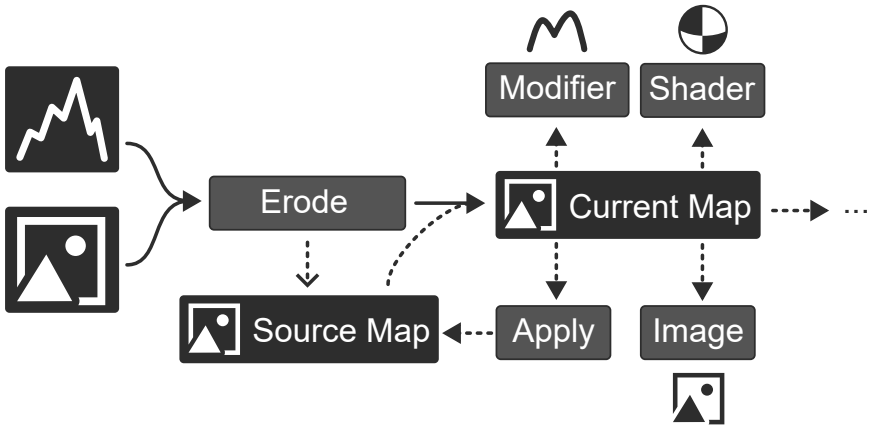
Je možné znovu vygenerovat zdrojovou mapu, pokud se chce uživatel vrátit, nebo pokud došlo ke změně vstupu. Daným postupem tvorby a uchovávání výškové mapy se odděluje potenciálně náročný výpočet povrchu od samotné eroze, u které lze předpokládat, že bude prováděna vícekrát se stejným terénem.

Při prvním vytvoření zdrojové mapy je také uložena její kopie, dále označovaná jako modelová mapa. Vstupní objekty nebo obrázky pak nemusí být znovu převáděny na výškové mapy při aplikování výsledku, kde je daná textura potřeba.

3.1.1 Aplikování výsledku

Výsledkem eroze je současná výšková mapa, pro kterou je automaticky generován náhled. Při práci s objektem jsou rovněž umožněny další způsoby, jak simulovanou erozi na objekt nanést. Pracovní postup včetně systému výškových map a možností aplikování výsledku je zobrazen v ilustraci 3.2. Navrhované řešení nabízí aplikování skrze modifikátory a materiály zmíněné v teoretické sekci 2.8.1.

Výše zmíněné úpravy je rozšíření schopno zcela automaticky generovat, přepisovat a adaptovat na materiály i modifikátory již vytvořené uživatelem. Výstupem může také být převod výškové mapy na obrázek v aplikaci Blender.



Obrázek 3.2: Ilustrace systému výškových map s možnostmi aplikování výsledku. Eroze ze vstupů na pravo umí vytvořit automaticky zdrojovou mapu. Ze zdrojové mapy se dále erozí tvoří současná mapa. Pro erozi současné mapy ji lze nastavit jako zdroj. Jako způsoby aplikování současné mapy jsou zde uvedeny převody na modifikátor, na materiál a na obrázek.

3.1.2 Uvolnění zdrojů

Závěrečným krokem pro erozi je čištění dat. Díky rozdělení terénu a jeho výškové mapy jsou vytvořené textury uchovávány v rozšíření. Dokončil-li uživatel práci, nebo potřebuje-li uvolnit paměť, pak lze vytvořené textury vymazat vždy dostupnou operací. Jelikož velikost textury u objektu je po prvním vytvoření neměnná, je také možné touto operací textury uvolnit a zvolit jiné rozlišení.

3.2 Návrh funkcionality

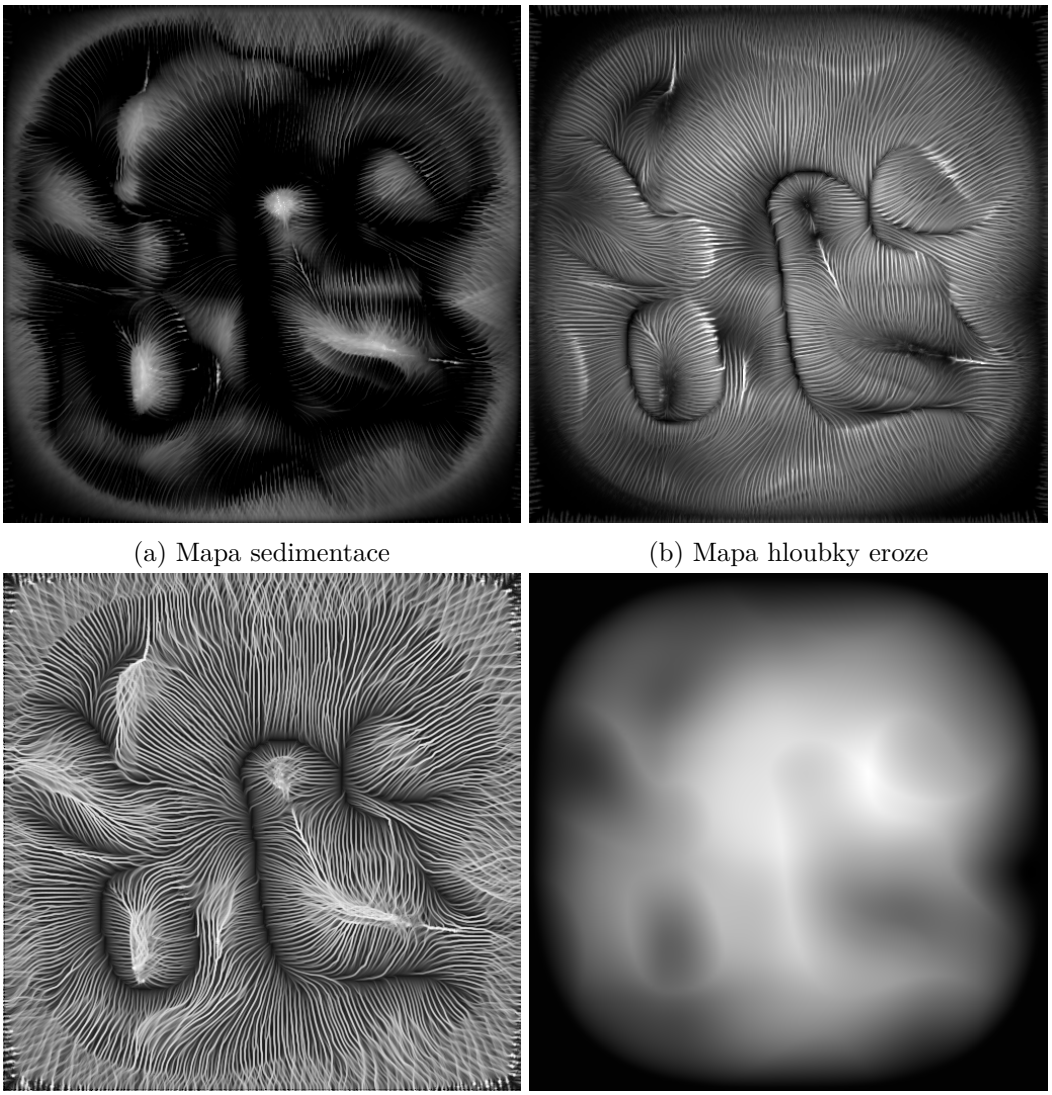
Mimo obecné požadavky, tedy kromě samotné eroze, by mělo rozšíření nabízet i další operace, jež jsou s použitými algoritmy spjaty. Uživatel pak bude schopný vygenerovat přídavná data týkající se povrchu a pohybu vody — specificky sedimentaci, hloubku eroze a koncentraci toků v jednotlivých bodech. Neboť tvorba výškové mapy je nutnou funkcí rozšíření, tak lze poskytovat generování přímo do samostatného obrázku jako samostatnou operaci dostupnou k užití. Zmíněné výstupy ilustruje obrázek 3.3. Požadavky, které z daných operací vzcházejí, jsou uvedeny v následujících podkapitolách.

3.2.1 Transformace objektů a tvorba výškové mapy

Pro maximální využití výškové mapy je nejdříve třeba transformovat vybraný objekt tak, aby zaplňoval co nejvíce její plochy. Navrženým řešením je vytvoření kvádrové obálky kolem výběru. Na základě jejích rozměrů je možné získat střed útvaru a matici jak pro posun, tak pro škálování do tvaru textury. Transformaci objektu ilustruje obrázek 3.4.

3.2.2 Aplikování výsledku eroze

Jak bylo zmíněno v sekci 3.1 pracovního postupu, mezi způsoby aplikování eroze nabízí rozšíření i úpravu materiálu objektu. V tomto případě je ale třeba řešit již uživatelem vytvořené prvky, nebo potenciálně předchozí běhy eroze.



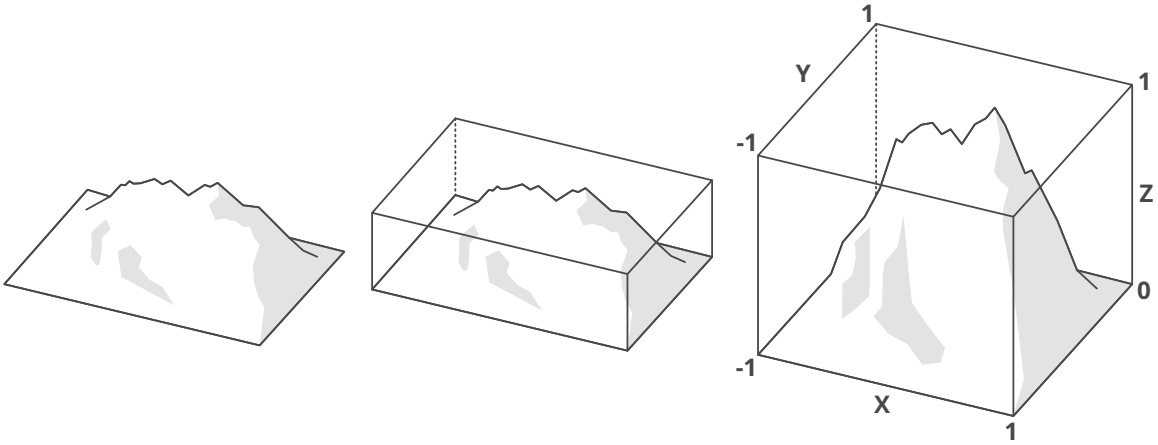
(a) Mapa sedimentace

(b) Mapa hloubky eroze

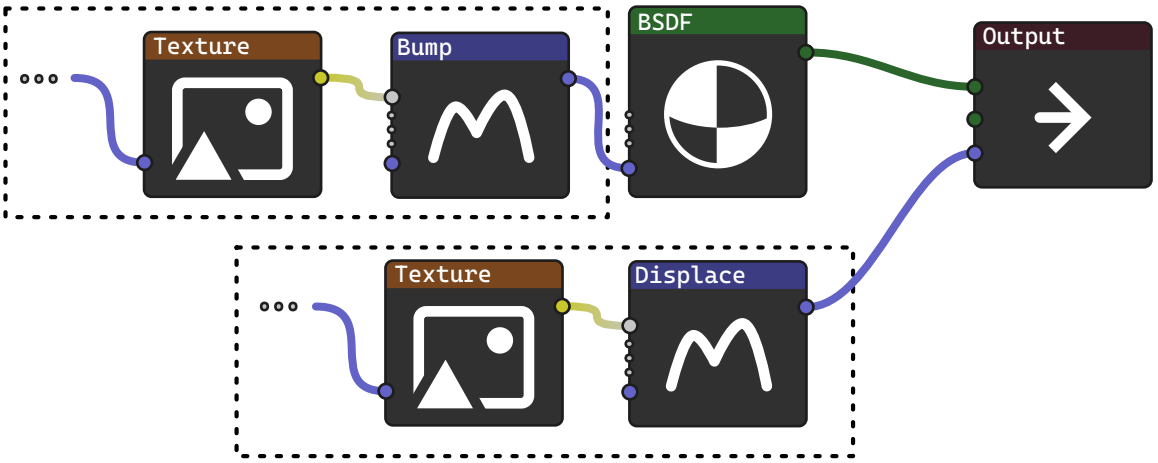
(c) Mapa koncentrace toků

(d) Výšková mapa

Obrázek 3.3: Ilustrace dalších výstupů rozšíření.



Obrázek 3.4: Příklad obálky objektu a škálování do ilustrovaných rozměrů. Výsledný povrch je v rozmezí specifikovaném v podkapitole OpenGL 2.8.2. Nejnižší bod povrchu je mapován na hodnotu 0 a nejvyšší na hodnotu 1.



Obrázek 3.5: Možnosti automatického doplnění materiálu vyznačeny obdélníky. Ilustrované uzly jsou vysvětleny v teoretické části 2.8.1.

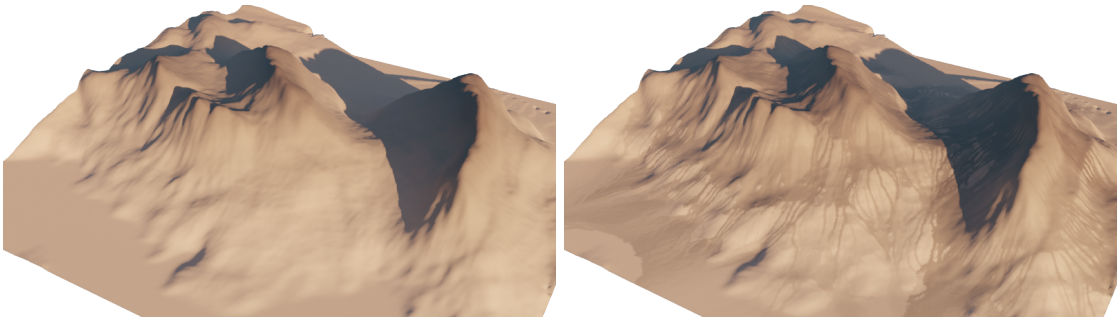
Není-li materiál definován, pak je rozšíření schopné vytvořit nový. Jinak je procházen a musí být vytvářeny všechny uzly, jež chybí pro správnou funkcionalitu materiálu. Příklad přidaných prvků je zobrazen v obrázku 3.5.

3.2.3 Transport barev a textury

Ne vždy je třeba upravit samotný model pro reálně vypadající krajiny, jako například u vzdálených kopců nebo zalesněných ploch. Potřeby uživatele může zde splňovat i barevná textura, jež pouze indikuje směr toku vody a mísení uvolněných materiálů. Za tímto účelem navržené rozšíření implementuje transport barev procesem eroze.

Myšlenkou rozšíření existujících algoritmů je dodání textury uživatelem a v průběhu procesu buď načítat barvu při erozi, nebo při sedimentaci na mapu nesenou barvu nanášet. Možný výsledek ilustruje obrázek 3.6.

Další funkcionalitou spjatou s texturami je převod barevného prostoru obrázků v aplikaci Blender. Při načítání a následném generování nesprávného formátu může dojít k ztmavení či zesvětlení textury. V případě eroze obrázku to má za následek i tvorbu nesprávné výškové mapy. Rošíření tedy musí být schopné různé barevné prostory rozlišit a navzájem převádět.



(a) Původní textura (b) Upravená textura

Obrázek 3.6: Ukázka transportu barev na modelu s 30,000 polygony.

3.2.4 Preference a instalace závislostí

Navrhované rozšíření vyžaduje externí závislosti. Aby byl projekt pro uživatele přívětivý, měl by být schopen automaticky dané externí knihovny stáhnout a nainstalovat. Aplikace Blender nabízí možnost tvorby vlastní stránky v seznamu rozšíření, kde bude navržená operace stažení umístěna. Zde budou také uvedeny obecné preference projektu.

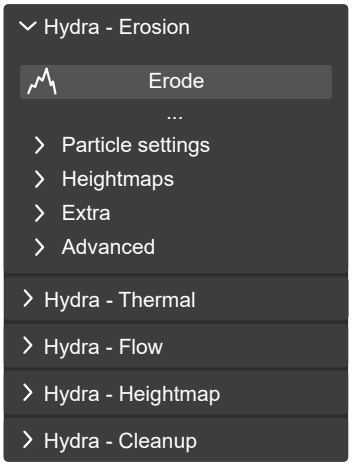
3.3 Uživatelské rozhraní

První volbou v návrhu rozhraní je celkové umístění v rámci aplikace. Například na rozdíl od rozšíření A.N.T. Landscapes, jež pro nastavení eroze využívá dočasného okna, byla zvolena vlastní záložka *Hydra* v pravém kontextovém menu. Návrh struktury záložek je ilustrován v obrázku 3.7.

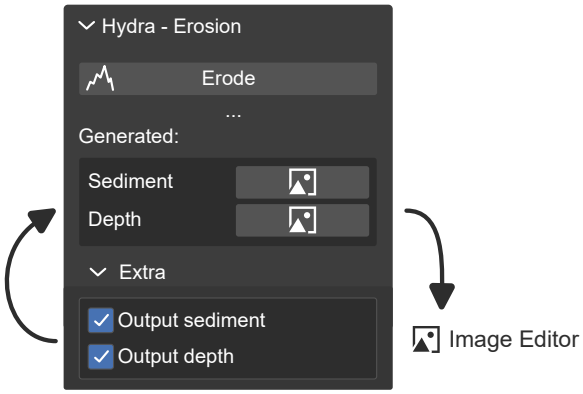
Jelikož se jedná o prvek rozhraní, jež není navázán přímo na určitou akci, tak samotné provedení operací bude spouštěno tlačítkem. Dané řešení dovoluje oddělit úpravu parametrů od zahájení s nimi spjaté operace, jež bude výpočetně náročná. Tím se také dá předejít zamrznutí a případnému pádu při posuvné uprávě parametrů, která by akci spouštěla mnohokrát.

Kromě eroze by mělo rozšíření nabízet další operace, jež mají rozdílné vstupy a kontexty. Funkce rozšíření jsou tedy rozděleny do několika samostatných panelů. Akce vázané přímo s objekty pak lze zobrazovat pouze tehdy, když je určitý objekt vybrán, například vyčištění alokované paměti je ale možné vždy.

Neboť rozšíření tvoří výstupy napříč mnoha editory a seznamy uvnitř aplikace Blender, tak musí být brán zřetel na informování uživatele a navigaci mezi výsledky. Příklad rozhraní, jež takovou funkcionalitu bude poskytovat, je ilustrován v obrázku 3.8.



Obrázek 3.7: Ilustrace struktury uživatelského rozhraní a přehled návrhu funkcionality



Obrázek 3.8: Návrh propojení výstupů. Seznam vytvořených textur je nabízen přímo v rozhraní a dovoluje zobrazit jednotlivé položky.

Kapitola 4

Implementace

V této kapitole je představeno řešení výše navržené funkcionality. Nejdříve je zde zmíněn souhrn inicializace a základní struktura rozšíření. Dále kapitola zachází do detailu uživatelského rozhraní a operátorů, jež jsou vstupní body jednotlivých algoritmů. Dalším bodem je generování výškových map, implementace jednotlivých typů eroze a přídavných operací. V poslední řadě je představeno řešení aplikování výsledků a navigace k nim.

4.1 Inicializace a struktura rozšíření

Tato kapitola zahrnuje registraci rozšíření v aplikaci, načítání a inicializaci modulů a základní strukturu aplikace.

4.1.1 Registrace

Prvním krokem spuštění rozšíření je vyhledání externí knihovny ModernGL, bez které nemůže rozšíření fungovat. Veškeré načítání uživatelských prvků a dalších modulů se v inicializaci odvíjí od přítomnosti této knihovny.

Je-li ModernGL nainstalováno se zjišťuje pokusným importováním v inicializačním souboru rozšíření. Ošetření případné výjimky pak nastavuje příznak neplatného spuštění, dle čehož se dále řídí uživatelské rozhraní. Není-li knihovna nainstalovaná, pak se načte pouze okno preferencí, jež uživatele provede instalací. Jinak se zbytek spuštění skládá z následujících kroků:

1. Registrace tříd uživatelského rozhraní
2. Vytvoření globálních dat a jejich inicializace
3. Napojení kontextu OpenGL
4. Načtení a překlad shader programů
5. Vytvoření globálních nastavení scény
6. Vytvoření individuálních nastavení entit

Třídy uživatelského rozhraní jsou registrovány individuálně pomocí funkce `register_class` z knihovny `bpy.utils`. Seznam tříd, jež se mají registrovat, uchovávají jednotlivé moduly v konstantních seznamech `EXPORTS`.

Pro získání OpenGL kontextu je využita schopnost ModernGL se napojit na již existující kontext aplikace Blender. Naopak vytvoření samostatného kontextu vede okamžitě k **pádu aplikace**. Získaný objekt je uložen do globální proměnné `common.data`, skrze kterou je k němu následně přistupováno.

Odregistrace

Seznam tříd uživatelského rozhraní je nejprve v opačném pořadí odregistrován, následně jsou smazána globální i objektová nastavení, jež by jinak mohla přetrvávat v souboru, a potom, pokud byla načtena knihovna ModernGL, dojde k uvolnění veškerých dat OpenGL.

4.1.2 Společná data

Ústředním prvkem rozšíření je modul `common`, jež slouží pro uchovávání alokovaných textur, vytvořených shader programů a předávání zpráv nebo chyb uživatelskému rozhraní. Ostatním modulům jsou tyto prvky poskytovány globální proměnnou `data`.

Mezi alokované textury patří především výškové mapy, jež jsou uchovávany ve slovníku. Proměnná `data` skrze metody zprostředkovává tvorbu map, jejich mazání a přístup k nim. Rovněž obsahuje seznamy řetězců a funkci na jejich zobrazení. Je tak možné při konání operací agregovat zprávy pro uživatele. Jako klíče výškových map slouží náhodné UUID.

4.1.3 Nastavení rozšíření

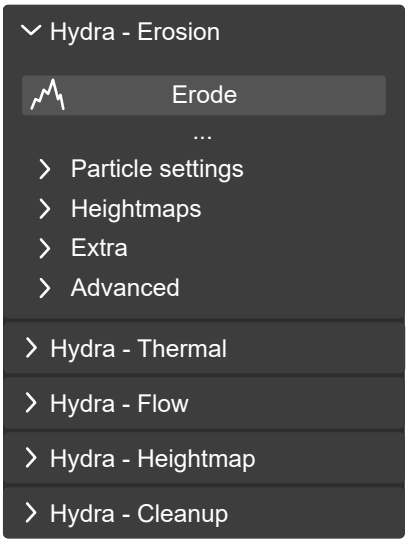
Aby nebylo třeba vždy znovu zadávat nastavení operací, tak je třeba někde uchovávat jejich hodnoty. Rozšíření definuje dvě skupiny nastavení pomocí třídy `types.PropertyGroup`, jež se ukládají v souboru Blender projektu.

Objekty a obrázky sdílejí stejnou skupinu nastevní. Jejich třídám je přiřazena jako vlastnost skrze typ `props.PointerProperty`. Hodnoty nastavení jsou nezávislé a unikátní pro každou instanci. K nastavením lze v rozšíření přistoupit skrze vlastnost `hydra` obou typů objektů.

Důležitými vlastnostmi entit jsou výškové mapy, jež jsou řetězcové hodnoty UUID a které slouží jako odkazy do slovníku map zmíněném výše. Jejich hodnoty jsou upravitelné pouze rozšířením.

Další unikátní nastavení entit, jež nesouvisí se vstupy algoritmů a které bude dále zmiňováno, je pravdivostní hodnota `is_generated`. Vlastnost indikuje, že entita byla vytvořena rozšířením a může být v budoucnu přepsána. Rozšíření nedovoluje u takto označených entit provádět erozi nebo vytvářet textury. Je ale nabízena operace, jež danou entitu osamostatní, čímž je znovu povolena veškerá funkcionalita rozšíření.

Rozšíření také vyžaduje nastavení, jež nemá smysl uchovávat pro každý objekt. Jedním z nich je rozlišení přímého exportu výškové mapy, jež je nezávislé na ostatních operacích. Aby byla hodnota nastavitelná uvnitř rozhraní, tak musí být někde definována jako vlastnost. Stejně jako u objektu byla tedy vytvořen ukazatel v typu `bpy.types.Scene`.



Obrázek 4.1: Ilustrace dělení panelu eroze na záložky

4.2 Uživatelské rozhraní a operátory

V této kapitole je uvedeno obecné rozložení uživatelského rozhraní, získání a ošetření možných vstupů při práci s obrázky i objekty, a dále princip volání dalších částí rozšíření.

Jak bylo zmíněno v kapitole návrhu 3.3, rozhraní bylo umístěno v bočním panelu okna. Prvky rozhraní jsou typu `Panel` a je nutné je individuálně registrovat při inicializaci. Každý panel zaobaluje jednotlivé funkce rozšíření.

V případě eroze bylo nutné řešit velké množství vstupů a zobrazených dat. Tento prvek je tedy dále dělen na několik menších záložek, jež jsou také typu `Panel`. Dané rozložení ilustruje obrázek 4.1. Stejně jako rodičovský prvek je nutné tyto panely individuálně registrovat. Zabalení prvků do sebe lze docílit nastavením vlastnosti `bl_parent_id` na identifikátor obklopujícího prvku.

4.2.1 Rozhraní objektů a obrázků

Rozhraní pro objekty bylo umístěno do okna 3D scény. Specificky je zobrazováno pouze v základním režimu daného okna zvaném `objectmode`. V jiných režimech nemusí být veškerá funkcionalita rozšíření proveditelná. Například při úpravě geometrie nelze aplikovat modifikátory.

Pro práci s texturami bylo rozhraní umístěno do editoru obrázků. U jednotlivých operací zde nedochází ke změně vstupu, režim okna tedy nemá na rozšíření vliv a není brán v potaz.

4.2.2 Získání a ošetření vstupů

Při implementaci uživatelského rozhraní bylo dbáno na zobrazení prvků pouze v případě, že jsou opravdu použitelné, ne se všemi objekty a texturami je rozšíření totiž schopné pracovat.

Podmínečné zobrazení panelů je řešeno skrze třídní metodu `poll`, která vrací pravdivostní hodnotu, má-li být panel zobrazen. Společnou kontrolou ve většině panelů je určení,

jestli byl objekt vygenerován tímto rozšířením. To určuje vlastnost `is_generated` v nastaveních objektu.

Objekty

Aktivní objekt lze kdykoliv získat z kontextu aplikace ve vlastnosti `active_object`. Kontext je také automaticky předáván vykreslovací funkci panelu jako argument. Všechny panely rozhraní, jež pracují s geometrií, ale potřebují kontrolovat, zda-li objekt skutečně geometrii má.

Prvním krokem je kontrola existence objektu, neboť výše zmíněná vlastnost může vracet prázdnou hodnotu. K tomu může dojít například krátce po smazání objektu. Aplikace Blender dále nabízí různé typy objektů, jež jsou rozlišitelné vlastností `type`. Typ, nad kterým je rozšíření schopné pracovat, se nazývá `MESH`. Objekt ale může mít stále prázdnou geometrii. Dále tedy kontrolují panely počet vrcholů, jež je přístupný skrze vlastnost objektu `data.vertices`. Tento seznam pro zvolený typ objektu vždy existuje. V případě prázdného seznamu se panely nezobrazí.

Obrázky

Aktivní obrázek není přímo definovaný v kontextu aplikace, ale náleží oknu editoru obrázků, ve kterém bylo rozhraní umístěno. K obrázku je přistupováno skrze vlastnost `image` proměnné aktivního okna, tedy `area.spaces.active`. Vlastnost může ale mít **prázdnou hodnotu**, jež je kontrolována. Příkladem toho je základní textura `Render Result`, dokud do ní není zakresleno.

4.2.3 Operátory

Funkce rozšíření jsou z uživatelského rozhraní spouštěny skrze takzvané operátory. Celý algoritmus, jež má být takto proveden, probíhá zcela v metodě operátoru `invoke`. Zodpovědností operátoru je tedy provedení přípravy, spuštění a následného úklidu požadované funkcionality pomocí modulů, jež jsou rozebrány v následujících kapitolách.

Obecný postup operace se skládá z volání příslušných modulů a navigace k výsledkům. Společnou součástí operátorů je také resetování zpráv v modulu `common` a po dokončení provádění i jejich výpis.

Aplikování výsledků

Většina operátorů pro aplikování výsledných map je stejná jak pro práci s obrázky, tak s objekty. Neboť musí ale získávat vstupní entity jiným způsobem, tak je u daných operátorů definovaná pravdivostní hodnota, dle které vstupy a postupy rozlišuje.

Tato hodnota je implementována vlastností operátoru a přistupuje se k ní ve vykreslovací funkci rozhraní. Pro přidání operátoru lze použít Blender metodu `operator`. Ta přidá tlačítko a zároveň vrací objekt, přes který lze operátoru předávat další vstupy skrze nastavení vlastností.

Pro všechny obrázky vytvořené rozšířením dále platí, že jsou zabaleny do souboru projektu Blender. Toho lze docílit voláním metody `pack`. Bez zabalení do souboru může dojít ke **smazání dat** při načtení obrázků.

4.2.4 Preference

Rozšíření nabízí dvě globální možnosti v preferencích, jež jsou směr dělení oken a přeskočení indexace. Směr dělení je aplikován při navigaci k výsledku v sekci 4.6 a přeskočení indexace ovlivňuje generaci výškové mapy, což je rozebráno v kapitole 4.3. Tento proces je rozebrán v kapitole 3.

Pro čtení preferencí je třeba přistoupit k vlastnosti `preferences` kontextu a zde vybrat ze seznamu správné rozšíření. V daném objektu je pak seznam nastavení znovu pod názvem `preferences`. Pro zkrácení zápisu byl přístup implementován jako funkce v modulu `common`.

4.3 Výškové mapy

V této kapitole je detailně uveden proces získání výškových map ze vstupních entit. Nejprve je uveden převod z obrázků a následně algoritmus vykreslení geometrie a nutných transformací.

Modul výškových map slouží nejen k přímé tvorbě obrázku, ale je také součástí přípravných fází veškeré eroze. Ústřední funkcí je zde `prepareHeightmap`, jež je volána v případě, kdy objekt nemá vytvořenou zdrojovou výškovou mapu, ze které další algoritmy čerpají.

Výsledkem funkce je kolmá projekce na texturu podél osy Z, kde černá barva má nejnižší výšku a bílá nejvyšší. Mapa je tedy normalizována.

4.3.1 Generování z obrázku

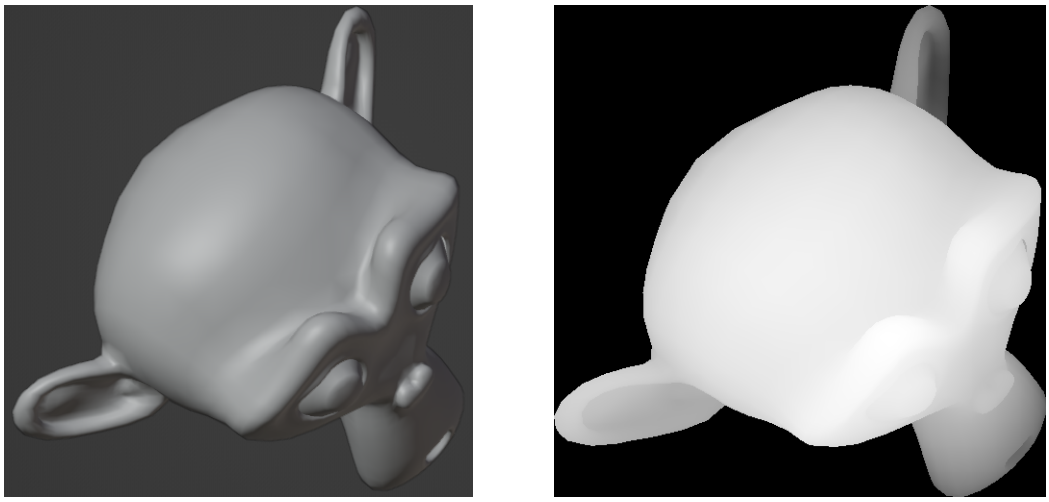
Pro vytváření textury z obrázků je převeden seznam pixelů na `numpy` vektor desetinných čísel. Pixelová data v obrázcích aplikace Blender vždy obsahují tři barevné složky a alfa kanál. Výběrem každého čtvrtého prvku, počínaje prvním, je následně extrahován pouze červený kanál textury, jež rozšíření považuje za výškovou složku. V různobarevných obrázcích tedy další kanály nejsou brány v potaz, stejně jako alfa kanál obrázku.

Výsledek je následně funkcí `copy` převeden na nový vektor, neboť vytvářená textura vyžaduje, aby vstupní data byla **spojitě uložena v paměti**. Z vektoru je pak vytvořena ModernGL textura.

Problém, jež musí být dále řešen v implementaci, je barevný prostor obrázku. Pokud vstup využívá celočíselná data, tedy neplatí `img.is_float`, pak je barevný prostor obrázku sRGB. Hodnoty jeho pixelů bez správného převodu nemusí odpovídat výškám, jež byly do textury původně uloženy. Funkce linearizace je zobrazena v algoritmu 2. Daná funkce je implementována jako výpočetní shader. Neboť výpočet není závislý na okolí pixelu, tak je funkce volána nad již vytvořenou texturou, kterou rovnou přepisuje.

Algorithm 2: Linearizace kanálu barvy [12]

if $channel \leq 0.04045$ **then**
  return $channel \div 12.92$
else
  return $[(channel + 0.055) \div 1.055]^{2.4}$



(a) Původní objekt (b) Odpovídající výšková mapa

Obrázek 4.2: Příklad převodu objektu na výškovou mapu. Objekt obsahuje překrývající se geometrii, kterou je algoritmus schopný správně zpracovat. Objekt je také trochu deformován do čtvercové textury.

4.3.2 Generování z objektu

Pro objekty je převod na výškovou mapu, ilustrovaný v obrázku 4.2, složitější a sestává z následujících kroků:

1. Získání povrchu objektu
2. Transformace povrchu
3. Nastavení pomocných vlastností objektu
4. Vykreslení do textury

Získání povrchu

Jako skutečný povrch tělesa je brán ten, který právě vidí uživatel, tedy včetně modifikátorů a dalších úprav. Kvůli zvolenému způsobu získání geometrie je **ignorována lokální transformace** objektu. Specificky není brána v potaz jeho rotace, neboť posun a škálování dále upravuje vytvořený algoritmus. Průmět modelu na texturu je tedy prováděn pouze na lokální ose Z.

Získání dat z objektu `obj` je definováno v algoritmu 4.1. V daném algoritmu `depsgraph` uchovává informace ohledně současné scény, pomocí kterých lze povrch tělesa vyhodnotit. V proměnné `mesh` pak lze najít seznam vrcholů objektu.

```
depsgraph = bpy.context.evaluated_depsgraph_get()
eval = obj.evaluated_get(depsgraph)
mesh = bpy.data.meshes.new_from_object(eval)
```

Výpis 4.1: Algoritmus vyhodnocení geometrie *mesh* vstupního objektu *obj* pomocí stromu závislostí Blender scény *depsgraph*.

Neboť povrch tělesa může sestávat i ze složitějších polygonů, tak před vykreslením do výškové mapy jsou polygony ještě explicitně rozděleny na trojúhelníky. Toho lze dosáhnout vytvořením `bmesh` objektu a funkcí `bmesh.ops.triangulate`. Podobně jako stejná uživatelská operace nabízí různé způsoby štěpení, ty jsou ale ponechány na základním, tedy `BEAUTY`[1].

Vytvořený objekt `bmesh` a seznam vrcholů jsou následně použity k převodu do OpenGL kontextu. Zde se algoritmus dle operačního systému (zjištěného funkcí `platform.system`) dělí na **dvě větve**. V obou případech je cílem vytvořit seznam trojic vrcholů jež trojúhelníky tvoří. Povrch tak může být tvořen libovolným počtem nespojitých ploch a polygony mohou mít libovolné pořadí. Tento přístup je také nutný, neboť aplikace Blender žádné pořadí polygonů nezaručuje. Daný způsob uložení vrcholů odpovídá formátu `TRIANGLES`, jež bude specifikován při vykreslování.

Na platformě Windows lze pro zefektivnění vykreslování využít indexace vrcholů. Souřadnice vrcholů jsou nalezeny ve vlastnosti *co* jednotlivých prvků seznamu vrcholů. Indexy pak lze získat algoritmem 3 nad dříve vytvořeným `bmesh` objektem. Procedůře `Append` zde přidává argument na konec seznamu indexů.

Algorithm 3: Získání seznamu indexů

```
for face in bmesh.faces do
    for vertex in face.verts do
        Append(indices, vertex.index)
```

Při testování na platformě Ubuntu a v prostředí WSL knihovna ModernGL vykazovala neočekávané chování při využití indexace. Problém je více rozebrán v sekci 5.2. Zde je tedy využit celý seznam vrcholů s duplicitními prvky. Postup je stejný jako algoritmus 3, akorát se akumulují přímo pozice vrcholů.

Výsledné seznamy pak slouží k vytvoření objektu `VertexArray`, jež potřebuje jako vstupy ModernGL typ `Buffer`. Oba seznamy jsou nejprve převedeny na `numpy` vektory a pozice vrcholů ještě na desetinná čísla o čtyřech bytech (formát `f4`). Následně jsou konvertovány na byty, z nichž lze vytvořit `Buffer` objekty. Celý převod tedy vypadá následovně:

[1] Dokumentace: https://docs.blender.org/api/current/bmesh.ops.html#bmesh.ops.triangulate

```
ctx.buffer(data=np.array(vertices).astype('f4').tobytes())
```

Výpis 4.2: Algoritmus vyhodnocení geometrie *mesh* vstupního objektu *obj* pomocí stromu závislostí Blender scény *depsgraph*.

Transformace

Posledním prvkem potřebným k výpočtu mapy je transformační matice objektu. Cílem dané operace je tvarovat objekt tak, aby zaplnil co nejvíce výsledné textury. To tedy znamená, že musí být v OpenGL transformován do krychle o souřadnicích od -1 do 1. Pro jednoduchost následného výpočtu je souřadnice Z rovnou převedena do rozmezí od -1 do 0, vyšší terén je tedy blíže k pozorovateli. To také dovoluje snímat překrývající povrch, jelikož nižší (a tedy vzdálenější) fragmenty budou zahozeny.

K vytvoření matice tranformace využívá modul kvádrového obalu tělesa `bound_box` počítaného aplikací Blender. Jedná se o matici obsahující souřadnice jeho osmi vrcholů. Střed daného kvádru je pak průmerem souřadnic opačných stěn a škálovací vektor v je pak dán vzorcem 4.1.

$$\begin{aligned} s_x &= \frac{2}{\left(v_x^+ - v_x^-\right)} \\ s_y &= \frac{2}{\left(v_y^+ - v_y^-\right)} \\ s_z &= \frac{1}{\left(v_z^+ - v_z^-\right)} \end{aligned} \tag{4.1}$$

V dané rovnici platí $v_{x,y,z}^+ > v_{x,y,z}^-$. Jsou-li strany kvádru pronásobeny odpovídajícími prvky vektoru, pak je celý model v mezích uvedených výše. Dále je třeba započíst posun objektu. Pro střed kvádru c a škálovací vektor s je pak transformační matice dána vzorcem:

$$M = \begin{pmatrix} s_x & 0 & 0 & -c_x \cdot s_x \\ 0 & s_y & 0 & -c_y \cdot s_y \\ 0 & 0 & -s_z & 0{,}5 + c_z \cdot s_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

Důvodem záporného znaménka u osy Z je překlopení směrem ke kameře, jež je v záporném směru. Společně s výpočtem dané matice se **nastavuje poměr stran** objektu — dvě proměnné v datech objektu ve směrech XY a Z, jež jsou následně využity v algoritmech eroze. Jejich výpočet je dán:

$$\begin{aligned} d_{xy} &= \frac{s_x}{s_y} \\ d_z &= \frac{s_x}{s_z} \end{aligned} \tag{4.3}$$

Volání shaderu

Nyní jsou již připravena data pro vykreslení. Následovně je vytvořen `Framebuffer` objekt (dále zvaný FBO) s dvěma napojenými texturami — hloubkovou, jež dovoluje překrývání v modelu, a barevnou, jež bude výsledná výšková mapa. Nejprve je ale třeba izolovat vykreslování od zbytku aplikace.

Aby nebylo ovlivněno rozhraní programu, tak je z daného FBO vytvořen ještě objekt zvaný `Scope`[2], ve kterém samotné vykreslení proběhne. Jemu je nastaven příznak `DEPTH_TEST`, jež povoluje zahazování fragmentů na základě hloubky. Příznak `CULL_FACE`, jež zahazuje odvrácené polygony, nastaven není, orientace polygonů v Blenderu tedy neovlivní jejich vykreslení. Možnost takto nezobrazovat polygony přímo v Blenderu tedy také není brána v potaz a vyhodnocený porvch může vypadat jinak, než jak ho vidí uživatel.

Nastavení příznaků lze i mimo daný objekt, vede to ale k **zneviditelnění** většiny uživatelského rozhraní. Nápodobně použití `FBO` napřímo vede k **uzamčení** modrého a zeleného kanálu celé aplikace, neboť použitá textura má pouze červený. Tyto změny **nelze vrátit**, leda restartováním aplikace.

Samotné vykreslení se skládá ze dvou částí — transformace modelu ve vertex shaderu a následné uložení výšky ve fragment shaderu. Vykreslování je voláno s parametrem `TRIANGLES`, jež udává výše zmíněný formát vstupních vrcholů po trojicích.

První fáze tedy převádí pozice vrcholů na homogenní souřadnice (typ `vec4`) a následně je násobí transformační maticí, jež přijímá jako *uniform* proměnnou. Výslednou pozici ukládá jak do `gl_Position` tak do vstupu fragment shaderu. Neboť výšky jsou v tomto bodě záporné, tak je výsledná barva (její první komponenta *red*), nastavena následovně:

$$red = 1 - pos_z \tag{4.4}$$

Tím je zápis do textury dokončen a nyní uchovává mapu modelu.

4.3.3 Výsledné nastavení

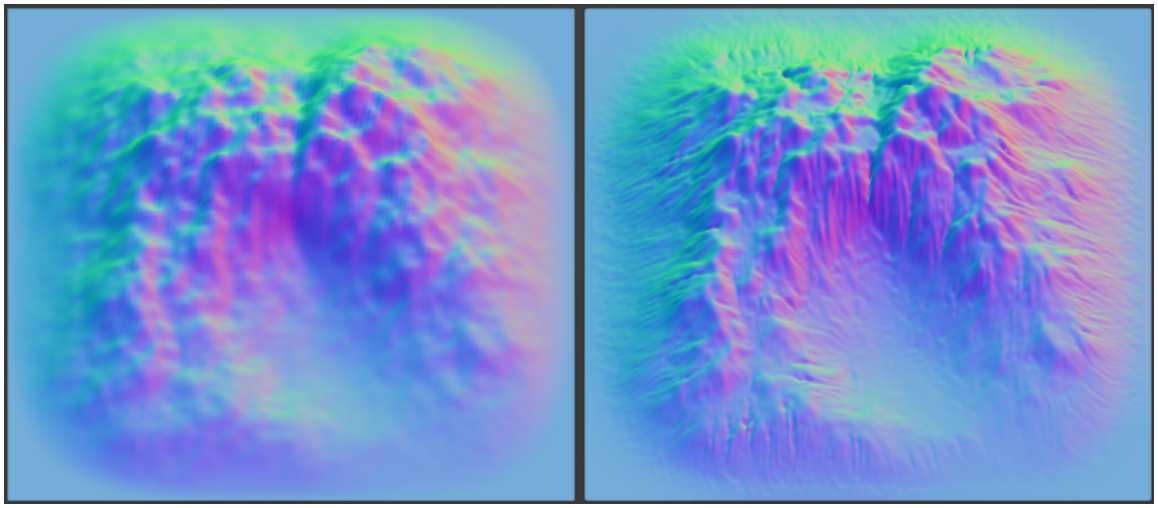
Po vytvoření samotné textury výškové mapy je třeba ji ještě uložit do entity. Základní mapa objektu je uvolněna a na její místo je dosazena nová textura. Neexistuje-li základní mapa pro entitu před nastavením, pak dojde i k uvolnění zdrojové mapy a nahrazením kopií nové textury.

4.4 Eroze

Hlavní funkcionalitou rozšíření je právě eroze, jež je v samostatných modulech rozdělena na erozi termální a vodní. Oba typy mají stejnou strukturu, jež se dělí na tři funkce:

1. Příprava eroze
2. Provedení
3. Uložení a úklid

[2] Ukázka práce s objektem: https://moderngl.readthedocs.io/en/latest/reference/scope.html



Obrázek 4.3: Ilustrace výsledku vodní eroze. Vlevo je původní povrch, vpravo je výsledná geometrie po erozi.

Vstupem všech tří funkcí je buď objekt, nebo obrázek, na které se eroze bude aplikovat. Zbytek potřebných dat je přístupný buď globálně, nebo skrze tyto entity. Důvodem rozdělení na tři funkce je potenciální rozšíření na responzivní uživatelské rozhraní, kdy fáze *provedení* by byla volána několikrát, v současném stavu ale tato funkcionalita není implementována.

Společná pro přípravné funkce obou typů je kontrola existence *modelové* mapy odkazované objektem. Není-li nalezena, pak ještě nebyla vytvořena, nebo je neplatná. V tom případě je voláno generování výškové mapy probrané v sekci 4.3. Součástí inicializace je také tvorba textur a jejich uložení do globální proměnné `common.data`, neboť musí přetrvat napříč zmíněnými funkcemi.

Společný výstup obou modulů je přímé nastavení *současné* mapy entity a případné uvolnění přepisované textury. Následující podkapitoly zacházejí do detailu jednotlivých algoritmů eroze.

4.4.1 Vodní eroze

Model vodní eroze použitý v rozšíření je založený na simulaci částicemi. Tato kapitola se bude zaměřovat na odlišnosti oproti typu simulace popsaném v teoretické části. Tato kapitola se nejprve zaměří na vstupy modelu a až dále na proces simulace včetně napojení na rozšíření. Příklad výsledku vodní eroze ilustruje obrázek 4.3.

Vstupy modelu

Simulační algoritmus byl obohacen o další výstupy týkající se procesu eroze, zavádí tedy několik dalších nastavení. Veškeré konstanty modelu jsou nastavitelné externě jako *uniform* proměnné:

- Počet iterací (životnost kapky)
- Síla eroze

- Síla depozice
- Maximální rychlost
- Odpor
- Faktor kapacity
- Kontrast eroze
- Kontrast depozice
- Síla zbarvení

Veškeré hodnoty zde **musí být kladné**. Pro síly eroze a sedimentace platí, že musí být menší než 0,5, Vyšší hodnoty mohou způsobit oscilaci povrchu a nestabilitu eroze. Dále odpor je zde **násobitel rychlosti**, tedy platí $drag <= 1$. Má-li kapka například v každé iteraci ztratit 20 % rychlosti, musí být proměnná odporu nastavena na 0,8.

Pro faktor kapacity platí, že by měl být řádově v setinách, naopak kontrasty jsou řádově v desítkách. Síla zbarvení je znovu násobitel, tedy menší než 1. Hodnota maximální rychlosti může být v podstatě libovolná a její základní hodnota je nastavena na 2. Zrychlení nápodobně může mít i vyšší hodnoty, ale nejlépe by měla být menší než 1. Porovnání vlivu různých vstupů na výslednou mapu lze vidět v ilustraci 4.4.

Vstupní textury algoritmu jsou následující:

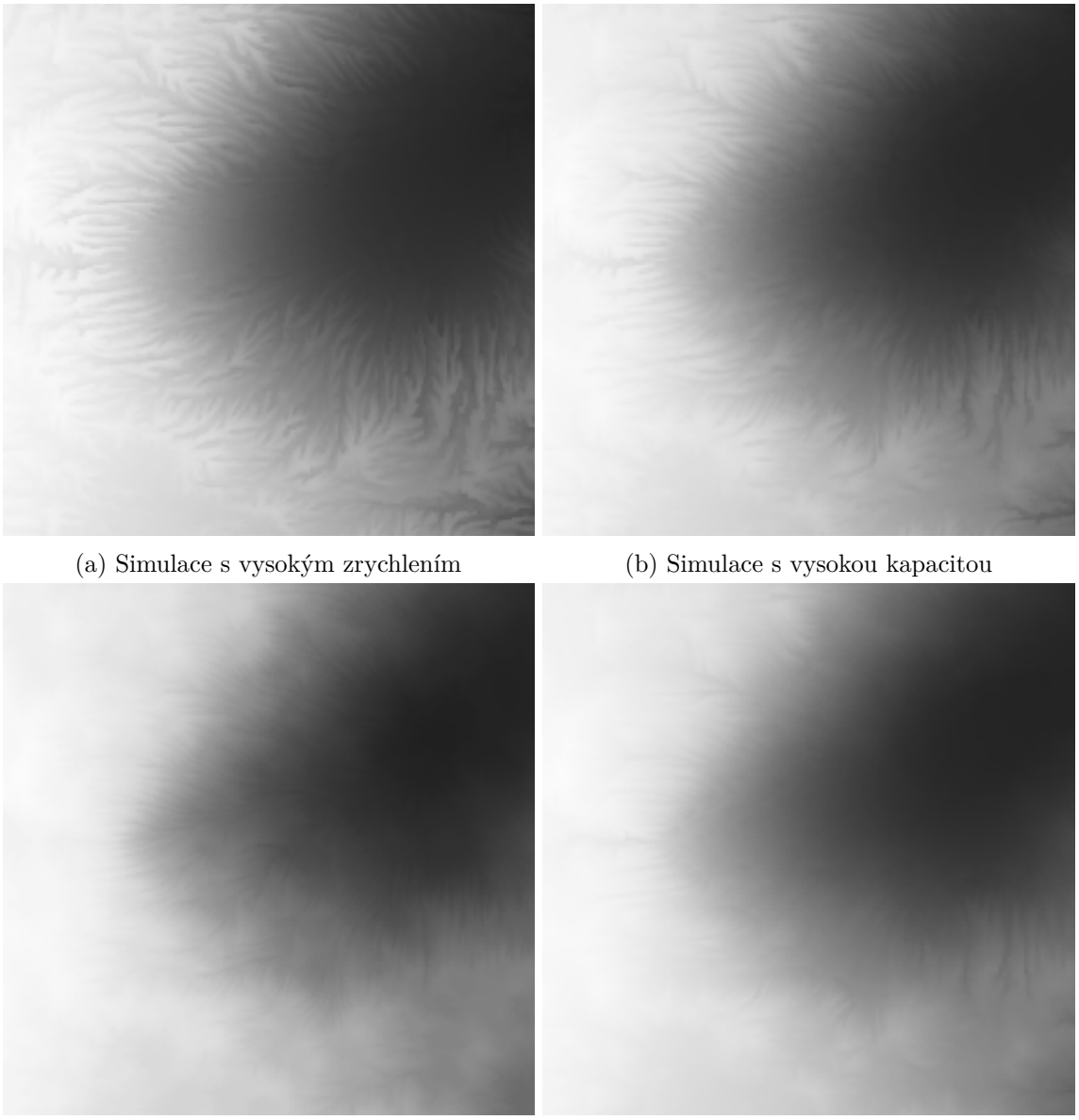
- Výšková mapa
- Hloubková mapa (množství eroze)
- Mapa sedimentu
- Mapa barev

Tyto textury jsou typu `r32f`, tedy s pohyblivou řádovou čárkou a jedním kanálem, až na barevnou texturu, jež má čtyři. Společně s těmito texturami jsou ještě definovány dva pravdivostní vstupy — povolení zápisu barvy a povolení zápisu sedimentace či množství eroze. Není-li tedy požadován odpovídající výstup uživatelem, pak není ani simulován. V případě zákazu přesunu barvy se pak jedná o znatelné zrychlení algoritmu.

Simulace

První fází simulace je inicializace proměnných kapky. Každá částice začíná s nulovou rychlostí, kapacitou a nasycením. V případě transportu barev je také načtena barva z pixelu v počáteční pozici. Pak následuje cyklus života kapky, jenž je limitovaný vstupním počtem iterací. Postup algoritmu je následující:

1. Zrychlení částice
2. Normalizace rychlosti



(a) Simulace s vysokým zrychlením

(b) Simulace s vysokou kapacitou

(c) Simulace s krátkou životností kapek

(d) Simulace s vysokou sílou eroze

Obrázek 4.4: Příklady vlivu různých parametrů na výslednou výškovou mapu.

3. Eroze a sedimentace
4. Posun částice
5. Kontrola polohy

V každé iteraci jsou na začátku získány výšky sousedních buněk ve čtyřokolí. Pro účely dalších výpočtů je ještě zjištěno jejich minimum a maximum. U výšek okolních buněk nedochází k interpolaci, počítá se pouze se zaokrouhlením současné pozice kapky na nejbližší texel. Stejně je získána i výška přímo pod kapkou.

Proces zrychlení kapky je pak odlišný od modelu zmíněného v teoretické části 2.7. Náklon povrchu je uvažován **jen ve směru pohybu** kapky a to v nezávisle v osách X a Y. Pro výšku kapky h na souřadnicích x, y, rychlost v a konstantu zrychlení a je rychlost přepočítána dle algoritmu 4.

Algorithm 4: Výpočet zrychlení částice

if $v_x > 0$ *and* $(h - h_{+x}) \leq drop_{max}$ **then**
  $v_x \leftarrow v_x + a \cdot (h - h_{+x})$
else if $(h - h_{-x}) \leq drop_{max}$ **then**
  $v_x \leftarrow v_x - a \cdot (h - h_{-x})$

if $v_y > 0$ *and* $(h - h_{+y}) \leq drop_{max}$ **then**
  $v_y \leftarrow v_y + a \cdot (h - h_{+y})$
else if $(h - h_{-y}) \leq drop_{max}$ **then**
  $v_y \leftarrow v_y - a \cdot (h - h_{-y})$

if $h < min_h$ **then**
  $v \leftarrow 0$

Je-li skok mezi současnou a následující buňkou příliš velký, pak rychlost zůstane nezměněna. To zabraňuje tvorbě artefaktů na okrajích nebo vedle prázdných prostor textury. Velikost přijatelného sestupu $drop_{max}$ kapky je nastavitelná uživatelem.

Závěr algoritmu zastaví kapku, nachází-li se v prohlubni, tedy je-li současná výška nižší než celé okolí. Tento krok omezuje vytváření prohlubní pod strmými povrchy při silněji nastavených vstupech.

Následujícím krokem je nastavení kapacity a normalizace rychlosti, aby kapka nepřeskočila texely při pohybu. Nulová rychlost je ponechána jako nulová. Pro maximální rychlost v_{max} a faktor kapacity f_c je postupováno dle algoritmu 5.

Algorithm 5: Normalizace rychlosti a výpočet kapacity

$capacity \leftarrow f_c \cdot |v|$

if $|v| \neq 0$ **then**
  $v \leftarrow v_{max} \cdot v \ / |v|$

Eroze a sedimentace

Stejně jako v modelu z teoretické části, následující krok eroze a sedimentace se odvíjí od nasycení a kapacity. Je-li kapacita větší než nasycení, pak dojde k erozi, jinak je přebytek upuštěn na terén. Síla eroze a sedimentace zde pouze násobí množství materiálu, jež je takto přesunuto. Dále oba procesy mají přídavnou podmínku. Sedimentace **nemůže proběhnout**, pokud je současný bod výše než zbytek okolí. Množství uloženého sedimentu je také omezeno do výšky nejvyššího souseda. Nápodobně eroze nemůže prohloubit bod pod minimum okolí.

Sedimentace probíhá pouze přímo v bodě pod částicí (zaokrouhleným na nejbližší pixel), jak bylo zmíněno v kapitole 2.7. Množství sedimentu je v této fázi také potenciálně přičteno do textury depozice. Dané množství je násobeno konstantou kontrastu, aby byla změna viditelná.

Eroze je naopak aplikovaná na nejbližší čtyři pixely ke kapce a relativní vliv je získán interpolací mezi buňkami. Pro polohu kapky pos, požadované množství materiálu m a minimum okolí h_{min} je eroze provedena dle algoritmu 6. Výstupy algoritmu jsou také množství erodovaného materiálu amt v jednotlivých buňkách.

Algorithm 6: Interpolace eroze

$corner \leftarrow pos - (0.5, 0.5)$ $f \leftarrow corner - \lfloor corner \rfloor$ // Souřadnice v rozmezí $[0, 1)$

if $h > h_{min}$ **then**
  // Množství zvětraného materiálu pro danou buňku
  $amt \leftarrow m \cdot (1 - f_x) \cdot (1 - f_y)$
  $h \leftarrow h - amt$

// Soused v kladném směru osy X
if $h_{+x} > h_{min}$ **then**
  $amt_{+x} \leftarrow m \cdot f_x \cdot (1 - f_y)$
  $h_{+x} \leftarrow h_{+x} - amt_{+x}$

if $h_{+y} > h_{min}$ **then**
  $amt_{+y} \leftarrow m \cdot (1 - f_x) \cdot f_y$
  $h_{+y} \leftarrow h_{+y} - amt_{+y}$

if $h_{+xy} > h_{min}$ **then**
  $amt_{+xy} \leftarrow m \cdot f_x \cdot f_y$
  $h_{+xy} \leftarrow h_{+xy} - amt_{+xy}$

Proměnná $corner$ v algoritmu 6 slouží k získání nejbližší čtveřice buněk ke kapce. Zaokrouhlením pozice kapky by byla získána buňka, ve které se nachází. Zaokrouhlením pozice posunuté o polovinu buňky je získán nejnižší střed, jež je ke kapce blíže než 1 v obou souřadnicích.

Neboť vzdálenosti buněk jsou vždy rovny jedné, tak následným odečtením lze přímo získat váhy dané čtveřice buněk. Podmínečné vyhodnocení jednotlivých sousedů je dále

znovu za účelem, aby nemohla kapka vytvářet díry do povrchu. K nasycení je následně přičtena suma zvětraného množství materiálu, tedy:

$$saturation \leftarrow saturation + amt + amt_{+x} + amt_{+y} + amt_{+xy} \tag{4.5}$$

Závěr iterace přičítá normalizovanou rychlost k pozici kapky. Zde je také kontrolována pozice částice, jestli není záporná. Pokud ano, cyklus je rovnou ukončen. Důvodem k této kontrole je transport barev, jež je rozebrán dále. Při záporných hodnotách pozice může docházet k tvorbě vizuálních artefaktů na okraji textury.

Volání z rozšíření

Algoritmus eroze byl ponechán jako fragment shader, neboť přepsání na výpočetní shader nevedlo ke znatelnému zrychlení. Při každém volání funkce *provedení* musí být znovu vytvořeno VAO. Je-li ponecháno v proměnné zatímco aplikace Blender znovu převezme kontrolu nad kontextem, pak při znovupoužití již nebude správně fungovat.

Než je možné začít vykreslení, je nejprve nutné mapovat správně hodnoty vstupů. Rozsahy prezentované uživateli byly nastaveny obecně do rozsahu $[0, 1]$, aby bylo rozhraní přívětivé. Tento rozsah je ale třeba transformovat na povolené hodnoty zmíněné výše ve vstupech algoritmu. V případě eroze se jedná pouze o násobení zmíněnými hodnotami a v případě odporu odečtení od hodnoty 1, aby s vyšší hodnotou správně klesal násobitel rychlosti.

Jako vertex shader algoritmu slouží soubor zvaný `identity.vert`, jež nijak nemění pozice vrcholů. Vstupní model se skládá ze dvou trojúhelníků, jež tvoří čtverec o souřadnicích od -1 do 1, tedy zabírající celé okno. Souřadnice ve zpracovávané textuře je získána přímo ze zabudované proměnné `gl_FragCoord`.

Neboť simulace **neřeší synchronizaci** kapek a časů zápisu nebo čtení textur, tak může dojít k **vadným přepisům** map. Aby se zmírnil vliv částic mezi sebou, tak simulace začíná vždy jen na podmnožině textury. V základním nastavení je kapka vytvořena na každém čtvrtém řádku a sloupci, tedy jedna na každý čtverec 4 pixelů. Velikost tohoto dělení je zde značena *subdiv*. Různé terény potřebují různé odstupy kapek a menší hodnoty vedou k rychlejší simulaci, hodnota je tedy nastavitelná uživatelem. Jako vstup algoritmu tedy figuruje dvourozměrný vektor posunu `off` a uvnitř kódu `GLSL` je počáteční pozice kapky počítána následovně:

$$base = off + subdiv \cdot gl\_FragCoord_{XY} + (0.5, 0.5) \tag{4.6}$$

Samotné volání vykreslení tedy probíhá ve třech cyklech dle algoritmu 7.

Algorithm 7: Cyklus vykreslení

```
for i from 0 to iterations − 1 do
    for y from 0 to subdiv − 1 do
        for x from 0 to subdiv − 1 do
            SetUniform(off, x, y) Render(TRIANGLES)
```

Kvůli implementaci jako fragment shader je také v přípravné funkci potřeba vytvořit FBO, do kterého se bude vykreslení provádět. Data ve výsledné textuře nemají žádný význam, pouze pro potenciální ladění, a v použitém kódu je výsledná barva nastavena na černou. Kvůli provádění vykreslování po částech jsou rozměry FBO děleny a zaokrouhleny nahoru, tedy pro rozměry výškové mapy w_H a h_H platí:

$$\begin{aligned} w_{FBO} &= \left\lceil \frac{w_H}{subdiv} \right\rceil \\ h_{FBO} &= \left\lceil \frac{h_H}{subdiv} \right\rceil \end{aligned} \tag{4.7}$$

Objekt FBO může přetrvat mezi funkcemi na rozdíl od vytvořeného VAO. Zakončení prováděcí funkce tedy také zahrnuje uvolnění VAO.

Uložení a úklid

Jak bylo zmíněno na začátku kapitoly, po simulaci eroze je nastavena současná mapa a případně je uvolněna existující textura. Mapy tvořené vodní erozí rozšíření označuje základním názvem `Particle 1`, pro rozlišení od automaticky tvořené mapy nebo termální eroze. Index nové textury se dále odvíjí od názvu zdrojové mapy.

Neboť výškové mapy jsou čistě pod správou rozšíření a není dovoleno jejich názvy měnit, tak odvození indexu lze jednoduše implementovat regulárním výrazem. V názvu zdroje je hledáno číslo na konci řetězce, tedy výrazem `\d+$`. Je-li nalezeno, pak je následně parsováno, inkrementováno a přidáno na konec nového názvu. Jinak je použito základní jméno pro typ eroze.

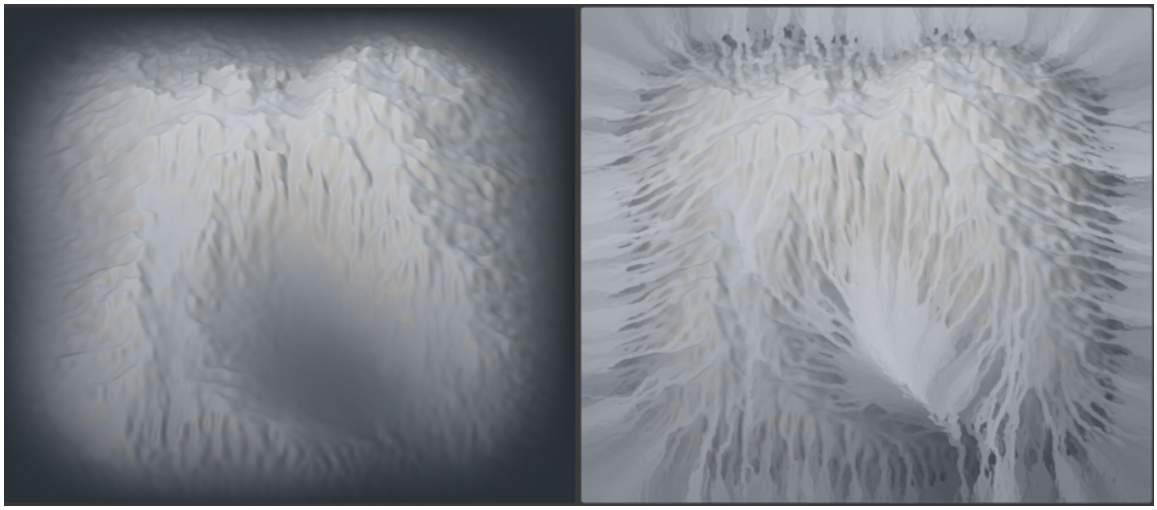
Mimo nastavení výškové mapy vodní eroze také vytváří až tři přídavné textury — hloubku, sedimentaci a transportovanou barvu. Další nakládání s těmito texturami po jejich vytvoření je už mimo kompetenci modulu, tudíž jsou vráceny jako seznam o pevné délce. Nejsou-li některé generovány, pak je na jejich pozici hodnota `None`. Finální kroky funkce pak spočívají v uvolnění FBO i `Scope` objektů.

4.4.2 Transport barev

Rozšíření jako součást vodní eroze také nabízí transport barev, ale jelikož textury poskytnuté uživatelem nejsou nějak omezeny rozlišením a mohou být zcela nezávislé na zvětrávaném objektu, tak jejich zpracování vyžaduje navíc krok překreslení. Ukázka transportu barev je v ilustraci 4.5.

Překreslení

Jak bylo zmíněno v generování výškové mapy, obrázky v aplikaci Blender vždy vrací čtyři kanály jako souvislý seznam pixelů. V případě barevné mapy tedy není třeba seznam filtrovat a je rovnou převeden na ModernGL texturu. Následně je vytvořen FBO o stejné velikosti, na kterém je překreslovací program spuštěn. Jako vertex shader algoritmu slouží znovu soubor zvaný `identity.vert`. Ten pro další fázi vykreslování vypočítává `UV` souřadnice z pozic vrcholů $pos_{x,y}$:



Obrázek 4.5: Ilustrace transportu barev. Vlevo je původní textura, vpravo je výsledek po transportu erozí. Výška povrchu je nezměněna.

$$\begin{aligned} u &= \frac{1}{2} pos_x + \frac{1}{2} \\ v &= \frac{1}{2} pos_y + \frac{1}{2} \end{aligned} \tag{4.8}$$

Mapování obrázku na texturu o správné velikosti a jeho interpolace je zde provedena automaticky díky OpenGL. Obrázek je předán jako typ `sampler2D`, ze kterého jsou následně funkcí `texture` na UV souřadnicích čteny pixely.

Podobně jako u výškové mapy musí být dále v implementaci řešen barevný prostor obrázku. Veškeré textury vytvořené rozšířením mají lineární barevný prostor a využívají čísla s pohyblivou řádovou čárkou. Při použití sRGB obrázku jako vstupu by výsledek byl znatelně tmavší po zpětném převodu. V tomto případě je provedena linearizace jednotlivých kanálů rovnou při mapování obrázku. Proces je tedy stejný jako u výškové mapy v algoritmu 2.

Simulace

Transport barev je v kódu eroze implementován tak, že každá částice uchovává svou aktuální barvu jednoduše reprezentovanou vektorem. Její hodnota je inicializovaná na barvu texelu, kde částice začíná. Dojde-li v iteraci k erozi materiálu, pak je přímo pod kapkou získána barva povrchu z textury a ta je přičtena k současné barvě. Množství, jež je nahrazené, ovládá vstupní proměnná, zde dále značená f_{col}. Výpočet nové barvy částice je následující:

$$color = f_{col} \cdot map_{col}\,(pos) + color \cdot (1 - f_{col}) \tag{4.9}$$

Barva je při erozi snímána jen z nejbližšího texelu. Naopak při sedimentaci dochází k interpolaci mezi sousedními buňkami. Proces transportu barev se touto volbou liší od procesu eroze, kde dochází k sedimentaci pouze přímo pod částicí. Oblasti naneseného materiálu

a míchané barvy tedy nebudou odpovídat. Přístup byl ale zvolen pro vytvoření jemnější výsledné textury.

Proces nanesení barvy je podobný erozi okolí zmíněné výše v algoritmu 6. V algoritmu 8 jsou nejdříve získány váhy okolních texelů a následně je vypočítán vliv na jednotlivé buňky, zde značený jako m. Množství nahrazené barvy f_{col}, jež je stejné jako pro zbarvení kapky, je pak vlivem násobeno.

Algorithm 8: Interpolace barvy

$corner \leftarrow pos - (0.5, 0.5)$
$f \leftarrow corner - \lfloor corner \rfloor$ // Souřadnice v rozmezí $[0, 1)$

$m \leftarrow f_{col} \cdot (1 - f_x) \cdot (1 - f_y)$
$map \leftarrow map \cdot (1 - m) + color \cdot m$

$m_{+x} \leftarrow f_{col} \cdot f_x \cdot (1 - f_y)$
$map_{+x} \leftarrow map_{+x} \cdot (1 - m_{+x}) + color \cdot m_{+x}$

$m_{+y} \leftarrow f_{col} \cdot (1 - f_x) \cdot f_y$
$map_{+y} \leftarrow map_{+y} \cdot (1 - m_{+y}) + color \cdot m_{+y}$

$m_{+xy} \leftarrow f_{col} \cdot f_x \cdot f_y$
$map_{+xy} \leftarrow map_{+xy} \cdot (1 - m_{+xy}) + color \cdot m_{+xy}$

4.4.3 Termální eroze

Algoritmus termální eroze je rozdělen na dvě části, které jsou implementovány skrze výpočetní shadery. Na rozdíl od vodní eroze simuluje erozi pomocí celulárních automatů.

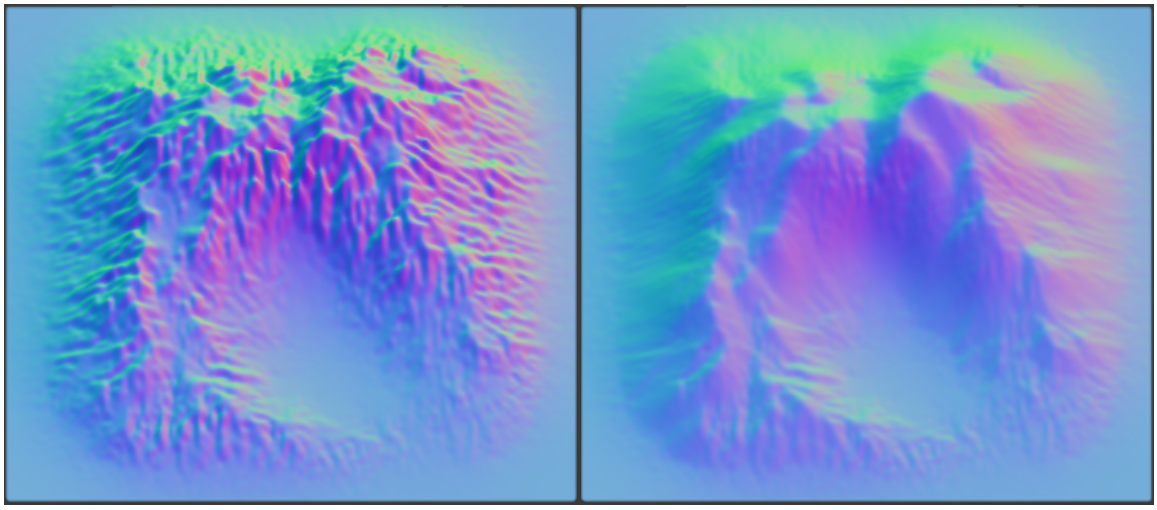
Mezi fázemi výpočtu je předávána pomocná textura. Zde je uchováváno množství, jež každá buňka vyžaduje od svého okolí, nebo jež je schopná poskytnout. Na základě dané textury je pak každá buňka schopna nezávisle vypočítat svou novou výšku bez ztráty materiálu. Hodnoty v této textuře jsou dále označovány jako *žádosti*. Ukázka termální eroze je v ilustraci 4.6.

Fáze výpočtu

V této fázi jsou prozkoumány sousední buňky a následně je vyplněna textura žádostí. V dané implementaci je uvažováno pouze čtyřokolí, textura tedy obsahuje čtyři kanály.

Prvním krokem je získání množství materiálu, jež by mělo být předáno okolí, zde značeno *supply*, a jež by mělo být sesunuto na zkoumanou buňku, značeno *demand*. Pro výšku buňky h a konstantu T_α, jež specifikuje maximální náklon povrchu jako první derivace výšky, je proces zkoumání okolí ilustrován algoritmem 9.

Výraz $T_\alpha \cdot l_n$ v tomto algoritmu určuje maximální rozdíl výšek, při kterém bude náklon povrchu menší nebo roven sypnému úhlu. Dále vytvořený vektor *demand* obsahuje pouze



Obrázek 4.6: Ilustrace simulace termální eroze. Vlevo je původní povrch, vpravo je výsledek eroze.

kladná čísla a vektor *supply* je naopak zcela záporný. Proměnné mx a mn tedy určují největší rozdíly výšek nahoru a dolů.

Algorithm 9: Prozkoumání okolí

$supply \leftarrow [0, 0, 0, 0]$
$demand \leftarrow [0, 0, 0, 0]$

for n *in neighbors* **do**
  if $|h_n - h| > T_\alpha \cdot l_n$ **then** // l_n je vzdálenost k buňce
    if $h_n > h$ **then**
      $demand_n \leftarrow h_n - h - T_\alpha \cdot l_n$
    else
      $supply_n \leftarrow h_n - h + T_\alpha \cdot l_n$

$mx \leftarrow max\,(demand)$
$mn \leftarrow min\,(supply)$

Hodnoty mn a mx jsou považovány za množství materiálu, jež je buňka maximálně schopná ztratit či získat. Vytvořené vektory jsou následně škálovány tak, aby dané množství přerozdělily mezi sousední buňky. Tento postup specifikuje algoritmus 10, jež dále využívá konstantu f_s, která specifikuje sílu eroze. Funkce `Clamp`, jež zde figuruje, omezí vstup na specifikované hranice. Je-li vstup roven `NaN`, k čemuž by došlo v případě nulových vektorů, pak je výstup roven nižší hranici. Komponenty výsledného vektoru jsou **záporné**, má-li být materiál přemístěn **do sousední buňky**. V případě **kladné** hodnoty dojde k přesunu **do zkoumané buňky**.

Algorithm 10: Výpočet žádostí

$C_d \leftarrow f_s \cdot mx \div \sum_{d \in demand} d$
$C_d \leftarrow$ `Clamp(`C_d`, 0, 1)`

$C_s \leftarrow f_s \cdot mn \div \sum_{s \in supply} s$
$C_s \leftarrow$ `Clamp(`C_s`, 0, 1)`

$request \leftarrow supply \cdot C_s + demand \cdot C_d$

Fáze přesunu

Tato fáze je spuštěna až po celkovém dokončení předchozí fáze. Každá buňka zde nezávisle přepočítá a uloží svoji výšku. Část tohoto procesu je ilustrována v algoritmu 11. Proměnná $r_{A \to B}$ zde značí hodnotu žádosti ze směru A směrem k B. Vektor žádostí zkoumané buňky je dále značen $request$ a její výška h.

V algoritmu dochází k porovnávání poskytovaného a požadovaného materiálu. Přičítána a odečítána je pak vždy nižší z daných hodnot, aby bylo zachováno maximální přesunutelné množství, jež si každá buňka stanovila. Daný proces je opakován pro všechny sousedy.

Algorithm 11: Přesun materiálu z levého souseda

```
if r_{L→R} < 0 then // Soused poskytuje materiál
    if −r_{L→R} > request_L then // Žádosti mají opačné znaménko
        h ← h + request_L
    else
        h ← h − r_{L→R}
else // Soused vyžaduje materiál
    if −r_{L→R} < request_L then
        h ← h + request_L
    else
        h ← h − r_{L→R}
```

Volání z rozšíření

Jak bylo zmíněno v úvodu kapitoly, programy jsou implementovány jako výpočetní shadery. Vykreslování vyžaduje tři textury — výškovou mapu, mapu žádostí a volnou texturu, jež se s výškovou mapou střídá.

Výpočet je možné provádět jak pro osy X a Y, tak pro diagonály. Uživatel je schopný směr eroze nastavit, včetně možnosti střídání směrů v každé druhé iteraci. Aby byl správně zachován sypný úhel, tak jsou délky mezi buňkami pro diagonály násobeny odmocninou ze dvou. Dané délky jsou programu předávány jako vstupní proměnné a jsou rozlišeny na osy X a Y (případně na hlavní a vedlejší diagonálu).

Algoritmus také respektuje poměr stran objektu, jež může být jiný než poměr stran textury. Délka mezi buňkami na ose X je vždy nastavena na hodnotu 1 a pro osu Y je využit poměr stran d_{xy} vypočítány při vytvoření výškové mapy.

Funkce výškové mapy také vytváří vertikální poměr stran vzhledem k ose X značený d_z. Ten je brán v potaz při převodu sypného úhlu na vstupní konstantu T_α. Pro šířku textury $width$ a sypný úhel α ve stupních je výpočet dán vzorcem 4.10. Specifikace poměrů stran je uvedena ve vzorci 4.3 v kapitole generování výškové mapy.

$$T_\alpha = \frac{\tan\left(\frac{\pi}{180} \cdot \alpha\right)}{width \cdot d_z} \tag{4.10}$$

4.5 Aplikování výsledků

Tato kapitola se zabývá nanesením výsledné výškové mapy po erozi na vstupní objekt, týká se tedy pouze práce s objekty. Při erozi obrázků je uživateli jenom zobrazena výsledná textura.

První a společnou částí aplikování textury je převod na rozdíl mezi výslednou mapou a původním modelem. Při zpětném nanesení textury tak bude odpovídat výsledné textuře. Operace je prováděna klonováním textury a použitím výpočetního shaderu nad tímto duplikátem. Výsledná mapa reprezentuje nulový rozdíl hodnotou 0,5. Maximální záporný rozdíl je zakreslen černou barvou a největší kladný rozdíl bílou.

Dále zmiňované metody nanesení textury jsou limitovány na osu Z, neboť posun v textuře je pouze výškový. Bez tohoto nastavení by došlo k deformaci dle normály povrchu, jež může mít libovolný směr. Také by mohlo docházet k vytváření převisů a dalších artefaktů.

4.5.1 Modifikátory

Jednou z možností deformace povrchu je takzvaný `Displacement` modifikátor. Výsledek eroze je díky modifikátoru okamžitě vyhodnocen a je možné změny povrchu kombinovat s dalšími modifikátory, jako je například teselace nebo vyhlazování geometrie. Jako vstup vyžaduje Blender texturu, do které lze nastavit vytvořený obrázek. Implementovaná funkce vytvoření modifikátoru se tedy skládá z následujících kroků:

1. Použitá textura je zapsána do obrázku, čímž je i aktualizován
2. Je získán seznam modifikátorů
   (a) Je-li seznam prázdný, pak je generován nový `Displacement` modifikátor
   (b) Není-li poslední prvek vytvořený rozšířením, pak je generován nový modifikátor
   (c) Jinak je vybrán poslední prvek seznamu
3. Pokud je již vytvořena Blender textura, tak je vybrána
4. Jinak je textura vytvořena a napojena
5. Textuře je nastaven vytvořený obrázek
6. Modifikátoru jsou nastaveny hodnoty
7. Pokud již existuje pomocný objekt, pak je vybrán
8. Jinak je vytvořen prázdný objekt

9. Pomocný objekt je transformován a nastaven modifikátoru
10. Pomocný objekt je nastaven jako potomek erodovaného objektu

Limitování modifikátoru na osu Z lze docílit přímo nastavením vlastnosti směru. Další vlastností jež musí být změněna je zdroj souřadnic. Vytvořenou Blender texturu není možné přímo přesouvat a škálovat, rozšíření toho tedy dosahuje pomocným objektem.

Nastavení transformace objektu je identické s výpočtem 4.1 při vytváření výškové mapy. Pozice objektu je nastavena dle středu obálky terénu a rozměry dle šířek ve vodorovné ploše. Osa Z objektu je nezměněna, neboť nemá na texturu žádný vliv.

Jelikož použitá textura je normalizována, pak musí být nastavena správná síla modifikátoru. Vertikální škálování objektu je uloženo ve vlastnosti zmíněné v rovnici výškové mapy 4.3. Díky převodu na rozdílovou mapu, jež změnší hodnoty na polovinu, je ale třeba brát dvojnásobek této vlastnosti, tedy:

$$strength = 2 \cdot d_z \tag{4.11}$$

4.5.2 Náhled

Náhled výškové mapy je pro objekty implementován také přes modifikátor. Rozdílem je sdílení textury a pomocného objekt, jež jsou stejné pro všechny náhledy. Při vytvoření nového náhledu je pomocný objekt přemístěn a přetransformován. Náhled výsledku eroze je ilustrován obrázkem 4.7.

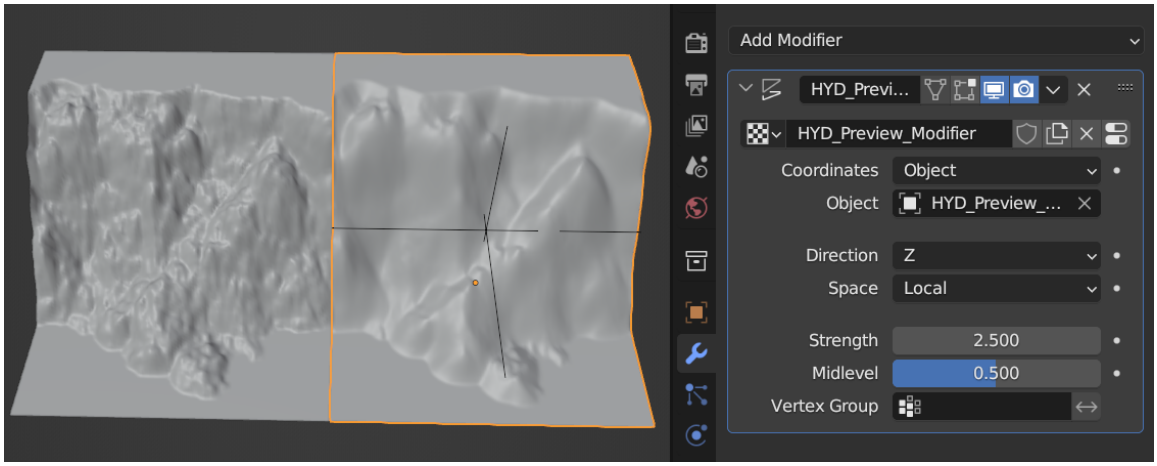
Pokud nahlížený objekt již má modifikátor vytvořený rozšířením, pak je zneviditelněn. Předpokladem je, že nahlížená eroze je nezávislá na předchozí operaci. Při rušení náhledu je naopak druhý modifikátor zpětně zviditelněn. V globálních datech rozšíření je uchováván poslední objekt, jež měl vytvořený náhled. Jeho jméno slouží ke smazání modifikátoru daného objektu při vytváření nového náhledu. Při přepsání sdílené textury by jinak došlo k neočekávaným změnám povrchu starého objektu.

Operace jež mohou současnou mapu objektu smazat automaticky náhled ruší. Při rušení je zcela smazán dočasný obrázek, pomocný objekt, modifikátor i Blender textura. Při odstranění je navíc ošetřeno potenciální přejmenování starého objektu, neboť uchovávané jméno by již nebylo platné. Je-li uchovávané jméno nesprávné, pak funkce prochází celý seznam objektů obsahujících náhledový modifikátor a následně jej odstraní.

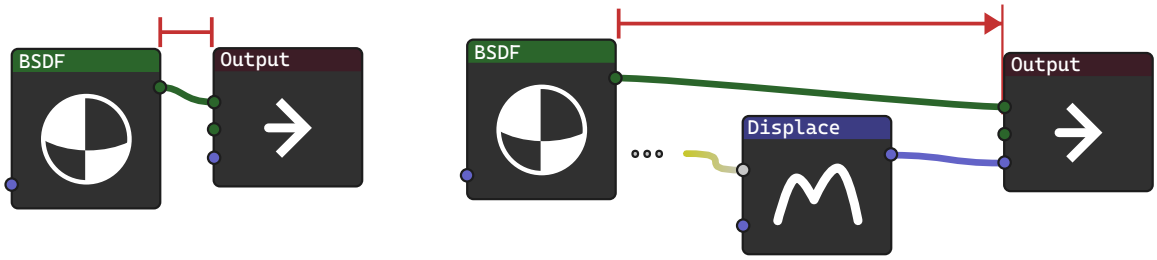
4.5.3 Materiály

Dalším způsobem nanesení eroze na objekt je využití materiálů. Pro tento účel jsou použitelné dva uzly, jež aplikace Blender nabízí, a to `Bump` mapování a `Displacement` mapování. Oba způsoby, jež jsou zde dále rozebrány, se pokouší o adaptovatelnost na uživatelem vytvořené materiály a jejich struktury.

Při implementaci byl také kladen důraz na vizuální jednoduchost výsledku. Společnou funkcí použitou pro oba způsoby je proto minimalizace uzlů materiálu. U vytvářených uzlů zabírajících velkou plochu nebo s mnoha nezapojenými vývody, jako je například uzel texturových souřadnic, jsou dané vývody schovány. Tento proces spočívá v nastavení vlastnosti



Obrázek 4.7: Příklad modifikátoru a náhledu eroze. U pravého terénu je vidět pomocný objekt jako černý kříž.



Obrázek 4.8: Ilustrace přesunu uzlů v materiálu. Výstupní prvek je posunut vpravo aby bylo místo pro nové prvky.

`hide` jak uzlu samotného, tak všech jeho vstupů a výstupů. Již zapojené vlastnosti zůstanou automaticky viditelné.

Za účelem úklidu modifikovaného materiálu jsou také existující uzly přesouvány, aby pro nové prvky bylo místo. V případě displacement mapování je posunut výstup materiálu vpravo. Posun ilustruje obrázek 4.8.

Dalším důležitým bodem jsou textury materiálu. Jako zdroj jejich mapování jsou napojeny generované souřadnice, jež pochází z uzlu `Texture Coordinates`. Důvodem implementace je kompatibilita s obecnými objekty a s terény rozšíření A.N.T. Landscape. Dané souřadnice odpovídají těm použitým při tvorbě výškové mapy, výsledek je tedy správně mapován na terén, i když nemá definovanou UV mapu. Například ve zmíněném rozšíření by jinak docházelo ke zobrazení pouze prvního pixelu textury napříč celým objektem.

Bump mapování

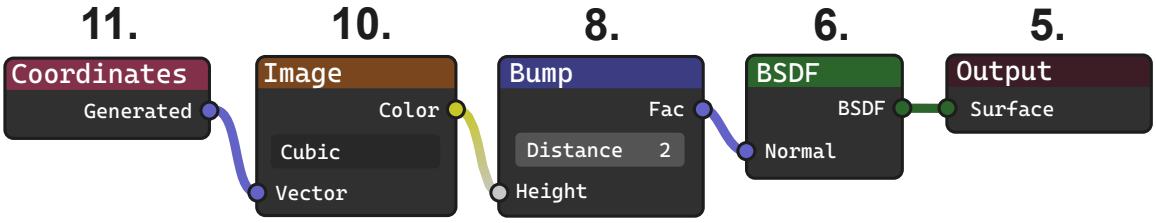
Tento způsob spočívá v užití `Bump` uzlu, jež musí být napojen jako vstupní normála libovolného `BSDF` uzlu, jež je podporuje. Výsledná deformace povrchu je pouze vizuální a funguje

jak pro Cycles, tak pro EEVEE vykreslování. Postup této metody se skládá z následujících kroků:

1. Použitá textura je zapsána do obrázku, čímž je i aktualizován
2. Nemá-li objekt materiál, pak je vytvořen
3. První materiál v seznamu je vybrán
4. Proces je ukončen, je-li vytvořený `Bump` uzel již v materiálu
5. Nemá-li materiál výstup, pak je vytvořen
6. Nemá-li shader, pak je přidán `Principled BSDF` a napojen
7. Proces je ukončen, má-li BSDF jiný vektorový vstup než `Bump`
8. Pokud neexistuje, pak je vytvořen `Bump` uzel a napojen
9. Proces je ukončen, má-li `Bump` uzel zapojený vstup
10. Je vytvořena textura a napojena
11. Textura je napojena na generované souřadnice

Dané kroky jsou také ilustrovány v obrázku 4.9. `Principled BSDF` uzel byl zvolen, neboť se jedná také o základní materiál tvořený aplikací. Uživateli také poskytuje mnoho možností jak materiál dále upravit. Zmíněné hledání a vytvoření výstupu se dále dělí na podkroky dle vyhodnocovaného materiálu. Blender dovoluje specifikovat, pro který vykreslovací engine má být použit. Pořadí hledání je následující:

1. Pro oba
2. Jen pro Cycles
3. Jen pro EEVEE



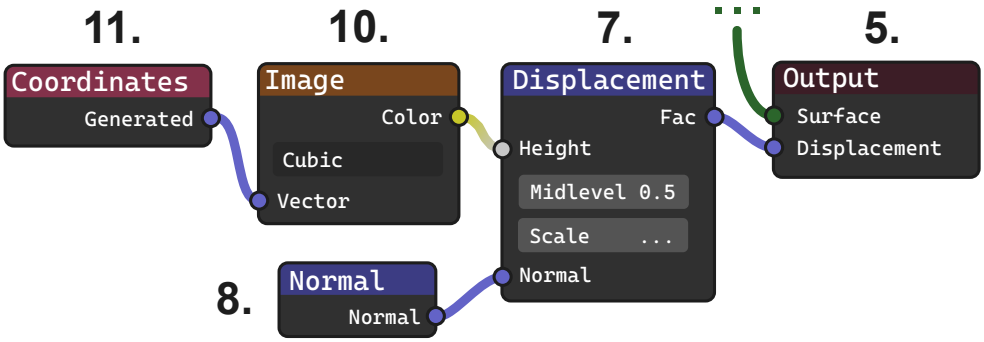
Obrázek 4.9: Ilustrace konfigurace pro bump mapování s očíslovanými kroky tvorby uzlů

Displacement mapování

Druhým přístupem skrze materiály je pomocí `Displacement` uzlu, jež se napojuje přímo na výstup materiálu. Oproti bump mapování nepotřebuje vytvářet `BSDF` uzel. Kroky, jež jsou dále zobrazeny v ilustraci 4.10, jsou následující:

1. Použitá textura je zapsána do obrázku, čímž je i aktualizován
2. Nemá-li objekt materiál, pak je vytvořen
3. První materiál v seznamu je vybrán
4. Proces je ukončen, je-li vytvořený `Displacement` uzel již v materiálu
5. Nemá-li materiál výstup, pak je vytvořen
6. Proces je ukončen, má-li výstup jiný vektorový vstup než `Displacement`
7. Je vybrán nebo vytvořen `Displacement` uzel
8. Pokud byl uzel vytvořen, pak je limitován na osu Z
9. Proces je ukončen pokud má `Displacement` uzel zapojený vstup
10. Je vytvořena textura a napojena
11. Textura je napojena na generované souřadnice

`Displacement` je konfigurován tak, aby byl posun vrcholů pouze vertikální, neboť textura eroze má pouze vertikální rozdíly výšek. To je implementováno skrze vektorový vstup uzlu. Zde je zapojen výstup prvku `Normal`, bez jakýchkoliv změn jeho nastavení. Přímé zadání vektoru směřující v ose Z je možné, ale nemá správný účinek.



Obrázek 4.10: Ilustrace konfigurace pro displacement mapování s očíslováním kroků tvorby uzlů

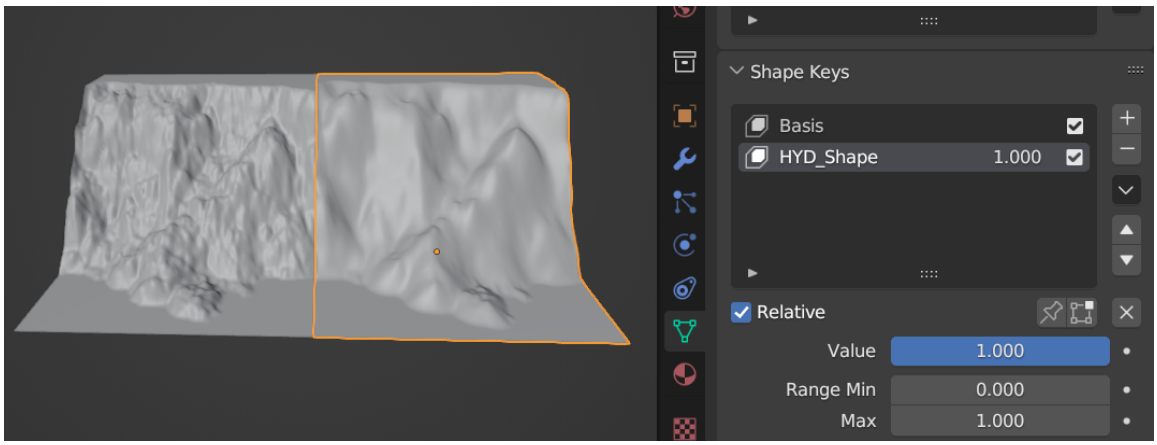
4.5.4 Přímé zapsání a Shape Keys

Přímý zápis eroze do modelu je nabízen v případě, kdy je zapnutý náhled eroze, nebo je vytvořený modifikátor. Zápis spočívá v tom, že je procházen seznam modifikátorů až po položku vytvořenou rozšířením a postupně jsou aplikovány. Je-li takových položek více,

pak je aplikována pouze první. Modifikátory vytvořené rozšířením se identifikují pomocí předpony `HYD`.

Podobným způsobem funguje využití shape keys. Zde je vybrán generovaný modifikátor ze seznamu a operací `modifier_apply_as_shapekey` z modulu `ops.object` je aplikován. Příklad výsledku je ilustrován v obrázku 4.11. Pokud již vrstva byla vygenerován, pak je nejdříve ostraněna. Použitý modifikátor je vyjmut ze seznamu. Po použití této operace Blender zabraňuje aplikování modifikátorů, výše zmíněný přímý zápis je poté tedy vypnut rozšířením, dokud vrstvy nejsou odstraněny.

Neboť přímé zapsání změní samotný model, tak současná výšková mapa je zároveň zapsána jako zdrojová i modelová. Aplikování jako Shape Key zapíše současnou mapu jako zdrojovou, ale ne modelovou, neboť seznam Shape Key stále zachovává původní model jako bázi.



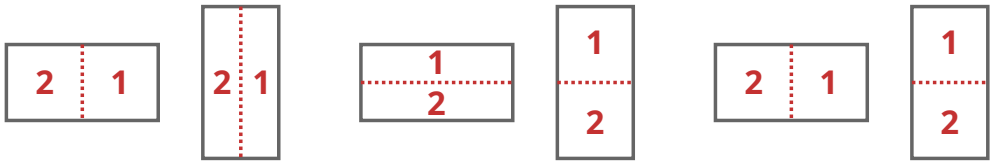
Obrázek 4.11: Ilustrace konfigurace pro Shape Keys. Vrstva vytvořená rozšířením je `HYD_Shape`. Pro hodnotu 1 má stejný výsledek jako modifikátor.

4.6 Navigace

V rozšíření je dán důraz na správné informování uživatele. Jsou-li výsledky zobrazitelné uvnitř aplikace, pak se snaží rozšíření k těmto výsledkům uživatele navigovat. Za tímto účelem bylo třeba implementovat funkci, jež vybere nebo vytvoří okno uvnitř aplikace.

Funkce výběru nebo vytvoření okna se snaží brát ohled na existující rozestavení rozhraní. Pro danou funkci bylo zvoleno pořadí výběru oken uvedené níže. K jednotlivým oknům aplikace lze přistoupit skrze seznam `screen.areas` uvnitř kontextu.

1. Existující okno správného typu, jež není aktivní
2. Rozdělení aktivního okna, je-li správného typu
3. Rozdělení okna 3D pohledu
4. Poslední okno v seznamu, jež není přehled scény nebo okno vlastností
5. Poslední okno v seznamu



(a) Horizontální dělení (b) Vertikální dělení (c) Dělení podle delšího rozměru

Obrázek 4.12: Možnosti rozdělení okna. Číslem 1 je značena část, kde zůstane původní okno. Nové okno bude v části 2.

Dělení okna záleží na nastavených preferencích rozšíření. Možnosti, které jsou ilustrovány v obrázku 4.12, jsou následující:

- Vždy vodorovně
- Vždy vertikálně
- Dle delšího rozměru okna

Rozdělení okna lze provést funkcí `area_split` modulu `ops.screen`, nejdříve je ale třeba dané okno **dočasně vybrat**. Toho je dosaženo voláním metody kontextu `temp_override` s nastaveným argumentem `area`. Okno vybrané uživatelem zůstane po ukončení procesu zachované.

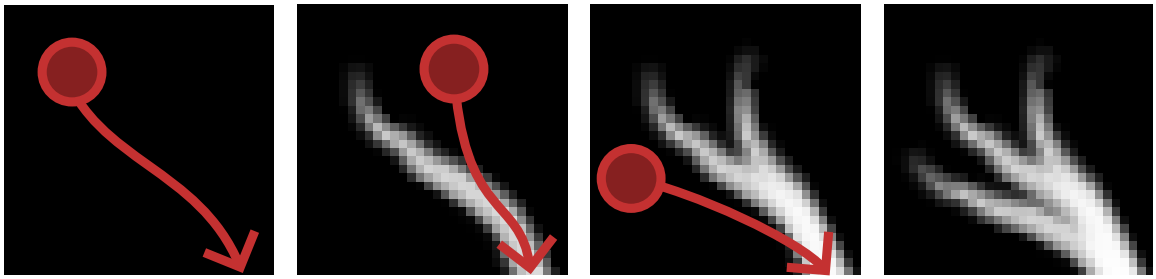
Nově vytvořené okno lze najít na konci seznamu poskytnutého kontextem. Typ okna lze přepínat nastavením vlastnosti `type` a v případě grafového editoru je ještě třeba nastavit podtyp, tedy vlastnost `ui_type`.

Ve vytvořeném okně nelze přímo nastavovat vlastnosti požadované jednotlivými typy navigace, nejprve je nutné přistoupit k první položce seznamu `spaces` okna, jež bude dále označována jako oblast. Tato položka **vždy existuje**.

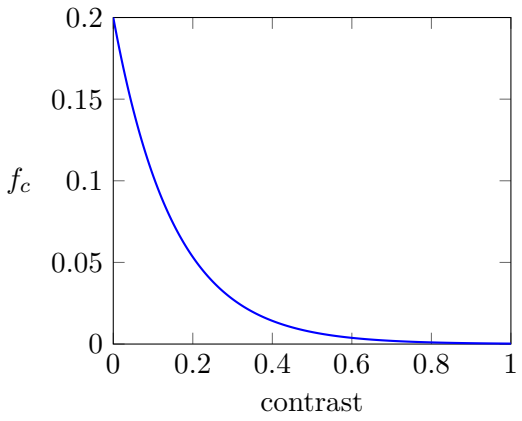
Při zobrazení generované krajiny je také přenastaven pohled uživatele, aby byl objekt viditelný. K vlastnostem kamery v 3D pohledu lze přistoupit skrze vlastnost `region_3d` oblasti. Pozice je nastavena přímo na pozici objektu. Z estetických důvodu byl zvolen izometrický úhel pohledu, perspektivní nastavení pohledu ale zůstává zachováno. Specificky byl zvolen pohled, jež svírá stejný úhel jako úhlopříčka krychle. Ten odpovídá rotaci $\frac{3}{\pi}$ na ose X a úhlu $\frac{\pi}{4}$ na ose Z. Úhly je možné zadat vytvořením objektu `Euler`. Rotace pohledu přijímá pouze quaterniony, jež lze následně vytvořit metodou `to_quaternion` daného objektu.

4.7 Mapa toků

Principem mapy toků je zakreslování tras pohybu kapek po povrchu. Pro výpočet pohybu kapek je využito stejného algoritmu 4 jako pro vodní erozi, nekontroluje ale maximální sestup kapek. Na rozdíl od eroze také neprovádí více iterací. Neboť algoritmus je pro dostatečné odstupy kapek deterministický, tak každá iterace by vytvářela stejnou mapu. Další odlišností je, že algoritmus je implementován skrze výpočetní shader.



Obrázek 4.13: Ilustrace zakreslování pohybu kapek. Červenou je zakreslena kapka a její trasa pohybu. Místa, kde se více tras překrývá, jsou světlejší.



Obrázek 4.14: Graf převodu uživatelské proměnné *contrast* na vstupní konstantu kontrastu f_c

Vzorec změny koncentrace je podobný jako u transportu barev. Cílem je, aby se pro libovolně velké množství částic výsledná hodnota pouze blížila jedné. Pro konstantu kontrastu f_c je vzorec nového toku w_{t+1} následující:

$$w_{t+1} = w_t \cdot (1 - f_c) + f_c \tag{4.12}$$

Zmenšení konstanty f_c vede k temnějšímu výsledku, užším tokům a většímu kontrastu. Konstanta je v uživatelském rozhraní reprezentována posuvníkem od 0 do 1, dané hodnoty ale ovlivňují výsledek nelineárně. Mezi uživatelem zvolenou hodnotou *contrast* a použitou konstantou f_c byl zaveden převod tak, aby stejný posun hodnoty vedl k podobně velké vizuální změně. Subjektivně, skrze testování hodnot, byl zvolen vzorec 4.13. Její graf je zobrazen v ilustraci 4.14.

$$f_c = 0{,}2 \cdot e^{-6{,}61 \cdot contrast} \tag{4.13}$$

Co se týče dalších vstupů algoritmu, jsou vyžadovány dvě textury — výšková mapa, jež je pouze čtena, a mapa toků, do které je přímo zapisováno. Parametry simulace jsou sdílené s erozí. Je-li tedy provedena vodní eroze, pak následující mapa toků bude simulovat stejný pohyb kapek. Simulace toků ale pracuje s vlastní mezerou mezi počátečními pozicemi, neboť je obecně vyžadována vyšší než u eroze.

Neboť pro vytvoření textury toků je vyžadována výšková mapa, tak je podobně jako u eroze dle potřeby automaticky vytvořena. V případě, že již objekt výškové mapy má, je hledána nejdříve *současná* mapa. Pokud neexistuje, bere algoritmus *zdrojovou* mapu.

Algoritmus pro velmi malé deformace povrchu může vytvořit nežádoucí černé pixely. Po generování mapy toků je tedy přidán krok zprůměrování černých pixelů s okolními buňkami. Krok je také implementován jako výpočetní shader přímo nad výslednou texturou.

4.8 Generování terénů

Při práci s obrázky rozšíření také nabízí možnost vygenerovat odpovídající krajinu jako objekt. Tento terén je následně umístěn na 3D kurzor. Daná operace může sloužit jak pro importované výškové mapy tak pro obrázky vygenerované z modelu uvnitř aplikace, neboť výsledný terén lépe zobrazí detail textury.

Geometrie vytvořená operací se skládá ze čtvercové mřížky na základě rozlišení vybrané textury, zde značené *width* a *height*. Jelikož vytvoření vrcholu pro každý pixel nemusí být třeba, tak je nabízen dělitel tohoto rozlišení *sub*. Výsledné rozlišení *res* je následně zaokrouhleno nahoru:

$$\begin{aligned} res_x &= \lceil width \div sub \rceil \\ res_y &= \lceil height \div sub \rceil \end{aligned} \tag{4.14}$$

Pro vytvoření terénu je využita funkce `ops.mesh.primitive_grid_add`, která přidá do scény objekt mřížky. Při volání této funkce je možné zadat rozlišení geometrie objektu. Funkce následně automaticky vybere vytvořený objekt, je tedy možné ho získat z kontextu aplikace. Objekt je přejmenován dle názvu vstupní textury a je mu nastavena vlastnost rozšíření `is_generated`. To zabraňuje erozi objektu pro případ, že by byl přepsán opětovným generováním terénu.

Krajina vytvářená rozšířením používá hladké stínování povrchu. Toho lze docílit nastavením vlastnosti `use_smooth` pro všechny polygony vytvořeného povrchu. Daný seznam je přístupný skrze vlastnost `polygons`.

Pro nanesení textury na vytvořený objekt je využit `Displace` modifikátor. Pro něj je vytvořena Blender textura a obrázek s výškovou mapu. Modifikátor je okamžitě aplikován a textura i obrázek jsou vymazány.

4.9 Instalace ModernGL

Neboť rozšíření vyžaduje externí balíček pro jazyk Python, je třeba vyřešit jeho instalaci. Využitím nástroje PIP, který jsou součástí instalace aplikace Blender, lze automaticky stáhnout i aktualizovat externí knihovny. Zde představený kód je převzatý od autora Roberta Gützkowa [4]. K instalaci knihovny slouží následující příkaz:

```
sys.executable -m pip install --upgrade moderngl
```

Příkaz je volán jako podproces pomocí knihovny `subprocess`. Je mu také nastaven argument prostředí, kde je povolen parametr `PYTHONNOUSERSITE`[3]. Díky němu instalace ignoruje uživatelské balíčky. Jinak by instalace neproběhla, pokud by už uživatel v systému ModernGL měl, přestože ve skriptech by Blender ke knihovně neměl přístup.

Pro daný postup je nejspíše třeba spuštění aplikace s vyššími oprávněními. Například základní instalce Blender ve Windows se nachází ve chráněné složce. Při nedostatečném oprávnění volání podprocesu vytvoří výjimku. Při jejím ošetření je pak v rozšíření nastaven text preferencí na patřičnou zprávu uživateli.

Po úspěšné instalaci je třeba aplikaci restartovat. Funkce knihovny jsou následně při normálním spuštění přístupné všem skriptům. Existující příklad podobného procesu instalace závislostí obsahuje zásuvný modul *DeepBump*[4].

4.9.1 Odinstalace

Nástroj PIP je také schopný odinstalace balíčků, ale při spuštění z aplikace Blender je zcela neodstraní, pouze změní prefix jejich složek. Odstranění balíčků je tedy možné pouze manuálně ze složky `site-packages` v adresářové struktuře aplikace Blender.

[3]Detail prostředí: https://docs.python.org/3/using/cmdline.html#envvar-PYTHONNOUSERSITE
[4]Popis instalace je uveden na stránce projektu: https://github.com/HugoTini/DeepBump

Kapitola 5

Měření a limitace

V této kapitole jsou představeny výsledky měření časové náročnosti jednotlivých operací rozšíření a následně zjištěné limitace implementovaného projektu.

5.1 Měření

Projekt byl testován na dvou grafických kartách — RTX 3080 a GTX 745. Pro měření byly algoritmy spouštěny mimo aplikaci Blender a vstupy byly předpřipravené textury. Pokud není jinak řečeno, pak je vodní eroze nastavena na 50 iterací a 50 kroků životnosti částice. Každé měření bylo zprůměrováno z 50 běhů.

Doba provedení simulace je uvažována od prvního volání ModernGL vykreslení do ukončení veškerého vykreslování, tedy po dokončení funkce `finish` ModernGL kontextu.

5.1.1 Velikost textury

Implementovaný algoritmus simuluje na každý pixel textury jednu kapku. Velikostí textury je tedy silně ovlivněna doba simulace. V tabluce 5.1 jsou uvedena měření celkové doby simulace v závislosti na velikosti textury. Odstup částic při simulaci byl nastaven na hodnotu 4. V tabulce 5.2 jsou doby přepočtené na jednu iteraci milionu částic.

Rozlišení textury	128 px	256 px	512 px	1024 px
RTX 3080 12GB	3 ms	9 ms	20 ms	72 ms
GTX 745 4GB	38 ms	172 ms	804 ms	3575 ms

Tabulka 5.1: Tabulka doby simulace 50 iterací. Odstup částic byl nastaven na 4 pixely.

Rozlišení textury	128 px	256 px	512 px	1024 px
RTX 3080 12GB	3,46 ms	2,75 ms	1,54 ms	1,38 ms
GTX 745 4GB	46 ms	52 ms	61 ms	68 ms

Tabulka 5.2: Tabulka doby jedné iterace pro milion částic. Odstup částic byl nastaven na 4 pixely.

Při simulaci na větší textuře je více času stráveného výpočtem na grafické kartě. RTX 3080 s větším množstvím paměti toho využívá a výpočet se pro pevný počet částic zrychluje. Starší karta GTX 745 naopak zpomaluje.

5.1.2 Interpolace

Pro vodní erozi je možnost interpolace eroze mezi buňkami v okolí částice. Každá iterace pak vyžaduje více kroků. Tabulka 5.3 ilustruje závislost celkové doby simulace na povolení interpolace pro různé velikosti textur.

Rozlišení textury		128	256	512	1024
RTX 3080 12GB	Bez interpolace	23 ms	38 ms	48 ms	203 ms
	S interpolací	23 ms	62 ms	83 ms	344 ms
GTX 745 4GB	Bez interpolace	72 ms	282 ms	1301 ms	-
	S interpolací	103 ms	407 ms	1915 ms	-

Tabulka 5.3: Tabulka celkové doby simulace 50 iterací. Odstup částic byl nastaven na 16 pixelů.

5.1.3 Odstup částic

V algoritmu vodní eroze je použitý odstup částic, aby se předešlo interferenci mezi nimi. Následující měření se zabývají vlivem velikosti odstupu na dobu simulace. Také byl měřen vliv odstupu pro různé velikosti textury. Tabulka 5.4 ilustruje měření pro rozlišení 256 pixelů a tabulka 5.5 pro rozlišení 512 pixelů. Tabulky jsou také zobrazeny v grafech 5.1 a 5.2.

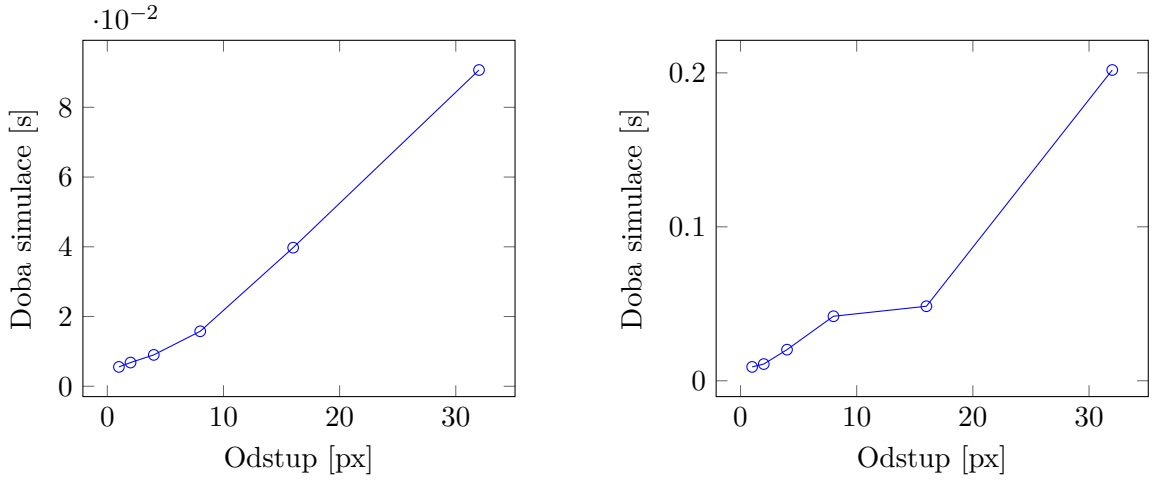
Odstup částic	1 px	2 px	4 px	8 px	16 px	32 px
RTX 3080 12GB	5,55 ms	6,80 ms	9 ms	16 ms	40 ms	90 ms
GTX 745 4GB	120 ms	142 ms	184 ms	236 ms	285 ms	334 ms

Tabulka 5.4: Tabulka celkové doby simulace. Rozlišení textury je 256 na 256 pixelů.

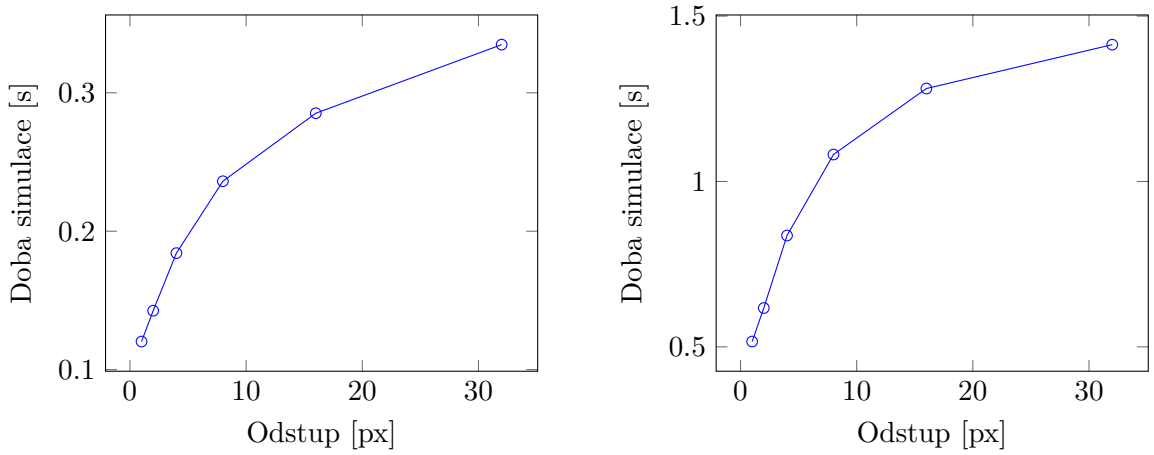
Odstup částic	1 px	2 px	4 px	8 px	16 px	32 px
RTX 3080 12GB	9 ms	11 ms	20 ms	42 ms	48 ms	202 ms
GTX 745 4GB	516 ms	617 ms	836 ms	1082 ms	1281 ms	1414 ms

Tabulka 5.5: Tabulka celkové doby simulace. Rozlišení textury je 512 na 512 pixelů.

Z měření je vidět, že pro větší odstupy se doba simulace drasticky zvyšuje. Čím větší je odstup částic, tím více volání algoritmu eroze je provedeno a tím více času je stráveného přepínáním kontextu mezi grafickou kartou a procesorem. Správný odstup kapek je tedy nejmenší, se kterým ale nedochází k interferenci kapek. Může tedy být pro každý porvch jiný.



Obrázek 5.1: Grafy závislosti doby simulace na velikosti odstupu pro kartu RTX 3080. Vlevo je graf pro texturu o rozlišení 256 a vpravo pro rozlišení 512.



Obrázek 5.2: Grafy závislosti doby simulace na velikosti odstupu pro kartu GTX 745. Vlevo je graf pro texturu o rozlišení 256 a vpravo pro rozlišení 512.

5.1.4 Termální eroze

Hlavním faktorem doby simulace pro termální erozi je velikost textury. V tabulce 5.6 je uvedeno měření závislosti celkové doby simulace na rozlišení vstupní výškové mapy.

Rozlišení textury	128 px	256 px	512 px	1024 px	2048 px
RTX 3080 12GB	1.80 ms	5.75 ms	19 ms	79 ms	318 ms
GTX 745 4GB	22 ms	85 ms	427 ms	1708 ms	11054 ms

Tabulka 5.6: Tabulka celkové doby simulace termální eroze po dobu 50 iterací.

5.2 Limitace

Při testování na operačním systému Fedora nebyla knihovna ModernGL schopna najít OpenGL kontext aplikace Blender. Vytvoření samostatného kontextu stále způsobilo pád aplikace, na operačním systému tedy není možné rozšíření použít.

Obecně pro operační systém Linux také knihovna ModernGL vykazuje neočekávané chování při indexaci vrcholů. Vykreslování možné je, ale dochází k zahazování polygonů v zásvislosti na přesném pořadí a pozici vrcholů. Ze všech permutací indexů pro každý trojúhelník obvykle funguje pouze jedna. Neboť jiné metody vykreslení, jež přidávají polygon s každým dalším vrcholem nejsou pro libovolný vstupní model použitelné, byla indexace u jiných operačních systémů než Windows vypnuta.

V implementovaném rozšíření dále existuje vazba objektů na jejich výškové mapy, opačný vztah ale není kontrolován. Při duplikaci objektů tedy dochází i ke sdílení vnitřně uchovávaných map, mohou tedy být nekorektně měněny. Již vytvořené obrázky ale zůstavají zachovávány a následující výstupy budou správně generovány dle nového jména objektu.

Kapitola 6

Závěr a budoucí práce

Výsledkem této práce je rozšíření schopné jak vodní, tak termální eroze libovolného terénu uvnitř aplikace Blender. Pro zrychlení výpočtu plně využívá grafické karty a schopností OpenGL. Rovněž nabízí mnoho způsobů, jak výsledné mapy aplikovat a usnadňuje práci s nimi. Rozšíření také klade důraz na automatizaci kroků a zpětnou vazbu uživateli, včetně navigace mezi vytvářenými výsledky.

Kromě práce nad objekty jsou veškeré operace přístupné i pro textury. Každý parametr algoritmů je upravitelný uživatelem a to s ohledem na stabilní výsledky pro libovolný vstup. Rozšíření je použitelné jak se špičkovou grafickou kartou, tak se starým GPU.

Vytvořený projekt je dále rozšiřitelný. Dalším plánovaným krokem je implementace eroze pomocí sil a rour zmíněné v teoretické části. Pro práci s obrázky je rovněž možné rozšířit algoritmy na vícevrstvou erozi.

Dalším možným rozšířením je responzivní rozhraní při erozi. Kód obou typů eroze je již připravený na inkrementální provádění, včetně možnosti přerušení uprostřed simulace.

Literatura

[1] Bayer, H. T. *Implementation of a method for hydraulic erosion.* München, Germany, 2015. Bachelor's Thesis. Technische Universität München. Dostupné z: https://www.firespark.de/resources/downloads/implementationofamethodeforhydraulicerosion.pdf.

[2] Beneš, B. Visual simulation of hydraulic erosion. *Journal of WSCG.* UNION Agency. 2002, sv. 10, 1-2, s. 79–86. ISSN 1213-6964. Dostupné z: http://hdl.handle.net/11025/5963.

[3] Dušan, Z. *Soil Erosion.* Elsevier, 1982. ISBN 978-0-444-99725-8. Dostupné z: https://books.google.com/books?id=o8ny2dUkpM8C&pg=PA48.

[4] Gützkow, R. *Install Dependencies Example*, 3. prosince 2020. Dostupné z: https://github.com/robertguetzkow/blender-python-examples/blob/master/add_ons/install_dependencies/install_dependencies.py.

[5] Jákó, B. a Tóth, B. Fast Hydraulic and Thermal Erosion on GPU. In: *Eurographics.* 2011. Dostupné z: https://old.cescg.org/CESCG-2011/papers/TUBudapest-Jako-Balazs.pdf.

[6] Julien, P. Y. a Simons, D. B. Sediment transport capacity relations for overland flow. *Transactions of the ASAE.* St. Joseph, Michigan, USA: American Society of Agricultural Engineers. 1985, sv. 28, č. 3, s. 755–762. Dostupné z: https://www.engr.colostate.edu/~pierre/ce_old/Projects/Paperspdf/Julien-Simons-ASAE85.pdf.

[7] Mei, X., Decaudin, P. a Hu, B.-G. Fast Hydraulic Erosion Simulation and Visualization on GPU. In:. říjen 2007, s. 47 – 56. DOI: 10.1109/PG.2007.15. ISBN 978-0-7695-3009-3. Dostupné z: https://xing-mei.github.io/files/erosion.pdf.

[8] Musgrave, F. K., Kolb, C. E. a Mace, R. S. The Synthesis and Rendering of Eroded Fractal Terrains. *SIGGRAPH Comput. Graph.* New York, NY, USA: Association for Computing Machinery. jul 1989, sv. 23, č. 3, s. 41–50. DOI: 10.1145/74334.74337. ISSN 0097-8930. Dostupné z: https://doi.org/10.1145/74334.74337.

[9] O'Brien, J. a Hodgins, J. Dynamic Simulation of Splashing Fluids. In:. Květen 1995, sv. 95, s. 198–205, 220. DOI: 10.1109/CA.1995.393532. ISBN 0-8186-7062-2.

[10] Selle, A., Fedkiw, R., Kim, B., Liu, Y. a Rossignac, J. An Unconditionally Stable MacCormack Method. *J. Sci. Comput.* Červen 2008, sv. 35, s. 350–371. DOI: 10.1007/s10915-007-9166-4.

[11] Stava, O., Benes, B., Brisbin, M. a Krivanek, J. Interactive Terrain Modeling Using Hydraulic Erosion. In:. červenec 2008, s. 201–210.

[12] Stokes, M., Anderson, M., Chandrasekar, S. a Motta, R. *A Standard Default Color Space for the Internet - sRGB*, 05. září 1996. Dostupné z: https://www.w3.org/Graphics/Color/sRGB.html.