Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Oskar Korošec

Proceduralno generiranje tropskega otoka

DIPLOMSKO DELO
UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

izr. prof. dr. Iztok Lebar Bajec
MENTOR

Ljubljana, 2016

© 2016, Oskar Korošec

Rezultati diplomskega dela so intelektualna lastnina avtorja ter Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Univerza
v Ljubljani
Fakulteta za računalništvo
in informatiko

Tematika naloge:

V diplomskem delu zasnujte agloritem za avtomatizirano programsko gradnjo (proceduralno gradnjo) vulkanskih otokov. Zgledujte se po sorodnih algoritmih, ki temeljijo na eroziji in prenosu sedimentacije. Rešitev implementirajte na tak način, da bo za procesiranje izkoriščala grafično procesno enoto (GPE).

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani izjavljam, da sem avtor dela, da slednje ne vsebuje materiala, ki bi ga kdorkoli predhodno že objavil ali oddal v obravnavo za pridobitev naziva na univerzi ali drugem visokošolskem zavodu, razen v primerih kjer so navedeni viri.
S svojim podpisom zagotavljam, da:

- sem delo izdelal samostojno pod mentorstvom izr. prof. dr. Iztoka Lebarja Bajca,
- so elektronska oblika dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko in
- soglašam z javno objavo elektronske oblike dela v zbirki "Dela FRI".

— Oskar Korošec, Ljubljana, november 2016.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Oskar Korošec

Proceduralno generiranje tropskega otoka

POVZETEK

V računalniški grafiki velikokrat nastane potreba po prikazu večjih področij terena. Ročno oblikovanje terena zahteva znanje in je časovno potratno. S proceduralnim generiranjem lahko z minimalnim poseganjem ali brez njega ustvarimo večja področja terena z verodostojnim videzom v razmeroma kratkem času.

V diplomskem delu je predstavljen proces generiranja terena tropskega otoka. Začne se z generiranjem baznega terena s pomočjo šuma *Simplex*, sledi preoblikovanje s simuliranjem procesov hidravlične in termične erozije, ki pripomoreta k verodostojnosti končnega terena. Okoli obale tropskih otokov velikokrat rastejo koralni grebeni, zato simuliramo tudi njihovo rast. Med simulacijo omogočamo vizualizacijo sprememb terena v realnem času. Na teren dinamično nanašamo teksture glede na njegovo obliko. Rezultat je verodostojen tridimenzionalni prikaz terena tropskega otoka z nanesenimi teksturami.

Ključne besede: proceduralno, generiranje terena, hidravlična erozija, termična erozija, rast koral, simulacija, dinamično nanašanje tekstur, GPE

University of Ljubljana
Faculty of Computer and Information Science

Oskar Korošec

Procuderal generation of a tropic island

ABSTRACT

A need to visualize larger areas of terrain often arises in the field of computer graphics. Knowledge and effort are required when designing terrain manually. Using procedural generation, larger areas of realistic-looking terrain can be generated with or without minimal intervention in a relatively short time.

The graduation thesis presents a process of generating terrain of a tropical island. The process starts with generation of the base terrain using *Simplex* noise, followed by transformation using simulation of hydraulic and thermal erosion processes that add a realistic look of the terrain. Coral reefs often grow around tropical islands, therefore, we also simulate their growth. Real-time visualization is enabled during the simulation in order to observe the changes of the terrain. Textures are dynamically applied to the terrain based on its shape. The result is a realistic three-dimensional visualization of the textured tropical island.

Key words: procedural, terrain generation, hydraulic erosion, thermal erosion, coral growth, simulation, dynamic texturing, GPU

ZAHVALA

Zahvaljujem se izr. prof. dr. Iztoku Lebarju Bajcu za pomoč in napotke pri izdelavi diplomskega dela. Posebej se zahvaljujem tudi družini in partnerici za podporo med študijem.

— Oskar Korošec, Ljubljana, november 2016.

KAZALO

1 Uvod

1.1 Motivacija

V računalniški grafiki se pogosto upodablja verodostojni teren, kar posebej velja na področju računalniških iger in filmske produkcije. Ročno oblikovanje velikih področij terena je zahtevno in časovno potratno. Delo si lahko olajšamo in poenostavimo s proceduralnim generiranjem. Z različnimi metodami in algoritmi je možno generirati verodostojen teren brez poseganja oblikovalcev. Velika prednost tega pri uporabi v računalniških igrah je, da se lahko teren, in s tem tudi igralna površina, generira v času izvajanja igre na uporabnikovi napravi. Na ta način je možno doseči praktično neskončno mnogo igralnih površin. Prostorska zahtevnost programske opreme je zato manjša, saj ni potrebe po vključevanju podatkov o terenu.

Proceduralno generiranje je mnogokrat računsko intenzivna operacija, kar je odvisno od uporabljenih algoritmov. V zadnjih letih je prišlo do velikega napredka na področju grafičnih procesnih enot (GPE) s programabilnimi senčilniki. GPE so zmožne hitrega procesiranja velike količine podatkov zaradi njihove visoko paralelne arhitekture. Veliko algoritmov, uporabnih za proceduralno generiranje, lahko paraleliziramo in pospešimo z

izvajanjem na GPE.

V tem diplomskem delu se bomo osredotočili na generiranje tropskega otoka, primernega za uporabo v računalniški igri. Omejili se bomo na generiranje in teksturiranje terena, brez postavljanja morebitnih objektov, kot so drevesa in skale. Najprej bomo generirali bazni teren s pomočjo šuma *Simplex*, nato pa za bolj verodostojen videz simulirali hidravlično in termično erozijo. Ker v tropskih morjih okoli otokov rastejo koralni grebeni, bomo simulirali tudi njihovo rast. Na koncu bomo na teren nanesli še teksturo, ustrezno značilnostim terena.

1.2 Delovno okolje

Generator terena smo razvili v brezplačni verziji pogona za igre Unity 5. Programska koda je napisana v C#, HLSL in ShaderLab. Razvojna strojna oprema je prenosni računalnik z grafično kartico Nvidia GTX 760M, procesorjem Intel Core i7 4700-HQ in 8 GB delovnega pomnilnika.

1.3 Pregled področja

Proceduralno generiranje vsebine je široko področje z veliko načini uporabe, kot je generiranje imen, vse do generiranja celotnih planetov. V računalniških igrah generirane vsebine uporabljamo že od samega začetka. Igri Rogue[1] in Elite[2] sta dobra primera uporabnosti te metode. Rogue je proceduralno zgenerirala več temnic, skozi katere se mora igralec prebiti. Vsakič, ko se je igra začela, so se te ponovno zgenerirale, kar pomeni, da je igralec vedno raziskoval drugačne temnice. Igra Elite pa je izkoristila proceduralno generiranje zaradi omejitev spomina takratne strojne opreme. Planeti in galaksije v igri Elite so zato predstavljeni z vhodnimi parametri, na podlagi katerih so algoritmi vsakič zgenerirali enake galaksije in planete.

V tem diplomskem delu smo se osredotočili na generiranje terena. Enega pomembnejših prispevkov na tem področju je naredil Ken Perlin. Predstavil je način generiranja tipa šuma, ki ga danes imenujemo Perlinov šum [1]. Kasneje je predstavil še šum Simplex [2], ki je za izračun hitrejši in izboljša videz. Šum ima v proceduralnem generiranju veliko aplikacij, ena od teh je generiranje terena.

[1] https://en.wikipedia.org/wiki/Rogue_(video_game)
[2] https://en.wikipedia.org/wiki/Elite_(video_game)

Musgrave, Kolb in Mace [3] so eni izmed prvih, ki so predstavili simuliranje erozije. Opisujejo termično in hidravlično erozijo ter način prikaza tridimenzionalnega terena. Simuliranje hidravlične erozije sta predstavila tudi Beneš in Forsbach [4]. Hidravlična erozija simulira pretakanje vode po terenu, ki spodjeda material, ga prenaša v svojem toku in nato odlaga.

Mei, Decaudin in Hu so v članku [5] predstavili implementacijo iz [4] na GPU. Jákó in Tóth [6] pa sta dodatno opisala še implementacijo termične erozije na GPU.

Za simuliranje rasti koral z namenom vizualizacije v računalniški grafiki nismo našli nobenih virov. Nakamura in Nakamori [7, 8] sta predstavila model, ki simulira rast koral in s tem koralnega grebena v namen znanstvenih raziskav. Z nekaj predelave smo ta model uporabili v naši simulaciji.

Maške [9] v svojem diplomskem delu opisuje generiranje terena s pomočjo hidravlične in termične erozije na GPE v programskem okolju XNA. V našem diplomskem delu bomo uporabili drugačen pristop h generiranju baznega terena, po simulaciji erozije bomo simulirali še rast koralnega grebena in na teren nanesli teksture.

2 Bazni teren

2.1 Predstavitev terena

Za predstavitev terena je na voljo več možnosti. Na primer: dve najpogostejši sta višinska karta in voksli. Višinska karta je rastrska slika, kjer vsak piksel predstavlja višino v pripadajoči točki na terenu. To razmerje je opisano kot $h(x, y) = T(x, y)$, kjer h predstavlja višino terena, T pa višinsko karto. Za hranjenje višinske karte potrebujemo samo en barvni kanal, zato je slika karte običajno sivinska. Uporabimo lahko tudi barvno sliko z več kanali, kjer višinsko karto zapišemo v enega od kanalov, medtem ko v ostalih hranimo dodatne višinske karte ali druge podatke. Od uporabljene bitne globine slike je odvisna natančnost, ki jo lahko dosežemo. Na primer: s sliko, ki za predstavitev enega kanala uporablja 8-bitno celoštevilsko vrednost, lahko dosežemo predstavitev 2^8 možnih višinskih vrednosti.

Predstavitev terena z voksli je še en popularnejših načinov. Voksli so kot nekakšni volumenski piksli. Prednost te predstavitve je, da lahko predstavimo tudi previse in jame. V predstavitvi z višinsko karto to ni mogoče, saj je vsaka točka na terenu predstavljena z eno samo vrednostjo. Kljub temu smo se v našem delu odločili za višinsko karto, ker

ima veliko manjšo prostorsko zahtevnost in so operacije nad njo preprostejše. V naši implementaciji zaradi višje natančnosti uporabljamo slike, ki za vsak kanal uporabljajo 32-bitno predstavitev s plavajočo vejico.

2.2 Šum Simplex

Pri generiranju terena se pogosto uporablja šum Simplex [2], ki je nadgradnja Perlinovega šuma [1]. Z njim je možno doseči hribovit videz. Šum Simplex je možno implementirati v poljubno mnogo dimenzijah. Ker za predstavitev uporabljamo višinsko karto, potrebujemo dvodimenzionalno funkcijo šuma. V naši rešitvi uporabljamo C# implementacijo šuma Simplex, ki je del prosto dostopnega paketa MinePackage[1].

Za boljši videz terena smo uporabili več oktav šuma[2]. Izraz oktava v glasbeni teoriji označuje interval med višinama dveh tonov, od katerih ima eden dvojno oziroma polovično frekvenco drugega. V primeru funkcije šuma to pomeni, da z višanjem frekvence šuma (skala šuma) zmanjšujemo tudi amplitudo, oz. da sta ta dva parametra soodvisna. Na eni višinski karti uporabimo več oktav šuma tako, da vse oktave skupaj seštejemo. To je prikazano v enačbi (2.1), kjer je n vrednost šuma Simplex v določenih koordinatah in N število oktav.

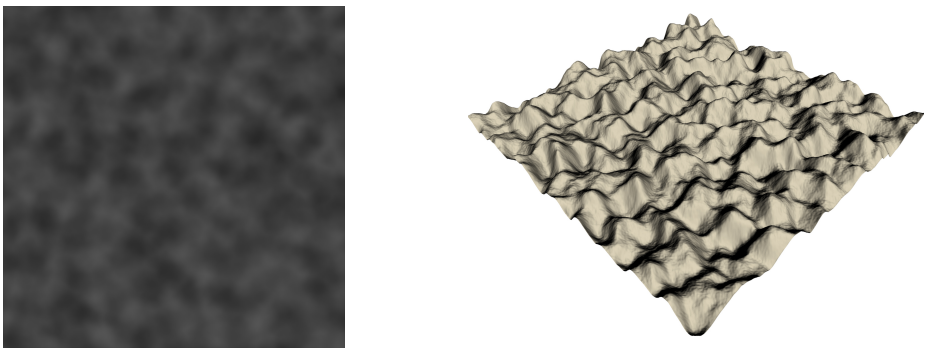
$$h(x, y) = \sum_{i=0}^{N} n(x2^i, y2^i)\frac{1}{2^i} \tag{2.1}$$

Prva oktava ima najvišjo amplitudo, zato največ pripomore k obliki terena, z vsako višjo oktavo pa se viša stopnja podrobnosti na terenu. Število oktav, ki jih je še smiselno uporabiti, je zato odvisno od želenega videza terena in resolucije višinske karte. V našem primeru smo zadovoljni z videzom po osmih oktavah, kar prikazuje slika 2.1.

2.3 Stožec

Cilj diplomskega dela je generirati teren v obliki otoka, zato je treba višinsko karto iz poglavja 2.2 preoblikovati. Želeli smo, da bo teren na sredini višinske karte višji kot ob robovih. To dosežemo z generiranjem dodatne višinske karte, ki definira teren v obliki stožca z nelinearno klančino. Obe višinski karti nato združimo v eno. Za generiranje višinske karte stožca uporabimo enačbi (2.2) in (2.3), kjer $r(x, y)$ označuje razdaljo točke

[1] https://sourceforge.net/projects/minepackage/
[2] http://www.redblobgames.com/maps/terrain-from-noise/



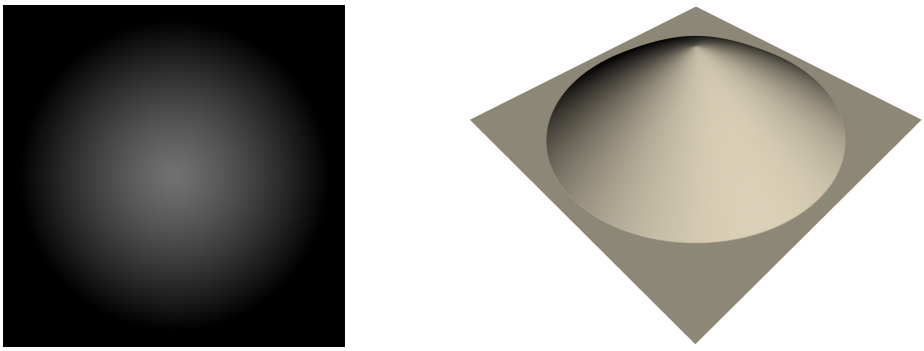
(a) (b)

Slika 2.1: (a) Primer višinske karte, generirane z osmimi oktavami šuma Simplex; (b) Višinska karta osmih oktav šuma Simplex, prikazana kot tridimenzionalni teren.

(x, y) do središča osnovne ploskve stožca, H višino stožca, R polmer stožca in p potenco, ki nadzoruje obliko klančine. Primer prikazuje slika 2.2.

$$r(x, y) = \sqrt{(x - x_C)^2 + (y - y_C)^2} \tag{2.2}$$

$$h(x, y) = \begin{cases} H(1 - (\frac{r(x,y)}{R})^p) & ; r(x, y) < R \\ 0 & ; \text{sicer} \end{cases} \tag{2.3}$$



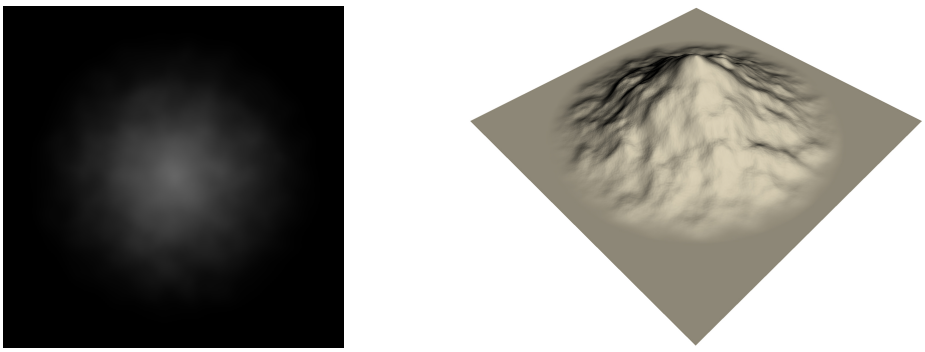
(a) (b)

Slika 2.2: (a) Primer višinske karte stožca; (b) Višinska karta stožca, prikazana kot tridimenzionalni teren.

2.4 Združevanje višinskih kart

Za združevanje višinskih kart smo se zgledovali po orodju Worldmachine[3], kjer je možno več višinskih kart združiti v eno na več načinov. Najboljše rezultate smo dobili, ko smo višinski karti združili s potenciranjem. Vsak piksel prve višinske karte smo potencirali na potenco vrednosti enako ležečega piksla v drugi višinski karti. Enačba (2.4) podrobneje opisuje ta postopek, kjer je h_1 višina v višinski karti v obliki stožca, h_2 pa višina v višinski karti, generirani s šumom Simplex. Moč združevanja določa s. Pri nižjih vrednostih s je združena višinska karta po obliki blizu karte stožca, pri višjih pa so vedno bolj opazne značilnosti karte šuma Simplex. Združeno višinsko karto prikazuje slika 2.3.

$$h_3(x, y) = h_1(x, y)^{1+h_2(x,y)s} \tag{2.4}$$



(a) (b)

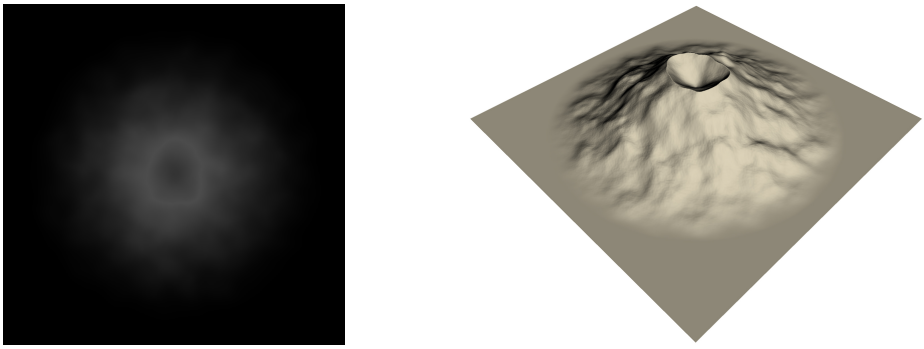
Slika 2.3: (a) Primer združene višinske karte; (b) Združena višinska karta, prikazana kot tridimenzionalni teren.

2.5 Vulkanski krater

Želeli smo, da otok dobi videz vulkana, zato smo dodali možnost generiranja vulkanskega kraterja. Generira se tako, da se vse vrednosti združene višinske karte, ki so višje od določenega praga, preko praga preslikajo navzdol. Preslikovanje je prikazano v enačbi (2.5), kjer je h_4 končna višinska karta baznega terena, C pa višina pragu in s tem višina, na kateri se ustvari vulkanski krater. Slika 2.4 prikazuje končno višinsko karto baznega terena.

[3] http://www.world-machine.com/

$$h_4(x,y) = \begin{cases} C - (h_3(x,y) - C) & ; h_3(x,y) > C \\ h_3(x,y) & ; \text{sicer} \end{cases} \tag{2.5}$$



(a) (b)

Slika 2.4: (a) Primer končne višinske karte baznega terena; (b) Končna višinska karta baznega terena, prikazana kot tridimenzionalni teren.

3 Erozija terena

3.1 Hidravlična erozija

V poglavju 2 smo opisali generiranje baznega terena. Kljub temu, da teren spominja na otok, je njegov videz neprepričljiv. S simuliranjem naravnih procesov, kot sta hidravlična in termična erozija, želimo videz izboljšati. Hidravlično erozijo smo implementirali po opisu v članku [5], ki opisuje metodo simuliranja hidravlične erozije na GPE. Implementirali smo tudi simulacijo termične erozije, katere GPE-verzija je opisana v članku [6]. Dejstvo, da sta algoritma prirejena za paralelno izvajanje na GPE, je pomembno in podrobneje opisano v poglavju 5.

Simuliranje hidravlične erozije se začne s simuliranjem pretakanja vode po površini terena. Na podlagi hitrosti vode in naklona terena določimo količino raztopljenega sedimenta ter njegov prenos. V predelih hitrejšega vodnega toka voda teren odnaša, v počasnejših pa odlaga. S tem se teren preoblikuje in dobi želen erodiran videz. Voda med simulacijo ves čas izhlapeva.

3.1.1 Model

Za opis simulacije bomo uporabljali naslednje oznake za uporabljene količine:

- višina terena h,
- višina gladine vode d,
- količina razstopljenega sedimenta s,
- izhodni tokovi vode $\vec{f} = \langle L, R, T, B \rangle$,
- hitrost vode v določeni točki $\vec{v} = \langle v_\mathrm{x}, v_\mathrm{y} \rangle$.

Vsaka od količin se hrani za vsako točko (celico) terena, kot prikazuje slika 3.1.

Vsaka iteracija simulacije je razbita na več zaporednih korakov:

1. dodajanje vode,
2. pretakanje vode po terenu in izračun vektorjev hitrosti $\vec{v}$ ter posodobitev višin vodne gladine d,
3. izračun topljenja terena in odlaganja sedimenta,
4. prenos sedimenta,
5. difuzija sedimenta,
6. izhlapevanje vode.

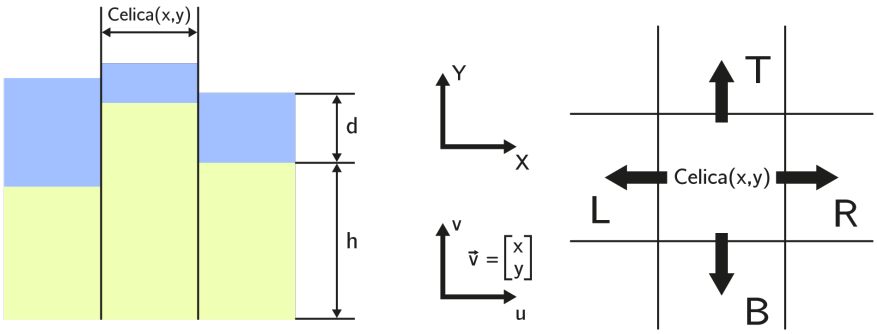
Vsak korak uporabi količine iz prejšnjega koraka in jih posodobi. S t označimo trenutni čas in z Δt časovni korak ene iteracije simulacije. Količine nato indeksiramo s časom. Vrednosti količin po končani iteraciji indeksiramo s $t+\Delta t$. Nekatere količine se izračunajo v večih podkorakih, te indeksiramo z $'$, $''$ itd., da med seboj ločimo vrednosti v različnih podkorakih. Sosednje celice trenutne celice indeksiramo z L, R, T, B.

3.1.2 Dodajanje vode

Dež predstavlja nastanek nove vode. Zaradi velikosti terena ni smiselno simulirati dežja kot posameznih kapelj, temveč kot uniformno rast gladine vode na celotnem terenu. Vmesno višino vode izračunamo kot

$$d'_{t+\Delta t} = d_t + \kappa_r \Delta t, \tag{3.1}$$

kjer κ_r označuje količino dežja v času t.

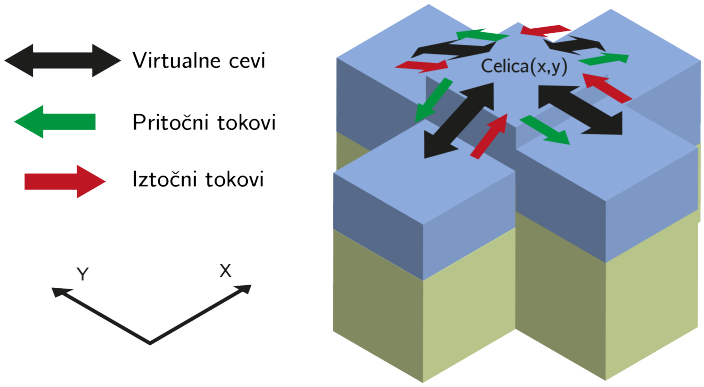


Slika 3.1: Prikaz količin, uporabljenih v simulaciji hidravlične erozije.

3.1.3 Simulacija pretakanja vode

Za simuliranje pretakanja vode se uporablja prirejen model, predstavljen v članku [10], ki povezuje sosednje celice z navideznimi cevmi, po katerih se med njimi izmenjuje voda (slika 3.2). Za vsako celico se hranijo njeni iztočni tokovi $\vec{f} = \langle L, R, T, B \rangle$ do štirih sosednjih celic, njeni pritočni tokovi pa so iztočni tokovi obravnavani celici sosednjih celic. V vsaki iteraciji simulacije se iztočni tokovi pospešijo zaradi razlike v hidrostatičnem tlaku med dvema celicama, višina vodne gladine pa se nato izračuna s kopičenjem vseh tokov v virtualnih ceveh. Iztočni tok L za celico (x, y) izračunamo po enačbi:

$$L'_{t+\Delta t} = \max\left(0, L_t + A\frac{g\Delta h_t^{\mathrm{L}}}{l}\Delta t\right) \tag{3.2}$$



Slika 3.2: Vsaka celica je s sosednjo povezana s štirimi virtualnimi cevmi.

kjer A predstavlja površino prereza cevi, l dolžino cevi med dvema sosednjima celicama, g gravitacijski pospešek in Δh_t^{L} razliko višin med obravnavano ter levo sosednjo celico

$(x-1, y)$, ki se izračuna po enačbi:

$$\Delta h_t^{\mathrm{L}} = h_t + d'_{t+\Delta t} - h_t^{\mathrm{L}} - d_{t+\Delta t}^{\mathrm{L}'} \tag{3.3}$$

Iztočni tokovi do ostalih sosedov R, T, B se izračunajo na podoben način. Možno je, da bi višina vodne gladine po izračunu postala negativna, če bi iz celice odteklo več vode, kot je ta vsebuje. V tem primeru je treba iztočne tokove omejiti s količino vode v celici, zato vse iztočne tokove skaliramo s faktorjem K, ki ga izračunamo po enačbi:

$$K = \min\left(1, \frac{d'_{t+\Delta t} l^2}{(L'_{t+\Delta t} + R'_{t+\Delta t} + T'_{t+\Delta t} + B'_{t+\Delta t})\Delta t}\right) \tag{3.4}$$

Vse iztočne tokove nato skaliramo s faktorjem K:

$$\vec{f}_{t+\Delta t} = K\vec{f}'_{t+\Delta t}. \tag{3.5}$$

Nove vrednosti iztočnih tokov uporabimo za izračun posodobljene višine vodne gladine d. Najprej za vsako celico prištejemo pritočne tokove sosednjih celic:

$$\vec{f}_{in} = \langle R_{t+\Delta t}^{\mathrm{L}}, L_{t+\Delta t}^{\mathrm{R}}, B_{t+\Delta t}^{\mathrm{T}}, T_{t+\Delta t}^{\mathrm{B}} \rangle \tag{3.6}$$

in odštejemo iztočne tokove. Izračun skupne spremembe volumna vode za celico (x, y) je tako:

$$\Delta V = (\sum \vec{f}_{in} - \sum \vec{f}_{t+\Delta t})\Delta t. \tag{3.7}$$

Višino vodne gladine za celico (x, y) nato posodobimo z:

$$d''_{t+\Delta t} = d'_{t+\Delta t} + \frac{\Delta V}{l^2}. \tag{3.8}$$

Za simuliranje procesa erozije potrebujemo hitrost vode $\vec{v}$ v vsaki celici. Upoštevamo samo horizontalni hitrosti po oseh X in Y, ki jih izračunamo iz iztočnih tokov. Povprečna količina vode na enoto časa, ki steče skozi celico (x, y) v smeri osi X, je podana z enačbo:

$$\Delta W_{\mathrm{X}} = \frac{R_{t+\Delta t}^{\mathrm{L}} - L_{t+\Delta t} + R_{t+\Delta t} - L_{t+\Delta t}^{\mathrm{R}}}{2}. \tag{3.9}$$

Hitrost nato izračunamo kot:

$$\vec{v} = \frac{\langle \Delta W_{\mathrm{x}}, \Delta W_{\mathrm{y}} \rangle}{l\bar{d}}. \tag{3.10}$$

V tem primeru $\bar{d}$ označuje povprečno višino vode v celici (x, y) med prvima dvema korakoma:

$$\bar{d} = \frac{d'_{t+\Delta t} + d''_{t+\Delta t}}{2}. \tag{3.11}$$

Pri simuliranju pretakanja vode je treba upoštevati robne pogoje. Iztekanje vode s terena smo omejili tako, da smo vsaki cevi, ki nima sosednje celice, nastavili vrednost pripadajočega iztočnega toka na 0. Na primer: za celice brez levega soseda $(0, y)$ tako nastavimo $L_{t+\Delta t}$ na 0. Enako storimo za pritočne tokove, kar v tem primeru pomeni, da je levi pritočni tok $R^{\mathrm{L}}_{t+\Delta t}$ enak 0.

3.1.4 Erozija in odlaganje terena

Voda med pretakanjem zaradi erozije pretvori del terena v sediment, ki ga prenaša in odlaga po terenu. Proces erozije in odlaganja za posamezno celico je večinoma odvisen od kapacitete sedimenta celice C_t, ki ga izračunamo po enačbi (3.12). V enačbi κ_{C} predstavlja konstanto kapacitete sedimenta, α pa lokalni naklon terena v tej točki.

$$C_t = \kappa_{\mathrm{C}} sin(\alpha)\, |\vec{v_t}| \tag{3.12}$$

Lokalni naklon terena α izračunamo po naslednjih enačbah:

$$\vec{m} = \begin{bmatrix} (h^{\mathrm{R}}_t - h^{\mathrm{L}}_t)/D \\ (h^{\mathrm{T}}_t - h^{\mathrm{B}}_t)/D \\ 1 \end{bmatrix} \tag{3.13}$$

$$\hat{n}_t = \frac{\vec{m}}{|\vec{m}|} \tag{3.14}$$

$$\alpha = acos(\hat{n}_t \cdot \hat{U}) \tag{3.15}$$

V enačbi (3.13) izračunamo $\vec{m}$, ki ga nato normaliziramo v (3.14), kjer je $\hat{n}$ normala terena v točki (x, y). Z D označujemo razdaljo med sosednjima celicama in s $\hat{U}$ enotski vektor v navpični smeri, v našem primeru $\hat{U} = [0, 0, 1]^T$. Na zelo položnih delih, kjer se α približuje 0, je kapaciteta sedimenta C zelo majhna. Na takšnih predelih proces erozije ne bo imel učinka, zato omejimo α s poljubno minimalno vrednostjo.

Na podlagi kapacitete sedimenta C_t in trenutne količine raztopljenega sedimenta s_t se odločimo, ali se bo del sedimenta odložil ter pretvoril v teren ali ne. Če velja $C_t > s_t$, poteče proces erozije in se del terena pretvori v sediment, v obratnem primeru pa se del sedimenta odloži in pretvori nazaj v teren. Novo višino terena in količino raztopljenega sedimenta izračunamo po enačbah:

$$h_{t+\Delta t} = \begin{cases} h_t + \kappa_{\mathrm{D}}(C_t - s_t) & ;\text{ če } C_t < s_t \\ h_t - \kappa_{\mathrm{S}}(C_t - s_t) & ;\text{ sicer} \end{cases} \tag{3.16}$$

$$s'_{t+\Delta t} = \begin{cases} s_t - \kappa_{\mathrm{D}}(C_t - s_t) & ; \text{če } C_t < s_t \\ s_t + \kappa_{\mathrm{S}}(C_t - s_t) & ; \text{sicer,} \end{cases} \tag{3.17}$$

kjer je κ_{S} konstanta topljenja materiala, κ_{D} pa konstanta odlaganja materiala.

3.1.5 Prenos sedimenta

Po končanem procesu erozije in odlaganja posodobimo količine sedimenta v vsaki celici glede na hitrosti vode $\vec{v}$. Ta proces opisuje advekcijska enačba:

$$\frac{\delta s}{\delta t} + (\vec{v}\nabla s) = 0. \tag{3.18}$$

Enačbo (3.18) rešimo z advekcijsko metodo. Novo vrednost sedimenta v celici (x, y) izračunamo z Eulerjevim korakom nazaj v času:

$$s''_{t+\Delta t} = s'_{t+\Delta t}(x - v_{\mathrm{x}}\Delta t, y - v_{\mathrm{y}}\Delta t), \tag{3.19}$$

kjer koordinate $(x - v_{\mathrm{x}}\Delta t, y - v_{\mathrm{y}}\Delta t)$ omejimo na območje terena.

Ta metoda za prenos sedimenta ne deluje zadovoljivo. Kot je Maške [9] omenil v svojem delu, je možno, da se v enem koraku količina sedimenta iz ene celice preslika v več celic. Opazili smo tudi, da sediment pri nižjih hitrostih vode večkrat potuje v ravnih črtah. Vzrok za to je, da se sediment v območjih nizke hitrosti prenaša med celicami, ki so neposredne sosede. To daje polju sedimenta grob in nenaraven videz, zato polje sedimenta dodatno gladimo s simuliranjem difuzije.

3.1.6 Difuzija sedimenta

Polje sedimenta dodatno gladimo s simuliranjem difuzije z dvodimenzionalno Laplacevo enačbo, ki jo numerično rešujemo z Jakobi iteracijami [11]. Izračun posamezne iteracije prikazuje enačba (3.20), kjer n_{U} predstavlja število sosednjih celic v Von Neummanovi okolici, ki držijo vodo ($d''_{t+\Delta t} > 0$).

$$s_{t+\Delta t} = \frac{s^{\mathrm{L}''}_{t+\Delta t} + s^{\mathrm{R}''}_{t+\Delta t} + s^{\mathrm{T}''}_{t+\Delta t} + s^{\mathrm{B}''}_{t+\Delta t}}{n_{\mathrm{U}}} \tag{3.20}$$

Robne pogoje smo poenostavili tako, da se koordinate celic omejijo na območje terena. Na primer: za vse celice, ki ležijo na levem robu terena $(0, y)$, se koordinate leve sosednje celice $(x - 1, y)$ omejijo na $(0, y)$. Celicam, ki ne vsebujejo vode, nastavimo vrednost $s_{t+\Delta t}$ na 0.

3.1.7 Izhlapevanje vode

Zadnji korak simulacije omogoča, da s terena odstranimo nekaj vode zaradi izhlapevanja. Predpostavimo, da je temperatura ozračja med simulacijo konstantna, zato hitrost izhlapevanja določimo s konstanto κ_{E}. Model izhlapevanja opisuje naslednja enačba:

$$d_{t+\Delta t} = d''_{t+\Delta t}(1 - \kappa_{\mathrm{E}}\Delta t). \tag{3.21}$$

3.2 Termična erozija

Hidravlična erozija oblikuje grob teren z veliko ostrimi robovi. To je posebej opazno na pobočjih ob vodnih strugah, ki se neprestano poglabljajo. S simuliranjem termične erozije zgladimo in izboljšamo videz terena. Termična erozija zajema dejavnike, ki drobijo površino terena v drobir, ta pa nato drsi po strmih pobočjih [3].

Podobno kot za hidravlično tudi za termično erozijo uporabimo model virtualnih cevi, le da v tem primeru cevi prenašajo drobir (terena) namesto vode. Zaradi boljšega rezultata smo uporabili Moorovo okolico, ki zajema vseh osem sosednjih celic $\mathbf{I} \in \mathrm{L}, \mathrm{R}, \mathrm{T}, \mathrm{B}, \mathrm{TL}, \mathrm{TR}, \mathrm{BL}, \mathrm{BR}$. To bi lahko uporabili že za hidravlično erozijo, vendar menimo, da bi veliko dodala h kompleksnosti implementacije brez večjih izboljšav pri rezultatu. Višino terena v trenutni celici označimo s h, njenih osem sosedov pa s $h^i, i \in \mathbf{I}$. Količina drobirja, ki jo lahko v eni iteraciji prenesemo v vsako od sosednjih celic, je omejena z M. Če to vrednost presežemo, začne algoritem oscilirati. Izračunamo jo po enačbi $M = \frac{H}{2}\Delta t$, kjer $H = h_t - min(h_t^i, i \in I)$ predstavlja razliko med višino terena v trenutni celici in višino terena v najnižji sosednji celici. Enačbo smo glede na izvirni članek [6] nekoliko poenostavili, v izvirnem članku izračun vključuje še lokalno trdnost terena.

Količino M nato razdelimo med nižje ležeče sosednje celice, ki z obravnavano celico tvorijo kot naklona, večji od kota mirovanja T. To je kritičen kot pobočja, ki ga lahko neki zrnat material še tvori. Kadar klančina preseže kot mirovanja, material zdrsne. V praksi ga izmerimo tako, da material počasi sipamo med dve prozorni plošči in opazujemo kot klančine, ki pri tem nastane.

Kot naklona med obravnavano in sosednjo celico $i \in \mathbf{I}$ izračunamo po enačbi:

$$\alpha^i = tan((h_t - h_t^i)/D), \tag{3.22}$$

kjer D predstavlja razdaljo med celicama. Na podlagi izračunanih kotov naklona sestavimo množico, ki vsebuje vse sosednje celice z naklonom, večjim od T:

$$\mathbf{J} = i \in \mathbf{I} | \alpha^i > T. \tag{3.23}$$

Vsaka celica iz množice **J** na podlagi višinske razlike z obravnavano celico dobi temu proporcionalno količino materiala, v ostale celice materiala ne prenašamo. Na primer: količino materiala, ki se prenese v levo sosednjo celico, če ta spada v množico **J**, izračunamo z enačbo:

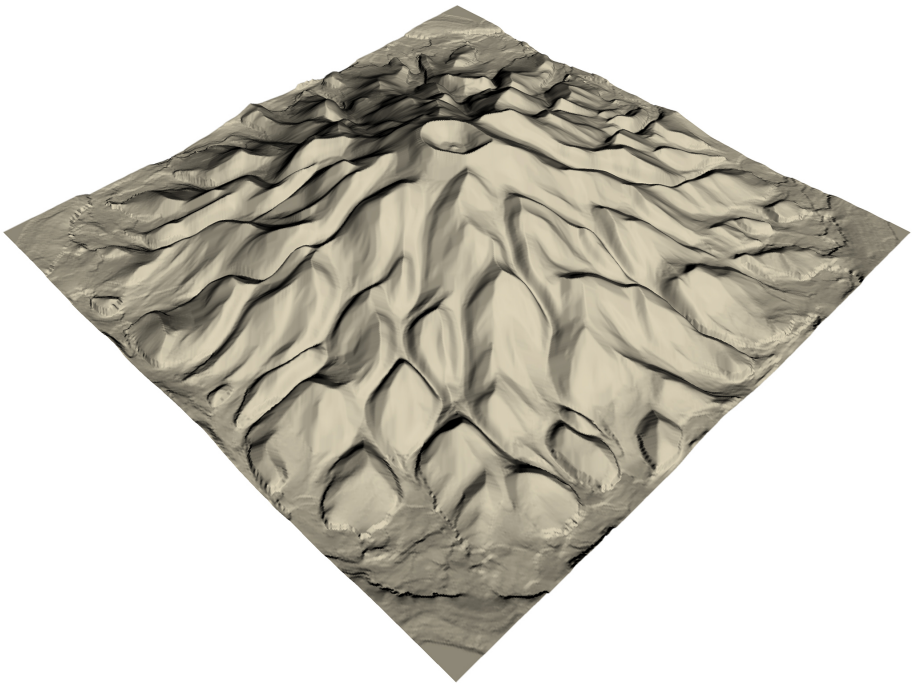
$$L = M \frac{h_t^{\mathrm{L}}}{\sum_{j \in \mathbf{J}} h_t^j}. \tag{3.24}$$

Materiala zaradi paralelnega izvajanja simulacije ne premaknemo direktno v celice, temveč količine prenešenega materiala za vsako sosednjo celico zapišemo v navidezne cevi. Z $\vec{m} = \langle L, R, T, B, TL, TR, BL, BR \rangle$ označimo iztočne tokove iz obravnavane celice do njenih sosedov. Pritočne tokove materiala, ki predstavljajo iztočne usreznih sosed, pa z $\vec{m}_{in} = \langle L^{\mathrm{R}}, R^{\mathrm{L}}, \ldots, BR^{\mathrm{TL}} \rangle$. Končno višino vsake celice izračunamo tako, da k trenutni višini prištejemo vse pritočne tokove materiala in odštejemo vse odtočne:

$$h_{t+\Delta t} = h_t + \sum \vec{m}_{in} - \sum \vec{m}. \tag{3.25}$$

Primer terena po končani simulaciji prikazuje slika 3.3.

(a)



(b)

Slika 3.3: (a) Višinska karta po končani hidravlični in termični eroziji; (b) Višinska karta, prikazana kot tridimenzionalni teren.

4 Rast koralnega grebena

4.1 Koralni greben

V tropskih morjih okoli otokov, predvsem takšnih vulkanskega nastanka, velikokrat rastejo korale, ki skozi čas tvorijo koralne grebene [12]. S simuliranjem rasti koralnih grebenov želimo videz otoka še bolj približati resničnosti in ga narediti zanimivejšega.

Korale rastejo v plitvinah toplejših morij. Za rast potrebujejo svetlobo, saj ožigalkarji živijo v sožitju s fotosinteznimi algami zooksantele (angl. zooxanthellae) [12]. Sestavljene so iz kolonij majhnih polipov, katerih apnenčasti zunanji skeleti se nalagajo in tvorijo koralne grebene. K tvorbi grebena pripomorejo tudi ostali organizmi, ki živijo v koralnem ekosistemu.

Ločimo več vrst koralnih grebenov, kar je opisal že Darwin v The Structure and Distribution of Coral Reefs [13]. Obalni grebeni so grebeni v prvi stopnji rasti. Ker se otok skozi čas potaplja in morska gladina viša, lahko obalni greben preraste v pregradni greben. Zaradi rasti se povečuje in oddaljuje od obale, tik ob obali pa nastane plitva laguna. Zadnja stopnja rasti je atol, ki nastane, ko je otok že pod gladino morja in ga koralni greben preraste. Atol je prepoznaven po veliki laguni, obrobljeni s koralnimi

otoki. V tem poglavju bomo opisali model, primeren za simuliraliranje rasti obalnih in pregradnih grebenov.

Nakamura in Nakamori [8] sta predstavila model rasti koralnega grebena na podlagi intenzitete svetlobe ter koncentracije anorganskega ogljika v vodi. Model je uporaben za dvodimenzionalne simulacije rasti koralnega grebena v pogledu s strani, mi pa potrebujemo model, s katerim bo mogoče simulirati rast koralnega grebena v pogledu iz zraka. Zato smo model predelali in poenostavili. Izvorni model uporablja veliko število konstant, ki smo jih večinoma združili, kjer je bilo to mogoče. Naš model ne uporablja več količin v resničnih enotah, saj je bil naš namen le vizualno simulirati nastanek koralnega grebena.

4.2 Model

V opisu simulacije uporabljamo dve glavni količini:

- višina terena (morskega dna) h in
- koncentracija anorganskega ogljika v vodi O.

Obe količini se hranita za vsako točko oziroma celico.

Vsaka iteracija simulacije je razbita na dva zaporedna koraka:

1. rast koral in
2. difuzija koncentracije anorganskega ogljika.

Količine bomo označevali kot v poglavju 3, tako da so na začetku iteracije indeksirane s časom t, po končani iteraciji pa s časom $t + \Delta t$. Vmesne vrednosti so označene z $'$,$''$ itd. Obravnavani celici sosednje celice indeksiramo z $\mathrm{L}, \mathrm{R}, \mathrm{T}, \mathrm{B}$. Ob začetku simulacije je vrednost O vsake celice nastavljena na O_0.

4.3 Rast koral

Kalcifikacija koral je v veliki meri odvisna od stopnje fotosinteze P, ki jo lahko izračunamo na podlagi globine vode g

$$g = W_t - h, \tag{4.1}$$

kjer W_t označuje globalno višino vode.

Korale zaradi plimovanja in ostalih dejavnikov običajno rastejo le pod določeno globino vode G. Stopnjo kalcifikacije koral c računamo po enačbi (4.2), kjer je κ_{B} konstanta hitrosti kalcifikacije in κ_{A} konstanta absorpcije vode podana v arbitrarnih enotah.

$$c = \begin{cases} O^2 \kappa_{\mathrm{B}} e^{-g\kappa_{\mathrm{A}}} & ; \text{če } G > g \\ 0 & ; \text{sicer} \end{cases} \tag{4.2}$$

Podatkov o višini koralnega grebena ne hranimo posebej, temveč ga štejemo kot teren. Rast grebena je neposredno odvisna od stopnje kalcifikacije. Končno višino terena po končani iteraciji izračunamo po enačbi:

$$h_{t+\Delta t} = h_t + c\Delta h. \tag{4.3}$$

Korale porabijo za rast določeno količino anorganskega ogljika v vodi, zato moramo vrednost O zmanjšati v sorazmerju z vrednostjo c. Spremembo vsebnosti anorganskega ogljika ΔO izračunamo po enačbi (4.4), vsebnost po končanem koraku pa po enačbi (4.5). S tem, da delimo z g, višino vode v obravnavani celici, v enačbi (4.5) dosežemo, da se pri enaki stopnji rasti koral koncentracija anorganskega ogljika v vodi zmanjša primerno količini vode nad to točko na terenu. Konstanta κ_{U} nadzoruje stopnjo porabe anorganskega ogljika.

$$\Delta O = \frac{c\Delta t}{g}\kappa_{\mathrm{U}} \tag{4.4}$$

$$O'_{t+\Delta t} = max(0, O_t - \Delta O) \tag{4.5}$$

4.4 Difuzija anorganskega ogljika

Koncentracija anorganskega ogljika v vodi je porazdeljena zaradi plimovanja in mešanja vode. Simulacija teče v časovnem razponu več tisoč let na velikem terenu, zato lahko porazdelitev anorganskega ogljika prikažemo z enostavno difuzijo, podobno kot smo to storili s sedimentom v poglavju 3.1.6.

Difuzijo anorganskega ogljika modelira enačba (4.6). Robne pogoje postavimo tako, da za vsako celico, ležečo tik ob robu, ne računamo Jakobijeve iteracije, temveč ji priredimo vrednost $O_{t+\Delta t}$ na O_0.

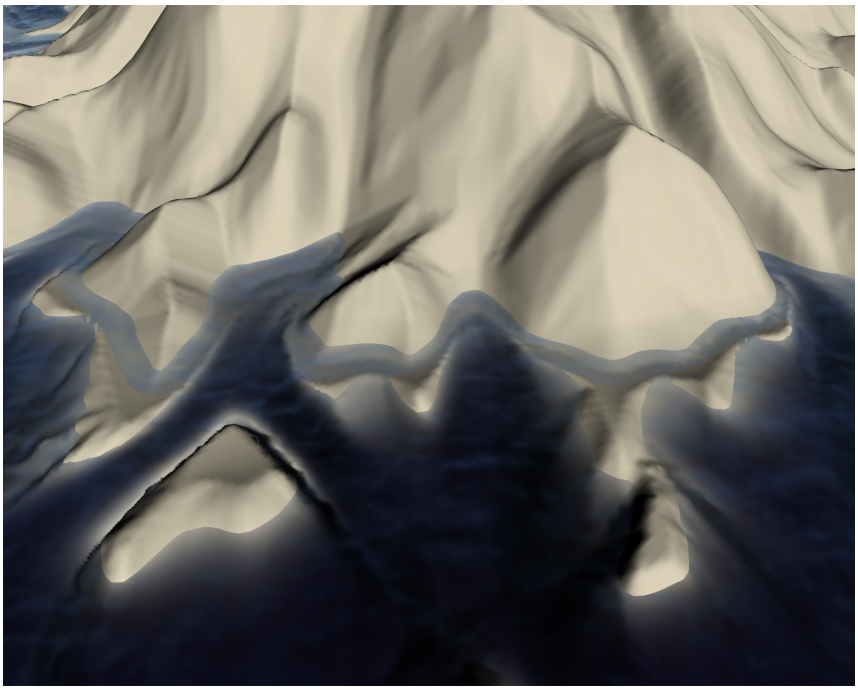
$$\frac{\delta O'_{t+\Delta t}}{\delta x^2} + \frac{\delta O'_{t+\Delta t}}{\delta y^2} = 0 \tag{4.6}$$

Enačbo (4.6) numerično rešujemo z Jakobijevo iteracijo (4.7), kjer n_{U} predstavlja število sosednjih celic Von Neumannovi okolici, ki ležijo pod vodno gladino ($g > 0$). Jakobijeve iteracije ne računamo, če je trenutna celica nad vodno gladino. Takšna celica ne more vsebovati koncentracije anorganskega ogljika in je v tem primeru $O_{t+\Delta t}$ prirejena vrednost 0.

$$O_{t+\Delta t} = \frac{O^{\mathrm{L}'}_{t+\Delta t} + O^{\mathrm{R}'}_{t+\Delta t} + O^{\mathrm{T}'}_{t+\Delta t} + O^{\mathrm{B}'}_{t+\Delta t}}{n_{\mathrm{U}}} \tag{4.7}$$

Na koncu iteracije zvišamo še globalno gladino vode po enačbi (4.8), kjer je κ_{D} konstanta rasti višine vodne gladine. Videz koralnega grebena po končani simulaciji prikazuje slika 4.1.

$$W_{t+\Delta t} = W_t + \kappa_{\mathrm{D}} \Delta t \tag{4.8}$$



Slika 4.1: Koralni greben ob obali otoka. Morska gladina je prikazana za boljšo predstavo.

5 Implementacija

5.1 Procesiranje podatkov na GPE

Simuliranje naravnih procesov (npr. hidravlična erozija) je običajno računsko zahtevna operacija. Čas izvajanja simulacije je pomemben, saj želimo rezultate erozije opazovati v realnem času, zato simulacijo izračunavamo na GPE. Uporabo GPE v namen, ki ni prikazovanje na ekranu, imenujemo GPGPU (angl. General-purpose computing on graphics processing units). Uporablja se na različnih področjih, med drugim tudi v znanosti in financah. Hitrost računanja na GPE omogoča uporaba visoko paralelne arhitekture z veliko jedri. Poglavitni problem je procesiranje podatkov na GPE, saj moramo uporabiti algoritme, ki so primerni za paralelno izvajanje. Brez teh algoritmov bi bil izračun počasen in GPE ne popolno izkoriščena.

Izvajanje poljubnih računskih operacij na GPE lahko dosežemo na več načinov. Po starejšem načinu vse potrebne podatke zapišemo v teksturo in jo nato prikažemo na kvadratu (angl. quad), ki zajema celoten zaslon. Procesiranje izvedemo v senčilniku fragmentov (angl. fragment shader). Novejše GPE v ta namen podpirajo različne vmesnike za programiranje aplikacij (angl. API), kot so CUDA, OpenCL in DirectCompute. V

naši implementaciji uporabljamo senčilnike *ComputeShader*[1], ki so podobni DirectCompute. Za shranjevanje podatkov nismo več omejeni na uporabo tekstur, uporabimo lahko tudi pomnilnike (angl. buffer), ostale osnovne podatkovne tipe ali sestavljene tipe (angl. struct). Simulacijo izvajamo nad višinsko karto, zato podatke shranjujemo v teksturi. Uporaba pomnilnika bi zahtevala prepis podatkov nazaj v teksturo po končani iteraciji simulacije, kar bi bilo časovno potratno.

5.2 Hranjenje podatkov med koraki simulacije

Uporabljeni algoritmi hranijo podatke (npr. višina vode) med posameznimi koraki simulacije. Ti podatki so vezani na posamezne točke v višinski karti, zato jih hranimo enako kot samo višinsko karto, tj. v teksturi. Za shranjevanje vsake količine potrebujemo en barvni kanal. Na ta način lahko v eni teksturi s tremi barvnimi kanali in kanalom prosojnosti (angl. alpha channel) hranimo štiri različne količine. V naši implementaciji uporabljamo teksture s štirimi kanali, kjer je vsak kanal s plavajočo vejico (angl. float). V Unity se ta format tekstur imenuje *ARGBFloat*. V senčilniku *ComputeShader* se ta format preslika v *float4*.

Med simulacijo hidravlične erozije za vsak piksel hranimo višino terena, višino vode in količino sedimenta v teksturi T_1; izhodne tokove do štirih sosedov v T_2; hitrost vode v T_3. Med termično erozijo ob višini terena hranimo še izhodne tokove terena do osmih sosedov, za kar potrebujemo teksturi T_4 in T_5. Za shranjevanje višine terena med termično erozijo uporabimo kar višino terena iz T_1, pri čemer ignoriramo ostale vrednosti. Enako storimo pri simulaciji rasti koral, kjer dodatno ob višini terena hranimo še vsebnost anorganskega ogljika v teksturi T_1. V teksturi T_2 ostaneta še dva prosta kanala. Pri rasti koral moramo hraniti le globalno višino vode na celotnem terenu, za kar potrebujemo eno spremenljivko v senčilniku, ki ji priredimo vrednost na začetku vsake iteracije.

5.3 Jedrne funkcije

Procesiranje simulacije izvajamo v namenskih senčilnikih, imenovanih *ComputeShader*. Delo smo razdelili med tri senčilnike: senčilnik S_H izvaja hidravlično erozijo, senčilnik S_T izvaja termično erozijo, senčilnik S_C pa izvaja simulacijo rasti koral. Simulacijo izvajamo v večih iteracijah, v posamezni iteraciji pa moramo določene teksture obdelati

[1] https://docs.unity3d.com/Manual/ComputeShaders.html

Senčilnik S_H	Senčilnik S_T	Senčilnik S_C
AddWaterRain	*ThermalErosionStep*	*InitializeCt*
ComputeFlux	*ThermalErosionAccumulate*	*SimulateGrowth*
ComputeWaterHeight		*SimulateCtDiffusion*
ComputeErosion		
ComputeSedimentTransport		
ComputeSedimentDiffusion		
ComputeEvaporation		

Tabela 5.1: jedrne funkcije uporabljenih senčilnikov.

večkrat. *ComputeShader* senčilnik je sestavljen iz več funkcij, med katerimi lahko poljubne označimo kot jedrne funkcije. Poženemo ga tako, da poženemo eno od njegovih jedrnih funkcij. Z uporabo več jedrnih funkcij lahko teksturo z enim senčilnikom obdelamo v različnih korakih. Naša koda je temu primerno organizirana, kar prikazuje tabela 5.1, napisana pa je v stilu DirectX 11 jezika HLSL.

Omenili smo že, da izhaja hitrost GPE-procesiranja iz velikega števila procesorjev, sposobnih paralelnega izvajanja. Ob zagonu jedrne funkcije določimo število skupin niti senčilnika, ki jih želimo pognati. Za vsako jedrno funkcijo določimo število niti v posamezni skupini[2]. S tem določamo skupno število niti, v katerih se bo izvedla posamezna jedrna funkcija. Optimalno število niti v posamezni skupini je odvisno od uporabljene GPE. V našem primeru smo ga določili s poskušanjem, in sicer 64 niti na posamezno skupino. Vse jedrne funkcije smo zasnovali tako, da vsaka obdela en piksel, zato poženemo toliko niti, kot je pikslov v teksturi.

5.4 Potek simulacije

Pred začetkom izvajanja simulacije generiramo višinsko karto baznega terena, kot smo opisali v poglavju 2, in jo shranimo v teksturo. Na tej točki je tekstura shranjena v glavnem pomnilniku računalnika (angl. RAM), zato jo prenesemo v pomnilnik GPE, v teksturo T_1. V pogonu Unity so teksture, ki so shranjene izključno v pomnilniku GPE, objekti tipa *RenderTexture*. Med simulacijo uporabljamo le *RenderTexture* in tekstur ne prenašamo med glavnim pomnilnikom ter pomnilnikom GPE, saj je to časovno potratna

[2] https://msdn.microsoft.com/en-us/library/windows/desktop/ff471566(v=vs.85).aspx

operacija. Vizualizacija se enako izvaja v senčilnikih, zato je prenašanje nepotrebno. Kljub temu moramo za vsako od tekstur T hraniti še njeno kopijo, saj rezulatatov posameznega koraka iteracije ne smemo pisati v isto teksturo. S tem bi zaradi paralelnosti izvajanja uničili rezultate ostalih niti. Vsaki jedrni funkciji moramo pred izvajanjem nastaviti potrebne vhodne in izhodne teksture. *ComputeShader* senčilniki uporabljajo za delo s teksturami objekta *Texture2D* in *RWTexture2D*. Prvi podpira samo branje, zato ga uporabljamo za vhodne teksture, medtem ko drugi podpira tudi pisanje in ga zato uporabljamo za izhodne teksture.

Iteracije simulacije izvajamo v zanki, v kateri v določenem vrstnem redu poženemo jedrne funkcije posameznega senčilnika. V namen nadzora simulacije smo implementirali enostaven grafični uporabniški vmesnik. S tem lahko nastavljamo določene parametre in vklapljamo ter izklapljamo posamezne korake simulacije. Vrstni red zagona senčilnikov načeloma ni pomemben, a moramo paziti, da senčilnika S_C ne uporabljamo hkrati kot S_H in S_T. Simulaciji erozije preoblikujeta teren in s tem videz koralnega grebena, kar ni zaželeno. Pomemben je vrstni red zagona jedrnih funkcij posameznega senčilnika. Jedrne funkcije poženemo v vrstnem redu od zgoraj navzdol, kot so prikazane v tabeli 5.1. V senčilniku S_H jedrne funkcije *AddWaterRain* ne poženemo za vsako iteracijo, ampak samo kadar želimo dodati več vode v simulacijo. Izjema je tudi senčilnik S_C, kjer *InitializeCt* zaženemo samo v prvi iteraciji, v kateri začnemo simulirati rast koral.

Vsaka jedrna funkcija bere iz vhodnih tekstur in rezultat zapiše v pripadajoče izhodne teksture. Vsaka naslednja jedrna funkcija potrebuje rezultat prejšnje v vhodni teksturi. Po vsakem klicu jedrne funkcije vrednosti izhodnih tekstur prepišemo nazaj v vhodne teksture. To je sicer časovno potratno, saj po vsakem zagonu jedrne funkcije poženemo še dodatno jedrno funkcijo senčilnika, ki kopira podatke med dvema teksturama. Kopiranje izvedemo tolikokrat, kolikor je tekstur, ki jih je jedrna funkcija simulacije spreminjala. Drugi pristop je uporabil Maške [9], ki je namesto kopiranja tekstur jedrnim funkcijam zaporedno menjaval vhodne in izhodne teksture tako, da so izhodne teksture prejšnje vhodne teksture naslednje. To je manj časovno potratno, a prispeva k zahtevnosti implementacije. V naši implementaciji uporabljamo veliko jedrnih funkcij in več senčilnikov, ki jih lahko poljubno vklapljamo ter izklapljamo. Dosegli smo tudi želeno hitrost simulacije, zato smo se odločili za rešitev s kopiranjem.

6 Vizualizacija

6.1 Prikaz terena

Višinska karta terena je dvodimenzionalna rastrska slika, teren pa želimo v realnem času prikazati kot tridimenzionalni objekt. V tem poglavju bomo opisali prikaz in senčenje tridimenzionalnega objekta, vključno z nanosom tekstur ter prikazom okoliške vode.

Višinska karta v vsakem pikslu hrani višino pripadajoče točke na terenu. Podrobnejši opis je v poglavju 2.1. Za prikaz terena smo uporabili metodo, imenovano *displacement mapping* [14]. Ta metoda nastavi vsakemu oglišču (angl. vertex) modela terena odmik na podlagi pripadajoče vrednosti z višinske karte. Odmik se ponavadi nastavlja v smeri normale oglišča. V našem primeru prikazujemo reliefni teren navpično iz ravne ploskve, zato odmik nastavljamo v smeri navpične osi.

Težava pri tem pristopu je, da naš model terena potrebuje veliko število oglišč, da se podrobnosti višinske karte prenesejo na model. Optimalno je, da ima ploskev enako število oglišč, kot ima višinska karta pikslov. Tako se vrednost vsakega piksla preslika na eno oglišče modela terena. V praksi to pomeni modele z izredno visokim številom oglišč in posledično velikim številom trikotnikov.

Moderne GPE podpirajo metodo, imenovano strojna teselacija (angl. hardware tesselation) [15], ki dinamično razdeli trikotnike modela na več manjših, pri čemer ustvari več oglišč. Namesto da na podlagi resolucije višinske karte uporabimo model terena s pripadajočim številom oglišč, lahko uporabimo strojno teselacijo. Dodatna prednost strojne teselacije je, da lahko njeno stopnjo prirejamo dinamično glede na oddaljenost pogleda od posameznega predela terena. Del terena v neposredni bližini ima veliko več vidnih podrobnosti kot del v daljavi, kjer podrobnosti ne bi bile opazne. V naši implementaciji uporabljamo strojno teselacijo fiksne stopnje in s tem dosežemo natančnejši videz.

Za pravilen izračun osvetlitve potrebujemo normale glede na lokalni naklon vrednosti v višinski karti. Izračunamo jih kot v poglavju 3.1.4 po enačbi (3.14).

Osvetlitev smo izračunali z Lambertovim modelom [16, str. 267-268]. To je enostaven model za senčenje razpršilnih (angl. diffuse) površin brez leska, kot je običajno za teren. Osvetljenost na površini modela izračunamo na podlagi vpadnega kota svetlobe $\hat{l}$ in normale terena v isti točki $\hat{n}$. Osvetljenost nato pomnožimo z barvo površine c_s in barvo luči c_l ter tako dobimo končno barvo piksla c_f, kot je opisano v naslednji enačbi:

$$c_f = (\hat{l} \cdot \hat{n}) c_s c_l \tag{6.1}$$

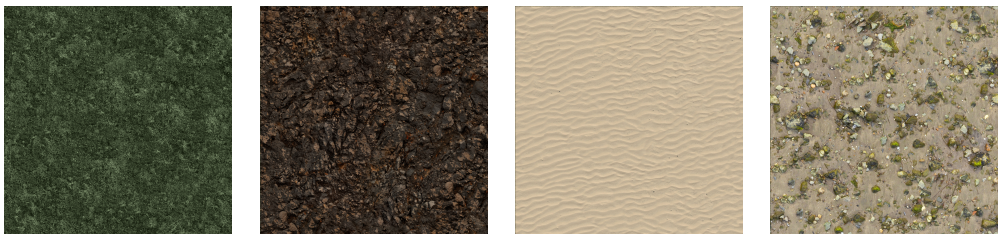
Za osvetljevanje terena potrebujemo vir svetlobe, ki daje vtis sončne svetlobe. Zaradi velike oddaljenosti sonca je vpadni kot svetlobe na vsako točko terena približno enak. Za osvetljevanje uporabljamo smerno oziroma oddaljeno luč [16, str. 264], ki nima pozicije. Luč definira samo vektor smeri, barvo in intenziteto. V zgornji enačbi (6.1) je intenziteta že všteta v barvo luči.

6.2 Teksturiranje

Želeli smo, da teren dobi videz tropskega otoka, k čemur veliko pripomorejo teksture. V senčilniku na podlagi vhodnih tekstur, višine in naklona terena ter višine vode v vsaki točki terena izračunamo barvo. Osnovne štiri teksture, ki jih združujemo, so: trava T_G (zelene površine), skalovje T_R, pesek T_S in morsko dno T_B. Prikazane so na slikah 6.1, dobili smo jih na spletni strani Textures[1].

Logika nanašanja tekstur temelji na opažanjih iz narave in osebne predstave o videzu tropskega otoka. Strma pobočja so redko poraščena, medtem ko na ravninah prevladujejo

[1] http://www.textures.com/



(a) (b) (c) (d)

Slika 6.1: Teksture, uporabljene za dinamično nanašanje; (a) tekstura trave T_G; (b) tekstura skalovja T_R; (c) tekstura peska T_S; (d) tekstura morskega dna T_B.

zelene površine. Za boljši videz smo v višjih in bolj strmih predelih teksturo trave rahlo posvetli ter dodali rumen odtenek. Na obalah je pogosta peščena plaža, zato smo pod gladino vode in na ožjem pasu obale nad njo nanesli teksturo peska, vendar le tam, kjer je obala dovolj položna. Na koncu smo pod gladino vode dodali še teksturo morskega dna.

Med teksturami smo želeli doseči mehke prehode. Prehod med dvema teksturama bi bil pretrd, če bi prisotnost vsake teksture v določeni točki računali binarno na podlagi mejne vrednosti. Za doseganje mehkih prehodov smo si pomagali s funkcijo *smoothstep*[2]. HLSL definira *smoothstep* s tremi parametri: minimalna vrednost, maksimalna vrednost in vhodna vrednost. Vhodne vrednosti med minimalno in maksimalno vrednostjo poda kot rezultat Hermitske interpolacije, sicer pa vrne 0 ali 1 glede na to, ali je vrednost manjša kot minimum ali večja kot maksimum. Tako lahko z minimalno in maksimalno vrednostjo definiramo poljubno širok pas, v katerem se določeni teksturi mešata, sicer pa se izbere ena izmed njiju.

Določamo lahko tudi gostoto ponavljanja tekstur (angl. tiling). Pri teksturiranju terena je to še posebej koristno, saj lahko manjšo teksturo trave ponovimo velikokrat in tako uporabimo manj pomnilnika. Pri ponavljanju mnogokrat pride do vidnih robov med ponovljenimi teksturami in ponovljene izrazite lastnosti v teksturi. Teksture moramo zato prirediti v orodju za obdelavo slik, da jih je čim manj.

[2] https://msdn.microsoft.com/en-us/library/windows/desktop/bb509658(v=vs.85).aspx

6.3 Prikaz vode

Otok v naravnem okolju je obkrožen z vodo, zato smo dodali prikaz okoliških voda. Senčilniki vode niso tema te diplomske naloge, zato smo uporabili senčilnik, ki je na voljo kot del standardnega paketa sredstev Unity, verzije 1.1.1[3] (angl. Unity Standard Assets Package). Paket vsebuje več uporabnih 3D-modelov, senčilnikov, tekstur, skript in ostalih sredstev, ki rešujejo pogoste težave pri razvoju iger, kot je tudi senčilnik vode.

Senčilnik vode smo za naše potrebe kljub temu nekoliko predelali, saj ni podpiral prikaza globine vode. V naravi ima voda v plitvejših predelih večjo prosojnost kot v globinah, kar smo želeli doseči tudi pri videzu našega tropskega otoka. Pogled na tropski otok je bolj verodostojen, če lahko plitke in globoke predele vizualno ločimo, zato smo izračunali globino vode v določeni točki po enačbi (6.2), kjer g označuje globino vode, W globalno višino vode in h višino terena. V vsaki točki površine vode z višinske karte preberemo pripadajočo vrednost višine terena, ki jo nato odštejemo od globalne višine vode.

$$g(x, y) = W - h(x, y) \tag{6.2}$$

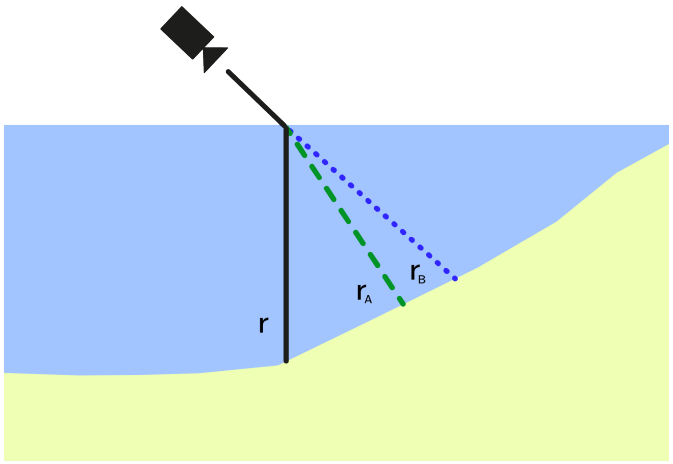
Tak pristop je hiter in preprost za izračun, a ne daje čisto pravilnih rezultatov, ko na površino vode ne gledamo naravnost od zgoraj navzdol. Prosojnost je odvisna od dolžine poti, ki jo svetloba prepotuje skozi vodo. Namesto vrednosti višinske karte bi morali v tej točki izračunati novo odmaknjeno pripadajočo točko na podlagi kota pogleda na vodno površino. Namesto globine vode bi bilo treba izračunati razdaljo od točke, kjer svetloba začne pot skozi vodo, in izračunano odmaknjeno točko, kjer jo konča. Pri tem bi morali upoštevati še lomljenje kota svetlobe zaradi refrakcije, kar prikazuje slika 6.2. Za naš namen pa daje preprostejša implementacija zadovoljive rezultate.

Intenziteta svetlobe l_W v globini je funkcija globine vode [7, str. 749]. Izračunamo jo po naslednji enačbi, kjer K nadzoruje moč slabljenja svetlobe:

$$l_W = e^{-g(x,y)K} \tag{6.3}$$

Končno barvo vode smo nato izračunali z linearno interpolacijo med dvema barvama po intenziteti svetlobe. Prva barva c_0 je barva tik pod gladino površine vode, ki je v našem primeru čisto prosojna, druga barva c_∞ pa je barva na neskončni globini, ki je

[3] https://www.assetstore.unity3d.com/en/#!/content/32351



Slika 6.2: Lomljenje žarka svetlobe pri prehodu iz zraka v vodo; r žarek, uporabljen v izračunu; r_A pravilno lomljen žarek; r_B žarek brez lomljenja.

v našem primeru neprosojna in temno modra. Vrednost l_W moramo pred interpolacijo omejiti na interval od 0 do 1, da z enačbo (6.4) ne izvajamo tudi linearne ekstrapolacije. Omejena intenziteta svetlobe je označena z l_{01}.

$$c_W = c_\infty + l_{01}(c_0 - c_\infty) \tag{6.4}$$

6.4 Implementacija senčilnikov

Opisana vizualizacija se izvaja v dveh senčilnikih. Prvi se uporablja za senčenje terena, drugi pa za senčenje okoliške vode. Napisana sta v jeziku ShaderLab, ki se v Unity uporablja za deklaracijo senčilnikov[4]. Posamezni senčilnik je v ShaderLab sestavljen iz enega ali večih podsenčilnikov, ki so napisani v HLSL podobnemu jeziku. Funkcijam podsenčilnika lahko poleg ostalega določimo vlogi senčilnika oglišč (angl. vertex shader) ali senčilnika fragmentov (angl. fragment shader). ShaderLab podpira tudi vloge z višjo abstrakcijo, kot je vloga senčilnika površine (angl. surface shader), kar poenostavi pisanje senčilnika za osvetljene površine, saj avtomatsko generira kodo za senčilnik oglišč in senčilnik fragmentov. Privzeto že podpira znane modele osvetlitve, a lahko vseeno implementiramo tudi svoje.

V senčilniku terena uporabljamo funkcijo z vlogo senčilnika oglišč za odmik točk

[4] https://docs.unity3d.com/Manual/SL-Shader.html

modela terena. Za nanašanje tekstur uporabljamo funkcijo z vlogo senčilnika površine. Model osvetlitve je implementiran v funkciji z vlogo osvetljevanja, ki jo uporabimo namesto vgrajenih modelov osvetlitve. Teselacijo omogočimo z enostavno funkcijo, ki vrača stopnjo teselacije. Risanje globine vode smo vstavili v funkcijo z vlogo senčilnika površine v senčilnik okoliške vode.

7 Rezultati

S programom smo zadovoljni, saj uspešno generira bazni teren, na katerem simulira hidravlično in termično erozijo ter na koncu še rast koralnega grebena. Končni teren z nanesenimi teksturami in izrisom morske gladine ima prepričljiv videz tropskega otoka. S spreminjanjem parametrov generiranja baznega terena lahko dobimo veliko različnih oblik otoka, kot prikazujejo slike 7.1, 7.2 in 7.3. Vrednosti konstant simulacije, ki smo jih uporabili pri generiranju teh slik, so prikazane v tabeli 7.1.

Uporabljamo višinske karte terena velikosti 512 krat 512 pikslov. Bazni teren se v povprečju generira v 3000 ms. Generiranje bi lahko pospešili, vendar ga ni treba, ker se izvede samo enkrat v celotnem procesu.

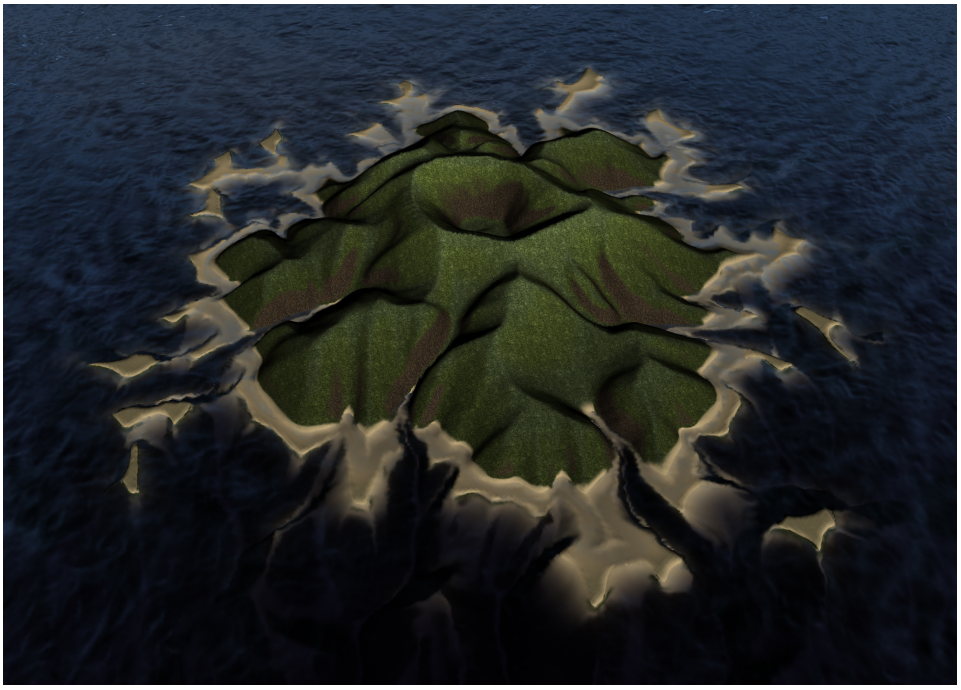
Simulacija teče hitro, s frekvenco osveževanja 94 slik na sekundo, ko teče samo hidravlična erozija, 108 slik na sekundo, ko teče samo termična erozija, in 85 slik na sekundo, ko tečeta obe hkrati. Rasti koralnega grebena ne poganjamo sočasno z erozijo, ampak ločeno, ta teče s frekvenco osveževanja 108 slik na sekundo. Najhitrejša je sama vizualizacija (124 slik na sekundo), ko ne teče nobena simulacija. V pogonu Unity3D smo imeli nastavljen prikaz z najvišjo stopnjo podrobnosti.

Spremembe na terenu je med simulacijo možno opazovati v realnem času in tudi vklapljati ter izklapljati padavine v hidravlični eroziji. Rezultat simulacije se precej spremeni, če padavine dodajamo v kratkih intervalih ali pa so aktivne skozi celotno simulacijo. Z videzom rezultatov simulacije smo zadovoljni, hidravlična in termična erozija skupaj oblikujeta razgiban teren brez večjih vidnih napak. Dodan korak difuzije sedimenta se dobro obnese, vendar ni fizično natančen. Transport sedimenta in logika nalaganja sedimenta potrebujeta še nekaj izboljšav. Težava modela hidravlične erozije je v tem, da je težko uravnovesiti vse konstante v simulaciji. Simulacija hitro postane nestabilna, kar povzroča večje napake na terenu. Maške [9] je v svojem diplomskem delu dosegel boljši transport in nalaganje sedimenta. Hitrosti izvajanja pa zaradi uporabe različne strojne opreme ne moremo primerjati z njegovo.

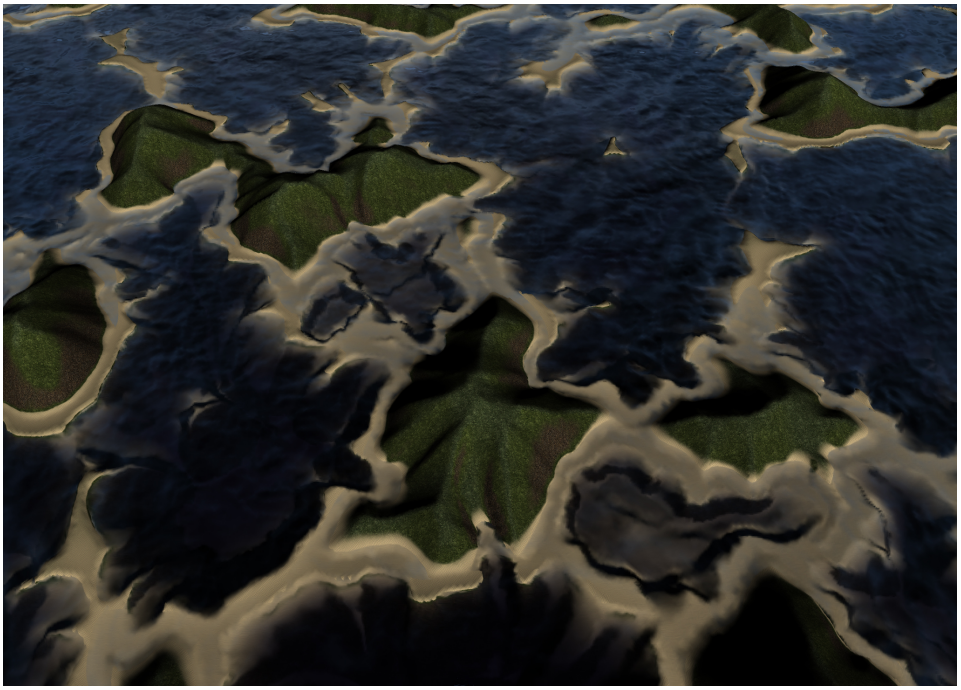
Simulacija rasti koralnega grebena daje prepričljive rezultate in teče hitro. Videz grebena bi lahko bil bolj razgiban, saj se trenutno ustvari popoln in neprekinjen greben ob obali otoka. Naravni grebeni so mnogokrat prekinjeni in različno oddaljeni od obale. Tako kot pri hidravlični eroziji smo imeli tudi v tej simulaciji težave z nastavitvijo konstant, saj ima že zelo majhna sprememba velik vpliv na rezultat.

Z dinamičnim nanašanjem tekstur smo zelo zadovoljni, deluje hitro in daje prepričljiv videz. Vhodne teksture lahko zamenjamo in tako na enostaven način spremenimo videz otoka. Nastavljamo lahko tudi različne parametre na senčilniku, kot sta gostota ponavljanja posamezne vhodne teksture in naklon terena, pri katerem namesto teksture trave nanesemo teksturo skale.

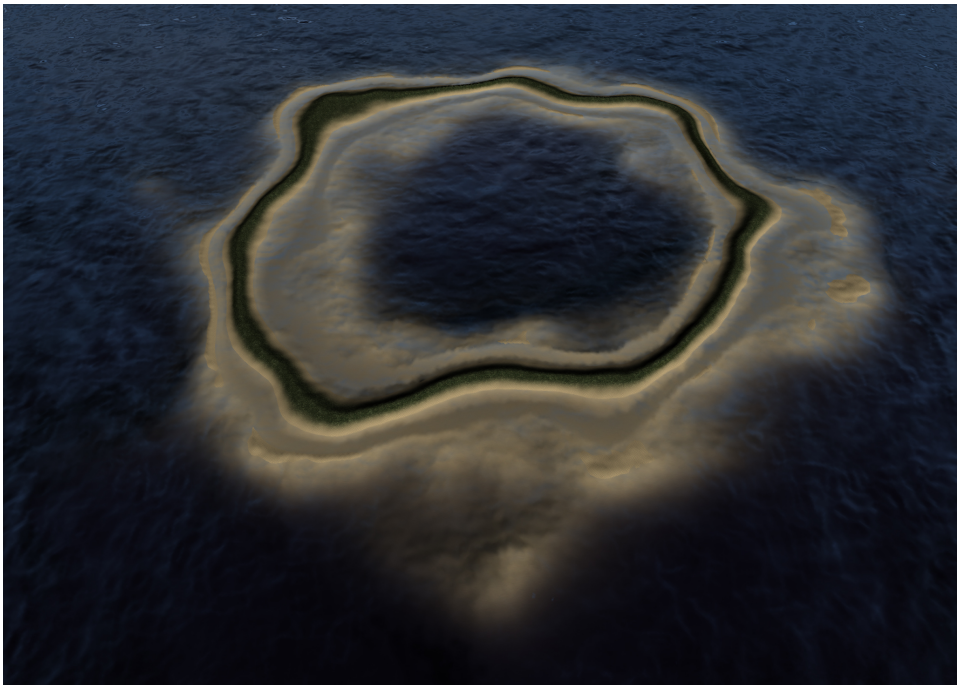
Gladina morja veliko pripomore k prepričljivemu videzu otoka. Predelani senčilnik morske gladine se dobro obnese, saj nam daje tudi vizualno informacijo o globini vode, kar je zelo koristno za videz koralnega grebena. Prikaz vode sicer ni fizično natančen, saj zanemari kot pogleda pri izrisu globine in jo vedno izrisuje, kot da je pogled od zgoraj navzdol. V praksi to ne povzroča težav, saj teren večino časa opazujemo iz zraka.



Slika 7.1: Končni generiran vulkanski otok.



Slika 7.2: Otočje, generirano brez preoblikovanja baznega terena v stožec.



Slika 7.3: Atol, generiran brez uporabe erozije s pomočjo vulkanskega kraterja.

Hidravlična erozija

Δt	κ_r	κ_C	κ_D	κ_S	κ_E	A	l	D
0.2	0.005	5	0.05	0.0005	0.1	2.5	2	2

Termična erozija

Δt	D
0.008	1

Rast koral

Δt	κ_A	κ_B	κ_U	G	O_0
0.01	10	0.25	0.04	0.001	0.25

Tabela 7.1: Vrednosti konstant pri generiranju terena na slikah 7.1, 7.2 in 7.3.

8 Sklepne ugotovitve

Končni izdelek je program, ki na podlagi vhodnih parametrov generira teren v obliki otoka, na katerem v realnem času simulira hidravlično in termično erozijo ter rast koralnega grebena. Na teren dinamično nanaša teksture in izrisuje okoliško gladino vode. Zadani cilj, da ima generiran otok prepričljiv videz tropskega otoka, smo po našem mnenju dosegli.

Začeli smo z generiranjem baznega terena na CPE. Za dosego razgibanega videza terena smo uporabili šum Simplex, katerega rezultate smo preoblikovali v obliko stožca in dodali vulkanski krater. Generiranje baznega terena je trajalo povprečno tri sekunde, kar v praksi ne predstavlja težav, saj se izvede samo enkrat. Z nastavljanjem parametrov s pomočjo grafičnega vmesnika je možno generirati veliko različnih oblik baznega terena. Nastavljamo lahko tako višino in premer oblike otoka, višino vulkanskega kraterja, razgibanost terena kot višino vodne gladine ter ostale.

V želji, da dobi bazni teren bolj prepričljiv videz, smo nato pognali simulaciji hidravlične in termične erozije. Simulirali smo rast koralnega grebena, saj teren predstavlja tropski otok. Vse simulacije se izvajajo hitro.

Med izvajanjem programa smo na teren dinamično nanašali teksture. Uporabili smo štiri osnovne teksture: travo, skalovje, pesek in morsko dno. Teksture smo glede na nagib terena v določeni točki mešali in dosegli prepričljiv videz otoka. Na bolj strmih predelih prevladuje tekstura skalovja, ob obali peska in pod vodno gladina morskega dna.

Implementacijo bi lahko izboljšali. Generiranje baznega terena bi lahko pospešili tako, da bi ga namesto v C# implementirali v nižjenivojskem jeziku (npr. C++) ali pa na senčilniku. Funkcije, ki generirajo bazni teren, smo zasnovali tako, da lahko izhod ene uporabimo kot vhod druge. S tem lahko preprosto dodamo več možnosti pri gradnji baznega terena, kot so dodatne oblike poleg oblike stožca.

Hitrost izvajanja simulacij na GPE bi lahko optimizirali, saj v trenutni implementaciji poteka nepotrebno kopiranje tekstur, kar upočasni proces. Terenu bi lahko izboljšali videz tako, da bi za njegovo predstavitev uporabili več plasti z različnimi trdotami, kar bi morali upoštevati pri simulacijah erozije [17].

Končni teren je uporaben v strateških igrah, kjer gledamo teren večinoma iz daljave. Teren nima dovolj podrobnosti za igre s pogledom iz prve osebe, kjer se igralec premika po terenu in ga raziskuje. Generiranje otoka želenega videza traja od 10 do 15 sekund, kar je težava pri uporabi v računalniški igri, vendar bi jo z izboljšanjem implementacije lahko odpravili.

Literatura

[1] K. Perlin, An image synthesizer, SIGGRAPH Comput. Graph. 19 (3) (1985) 287–296.

[2] K. Perlin, Noise hardware, Real-Time Shading SIGGRAPH Course Notes, 2001.

[3] F. K. Musgrave, C. E. Kolb, R. S. Mace, The synthesis and rendering of eroded fractal terrains, SIGGRAPH Comput. Graph. 23 (3) (1989) 41–50.

[4] B. Beneš, R. Forsbach, Visual simulation of hydraulic erosion, Journal of WSCG 2 (1-2) (2002) 79–86.

[5] X. Mei, P. Decaudin, B.-G. Hu, Fast hydraulic erosion simulation and visualization on gpu, in: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications, PG '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 47–56.

[6] B. Jákó, B. Tóth, Fast Hydraulic and Thermal Erosion on GPU, in: N. Avis, S. Lefebvre (Eds.), Eurographics 2011 - Short Papers, The Eurographics Association, 2011.

[7] T. Nakamura, T. Nakamori, A geochemical model for coral reef formation, Coral Reefs 26 (4) (2007) 741–755.

[8] T. Nakamura, T. Nakamori, A simulation model for coral reef formation: Reef topographies and growth patterns responding to relative sea-level histories, in: Sea Level Rise, Coastal Engineering, Shorelines and Tides, Nova Science Publishers, 2011, pp. 251–262.

[9] L. Maške, Simulacija in vizualizacija erozije terena z uporabo GPE, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2013.

[10] J. F. O'Brien, J. K. Hodgins, Dynamic simulation of splashing fluids, in: Proceedings of the Computer Animation, CA '95, IEEE Computer Society, Washington, DC, USA, 1995, p. 198.
url: http://dl.acm.org/citation.cfm?id=791214.791474

[11] J. M. Cecilia, J. M. García, M. Ujaldón, Cuda 2d stencil computations for the jacobi method, in: International Workshop on Applied Parallel Computing, Springer, 2010, pp. 173–183.

[12] M. Coe (Ed.), Oxfordova ilustrirana enclikopedija žive narave, Vol. 2, DZS, Ljubljana, 1995.

[13] C. Darwin, The structure and distribution of coral reefs (1842), Appleton and Co., New York, 366p.

[14] L. Szirmay-Kalos, T. Umenhoffer, Displacement mapping on the GPU - State of the Art, Computer Graphics Forum 27 (1) (2008) 1567–1592.

[15] H. Schäfer, M. Nießner, B. Keinert, M. Stamminger, C. T. Loop, State of the art report on real-time rendering with hardware tessellation., in: Eurographics (State of the Art Reports), 2014, pp. 93–117.

[16] E. Angel, D. Shreiner, Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL, 6th Edition, Addison-Wesley Publishing Company, USA, 2011.

[17] O. Šťava, B. Beneš, M. Brisbin, J. Křivánek, Interactive terrain modeling using hydraulic erosion, in: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '08, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2008, pp. 201–210.
url: http://dl.acm.org/citation.cfm?id=1632592.1632622