

Simulation-driven gym layout optimization

Andrej Jočić (marctan), Matic Stare (marcsin) in Martin Krpan

Collective behaviour course research seminar report

November 18, 2023

Iztok Lebar Bajec | izredni professor | mentor

TODO

TODO1 | TODO2

Introduction

The present work deals with gym layout optimisation. A common occurrence in gyms at peak hours is over-crowding of popular equipment and lots of wandering around searching for a free machine. Through simulating gym-goer behavior, we aimed to identify the most effective arrangement of exercise equipment to improve the flow, accessibility, and overall customer satisfaction. Our goal is to offer practical insights for gym owners, managers, and designers seeking to optimize their facility layout for the benefit of their clients and business success. We limit ourselves to the case of muscle hypertrophy training with resistance exercises.

Strength training. A typical workout routine splits the body into several muscle groups, cycling through them on a per-workout basis. This rotation can be done on a weekly schedule, or the workouts can be weekday-independant for greater flexibility. Popular splits include:

- **Push-pull-legs (PPL)** - Push muscles are those involved in pushing movements, such as the chest, frontal deltoids (shoulders), and triceps. Pull muscles are those involved in pulling movements, such as the back musculature, rear deltoids, and biceps. Legs are the lower body muscles, such as the quadriceps, hamstrings, and calves.
- **Upper-lower** - Upper body muscles are those above the waist, such as the chest, back, shoulders, and arms. Lower body muscles are those below the waist, such as the quadriceps, hamstrings, and calves.
- **Full body** - All muscle groups are trained in every workout.

The exercises in a particular workout are then selected to stimulate each muscle in the chosen group, ideally cycling through them to avoid fatigue-induced form breakdown. Each individual exercise usually consists of 2-4 sets (where a movement is repeated some number of times), and short breaks in between.

Related work. The only publication related to gym layout optimization we could find was ref. [1], which assumed all gym clients had fixed-order workout routines (including ones with weight loss as a goal) and optimised a circular gym layout to minimise backward movement. Unfortunately this does not give a good foundation for our work, as the assumptions differ too much from our model.

Thus we started from a crowd modelling review [2] for basic model design principles. We found two useful articles about incremental urban layout optimisation [3, 4] for inspiration.

Methods

Gym layout representation. For simplicity, we will represent a gym layout as a rectangular area with pre-set square equipment locations of equal size, distributed similarly as isles in a grocery store. Each location can be occupied by any piece of exercise equipment, and is never left empty. One corner of the area will serve as locker area entrance, where agents enter and exit the gym.

Note that such a discretization does not allow us to model shared resources such as the free weights next to a set of benches, or compound machines such as a cable harness with multiple weight stacks. Our set of supported machines won't include any multi-purpose equipment, such as a squat rack with a pull-up bar.

Gym-goer behaviour model.

Procedural generation of a tropic island and coral reef

In computer graphics there is frequent need for displaying large vistas of natural looking terrain. Designing such terrain by hand is typically time consuming. With procedural generation, on the other hand, larger areas of natural looking terrain can be generated with or without minimal intervention in a relatively short time. In this work we present a process of procedural generation of a tropical island with the associated coral reef. We start by generating a heightmap for the base terrain. The heightmap is then transformed by simulating the processes of hydraulic and thermal erosion to achieve a more natural look of the terrain. As coral reefs often grow around tropical islands, we also simulate their growth as part of the last step. Real-time visualization is enabled during the simulation, so that one can observe the evolution of the terrain. Here we dynamically apply textures to the terrain based on its local characteristics. The result is a natural looking model of the textured tropical island and coral reef.

Procedural generation | Terrain generation | Thermal and hydraulic erosion | Coral reef | Simulation | GPU

Customer pool generation. A simulation cycle (eg. one week of traffic) will begin with generating a pool of customers and their workout routines. We will sample from a small pre-defined set of routines with some probability distribution. For a start, we will parametrise the distribution ad hoc, based on our observations of people in our local gyms. Each agent will then cycle through the muscle groups in their routine on each consecutive workout until the end of the simulation. The rate at which agents enter the gym will vary throughout the day, with a peak during the evening hours (this peak should be especially pronounced on work days).

Workout simulation. An agent will enter the gym with a multiset of muscles to train (where multiplicity is necessary for muscles that will be trained on multiple pieces of equipment for diversity of stimulation). The agent's goal is to exhaust this multiset as quickly as possible and exit the gym. These multisets will be constructed ad hoc, in keeping with general workout programming practice. For example, a push workout could be associated with $\{chest, chest, triceps, frontal deltoids\}$. There must also be a maximum time limit, at which point the agent will leave regardless.

Optimisation criterion.

- sets per minute
- crowdeness
- ability to finish workout plan in available time
- cost of equipment/installation? (input to optimizer?)

Simulation. We will use Python's Mesa library for simulation, analysis, and visualization of agent-based models.

Generiranje začetnega terena. Ker je bil cilj generirati teren v obliki otoka smo želeli, da bo teren na sredini višinske karte višji kot ob robovih. To smo dosegli z generiranjem višinske karte, ki definira teren v obliki stožca z nelinearno klančino:

$$h_A(x, y) = \begin{cases} \kappa_H \left(1 - \left(\frac{r(x, y)}{\kappa_R}\right)^{\kappa_P}\right); & \text{če } r(x, y) < \kappa_R \\ 0; & \text{sicer,} \end{cases} \quad [1]$$

kjer κ_H označuje višino stožca, κ_R polmer stožca in κ_P potenco, ki nadzoruje obliko klančine,

$$r(x, y) = \sqrt{(x - x_C)^2 + (y - y_C)^2} \quad [2]$$

pa razdaljo točke (x, y) do središča osnovne ploskve stožca (x_C, y_C) .

Za izboljšanje verodostojnosti terena smo v naslednjem koraku generirali dodatno višinsko karto, ki smo jo potem združili s prvo. Za generiranje te smo uporabili šum Simplex [5], ki ga lahko implementiramo v poljubno mnogo dimenzijah in s katerim lahko dosežemo hribovit videz. V naši rešitvi uporabljamo C# implementacijo, ki je del prostot dostopnega paketa MinePackage¹. Za boljši videz terena smo uporabili več oktav šuma². Izraz oktava v glasbeni teoriji označuje interval med višinama dveh tonov, od katerih ima eden dvojno oziroma polovično frekvenco drugega. V primeru funkcije šuma z višanjem frekvence šuma (skala šuma) zmanjšujemo tudi amplitudo. Na eni višinski karti smo uporabili več oktav šuma tako, da smo vse oktave skupaj seštel:

$$h_B(x, y) = \sum_{i=0}^N \eta(x2^i, y2^i) \frac{1}{2^i}, \quad [3]$$

kjer je η vrednost šuma Simplex v koordinatah (x, y) in N število oktav.

Pri združevanju višinskih kart stožca in šuma smo se zgledovali po orodju World-machine³, kjer je možno več višinskih kart združiti v eno na več načinov. Najboljše rezultate smo dobili, ko smo višinski karti združili s potenciranjem:

$$h_C(x, y) = h_A(x, y)^{1+h_B(x, y)\kappa_N}, \quad [4]$$

kjer je h_A višina v višinski karti stožca, h_B pa višina v višinski karti šuma Simplex. Vpliv šuma določa parameter κ_N .

Ker smo želeli, da otok dobi videz vulkana, smo dodali možnost generiranja vulkanskega kraterja. Generirali smo ga tako, da smo vse vrednosti v združeni višinski karti h_C , ki so višje od določenega praga, preko praga preslikali navzdol. Postopek prikazuje

¹sourceforge.net/projects/minepackage

²www.redblobgames.com/maps/terrain-from-noise

³www.world-machine.com

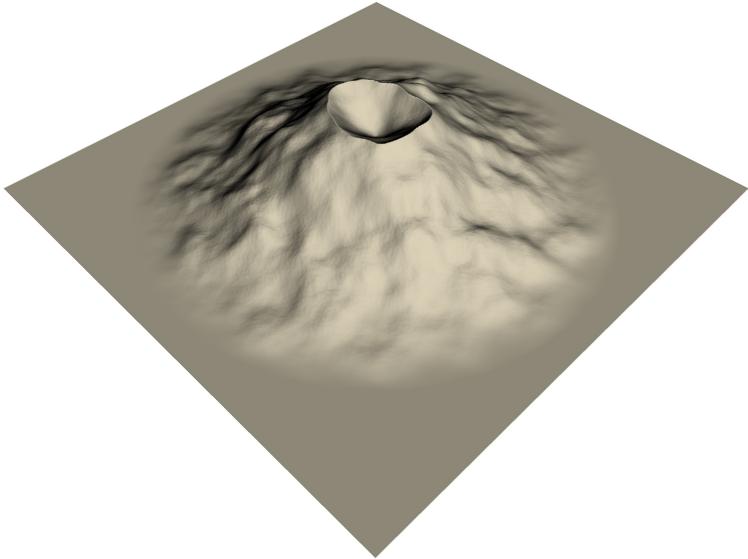


Figure 1. Začetni relief terena.

enačba (5), kjer je h končna višinska karta začetnega terena, κ_C pa višina pragu in s tem višina, na kateri se ustvari vulkanski krater. Slika 1 pa prikazuje relief generiran z uporabo končne višinske karte začetnega terena.

$$h_0(x, y) = \begin{cases} 2\kappa_C - h_C(x, y); & \text{če } h_C(x, y) > \kappa_C \\ h_C(x, y); & \text{sicer} \end{cases} \quad [5]$$

Erozija. Kljub temu, da že začetni teren spominja na otok, je njegov videz še nepreprečljiv. Simuliranje naravnih procesov, kot sta hidravlična in termična erozija, nam omogoča videz izboljšati. Hidravlično erozijo smo implementirali po viru [6], termično pa po [7]. Vira opisujeta metodi simuliranja teh procesov na GPE.

Hidravlična erozija. 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.

Za opis simulacije bomo uporabljali naslednje pomembne količine: višina terena, količina tekoče vode, količina raztopljenega sedimenta, odtočni tokovi vode, in hitrost vode. Pri tem se vsaka od količin hrani za vsako točko (celico) terena. S t dodatno označujemo trenutni čas in s τ časovni korak ene iteracije simulacije. Vsaka iteracija je razbita na več zaporednih korakov: dodajanje vode, pretakanje vode po terenu, izračun odloženega in stopljenega sedimenta, prenos sedimenta, difuzija sedimenta, izhlapevanje vode. Vsak korak uporabi količine iz prejšnjega in jih posodobi. Nekatere količine se izračunajo v večih podkorakih, zato te indeksiramo z ', " itd., da med seboj ločimo vrednosti v različnih podkorakih. Sosednje celice trenutne celice označujemo z L (leva), R (desna), T (gornja), B (spodnja).

Za simuliranje pretakanja vode uporabljamо prirejen model, ki povezuje sosednje celice z navideznimi cevmi, po katerih se med njimi izmenjuje voda [8]. Ob začetku simulacije privzemamo, da je višina vode po celotnem terenu ničelna; $d_0 = 0$. Dež predstavlja nastanek nove vode, kar za obravnavano celico (x, y) izračunamo kot

$$d_t' = d_t + r\tau, \quad [6]$$

kjer r označuje količino dežja, ki prispe v celico na enoto časa. Zaradi velikosti terena namreč ni smiseln simulariti dežja kot posameznih kapelj, temveč kot uniformno rast količine tekoče vode po posamezni celici terena.

Za vsako celico hranimo njene odtočne tokove L, R, T, B do štirih sosednjih celic, njeni pritočni tokovi pa so ustrezeni odtočni tokovi obravnavani celici sosednjih celic. V vsaki iteraciji simulacije se tok vode v odtočnih tokovih pospeši zaradi razlike v hidrostaticnem tlaku med celicami, količina tekoče vode pa se nato izračuna s kopiranjem vseh tokov v virtualnih cevih.

Odtočni tok v smeri proti levi celici za obravnavano celico izračunamo po enačbi:

$$L_t' = \max \left\{ 0, L_t + \tau \kappa_A \frac{\kappa_g \Delta h_t^L}{\ell} \right\}, \quad [7]$$

kjer κ_A predstavlja površino prereza cevi, ℓ dolžino cevi med dvema sosednjima celičama, κ_g gravitacijski pospešek in Δh_t^L razliko višin med obravnavano ter levo sosednjo celico $(x - 1, y)$, ki se izračuna po enačbi:

$$\Delta h_t^L = h_t + d_t' - h_t^L - d_t^{L'}. \quad [8]$$

Odtočni tokovi v smeri sosed R, T, B se izračunajo na enak način. Ker izračun odtočnih tokov upošteva višinske razlike do vsake sosedne individualno, obstaja možnost, da bi iz obravnavane celice odteklo več vode, kot je ta vsebuje. S tem bi količina tekoče vode v celici po izračunu postala negativna. Težavo se rešuje z omejitvijo moči vsakega izmed odtočnih tokov v odvisnosti od količine tekoče vode v celici in predvidene moči vseh odtočnih tokov [6, 9]. V primeru odtočnega toka v smeri proti levi celici se tako odtočni tok v naslednjem koraku simulacije izračuna kot:

$$L_{t+\tau} = L_t' \min \left\{ 1, \frac{d_t' \ell^2}{\tau \sum \mathbf{O}_t'} \right\}, \quad [9]$$

kjer \mathbf{O}_t' predstavlja množico vseh odtočnih tokov obravnavane celice $\{L_t', R_t', T_t', B_t'\}$ izračunanih z uporabo enačbe (7), $\sum \mathbf{O}_t'$ pa njihovo skupno vsoto.

Pretakanje vode po navideznih ceveh se odraža v spremembji količine tekoče vode

$$d_t'' = d_t' + \frac{\sum \mathbf{I}_{t+\tau} - \sum \mathbf{O}_{t+\tau}}{\ell^2} \tau, \quad [10]$$

kjer sta

$$\mathbf{I}_{t+\tau} = \{R_{t+\tau}^L, L_{t+\tau}^R, B_{t+\tau}^T, T_{t+\tau}^B\}$$

in

$$\mathbf{O}_{t+\tau} = \{L_{t+\tau}, R_{t+\tau}, T_{t+\tau}, B_{t+\tau}\}$$

po vrsti množici pritočnih in odtočnih tokov obravnavane celice v naslednjem časovnem koraku. Pri simuliraju pretakanja vode moramo upoštevati robne pogoje, ki izhajajo iz dejstva da je teren končen. Iztekanje vode s terena omejimo tako, da vsaki cevi, ki nima sosednje celice, postavimo vrednosti ustreznih odtočnih in pritočnih tokov na 0. Na primer: za celice brez levega soseda $(0, y)$ postavimo tako levi odtočni tok L , kot tudi desni pritočni tok R^L na 0.

Voda med pretakanjem zaradi erozije pretvori del terena v sediment

$$S = \kappa_S \| \mathbf{v}_t \| \sin \alpha, \quad [11]$$

ki ga nato prenaša in odlaga po terenu. V enačbi (11) κ_S predstavlja konstanto kapacitete sedimenta, α pa lokalni naklon terena v celici. Slednjega za obravnavano celico izračunamo kot $\alpha = \arccos(\hat{\mathbf{n}}_t \cdot \mathbf{k})$. Pri tem $\hat{\mathbf{n}}_t$ predstavlja normalo na teren, ki jo pridobimo kot

$$\mathbf{m} = \mathbf{i} \frac{h_t^R - h_t^L}{\kappa_M} + \mathbf{j} \frac{h_t^T - h_t^B}{\kappa_M} + \mathbf{k}, \quad \hat{\mathbf{n}}_t = \frac{\mathbf{m}}{\| \mathbf{m} \|}, \quad [12]$$

kjer je κ_M razdalja med sosednjimi celicami terena, \mathbf{i} , \mathbf{j} , \mathbf{k} pa po vrsti enotski vektorji, ki kažejo v smereh osi x, y in z. Hitrost vode v obravnavani celici $\mathbf{v}_t = \langle v_x, v_y \rangle$, ki jo potrebujemo v enačbi (11) pa izračunamo s pomočjo odtočnih in pritočnih tokov celice:

$$v_x = \frac{R_{t+\tau}^L - L_{t+\tau} + R_{t+\tau} - L_{t+\tau}^R}{\ell(d_t' + d_t'')}, \quad [13]$$

$$v_y = \frac{T_{t+\tau}^B - B_{t+\tau} + T_{t+\tau} - B_{t+\tau}^T}{\ell(d_t' + d_t'')}. \quad [14]$$

Na zelo položnih delih, kjer se α približuje 0, je vrednost S zelo nizka. Na takšnih predelih proces erozije ne bo imel občutnega učinka. To lahko omilimo na tak način, da α navzdol omejimo s poljubno minimalno vrednostjo. Na podlagi vrednosti S in raztopljenega sedimenta iz predhodne iteracije simulacije, s_t , se odločimo, ali se bo del sedimenta odložil ter pretvoril nazaj v teren ali ne. Ko velja $S > s_t$, se zgodi proces erozije in del terena se raztopi, v nasprotnem primeru se del raztopljenega sedimenta odloži in pretvori nazaj v teren. Novo višino terena in količino raztopljenega sedimenta izračunamo po enačbah:

$$h_{t+\tau} = \begin{cases} h_t - \kappa_E(S - s_t); & \text{če } S > s_t \\ h_t + \kappa_D(S - s_t); & \text{sicer} \end{cases} \quad [15]$$

$$s_t' = \begin{cases} s_t + \kappa_E(S - s_t); & \text{če } S > s_t \\ s_t - \kappa_D(S - s_t); & \text{sicer}, \end{cases} \quad [16]$$

kjer je κ_E konstanta topljenja materiala, κ_D pa konstanta odlaganja materiala.

Raztopljeni sediment prenaša vodni tok, kar opisuje advekcijska enačba $\frac{\delta s}{\delta t} + (\mathbf{v}_t \nabla s) = 0$. Enačbo rešujemo z Eulerjevim korakom nazaj v času, kar pomeni da količini raztopljenega sedimenta v obravnavani celici (x, y) priredimo vrednost, ki je odvisna od vodnega toka v njej

$$s_t''(x, y) = s_t'(x - v_x \tau, y - v_y \tau), \quad [17]$$

kjer koordinate $(x - v_x \tau, y - v_y \tau)$ omejimo na območje terena. Kot je že Maške [9] omenil, postopek ne deluje zadovoljivo saj je možno, da se v enem koraku količina sedimenta iz ene celice preslika v več celic. Opazili smo tudi, da se sediment pri nižjih hitrostih vode večkrat prenaša v ravnih črtah. Vzrok za to je, da se sediment v območjih nizke hitrosti prenaša med celicami, ki so neposredne sosedne. To daje polju sedimenta grob in nenanaren videz, zato ga mi dodatno gladimo s simuliranjem difuzije sedimenta.

Difuzijo simuliramo z dvodimensonalno Laplacevo enačbo, ki jo numerično rešujemo z Jakobi iteracijami [10]. Izračun iteracije prikazuje enačba

$$s_{t+\tau} = \frac{s_t^{L''} + s_t^{R''} + s_t^{T''} + s_t^{B''}}{n_W}, \quad [18]$$

kjer n_W predstavlja število celic izmed L, R, T in B, ki držijo vodo ($d_t'' > 0$). 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'' na 0.

V zadnjem koraku vsake iteracije simulacije hidravlične erozije upoštevamo, da se količina tekoče vode spreminja tudi zaradi izhlapevanja. V našem delu predpostavljamo, da je temperatura ozračja med simulacijo konstantna, zato hitrost izhlapevanja določamo s konstanto κ_T . Proces izhlapevanja opisuje enačba

$$d_{t+\tau} = d_t''(1 - \kappa_T \tau). \quad [19]$$

Termična erozija. Hidravlična erozija oblikuje grob teren z veliko ostrimi robovi, kar je opazno predvsem na pobočjih ob vodnih strugah, ki se neprestano poglabljajo. S simuliranjem termične erozije, ki modelira proces razgradnje kamnin v drobir zaradi temperturnih sprememb in njegovo nadaljne razsipanje, teren zgladimo in izboljšamo njegov videz. Podobno kot za hidravlično lahko tudi za termično erozijo uporabljamo model virtualnih cevi, le da v tem primeru cevi prenašajo drobir (terena) namesto vode. Vendar se običajno [7, 9] zaradi boljšega rezultata pri termični eroziji namesto von Neumannove okolice, ki zaobjema zgolj najbližje štiri sosedne obravnavane celice, uporablja Moorova okolica, ki zajema vseh osem najbližjih sosednjih celic $\mathbf{N} \in \{L, R, T, B, TL, TR, BL, BR\}$. To bi lahko sicer storili že pri hidravlični eroziji, vendar menimo, da bi povzdignilo kompleksnost implementacije brez večjih izboljšav pri rezultatu.

Količina drobirja, ki jo lahko v eni iteraciji prenesemo v vsako od sosednjih celic, je omejena z $M_t = \frac{H_t}{2} \tau \kappa_M$, kjer je $H_t = h_t - \min \{h_t^i | i \in \mathbf{N}\}$ razlika v višini terena v obravnavani celici in najnižjo višino terena v njej sosednjih celicah. Če to vrednost presežemo, začne algoritem oscilirati [7]. Nadaljni proces smo glede na vir [7] nekoliko poenostavili; vir upošteva tudi lokalno trdnost terena, ki je v našem delu ne.

Drobir se pod vplivom gravitacije razsipa med nižje ležeče sosednje celice, ki z obravnavano celico tvorijo kot naklona, večji od kota mirovanja κ_α . To je kritičen kot pobočja, ki ga lahko nek zrnat material še tvori. Kadar klančina preseže kot mirovanja, material zdrsne.

Kote naklona med obravnavano in posamezno sosednjo celico $i \in \mathbf{N}$ izračunamo kot $\alpha_t^i = \arctan \frac{h_t - h_t^i}{\kappa_M}$. Na podlagi teh sestavimo množico, ki vsebuje vse sosednje celice z naklonom, večjim od kota mirovanja κ_α :

$$\mathbf{R}_t = \{i \in \mathbf{N} | \alpha_t^i > \kappa_\alpha\}. \quad [20]$$

V vsako celico iz množice \mathbf{R}_t na podlagi višinske razlike z obravnavano celico prenesemo temu proporcionalno količino drobirja, v ostale celice ga ne prenašamo. Na primer: količino drobirja, ki se prenese v levo sosednjo celico, če ta spada v množico \mathbf{R}_t , izračunamo z enačbo:

$$L_t = M_t \frac{h_t^L}{\sum_{i \in \mathbf{R}_t} h_t^i}. \quad [21]$$

Zaradi paralelnega izvajanja simulacije drobirja ne prestavimo v celice takoj, temveč za ta proces uporabimo navidezne cevi. Končno višino vsake celice izračunamo podobno kot pri hidravlični eroziji, in sicer tako, da k trenutni višini prištejemo vse pritočne tokove in odštejemo vse odtočne:

$$h_{t+\tau} = h_t + \sum \mathbf{M}_t^{\text{in}} - \sum \mathbf{M}_t^{\text{out}}, \quad [22]$$

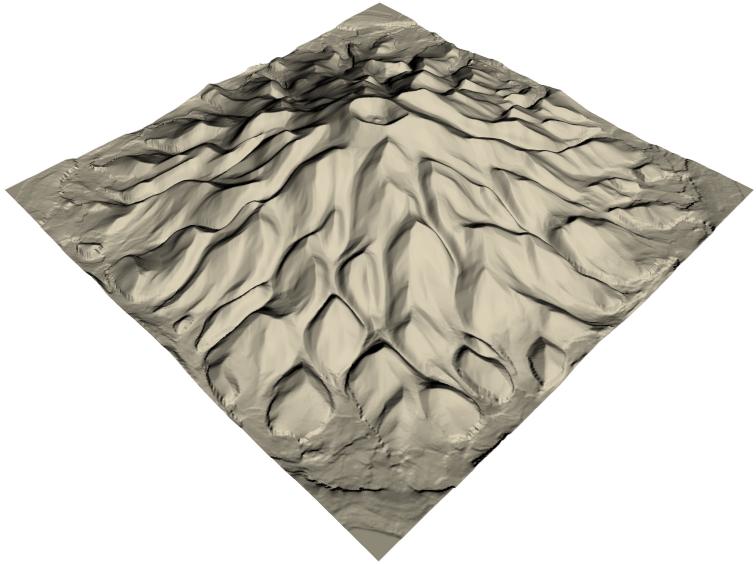


Figure 2. Relief terena po hidravlični in termični eroziji.

le da tokrat z

$$\mathbf{M}_t^{\text{out}} = \{L_t, R_t, T_t, B_t, TL_t, TR_t, BL_t, BR_t\}$$

označimo drobir, ki se iz obravnavane celice razsipa v sosednje celice preko ustreznih virtualnih cevi (odtočni tokovi drobirja), z

$$\mathbf{M}_t^{\text{in}} = \langle R_t^{\text{L}}, L_t^{\text{R}}, \dots, TL_t^{\text{BR}} \rangle$$

pa pritočne tokove drobirja (odtočne tokove drobirja usreznih sosed). Primer terena po končani simulaciji prikazuje slika 2.

Rast koralnega grebena. V tropskih morjih okoli otokov, predvsem takšnih vulkanskega nastanka, velikokrat rastejo korale, ki skozi čas tvorijo koralne grebene [11]. Korale rastejo v plitvinah toplejih morij. Za rast potrebujemo svetlobo, saj ožigalkarji živijo v sožitju s fotosinteznimi algami zoooksantele (angl. *zooxanthellae*) [11]. Sestavljeni so iz kolonij majhnih polipov, katerih apnenčasti zunanjki skeleti se nalagajo in tvorijo koralne grebene. K tvorbi grebena pripomorejo tudi ostali organizmi, ki živijo v koralnem ekosistemu. S simuliranjem rasti koralnih grebenov želimo videz otoka še bolj približati resničnosti in ga narediti zanimivejšega.

Model rasti koralnega grebena predstavljen v nadaljevanju temelji na modelu Nakamure in Nakamorijs [12]. Avtorja sta predstavila model rasti koralnega grebena na podlagi intenzitete svetlobe ter koncentracije anorganskega ogljika v vodi, a za dvodimenzionalne simulacije rasti v pogledu s strani. Ker za naše potrebe potrebujemo pristop, s katerim je mogoče rast simulirati v pogledu iz zraka, smo njun model priredili in poenostavili.

V opisu bomo uporabljali dve pomembni količini: višina terena (morskega dna) in koncentracija anorganskega ogljika v vodi. Obe količini se hranita za vsako točko oziroma celico terena. Vsaka iteracija simulacije pa je razbita na dva zaporedna koraka; rast koral in difuzijo koncentracije anorganskega ogljika.

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

$$g_t = w_t - h_t, \quad [23]$$

kjer w_t označuje globalno višino morske gladine v trenutni iteraciji simulacije. Korale zaradi plimovanja in ostalih dejavnikov namreč običajno rastejo le do določene globine vode κ_G . Stopnjo kalcifikacije koral izračunamo z uporabo enačbe

$$c_t = \begin{cases} O_t^2 \kappa_B e^{-g_t \kappa_A}; & \text{če } g_t < \kappa_G \\ 0; & \text{sicer,} \end{cases} \quad [24]$$

kjer je O_t vrednost koncentracije anorganskega ogljika v vodi v trenutni iteraciji simulacije, κ_B konstanta hitrosti kalcifikacije in κ_A konstanta absorbcije vode. V prvi iteraciji simulacije je vrednost O_0 vsake celice postavljena na O_0 .

Korale za rast porabijo določeno količino anorganskega ogljika, zato moramo vrednost njegove koncentracije v vodi zmanjšati v sorazmerju s stopnjo kalcifikacije. Spremembo izračunamo po enačbi

$$O_t' = \max \left\{ 0, O_t - \frac{c_t \tau}{g_t} \kappa_U \right\}, \quad [25]$$

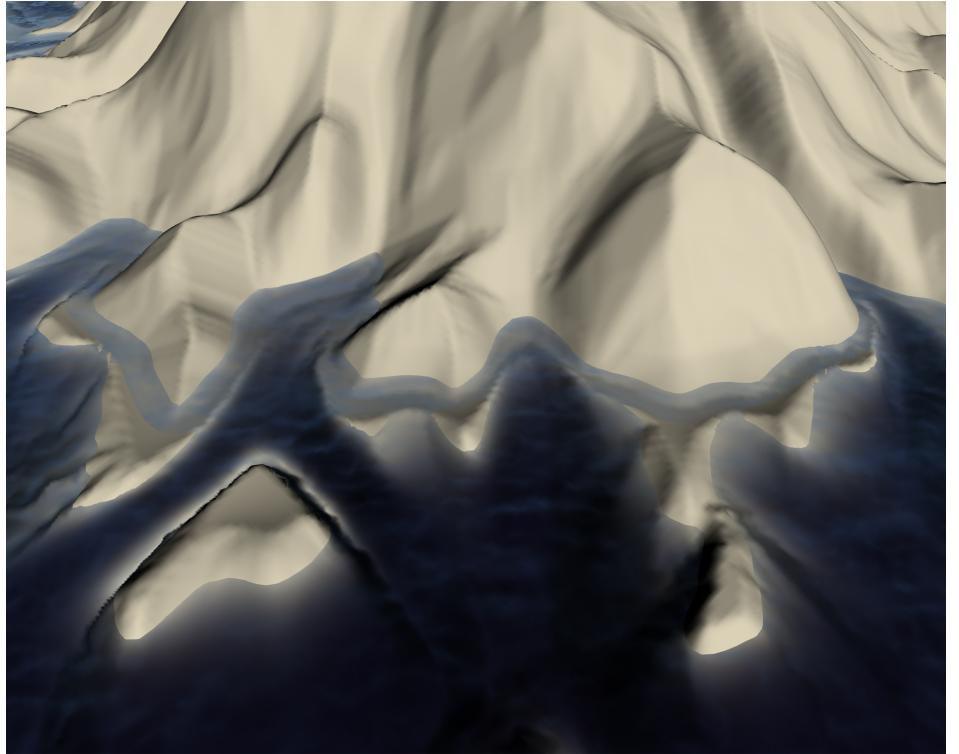


Figure 3. Koralni greben ob obali otoka. Morska gladina je prikazana za boljšo predstavo.

kjer κ_U nadzoruje stopnjo porabe anorganskega ogljika. S tem, da v enačbi (25) stopnjo kalcifikacije delimo z višino vode, dosežemo, da se pri enaki stopnji rasti koral koncentracija anorganskega ogljika v vodi zmanjša primerno količini vode nad to točko na terenu.

Na koncentracijo anorganskega ogljika v vodi vpliva tudi plimovanje in mešanje vode. Ker v našem primeru simulacija teče v časovnem razponu več tisoč let in to na velikem terenu, lahko ta vpliv prikažemo s preprosto difuzijo, podobno kot smo to storili s sedimentom pri hidravlični eroziji

$$O_{t+\tau} = \frac{O_t^{L'} + O_t^{R'} + O_t^{T'} + O_t^{B'}}{n_U}, \quad [26]$$

kjer n_U predstavlja število najbližjih sosednjih celic v von Neumannovi okolici, ki ležijo pod morsko gladino ($g_t > 0$) [10]. Pri robnih celicah, celicah, ki ležijo tik ob robu terena in zato nimajo levega ali desnega, ali zgornjega, ali spodnjega soseda, Jakobi-jeve iteracije ne računamo, temveč $O_{t+\tau}$ priredimo vrednost O_0 . Jakobi-jeve iteracije prav tako ne računamo pri celicah, ki so nad gladino morja, a za razliko od prej v teh primerih $O_{t+\tau}$ priredimo vrednost 0, ker celice ne izpolnjujejo pogojev za nastanek koral.

V našem delu podatkov o višini koralnega grebena ne hranimo ločeno, temveč ga štejemo kot teren. Rast grebena (in s tem dvig višine terena) je neposredno odvisna od stopnje kalcifikacije. Višino terena po iteraciji simulacije izračunamo z enačbo

$$h_{t+\tau} = h_t + c_t \tau. \quad [27]$$

Rast koralnega grebena in posledično spremembu višine terena vpliva tudi na globalno višino morske gladine. Na spremembe v višini morske gladine seveda vplivajo tudi drugi dejavniki. V našem delu, po vzoru Nakamure in Nakamora [12] privzemamo konstantno rast višine morske gladine. Globalno višino morske gladine ob zaključku iteracije rasti koralnega grebena tako izračunamo po enačbi

$$w_{t+\tau} = w_t + \kappa_W \tau, \quad [28]$$

kjer je κ_W konstanta rasti višine morske gladine. Vpliv simulacije rasti koralnega grebena na končni izgled proceduralno generiranega otoka prikazuje slika 3.

Implementacija. Simuliranje naravnih procesov kot so hidravlična in termična erozija je običajno računsko zahtevna operacija. Čas izvajanja simulacije je pomemben še posebno takrat ko želimo rezultate opazovati v realnem času, zato v teh primerih posegamo po GPE. V naši implementaciji uporabljamo okolje Unity in senčilnike

*ComputeShader*⁴. Simulacijo izvajamo nad višinsko karto, zato podatke o slednji shranjujemo v teksturi. Za hranjenje višinske karte v osnovi potrebujemo zgolj en barvni kanal, zato je teksura višinske karte običajno sivinska. V naši implementaciji za posamezen barvni kanal uporabljamo 32-bitno predstavitev s plavajočo vejico (*float*).

Vsek izmed predstavljenih algoritmov hrani podatke o nekaterih količinah (npr. višina vode), ki se uporabljajo med posameznimi koraki in iteracijami simulacije. Ti podatki so vezani na posamezne točke v višinski karti, zato jih hranimo na enak način kot višinsko karto, tj. v teksturi. Za shranjevanje vsake količine tako kot za višinsko karto potrebujemo en barvni kanal. Ker v naši implementaciji uporabljamo tekštore s štirimi kanali, ki se v senčilniku preslika v *float4*, nam to omogoča v eni teksuri shranjevati podatke o več različnih količinah.

Razvili smo tri senčilnike, pri čemer prvi izvaja hidravlično erozijo, drugi termično erozijo, tretji pa skrbi za rasti koral. Vsak senčilnik simulacijo pripadajočega procesa izvaja v več iteracijah, v posamezni iteraciji pa določene tekštore obdela večkrat. ComputeShader senčilnik je sestavljen iz več funkcij, med katerimi lahko poljubne označimo kot jedrne. Poženemo ga tako, da poženemo eno od jedrnih funkcij. Z uporabo več jedrnih funkcij lahko tekšturo z enim senčilnikom obdelamo v več različnih korakih. Naša koda je organizirana temu primerno, napisana pa je v jeziku HLSL različice DirectX 11.

V prvem koraku naše implementacije generiramo višinsko karto začetnega terena in jo shranimo v obliki tekštore. Na tej točki je pripadajoča tekstura shranjena v glavnem pomnilniku računalnika, zato jo pred začetkom izvajanja simulacije naravnih procesov prenesemo v pomnilnik GPE. V pogonu Unity so tekštore, ki so shranjene izključno v pomnilniku GPE, objekti tipa RENDERTEXTURE. V senčilnikih uporabljamo le takšen tip tekštore; tekstur ne prenašamo med glavnim pomnilnikom računalnika ter pomnilnikom GPE, saj je to časovno potratna operacija. Ker se vizualizacija prav tako izvaja v senčilnikih, je prenašanje nepotrebno. Ker pa rezultatov posameznega koraka iteracije ne smemo pisati v isto tekšturo, ker bi zaradi paralelnosti izvajanja s tem lahko uničili rezultate ostalih niti, moramo poleg vsake teksture hraniti še njeno kopijo. Vsaki jedrni funkciji moramo tako pred izvajanjem nastaviti potrebne vhodne in izhodne tekštore.

Iteracije simulacije izvajamo v zanki, v kateri v določenem vrstnem redu poženemo jedrne funkcije ustreznega senčilnika. V namen nadzora simulacije smo implementirali enostaven grafični uporabniški vmesnik. Z njim lahko spremojamo določene parameterje in vklapljammo ter izklapljammo posamezne korake simulacije. Vrstni red zagona senčilnikov načeloma ni pomemben, a moramo paziti, da tistega, ki je namenjen rasti koral, ne uporabljammo hkrati z hidravlično oz. termično erozijo. Koralnega grebena namreč ne hranimo ločeno ampak kot teren, erozija (tako hidravlična kot termična) pa teren preoblikuje in posledično spremeni videz koralnega grebena, kar ni zaželeno.

Vizualizacija. Višinska karta v vsakem pikslu hrani višino pripadajoče točke na terenu. Za prikaz terena uporabljamo metodo imenovano odmiki (angl. *displacement mapping*) [13]. Njena težava je, da za prenos podrobnosti višinske karte na model potrebuje veliko število ogljišč. V optimalnem primeru bi bil model predstavljen z enakim številom ogljišč, kot ima višinska karta piksov. V praksi to pomeni modele z izredno visokim številom ogljišč in posledično velikim številom trikotnikov. Problem lahko rešimo z uporabo strojne teselacije (angl. *hardware tessellation*) [14], ki model v osnovi predstavi z majhnim številom ogljišč in trikotnikov, te pa nato z uporabo senčilnikov dinamično deli na več manjših, pri čemer ustvari več ogljišč in več podrobnosti. V naši implementaciji uporabljamo strojno teselacijo fiksne stopnje.

Osvetlitev izračunavamo z Lambertovim modelom [15, str. 267–268]

$$c = (\hat{\mathbf{l}} \cdot \hat{\mathbf{n}}) c_m c_l, \quad [29]$$

kjer osvetlenost v neki točki izračunamo na podlagi vpadnega kota svetlobe $\hat{\mathbf{l}}$ in normalne na teren v tej točki $\hat{\mathbf{n}}$. Parametra c_m in c_l po vrsti predstavljata barvo materiala v točki in barvo vira svetlobe. Za osvetljevanje uporabljamo smerno oziroma oddaljeno luč [15, str. 264], ki nima pozicije. Luč definira samo vektor smeri, barvo in intenziteto. V enačbi (29) je intenziteta že všteta v barvo luči.

Naš cilj je bil generiranje tropskega otoka, k čemur veliko pripomorejo tekštore. Osnovne štiri tekštore, ki jih uporabljamo, so: trava (zelene površine), skalovje, pešek in morsko dno. Texture smo dobili na spletni strani Textures⁵. V senčilniku na podlagi višine in naklona terena ter višine vode v vsaki točki terena vzorčne tekštore združujemo in ob upoštevanju osvetlitve izračunamo barvo v tej točki. Logika združevanja tekštore temelji na opažanjih iz narave in osebne predstave o videzu tropskega

⁴docs.unity3d.com/Manual/ComputeShaders

⁵www.textures.com



Figure 4. Atol, generiran s pomočjo vulkanskega kraterja brez simuliranja posledic erozije (levo) vulkanski otok generiran z simulacijo hidravlične erozije, termične erozije in rasti koralnega grebena.

otoka. Strma pobočja so redko poraščena, medtem ko na ravninah prevladujejo zeleni 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 namesli teksturo peska, vendar le tam, kjer je obala dovolj položna. Na koncu smo pod gladino vode dodali še teksturo morskega dna. Za doseganje mehkih prehodov med teksturami smo si pomagali s funkcijo SMOOTHSTEP, ki je del jezika HLSL. Funkcija nam omogoča definicijo poljubno širokega pasu, v katerem se dve teksturi mešata, izven pasu pa je izbrana ena izmed njiju.

Ker je otok v naravnem okolju obkrožen z morjem, smo dodali prikaz tudi slednjega. Morja v okolici tropskih otokov so običajno zelo čista zato v plitvih predelih pogosto vidimo dno. Intenziteta svetlobe v neki točki v globini morja je funkcija dolžine poti svetlobnih žarkov skozi vodno maso [16]. V naši aplikaciji na teren in morje v pretežni meri gledamo od zelo daleč, zato pri izračunu barve morske gladine zanemarjamо dejstvo, da se svetlobni žarki, ko iz zraka prehajajo v vodo, lomijo kar vpliva na dolžino njihove poti skozi vodno maso. Barvo morske gladine tako določamo zgolj s spremenjanjem prosojnosti osnovne, temno modre barve v odvisnosti globine vode v obravnavani točki

$$c_w \cdot a = e^{-g_t \kappa_A}, \quad [30]$$

kjer κ_A nadzoruje moč slabljenja svetlobe (motnost vodne mase). Izračun opravlja ločen senčilnik, ki skrbi zgolj za izračun barve vsake točke na površini, ki predstavlja morsko gladino.

Rezultati

Uporabljamo višinske karte velikosti 512×512 točk. Začetni teren se, predvsem zato ker je opravilo prepuščeno centralni procesni enoti, v povprečju zgradi v 3 s. Ker se ta korak izvede samo enkrat v celotnem procesu to ni moteče. Nadaljnja simulacija naravnih procesov se izvaja hitro; s frekvenco 94 Hz (iteracij na sekundo), ko se izvaja zgolj proces hidravlične erozije, 108 Hz, ko se izvaja zgolj proces termične erozije, in 85 Hz, ko se izvajata oba procesa erozije sočasno. Simulacije rasti koralnega grebena ne izvajamo sočasno z erozijo, ampak ločeno, ta se izvaja s frekvenco 108 Hz. Najhitrejša je vizualizacija, ki takrat, ko se ne izvaja nobena simulacija, dosega prikaz z 124 fps (slik na sekundo). V celoti generiranje otoka želenega videza traja od 10 do 15 sekund. Vse vrednosti veljajo za prenosni računalnik s procesorjem Intel Core i7 4700-HQ, 8 GB delovnega pomnilnika in grafično kartico nVidia GTX 760M, pri čemer smo v pogonu Unity 5 imeli nastavljen prikaz z najvišjo stopnjo podrobnosti.

Predstavljen postopek torej uspešno generira začetni teren, na katerem simulira hidravlično in termično erozijo ter nato še rast koralnega grebena. Končni teren z nanesenimi teksturami in izrisom morske gladine ima prepričljiv videz tropskega otoka. S spremenjanjem parametrov lahko dobimo veliko različnih oblik otoka, kot prikazujeta primera na sliki 4.

Z videzom tropskega otoka 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 se izvaja hitro. Videz grebena bi lahko bil bolj razgiban, saj se trenutno ob obali otoka ustvari popoln in neprekinjen greben. Naravni grebeni so mnogokrat prekinjeni in različno oddaljeni od obale. Tako kot pri hidravlični eroziji smo tudi v primeru simulacije rasti koralnega grebena imeli težave z uravnovešanjem konstant, saj ima že zelo majhna spremembu velik vpliv na končni rezultat.

Dinamično nanašanje tekstur deluje hitro in daje prepričljiv videz. Vzorčne tekture lahko zamenjamo in tako na preprost način spremenimo videz otoka. Videz otoka lahko spreminja tudi s spremenjanjem parametrov v senčilniku, kot sta na primer gostota ponavljanja posamezne vzorčne tekture in naklon terena, pri katerem namesto tekture trave nanesemo teksturo skale.

Gladina morja veliko pripomore k prepričljivosti vizualizacije otoka. Senčilnik se dobro obnese, saj nam daje tudi vizualno informacijo o globini vode, kar je še posebno koristno v okolici koralnega grebena. Prikaz sicer ni fizikalno natančen, saj zanemari kot pogleda. Globino vedno izrisuje, kot da je pogled od zgoraj navzdol. V praksi pa to ne povzroča težav, saj teren večino časa opazujemo iz zraka.

Sklep

V delu smo predstavili postopek, ki na podlagi vhodnih parametrov generira teren v obliki otoka, na katerem v realnem času nato simulira hidravlično in termično erozijo ter za tem še rast koralnega grebena. Na teren dinamično nanaša tekture in izrisuje gladino morja. Zadani cilj, da ima generiran teren prepričljiv videz tropskega otoka z okoliškim koralnim grebenom, smo po našem mnenju dosegli.

Končni teren je uporaben predvsem v strateških igrah, kjer nanj večinoma gledamo iz oddaljenosti. Teren namreč nima dovolj podrobnosti za prvo osebne igre, kjer se igralec po terenu sprehaja in ga raziskuje. Čas potreben za generiranje otoka želenega videza je pri uporabi v računalniški igri lahko težava, vendar bi jo z izboljšanjem implementacije lahko odpravili. Generiranje začetnega terena bi lahko pospešili tako, da bi ga namesto v jeziku C# implementirali v niženivojskem jeziku (npr. C++) ali pa na senčilniku. Hitrost izvajanja simulacij naravnih procesov na GPE bi lahko optimizirali, saj se v trenutni implementaciji opravlja nepotrebno kopiranje tekstur, kar izvajanja simulacij upočasni.

Terenu bi lahko dodatno izboljšali videz s tem, da bi za njegovo predstavitev uporabili več plasti z različnimi trdotami, kar bi morali nato upoštevati tudi pri simulacijah procesa erozije [17]. Funkcije, ki generirajo začetni teren, pa smo že zasnovali tako, da lahko izhod ene uporabimo kot vhod druge. S tem imamo več možnosti pri gradnji začetnega terena; nismo omejeni zgolj na obliko stožca.

CONTRIBUTIONS. JN je napisal uvod in skrbel za jezik, MK je izvedel analizo podatkov in izdelal slike, IC je implementiral algoritem in pognal eksperimente.

Bibliography

1. Küçükdeniz T (2022) Gym layout modeling and optimization. *Management Science Letters* 12(3):185–192.
2. Yang S, Li T, Gong X, Peng B, Hu J (2020) A review on crowd simulation and modeling. *Graphical Models* 111:101081.
3. Feng T, Yu LF, Yeung SK, Yin K, Zhou K (2016) Crowd-driven mid-scale layout design. *ACM Trans. Graph.* 35(4):132–1.
4. Mathew CT, Knob PR, Musse SR, Aliaga DG (2019) Urban walkability design using virtual population simulation in *Computer Graphics Forum*. (Wiley Online Library), Vol. 38, pp. 455–469.
5. Perlin K (2002) Noise hardware in *Real-Time Shading Languages – SIGGRAPH Course Notes*, ed. Olano M.
6. Mei X, Decaudin P, Hu BG (2007) Fast hydraulic erosion simulation and visualization on GPU in *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*. (IEEE), pp. 47–56.
7. Jákó B, Tóth B (2011) Fast hydraulic and thermal erosion on GPU in *Eurographics 2011 - Short Papers*, eds. Avis N, Lefebvre S. (The Eurographics Association), pp. 57–60.
8. O'Brien JF, Hodgins JK (1995) Dynamic simulation of splashing fluids in *Proceedings of Computer Animation '95*. (IEEE), pp. 198–205.
9. Maške L (2013) *Simulacija in vizualizacija erozije terena z uporabo GPE*. (Fakulteta za računalništvo in informatiko, Univerza v Ljubljani). Diplomsko delo.
10. Cecilia JM, García JM, Ujaldón M (2010) CUDA 2D stencil computations for the Jacobi method in *International Workshop on Applied Parallel Computing*. (Springer), pp. 173–183.
11. Dermastia M, Kiauta T, Turk T (1995) *Oxfordova ilustrirana enciklopédia žive narave* ed. Coe MJ. (DZS, Ljubljana), 2 edition.
12. Nakamura T, Nakamori T (2011) 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), pp. 251–262.
13. Szirmay-Kalos L, Umenhofer T (2008) Displacement mapping on the GPU – state of the art. *Computer Graphics Forum* 27(6):1567–1592.
14. Nießner M et al. (2016) Real-time rendering techniques with hardware tessellation. *Computer Graphics Forum* 35(1):113–137.
15. Angel E, Shreiner D (2011) *Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL*. (Addison-Wesley Publishing Company, USA), 6th edition.
16. Nakamura T, Nakamori T (2007) A geochemical model for coral reef formation. *Coral Reefs* 26(4):741–755.
17. Št'ava O, Beneš B, Brisbin M, Krivánek J (2008) Interactive terrain modeling using hydraulic erosion in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. (Eurographics Association), pp. 201–210.