

VILNIAUS JÉZUITŲ GIMNAZIJA

INFORMACINĖS TECHNOLOGIJOS
IV GIM. KL.

Brandos darbas

**VIRTUALAUS RELJEFO GENERAVIMO IR
REDAGAVIMO PROGRAMOS KŪRIMAS**

Atliko: IV gim. kl. mokinys(-ė)
Kristupas Trečiokas

(parašas)

Darbo vadovas: Informacinių technologijų mokytojas
Gvidas Tupinis

(parašas)

Vilnius

2024

Turinys

Naudojamų savokų žodynas	2
Įvadas	3
1. Rust programavimo kalba	4
1.1 Atminties apsauga	4
1.2 Efektyvumas	5
1.3 Išskirtinės Rust savybės	5
1.4.1 Alternatyvos	6
2. Programos kūrimas	7
2.1 Virtualaus reljefo generavimas	7
2.1.1 Reljefo tinklelio generavimas	7
2.1.2 Reljefo aukščių generavimo algoritmas	7
2.1.3 Reljefo fragmentų valdymas	9
2.2 Reljefo aukščių redagavimas	11
2.3 Reljefo tekštūrų redagavimas	12
2.4 3D Objektų įterpimas	12
2.5 Išsaugojimo sistema	13
2.6 LOD sistema	14
2.7 Galutinė programos versija	15
Išvados	16
Literatūra	17
Įsivertinimas	19
Priedai	20
Priedas 1. Bevy variklio demonstracija	20
Priedas 2. Plokštumos tinklelio generavimo kodas	21
Priedas 3. Darbo planas	23

Naudojamų savokų žodynas

Aukščių žemėlapis – (angl. heightmap) 2D masyvas, kuriame saugomi reljefo auksčiai, priskirti kiekvienam reljefo laukeliui/taškui.

Aukšto lygio programavimo kalba – (angl. high-level programming language) programavimo kalba, natūraliai atspindinti programavimo savokas, nepriklausoma nuo kompiuterio arba operacinės sistemos. [1]

Atminties apsauga – (angl. memory safety, memory protection) - operatyviosios atminties paskirstymas programoms ir priemonės, užtikrinančios, kad programa negalėtų pakeisti svetimų duomenų, t. y. esančių kitoms programoms skirtose atminties srityse. [1]

ECS – Entity component system (liet. subjektų-komponentų sistema) - programinės įrangos architektūrinis modelis, skirtas saugoti duomenis bei gauti prieigą prie duomenų apie pasaulio objektus. Dažnai naudojama žaidimų varikliuose.

GC – (Garbage collection) - automatinio atminties valdymo forma, užtikrinanti atminties saugumą, tačiau sumažinanti programos našumą.

Gija – lygiagrečiojo programavimo savoka, reiškianti vieno proceso viduje vienu metu vykdomas darbų grandinę.

Šešeliuoklė – (angl. shader) – tai kompiuterio programa, kuri apskaičiuoja tinkamus šviesos, tamsos ir spalvų lygius atvaizduojant 3D aplinką - šis procesas vadinamas šešeliavimu.

Tipų sauga – (angl. type safety) programavimo kalbos kontrolė, kuri užkerta kelią tipų klaidoms.

UV atvaizdavimas – (angl. UV mapping) 3D modeliavimo procesas, kurio metu 3D modelio paviršius projektuojamas į 2D vaizdą tam, kad būtų galima atvaizduoti tekstūrą. Raidės "U" ir "V" žymi 2D tekstūros ašis, nes "X", "Y" ir "Z" jau naudojamos 3D objekto ašims erdvėje žymėti.

Žemo lygio programavimo kalba – (angl. low-level programming language) Programavimo kalba, priklausoma nuo kompiuterio arba operacinės sistemos architektūros. [1]

Įvadas

Virtualus reljefas – tai kompiuteriu sukurtas trimatis (3D) skaitmeninis kraštovaizdžio ar lauko aplinkos atvaizdas. Jis dažnai naudojamas įvairiose programose, išskaitant kompiuterinius žaidimus, simuliacijas, virtualią realybę, geografinės informacines sistemas (GIS)[2]. Virtualios vietovės paprastai kuriamos siekiant imituoti realaus pasaulio geografinės savybes, pavyzdžiui, kalnus, slėnius, upes, miškus ir kitus gamtinius ar žmogaus sukurtus elementus.

Tam, kad būtų galima sukurti virtualų reljefą, reikalinga virtualaus reljefo generavimo ir redagavimo programa. Yra kelios populiarios reljefo kūrimo ir redagavimo programos, naujojamos įvairiose pramonės šakose: *World Machine*, *Unity 3D Terrain*, *Unreal Engine Landscape*, *Terragen*, *L3DT* ir kt. Tačiau, dabartinės programos turi trūkumų, dėl kurių naudojimas ar pačios programos redagavimas yra apsunkinamas:

- Programos yra per sudėtingos ir turi per daug funkcijų, tad įprastiniams vartotojui jos gali pasirodyti neintuityvios, gali būti sunku rasti tai, ko ieškoma.
- Programos yra pasenusios. Dauguma komercinių virtualaus reljefo generavimo ir redagavimo programų yra atnaujinamos iki šiol, tačiau paprastos ir turinčios mažiau funkcijų yra pasenusios. Minėta programa *L3DT* buvo atnaujinta 2018 m.[3] ir programos grafika bei vartotojo sąsaja neatitinka šiuolaikinių standartų.
- Programos yra ne atviro kodo. Dauguma virtualaus reljefo generavimo ir redagavimo programų yra uždaros kodo, tad jų neįmanoma redaguoti. Taip gali būti sunku įgyvendinti naujas funkcijas ar pagerinti programos efektyvumą.

Dėl šių priežasčių buvo išsikeltas darbo tikslas – sukurti funkcionalią programą virtualaus reljefo vaizdavimui, generavimui ir redagavimui. Šiam tikslui įgyvendinti buvo išsikelti tokie uždaviniai:

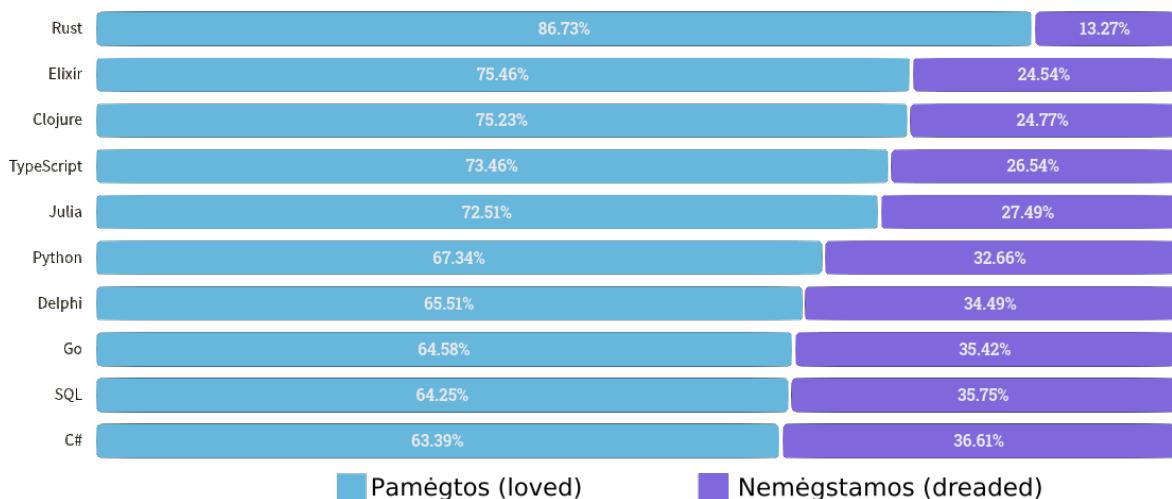
- Įvertinti *Rust* galimybes kuriant reljefo generavimo programas
- Sukurti 3d reljefo tinklelio generavimo algoritma
- Sukurti reljefo aukščių bei tekstūrų redagavimo sistemą
- Programai pritaikyti LOD optimizavimo metodą

Sukurta programa neturi jokių minėtų trūkumų – ji intuityvi, paprasta, lengvai naujodama ir efektyvi. Taip pat, programos kodas yra viešas ir lengvai redaguojamas. Programai kurti buvo pasirinkta *Rust* programavimo kalba dėl jos išskirtinių savybių, reikalingų būtent virtualaus reljefo generavimo ir redagavimo programai.

1. Rust programavimo kalba

Reljefo generavimo ir redagavimo programai tinkamiausios yra žemo lygio programavimo kalbos, nes tokio tipo programoms būdingas našumas. Šiai programai buvo pasirinkta *Rust* programavimo kalba.

Rust – ganėtinai nauja (sukurta 2015 m.) sistemų programavimo kalba. *Rust* – moderni, atviro kodo programavimo kalba, pamėgta daugelio programuotojų [4].



1 pav.: 2022 m. *Stackoverflow* apklausa [4], rodanti vartotojų labiausiai pamėgtas programavimo kalbas

Rust programavimo kalba geriausiai atspindi 3 aspektuose: [5]

- Atminties apsauga nenaudojant GC (Garbage collection)
- Greitis, ekspresyvumas ir našumas, prilygstantis *C++*
- Tipų sauga, prilygstanti *Java*

1.1 Atminties apsauga

Viena iš priežasčių, dėl ko buvo pasirinkta naudoti *Rust* – atminties apsauga be didelių našumo sąnaudų. *Rust* programavimo kalboje atmintis valdoma naudojant nuosavybės (angl. ownership) sistemą, kurioje galioja šios taisyklės: [6]

- Kiekviena *Rust* vertė turi kintamąjį, kuris vadinamas „savininku“
- Vienu metu gali būti tik vienas kintamojo savininkas
- Kai savininkas išeina iš aprėpties (kintamasis nėra naudojamas), vertė panaikinama

Tokiu būdu visos nuorodos nukreipia į galiojančią atmintį ir taip *Rust* kalboje įmanomas atminties saugumas be GC ar nuorodų skaičiavimo. Programavimo kalbose be atminties saugos, kaip *C*, *C++* ar *Assembly* įmanoma sukurti rodykles, kurios nerodo į galiojančią atminties vietą. Bandymas priskirti jai reikšmę arba dereferencijuoti ją gali sukelti neapibrėžtą elgesį:

```
1 int* p = nullptr;    // Sukuriame neinicializuota rodykle
2 *p = 42;             // Bandymas priskirti vertę per neinicializuota rodykle
```

Kodo fragmentas 1: C++ neinicializuotos rodyklės

1.2 Efektyvumas

Renkantis programavimo kalbą, efektyvumas yra itin svarbu – nuo to priklauso programos greitis. Būtent todėl buvo pasirinkta *Rust* programavimo kalba. Kartu su *C* kalba, ji gerokai lenkia kitas kalbas – ji 50% efektyvesnė už *Java* ir apie 98% efektyvesnė už *Python*. *Rust* programos suvartoja apie 74% mažiau energijos, nei tokios pačios programos, parašytos naudojant *Python*. [7]

Total				
	Energija	Laikas	Atmintis	
(c) C	1.00	1.00	(c) Pascal	1.00
(c) Rust	1.03	1.04	(c) Go	1.05
(c) C++	1.34	1.56	(c) C	1.17
(c) Ada	1.70	1.85	(c) Fortran	1.24
(v) Java	1.98	1.89	(c) C++	1.34
(c) Pascal	2.14	2.14	(c) Ada	1.47
(c) Chapel	2.18	2.83	(c) Rust	1.54
(v) Lisp	2.27	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	3.09	(c) Haskell	2.45
(c) Fortran	2.52	3.14	(i) PHP	2.57
(c) Swift	2.79	3.40	(c) Swift	2.71
(c) Haskell	3.10	3.55	(i) Python	2.80
(v) C#	3.14	4.20	(c) Ocaml	2.82
(c) Go	3.23	4.20	(v) C#	2.85
(i) Dart	3.83	6.30	(i) Hack	3.34
(v) F#	4.13	6.52	(v) Racket	3.52
(i) JavaScript	4.45	6.67	(i) Ruby	3.97
(v) Racket	7.91	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	26.99	(v) F#	4.25
(i) Hack	24.02	27.64	(i) JavaScript	4.59
(i) PHP	29.30	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	43.44	(v) Java	6.01
(i) Lua	45.98	46.20	(i) Perl	6.62
(i) Jruby	46.54	59.34	(i) Lua	6.72
(i) Ruby	69.91	65.79	(v) Erlang	7.20
(i) Python	75.88	71.90	(i) Dart	8.64
(i) Perl	79.58	82.91	(i) Jruby	19.84

2 pav.: Programų, parašytų skirtingomis programavimo kalbomis resursų naudojimas[7]

1.3 Išskirtinės Rust savybės

Neskaitant kitų *Rust* ypatybių, kaip atminties apsauga ir efektyvumas, *Rust* turi tam tikras savybes, kurios padeda palaikyti švarų ir tvarkingą kodą. [6] Virtualaus reljefo generavimui

mo ir redagavimo programai kurti buvo pasirinktas žaidimų variklis *Bevy*. Šis žaidimo variklis buvo pasirinktas dėl kelių priežasčių:

- Variklyje naudojama ECS (Entity – component system).
- Variklis turi 3D atvaizdavimo įrankį, turintį daug funkcijų: apšvietimas, kameros, tekstuūros, medžiagos, pasirinktinės šešeliuoklės.
- Variklis yra itin našus.
- Variklis palaiko daugelį platformų, kaip *Linux*, *Windows*, *MacOS*, *iOS*, *Android*. Taip pat, galima kurti žiniatinklio programas.

Bevy variklio ypatumai bei ECS parodomai 1 priede.

1.4.1 Alternatyvos

Virtualaus reljefo generavimo ir redagavimo programai kaip pagrindą galima naudoti nebūtinai *Bevy*, egzistuoja alternatyvos, taip pat tinkamos tokiai programai kurti:

- **Žaidimų variklis *Amethyst*** – universalus atvirojo kodo žaidimų variklis, skirtas kurti 2D ir 3D žaidimams, žinomas dėl savo lankstumo, modulinės struktūros ir našumo. Tačiau, nuo 2022–08, "Amethyst" žaidimų variklio kūrimas yra sustabdytas, tad variklio kodas yra nebeatnaujinamas. [8]
- **Žaidimų variklis *Piston*** – modulinis atvirojo kodo žaidimų variklis. Jis sukurtas taip, kad programuotojams suteiktų paprastą ir lanksčią platformą 2D ir 3D žaidimams, simuliacijoms ir interaktyvioms programoms kurti. Deja "Piston" neturi ECS, todėl sudėtingos programos kūrimas būtų itin sudėtingas. [9]
- **Variklio kūrimas su *wgpu*.** *Wgpu* – grafinė biblioteka, pagrista *WebGPU* aplikacijų programavimo sasaja. Ji tinka bendrosios paskirties grafikai ir skaičiavimams, naudojant GPU. Tai nebūtų optimalus variantas tokiai programai, nes *wgpu* yra žemo lygio grafikos aplikacijų programavimo sasaja, o kuriant reljefo generavimo ir redagavimo programą reikia sudėtingos 3D atvaizdavimo logikos. Reikėtų įgyvendinti arba integruoti tokias funkcijas, kaip vietovės generavimo algoritmai, vietovės manipuliavimo įrankiai ir realaus laiko atvaizdavimas, o tai gali būti itin sudėtinga. [10]

2. Programos kūrimas

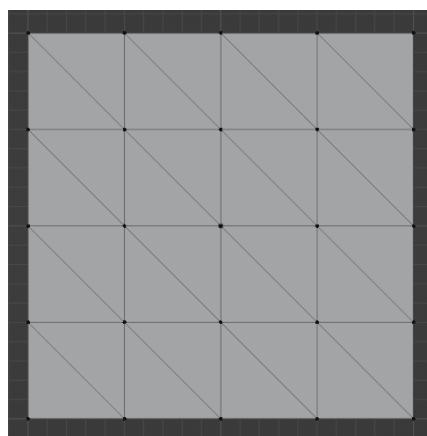
2.1 Virtualaus reljefo generavimas

2.1.1 Reljefo tinklelio generavimas

Pirmasis žingsnis kuriant virtualaus reljefo modelį - sugeneruoti reljefo tinklelį. Šioje programeje reljefo tinklelis yra plokštuma, padalinta į lygias dalis. Ši plokštuma, kaip ir kiekvienas virtualus 3D objektas, yra sudaryta iš trikampių. [11]

Tam naudojamas tinklelio sukūrimo algoritmas (2 priedas). Jį sudaro šie pagrindiniai etapai:

- **Inicjalizacija:** Inicializuojami vektoriai, kuriuose saugomos viršūnės, UV koordinatės ir viršūnių indeksai, sudarantys trikampius..
- **Viršūnių generavimas:** Generuojama kiekviena tinklelio viršūnė, apskaičiuojant jos padėti ir UV koordinates pagal plokštumos matmenis.
- **Trikampių formavimas:** Kiekvienam kvadratui, esančiam plokštumoje suformuojami du trikampiai. Šie trikampiai įtraukiami į trikampių sąrašą.
- **UV koordinačių priskyrimas:** Priskiriamos UV koordinatės kiekvienai viršūnei, užtikrinant tinkamą tekstūros atvaizdavimą.

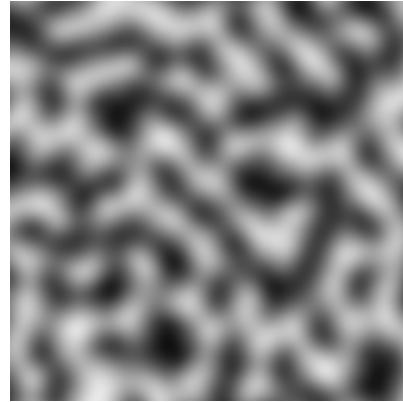


3 pav.: Sugeneruotas plokštumos (4x4) tinklelis

2.1.2 Reljefo aukščių generavimo algoritmas

Sugeneruotas tinklelis šiuo metu yra plokščias. Tam, kad galētume sukurti nelygū paviršių, turime naudoti aukščių žemėlapį. Tradiciniai metodai aukščių žemėlapiams kurti

naudojamos procedūrinės triukšmo funkcijos, iš kurių populiarus pasirinkimas - "Perlin" triukšmas. Tačiau, "Perlin" triukšmas pasižymi tam tikrais vizualiais artefaktais [12]. Taigi, buvo pasirinktas alternatyvus metodas, kuriame naudojamas "OpenSimplex" triukšmas - "Simplex" triukšmo variantas. Šis triukšmas neturi "Perlin" triukšmo artefaktų bei sukuria tolygesnį aukščių žemėlapį.



4 pav.: Sugeneruota 2D tekstūra naudojant "OpenSimplex" triukšmą [13]

Naudojamas algoritmas turi šiuos pagrindinius etapus:

- **InicIALIZACIJA:** Inicializuojamas masyvas aukščio vertėms saugoti.
- **Poslinkio apskaičiavimas:** Apskaičiuojamas poslinkis pagal reljefo fragmento koordinatę (Apie reljefo fragmentus žr. "Reljefo fragmentų valdymas"). Šis poslinkis naudojamas siekiant užtikrinti, kad reljefas būtų tolygus tarp fragmentų.
- **Triukšmo generavimas:** Iteruojama per kiekvieną aukščio žemėlapio tinklelio tašką ir apskaičiuojama aukščio vertė naudojant "OpenSimplex" triukšmą, pateikiant taško koordinates. Šis gautas aukštis padauginamas iš norimos vertės. Kuo ši vertė didesnė, tuo didesnė reljefo aukščių amplitudė.

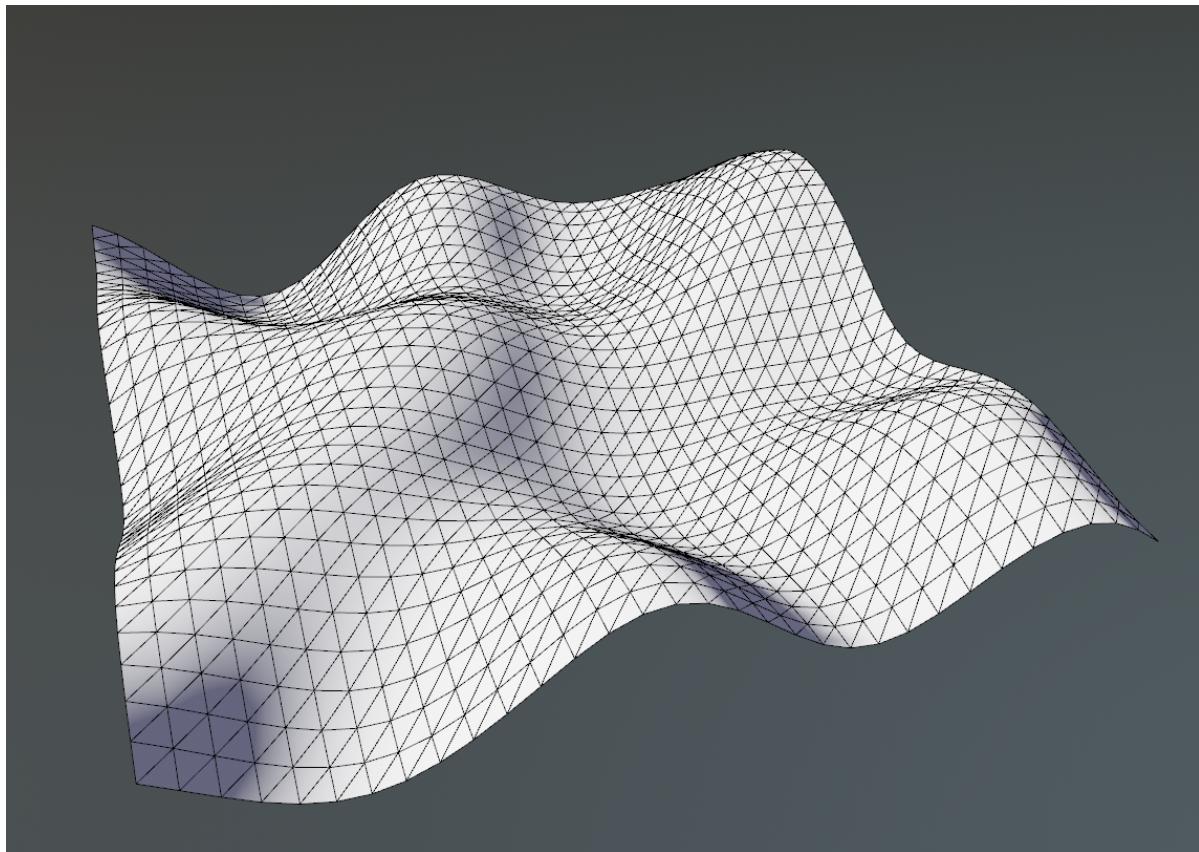
```

1 let mut heights = vec![0.0; self.chunk_size * self.chunk_size];
2 let (offset_x, offset_y) = (
3     chunk_pos.x as f64 * self.chunk_size as f64,
4     chunk_pos.y as f64 * self.chunk_size as f64
5 );
6 let hasher = PermutationTable::new(1);
7 for y in 0..self.chunk_size {
8     for x in 0..self.chunk_size {
9         let height_index = x + y * self.chunk_size;
10        heights[height_index] = open_simplex_2d(
11            [(x as f64 + offset_x) * terrain_scale,
12             (y as f64 + offset_y) * terrain_scale],
13             &hasher) as f32 * terrain_height;
14    }
}

```

Kodo fragmentas 2: Aukščių žemėlapio generavimo algoritmas

Kiekvienai viršūnės y kordinatei parinkus reikšmę iš aukščių žemėlapio, gauname 3D reljefą:



5 pav.: Sugeneruotas 3D reljefas

2.1.3 Reljefo fragmentų valdymas

Programoje reljefas yra sudarytas iš fragmentų. Fragmentas - tam tikra viso reljefo dalis, apibrėžta fragmento koordinatėmis. [14] Fragmento koordinacių veikimo principas parodomos 6 pav. Fragmentų sistema turi šiuos pagrindinius privalumus:

- **Reljefo išplėtimo galimybes:** Reljefas gali būti dinamiškai praplėčiamas be didelių resursų sąnaudų.
- **Nesudėtingas redagavimas:** Vartotojas gali kurti, ištrinti arba keisti atskirus fragmentus. Keičiant vieną fragmentą, atnaujinamas tik keičiamas fragmentas, o ne visas virtualus reljefas.
- **LOD valdymas:** Fragmentų sistemos naudojimas gerokai palengvina LOD sistemos implementaciją, kuri pagerina programos našumą (plačiau skyriuje LOD).

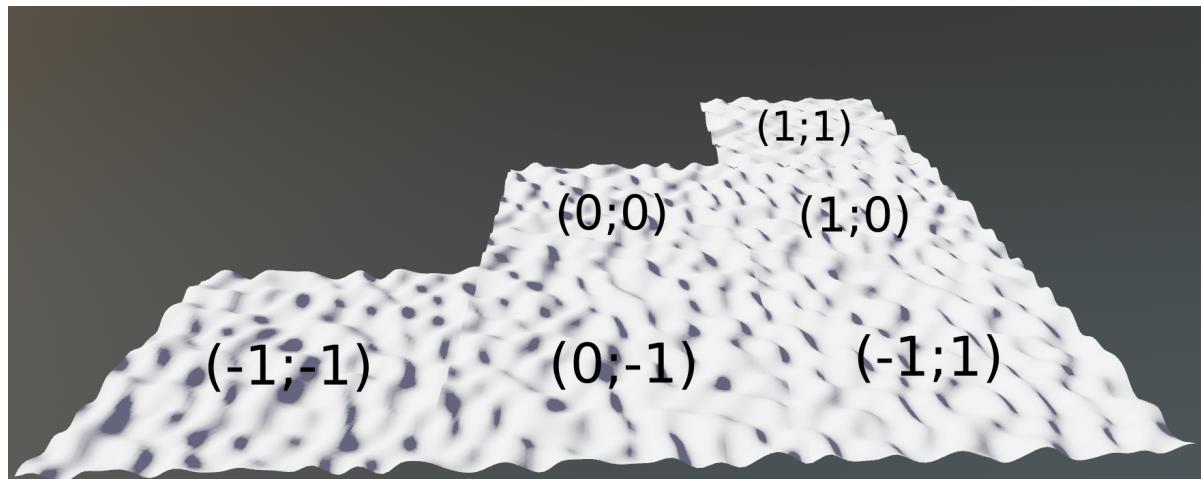
Naudojant fragmentų sistemą, programa turi turėti du koordinacių tipus - pasaulio ir fragmentų. Pasaulio koordinatės apibrėžia absoliučią objekto padėtį, o fragmentų koordinatės nurodo santykinę fragmentų padėtį. Ši funkcija naudojama paversti pasaulio koordinates į fragmentų koordinates:

```

1 pub fn world_to_chunk_pos(world_pos:IVec2) -> IVec2{
2     IVec2::new(
3         world_pos.x.div_euclid(chunk_size as i32),
4         world_pos.y.div_euclid(chunk_size as i32)
5     )
6 }
```

Kodo fragmentas 3: Funkcija, paverčianti pasaulio koordinates į fragmentų koordinates

`IVec2` - fragmentų koordinatė, sudaryta iš 2 sveikujų skaičių. Naudojama `div_euclid` funkcija tam, kad būtų užtikrinta, kad fragmento koordinatės bus apskaičiuotos teisingai, net jei pasaulio koordinatės yra neigiamos arba fragmento dydis nėra pasaulio koordinacių komponenčių kartotinis. Nuo sveikujų skaičių dalybos ši funkcija skiriasi tuo, kad jei reikšmė > 0 , tai gauta reikšmė bus apvalinama link 0, o jei reikšmė < 0 , tai gauta reikšmė bus apvalinama link $-\infty$ (pvz.: $-3/7 = 0$, o $(-3).div_euclid(7) = -1$).



6 pav.: Reljefo fragmentai ir atitinkamos koordinatės

Programoje fragmentų sistema naudojama aukščių žemėlapiams, tekstūroms, fragmentų objektams saugoti:

```

1 pub struct TextureMap{
2     pub textures:HashMap<IVec2, Handle<Image>>,
3     //...
4 }
```

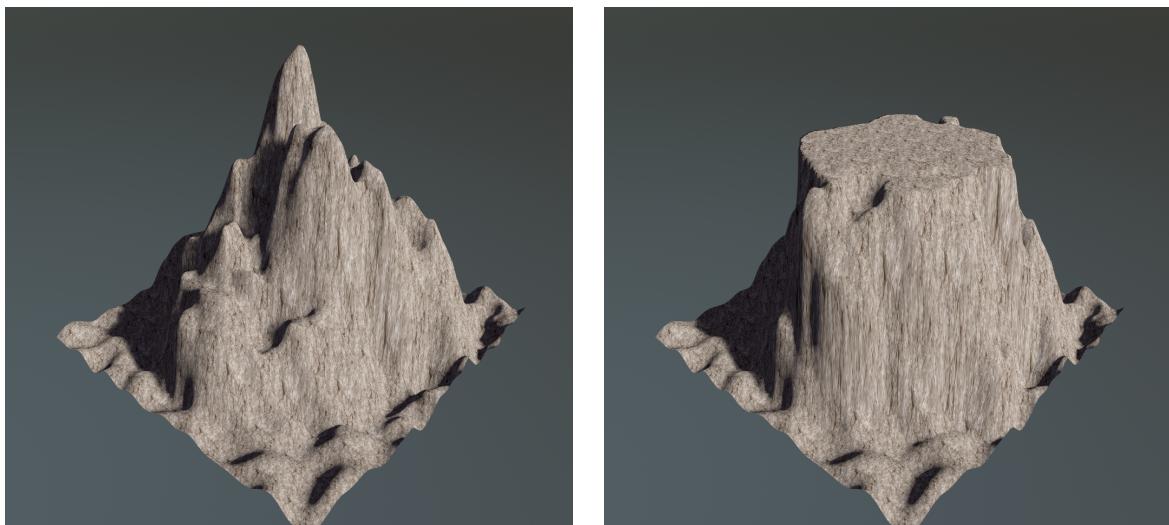
Kodo fragmentas 4: Tekstūrų saugojimas naudojant fragmentų sistemą

Šiuo atveju, tekstūros yra saugojamos `HashMap` (žodyno) struktūroje: žodyno raktas yra fragmento koordinatė, o saugoma reikšmė - tekstūra. Tai gerokai palengvina naujų fragmentų sukūrimą bei ištrynimą.

2.2 Reljefo aukščių redagavimas

Reljefo aukščių redagavimui naudojama *tapymo* sistema - vartotojas gali pasirinkti *teptuko* tekstūrą, intensyvumą ir, naudojant kompiuterio pele, *tapytį* ant reljefo. Programoje galimi 3 reljefo aukščių redagavimo tipai:

- **Padidinti/sumažinti:** Pagrindinis aukščio redagavimo metodas. Spaudžiant kairįjį pelės klavišą, reljefas kyla (aukščiai didėja), o laikant *Control* klavišą bei kairįjį pelės klavišą, reljefas leidžiasi (aukščiai mažėja).
- **Nustatyti aukštį:** Galima pasirinkti norimą reljefo aukštį ir pelės pagalba pakelti arba nuleisti pasirinktas reljefo dalis taip, kad jų vidutinis aukštis būtų artimesnis norimam aukščiui. Tokiu būdu galima sukurti plynaukštę:



7 pav.: Reljefo aukščio nustatymas, sukuriant plynaukštę

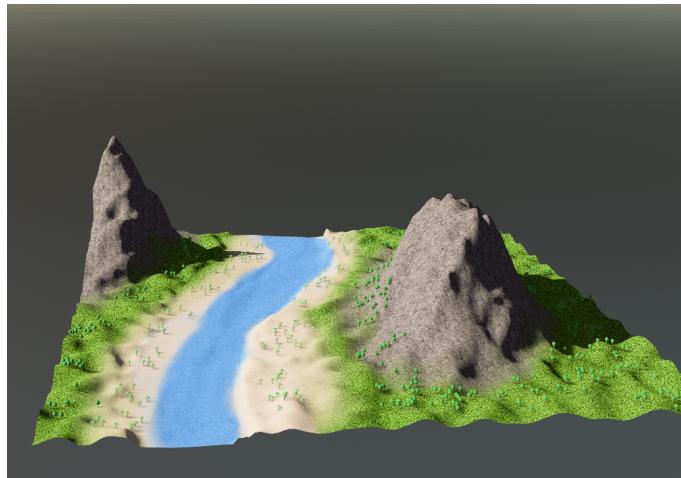
- **Išlyginti:** Išlyginami tam tikri iškilimai ar kiti nelygumai, išvedant aukščių vidurkį ir ji pritaikant kiekvienam taškui:



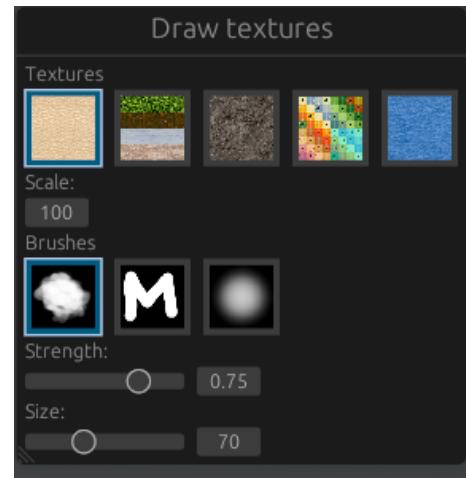
8 pav.: Reljefo išlyginimas

2.3 Reljefo tekstūrų redagavimas

Reljefo tekstūroms redaguoti naudojamas analogiškas metodas. Pirmiausia pasirenkama norima tekstūra, jos mastelis. Tada pasirenkama *teptuko* tekstūra, intensyvumas bei užimamas plotas.



Nutapytas reljefas



Tekstūrų redagavimo vartotojo sasaja

9 pav.: Reljefo tekstūrų redagavimas

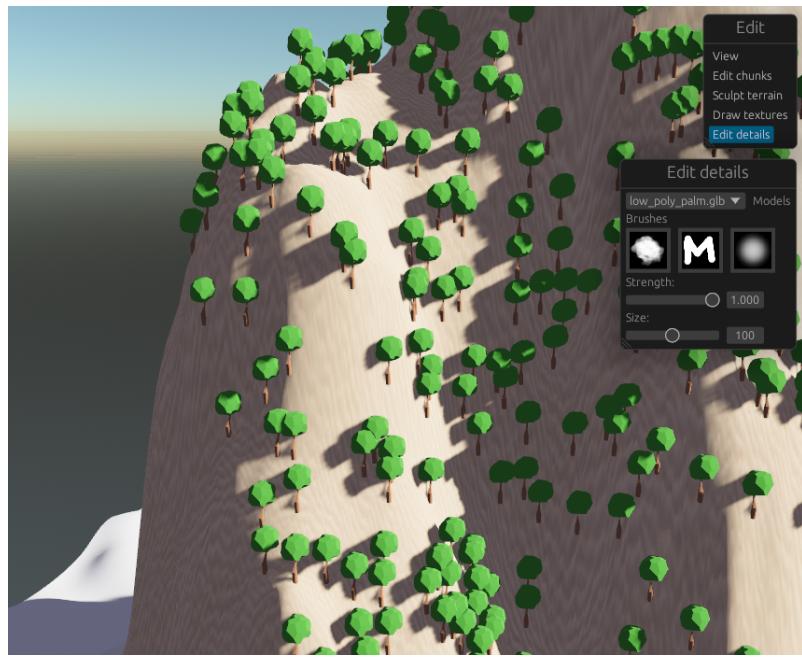
Nupiešta tekstūra priklauso nuo pasirinktos *teptuko* tekstūros. Taip galima tiksliai išgauti norimus raštus reljefe:



10 pav.: Skirtingi raštai reljefe

2.4 3D Objektų įterpimas

Programoje taip pat buvo įgyvendinta objektų įterpimo funkcija - pasitelkiant analogišką *teptuko* principą, ant reljefo galima kopijuoti/ištrinti norimą 3D modelį:



11 pav.: 3D objektai ant reljefo

2.5 Išsaugojimo sistema

Programoje sukurtą virtualų reljefą galima išsaugoti - tam naudojamas dvejetainis kodavimas pasitelkiant *bincode* biblioteką [15]. Reljefo faile įtraukiama šie duomenys:

- Reljefo fragmentų dydys
- Reljefo fragmento tekstūros dydis
- LOD informacija
- Reljefo aukščių ir tekstūrų žemėlapiai.
- Iterpti 3D objektai

```

1 #[derive(Serialize, Deserialize)]
2 pub struct TerrainData {
3     pub chunk_size: usize,
4     pub texture_size: usize,
5     pub lod: Vec<LODLevel>,
6     pub chunks: Vec<ChunkData>,
7     pub details: Vec<DetailData>,
8 }
```

Kodo fragmentas 5: Struktūra, skirta reljefo informacijos išsaugojimui

Vartotojas gali išsaugoti esamą reljefą bei įkelti jau išsaugotą failą.

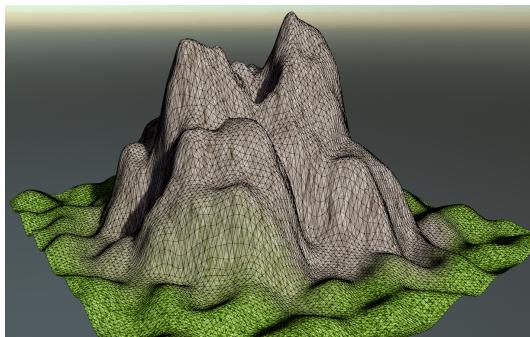
2.6 LOD sistema

Šiuo metu programoje kiekvienas fragmentas yra atvaizduojamas maksimalia raiška, nepriklausomai nuo atstumo iki stebėtojo, taip švaistant resursus. Tam, kad ši problema būtų išspręsta, programoje buvo pritaikyta LOD sistema. LOD - (angl. level of detail) detalumo lygis. Tai - kompiuterinės grafikos ir modeliavimo programose naudojamas metodas, kai reikia atvaizduoti sudėtingus 3D objektus. 3D objektų detalumo lygis yra koreguojamas pagal tokius veiksnius, kaip atstumas nuo stebėtojo, objekto svarba ar turimi iškekliai. Tolimi objektai neprivalo būti atvaizduojami aukščiausia kokybe, tad tam, kad būtų suraupyt ištekliai, šių objektų modeliai keičiami į tokius pačius, tačiau mažesnės rezoliucijos modelius. Pagrindinis LOD tikslas - subalansuoti vaizdo kokybę bei našumą. [14]

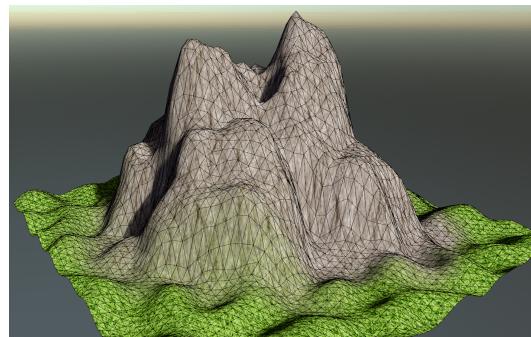
Programoje kiekvienam reljefo fragmentui sugeneruojami 3D modeliai su skirtinių tinklelio rezoliucijomis. Rezoliucija parenkama remiantis šia lygtimi:

$$\text{Tinklelio rezoliucija} = \frac{\text{Aukščiausios raiškos tinklelio rezoliucija}}{2^{\text{LOD indeksas}}}$$

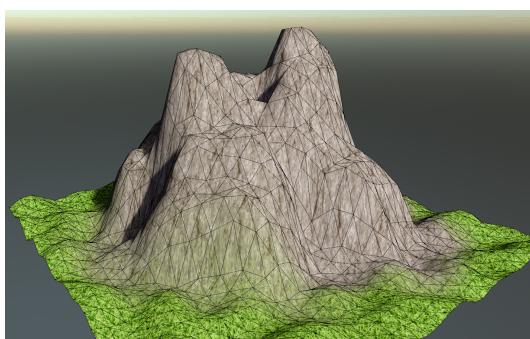
Taigi, kai $\text{LOD indeksas} = 0$, $\text{tinklelio rezoliucija} = \text{Aukščiausios raiškos tinklelio rezoliucija}$. Kai $\text{LOD indeksas} = 1$, $\text{tinklelio rezoliucija} = \text{Aukščiausios raiškos tinklelio rezoliucija} / 2$. Taigi, didinant LOD indeksą, tinklelio rezoliucija mažės dvigubai, o tinklelio taškų skaičius mažės keturgubai.



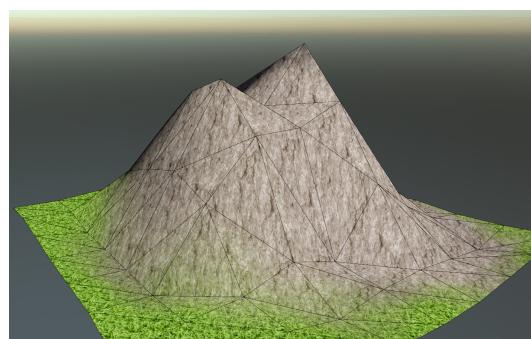
LOD 0: 16641 taškai



LOD 1: 4225 taškai



LOD 2: 1089 taškai



LOD 3: 289 taškai

12 pav.: Reljefo modelių raiškos lyginimas su skirtiniais detalumo lygiais

2.7 Galutinė programos versija

Galutinę programos versiją sudaro šios funkcijos:

- Reljefo aukščių ir tekstūrų redagavimas
- Reljefo fragmentų valdymas
- 3D objektų įterpimas
- LOD sistema
- Vartotojo sąsaja, paremta *egui* [16] biblioteka
- Išsaugojimo / įkėlimo sistema

Programą galima pasiekti https://github.com/0crispy/mountforge_editor.

Išvados

Darbo metu buvo įvertintos *Rust* programavimo kalbos galimybės bei pasirinktas programos kūrimui tinkamas žaidimų variklis – *Bevy*, atsižvelgiant į alternatyvas. Tada buvo išanalizuoti esami virtualaus reljefo generavimo bei redagavimo metodai ir pasirinkti programai tinkamiausi. Pasirinkus metodus, buvo sukurta programa. Reljefo generavimui buvo naudojamas 2D tinklelio generavimo algoritmas, o aukščiai buvo generuojami naudojant "Open-Simplex" triukšmo funkciją. Šis metodas padėjo sukurti lygų bei realistišką reljefą. Naudota fragmentų valdymo sistema gerokai palengvino redagavimą ir programos našumą. Tai taip pat leido įgyvendinti LOD sistemą, kuri padidino programos našumą. Aukščiams ir teksturoms redaguoti buvo sukurta *tapymo* sistema, naudojama daugelyje panašių programų, kuri leidžia vartotojui nesudētingai ir intuityviai keisti reljefą. Dėl *Rust* bei *Bevy* programos kodas yra švarus, suprantamas bei lengvai išplečiamas, o svarbiausia – našus. Taigi, sukurta programa yra nesudētinga, naudoja šiuolaikines atvaizdavimo technikas bei yra atviro kodo, todėl galima teigti, kad naudoti įrankiai bei metodai yra tinkami tokios ar panašių programų kūrimui.

Literatūra

- [1] Valentina Dagienė, Gintautas Grigas ir Tatjana Jevsikova. *Enciklopedinis kompiuterijos žodynas*. 2008. URL: <http://www.ims.mii.lt/EK%c5%bd/>.
- [2] Christian Dick. *Interactive Methods in Scientific Visualization. Terrain Rendering*. 2009. URL: <https://www.cs.cit.tum.de/fileadmin/w00cfj/cg/Research/Tutorials/Terrain.pdf>.
- [3] Aaron Torpy. *L3DT - Large 3D Terrain Generator*. ENG. URL: <http://www.bundysoft.com/L3DT/> (tikrinta 2023-11-06).
- [4] *Stack Overflow Developer Survey*. 2022. URL: <https://survey.stackoverflow.co/2022>.
- [5] Prabhu Eshwarla. *Practical System Programming for Rust Developers: Build fast and secure software for Linux/Unix systems with the help of practical examples*. en. Packt Publishing Ltd, 2020-12. ISBN: 978-1-80056-201-1.
- [6] Steve Klabnik ir Carol Nichols. *The Rust Programming Language (Covers Rust 2018)*. No Starch Press, 2019-08-12. ISBN: 978-1-71850-044-0.
- [7] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes ir João Saraiva. „Energy efficiency across programming languages: how do energy, time, and memory relate?“ en. *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. Vancouver BC Canada: ACM, 2017-10, p.p. 256–267. ISBN: 978-1-4503-5525-4. DOI: 10.1145/3136014.3136031. URL: <https://dl.acm.org/doi/10.1145/3136014.3136031>.
- [8] *Amethyst*. URL: <https://github.com/amethyst/amethyst>.
- [9] bvssvni. *Piston*. URL: <https://www.piston.rs/>.
- [10] *wgpu: portable graphics library for Rust*. URL: <https://wgpu.rs/> (tikrinta 2024-02-21).
- [11] J. R. Sack ir J. Urrutia. *Handbook of Computational Geometry*. en. Elsevier, 1999-12. ISBN: 978-0-08-052968-4.
- [12] Perlin Ken. *Noise hardware. In Real-Time Shading SIGGRAPH Course Notes*. 2001. URL: <https://redirect.cs.umbc.edu/~olano/s2002c36/ch02.pdf>.
- [13] Alex. *lmas/opensimplex*. 2024-01. URL: <https://github.com/lmas/opensimplex>.
- [14] Jack Xu. *Rust wgpu Procedural Terrains: Create Stunning Landscapes for Your Games*. en. UniCAD. Skyr. Level of Detail Algorithm, p. 321.
- [15] *bincode: A compact encoder / decoder pair that uses a binary zero-fluff encoding scheme*. URL: <https://github.com/bincode-org/bincode> (tikrinta 2024-02-21).

- [16] *egui: an easy-to-use GUI in pure Rust*. URL: <https://github.com/emilk/egui> (tikrinta 2024-02-21).

Įsivertinimas

Pagrindinis tikslas darant šį brandos darbą buvo išsiaiškinti, kaip generuoojamas reljefas bei pabandyti sukurti funkcionalią programą, kuri leistų vaizdo žaidimų kūrėjams kurti virtualius reljefo modelius. Taip pat, norėjau praplėsti savo "Rust" žinias ir išmoktus metodus pritaikyti savo ateities projektuose. Manau, kad sukurta programa puikiai atitinka mano pradiinius lūkesčius. Jos kūrimo metu daug sužinojau apie 3D objektų generavimą, tekstūrų valdymą bei programos architektūrą.

Sudėtingiausia proceso dalis buvo reljefo tinklelio bei aukščių generavimas. Aukščiai turėjo būti parenkami taip, kad perėjimai tarp gretimų fragmentų būtų lygūs bei nepastebimi. Tai pasiekti buvo itin sudėtinga. Taip pat turėjau surasti metodą, kaip kiekvienam reljefo taškui paskirti reikšmę iš aukščių žemėlapio. Tai buvo komplikuota, tačiau leido įsigilinti į generavimo metodus bei geriau suprasti tam tikrus 3D grafikos konceptus, kaip "normal" žemėlapiai, UV koordinatės, taškų indeksavimas.

Taip pat, brandos darbui sukurti buvo naudojama "LaTeX" teksto sudarymo sistema. Išmokti ir perprasti šios sistemos ypatumus užtruko daug laiko, tačiau tai leido sukurti, mano manymu, vizualiai gerai pateiktą brandos darbą. Manau, kad ši įrankjų naudosiu rašydamas kitus mokslinius darbus.

Vis dėlto, kūrimo metu reikėjo atsisakyti tam tikrų suplanuotų programos funkcijų, kaip hidraulinės erozijos simuliacija, realistiškas vandens bei debesų atvaizdavimas. Manau, kad šios funkcijos nėra itin reikalingos, be to, programa yra atviro kodo, tad šias funkcijas gali suprogramuoti bet koks programuotojas, norintis praplėsti programą.

Proceso metu labiausiai patiko matyti programos progresą ir kaip ji po truputį tampa įrankiu, kurį galima lyginti su kitomis reljefo kūrimo programomis. Buvo itin įdomu sukurti serializavimo ir deserializavimo funkcijas, kas leido išsaugoti sukurtus reljefo modelius. Esu patenkintas sukurta programa ir parašytu brandos darbu bei manau, kad tai buvo itin gera patirtis.



13 pav.: Mano sukurtas virtualus reljefas

Priedai

Priedas 1. Bevy variklio demonstracija

```
1 /*
2  * Viena iš išskirtinių Bevy savybių yra variklio ECS:
3   - Entity (liet. subjektas) – kiekvienas variklio objektas yra saugomas
4   atmintyje kaip 8 baitų sveikasis skaičius.
5   - Component (liet. komponentas) – tam tikri duomenys, kurie gali buti
6   priskirti subjektui. Atvirkščiai, kaip kituose varikliuose (pvz. Unity),
7   komponentai saugo tik duomenis, o ne objektų elgsena. Tai padeda
8   palaikyti tvarkingą kodą:
9 */
10 #derive[Component]
11 struct ManoKomponentas{
12     skaicius:f32 //Komponente saugomas skaičius
13 }
14 /*
15  - System (liet. sistema). Sistema atsakinga už tam tikrus komponentus
16  turinčių subjektų apdorojimą ir veikimą su jais. Sistemos – tai su
17  komponentais susijusi logika arba elgsena. Jos apibrėžia, kaip
18  komponentai sąveikauja ir laikui bėgant atsinaujina. Bevy variklyje
19  sistemas yra paprastos funkcijos. Bevy funkcijos kaip argumentus priima
20  Query (liet. užklausa) struktūras. Užklausoje galima nurodyti, kokius
21  komponentus norime pasiekti:
22 */
23 fn pasveikink_zmones(uzklausa: Query<&Zmogus>) {
24     for vardas in &uzklausa {
25         println!("labas {}!", zmogus.vardas);
26     }
27 }
28 /*
29 Šiame pavyzdje naudojame užklausą, kuri duoda prieigą prie visų komponentų
30 , kurių tipas – 'Zmogus'. Naudodami 'for' ciklą galime pereiti per už
31  klausą ir taip įvykdyti tam tikrus veiksmus su kiekvienu komponentu. Š
32  iuo atveju atspausdiname kiekvieno žmogaus vardą.
33
34 Tokia sistema atskiria duomenis nuo objektų elgesio. Dauguma šiuolaikinių ž
35  aidimų variklių naudoja klases, taip sujungdami duomenis su elgsena (pvz
36 . Unity, C#):
37 public class Zmogus : MonoBehaviour {
```

```

27     public string vardas;
28     void Start(){
29         Debug.Log("labas " + name + "!");
30     }
31 }
32
33 Toks objektų valdymo metodas nėra neteisingas, tačiau dirbant su itin didel
és apimties programomis, ECS gali padėti geriau organizuoti kodą ir plé
sti programos funkcionalumą, skatina kurti švarius, atsietus dizainus,
nes verčia suskaidyti programos duomenis ir logiką i pagrindinius
komponentus. Ši sistema taip pat padeda pagreitinti kodą, nes
optimizuoja prieiga prie atminties modelių ir palengvina lygiagretinimą.
34
35 Bevy žaidimų variklis yra itin našus ir didelė to priežastis – beveik
    viskas apdorojoma lygiagrečiai:
36 */
37 fn main() {
38     App::new()
39         .add_systems(Update, (sistemas1, sistemas2))
40         .run();
41 }
42 fn sistemas1(){
43     println!("1");
44 }
45 fn sistemas2(){
46     println!("2");
47 }
48 /*
49 Kadangi 'sistemas1' ir 'sistemas2' yra funkcijos, neprilausančios viena nuo
    kitos, jos yra vykdomos vienu metu – skirtingose gjose.
50 */

```

Priedas 2. Plokštumos tinklelio generavimo kodas

```

1 fn generate_mesh(&self, lod:usize) -> Mesh{
2     let chunk_size = self.chunk_size;
3     let mesh_size = self.mesh_size(lod);
4     let vertex_count = (mesh_size+1) * (mesh_size+1);
5     let mut vertices = vec![0.0; 3]; vertex_count];
6     let mut uvs = vec![0.0; 2]; vertex_count];
7     let mut triangles = vec![0; mesh_size * mesh_size * 6];
8     let mut triangle_index = 0;
9     let mut add_triangle = |t: [u32; 3]| {
10         triangles[triangle_index] = t[2];

```

```

11     triangles[triangle_index + 1] = t[1];
12     triangles[triangle_index + 2] = t[0];
13     triangle_index += 3;
14 }
15 for x in 0..=mesh_size {
16     for y in 0..=mesh_size {
17         let vertex_index = x + y * (mesh_size+1);
18         vertices[vertex_index] = [
19             x as f32 / mesh_size as f32 * chunk_size as f32,
20             0.0,
21             y as f32 / mesh_size as f32 * chunk_size as f32,
22         ];
23         if x < mesh_size && y < mesh_size {
24             add_triangle([
25                 vertex_index as u32,
26                 vertex_index as u32 + (mesh_size+1) as u32 + 1,
27                 vertex_index as u32 + (mesh_size+1) as u32,]);
28             add_triangle([
29                 vertex_index as u32 + (mesh_size+1) as u32 + 1,
30                 vertex_index as u32,
31                 vertex_index as u32 + 1,]);
32         }
33         uvs[x + y * (mesh_size+1)] = [
34             x as f32 / (mesh_size+1) as f32,
35             y as f32 / (mesh_size+1) as f32,];
36     }
37 }
38 let normals = vec![[0.0,1.0,0.0];vertices.len()];
39 let mut mesh = Mesh::new(PrimitiveTopology::TriangleList);
40 mesh.insert_attribute(Mesh::ATTRIBUTE_POSITION, vertices);
41 mesh.insert_attribute(Mesh::ATTRIBUTE_NORMAL, normals);
42 mesh.insert_attribute(Mesh::ATTRIBUTE_UV_0, uvs);
43 mesh.set_indices(Some(mesh::Indices::U32(triangles.to_vec())));
44 mesh
45 }

```

Priedas 3. Darbo planas

Veiksmas	Data	Rezultatas
Temos suformulavimas	2023-02-05	Suformuluota tema
Problemos analizė	2023-03-19	Išanalizuota problema
Tikslo ir uždavinijų formulavimas	2023-03-20	Suformuluoti tikslai ir uždaviniai
Programos kūrimas: Reljefo atvaizdavimas	2023-04-01	Sukurtas virtualus tinklelis reljefui atvaizduoti
Programos kūrimas: Grafinė sasaja, redagavimas	2023-04-20	Pridėti vartotojo sasajos elementai, reljefo redagavimo funkcija.
Programos kūrimas: Reljefo generavimas naudojant įvairių tipų triukšmą bei simuliujant hidraulinę eroziją.	2023-06-09	Pridėta reljefo generavimo funkcija.
Programos kūrimas: Vizualūs patobulinimai (šešeliai, debesys, vanduo), programos patalpinimas į svetainę	2023-09-06	Sukurta galutinė programos versija.
Įvado parengimas	2023-09-20	Parengtas įvadas
Išvadų ir pasiūlymų suformulavimas	2023-09-25	Suformuluotos išvados ir pasiūlymai
Anotacijos parengimas	2023-10-10	Parengta anotacija
Darbo turinio parengimas	2023-11-11	Parengtas darbo turinys