TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Martin Viksi 155153IAPB

GENERATIIVSE MAAILMA LOOMINE THREE.JS NÄITEL

Bakalaureusetöö

Juhendaja: Gert Kanter

Tehnikateaduse magister

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Martin Viksi

10.05.2019

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on teha veebirakendus, millega saab genereerida erinevate algoritmidega 3D maastikku veebis.

Töö on jaotatud kaheks osaks, esimeses osas on analüüsitud algoritme ja tööks vajalikke tehnoloogiad ning teises osas realiseeritakse praktiline osa. Töö tulemiks on töötav veebirakendus, millega saab genereerida maastikku kolmel erineval viisil.

Rakendus on kättesaadav aadressil: https://mviksi.github.io/iapb/

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 22 leheküljel, 7 peatükki, 18 joonist, 2 tabelit.

Abstract

3D Terrain Generation Based On Three.js Library

The aim of this thesis is to make web application, where it can generate 3D terrain on the web.

Work is divided into two parts, first part is for analysing algorithms and technologies and second part is to realize practical part. Result is a working web application, which can generate terrain three different ways.

Application is available at https://mviksi.github.io/iapb/

The thesis is in Estonian and contains 22 pages of text, 7 chapters, 18 figures, 2 tables.

Lühendite ja mõistete sõnastik

HTML	*Hypertext Markup Language*
Wireframe	Loodud geomeetria kujundi tippude vahel olevate jooned, mis aitavad materiali kujust paremini aru saada
Seed	Number või muu väärtus, mis genereeritakse juhuslikult.
CGI	*Computer-generated imagery*, arvuti poolt genereeritud pilt.
Skybox	Kuup, mille siseküljele on paigutatud tekstuurid.
Fragement shader	GLSL-i koodi osa punktide loomiseks.
Vertex shader	GLSL-i koodi osa pikslite värvimiseks.
WebGL	Veebigraafika raamistik.
Must kast	Mingi funktsiooni kasutamine, teadmata kuidas see töötab.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval püüeldakse ühe enam täieliku automatiseerimise poole, tootmisliinidest arvuti algoritmideni. Protseduurilise genereerimise algoritme kasutades on võimalik väheste sisendparameetritega luua paljusid variatsioone. Mängu maailmad lähevad järjest suuremaks ja maastiku käsitsi loomine on kallis ja ajamahukas protsess. Protseduurilise genereerimise algoritmidega on läbimängitavus potentsiaalselt lõpmatu ja neid tehnoloogiad kasutatakse ka filmides *CGI* loomisel.

Maailma genereerimiseks on olemas igasuguseid alla laetavaid programme, mis tihti tulevad kaasa liigsete funktsionaalsustega ja on aeglased. Veebibrauseri eeliseks on kättesaadavus ja kasutusmugavus, miinuseks mahajäänud graafika. Maastiku genereerimist tehakse tavaliselt kolmel eri viisil:

- Fraktaalne maastik.
- Füüsiline erosiooni simulatsioon.
- Maastiku süntees analüüsides tõelisi pilte või näidismaastiku.

Töö praktilises osas tutvutakse lähemalt maastiku sünteesiga näidispildi pealt ja fraktaalse maastiku genereerimisega. Teises peatükis käsitletakse erinevaid maastiku genereerimise viise. Kolmandas peatükis tutvustatakse erinevaid raamistikke ja vahendeid, millega tehakse praktiline veebirakendus. Käesolevas bakalaureusetöös kasutatakse kolme erinevat müra algoritmi ja tehakse lõplikku genereerimist. Viiendas peatükis võrreldakse algoritmide kiiruseid ja algoritme. Kuuendas peatükis räägitakse võimalikest edasiarendustest. Viimases peatükis tehakse kokkuvõte ja antakse ülevaade tehtud tööst.

2 Generatiivse sisuloomise võtted

Maastiku genereerimine on üks generatiivse sisuloomise mooduseid. Neid kasutatakse järjest rohkem, sest need on odavad ja vähendavad disainerite tööd. Tänu maastiku loomisele tõuseb mängitavus ja iga kord on uus kogemus [1]. Generatiivseks sisuloome näiteid mängudest:

- Rogue on üks esimesi mänge, milles genereeriti juhuslikult laburent, mis pani aluse ka teistele 2D laburendimängudele. Sellistes mängudes on oluline kirja panna erinevad reeglid, et mäng oleks läbitav [2].
- Mängus Borderlands on kasutatatud erinevate relvade genereerimist, mis on samuti Guinnessi maailmarekord: 17.75 miljonit erinevat relva mängus [3].
- Minecrafti mängus kasutatakse maailma loomiseks Perlini müra aga tekstuuri asemel kasutatakse kuupe, nagu on näha Joonis 1 [4].

Joonis 1. Minecrafti maailm [4].

2.1 Maastiku genereerimise müraalgoritmid

2.1.1 Võrgumüra

Võrgumüra ehk *lattice noise* hõlmab müra või müra gradientide interpoleerimist, mis on määratud võrepunktides, täisarvulisel skaalal. Kõige populaarsem võrgumüra on Perlini müra. Juhuslikud arvud jagatakse võrdselt võrku ära ja numbrid kaardistatakse tekstuuriruumi nii, et iga koordinaat on täisarv [5].

2.1.2 Väärtusmüra

Võrgumüra ehk *value noise* on väärtustel põhinev mürafunktsioon. Kui igas võrepunktis on juhuslik arv, siis uut väärtust saab interpoleerida nende juhuslike väärtuste vahel. See interpolatsioon toimub silumisoperatsioonina ja käitub kui madal-pääs filter. Seega, väärtusmüra on üks lihtsamaid viise, et genereerida stohhastilist funktsiooni [5].

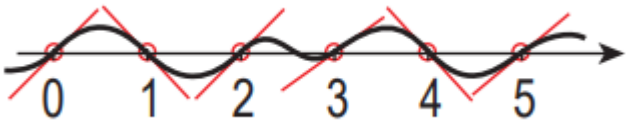
2.1.3 Gradientmüra

Gradientidel põhinev müra, mis pakub alternatiivset meetodit, kasutades võrgupunktis juhusliku numbri asemel gradientvektoreid [5].

2.1.4 Modifitseeritud Perlini müra

Ken Perlin, kes sai 1997. aastal tehnilise saavutuse auhinna Filmi- ja Kunstiakadeemialt [6], publitseeris uuendatud Perlini müra 2002. aastal [7]. Mängu arenduses saab seda kasutada nii maastiku loomisel, leegi efektidel kui ka pilvede tekstuuril [7].

Perlini müra on gradientmüra, mis tähendab, et pseudo juhuslikud gradiendid pannakse regulaarselt vahedega punktidesse ruumis ja interpoleeritakse nende punktide vahel. Selleks, et genereerida Perlini 1D müra, ühendatakse pseudo juhuslik gradient iga täisarvu koordinaadiga ja igal täisarvu punktil on Perlini müra väärtus null, vaata Joonis 2 [8].



Joonis 2. 1D Perlini müra [8].

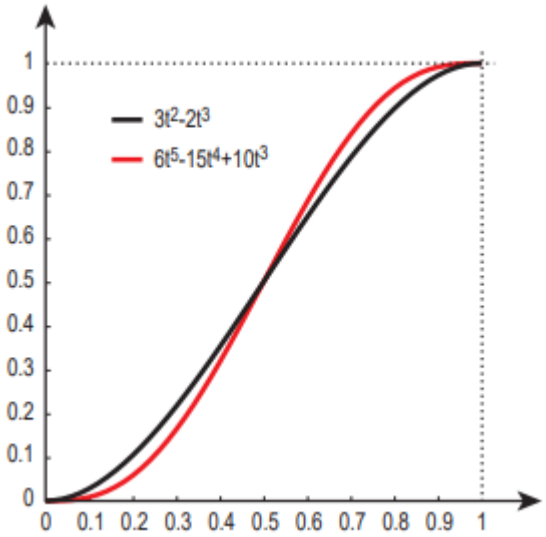
Tehniliselt töötab Perlini müra järgmiselt, et ruudu äärmistesse punktidesse lisatakse gradientvektorid, teatud vektorite kogumist, kus need asuvad kuubi keskel või äärtes. Järgmisena arvutatakse neli vahemaa vektorit etteantud punktist. Järgmisena võetakse skalaarkorrutis gradient ja vahemaa vektorist. Interpoleerides nelja väärtuse vahel saab kaalutud keskmise. Kuna lineaarne interpolatsioon näeb välja ebanaturaalne, on vaja lisada sujuvam üleminek gradientide vahel. Selle jaoks kasutatakse valemit: [7]

$$f(t) = 6t^5-15t^4+10t^3$$

Originaalselt Ken Perlin tegi Hermite segunemise funktsiooniga:

$$f(t) = 3t^2 – 2t^3$$

Kaks funktsiooni on väga sarnased nagu on näha Joonis 3, aga kasutatakse viienda astme polünoomi valemit, sest sellel on teine tuletis 0.



Joonis 3. Kahe funktsiooni erinevus.

Maastiku genereerimiseks ainult sellest ei piisa, sest muidu on liiga sujuv, selleks tuleb juurde lisada oktaave, vaata Joonis 4 [7].

```
public double OctavePerlin(double x, double y, double z, int octaves, double persistence) {
    double total = 0;
    double frequency = 1;
    double amplitude = 1;
    double maxValue = 0;  // Used for normalizing result to 0.0 - 1.0
    for(int i=0;i<octaves;i++) {
        total += perlin(x * frequency, y * frequency, z * frequency) * amplitude;

        maxValue += amplitude;

        amplitude *= persistence;
        frequency *= 2;
    }

    return total/maxValue;
}
```

Joonis 4. Kuvatõmmis kasutaja flafla2 modifitseeritud mürast [7].

2.1.5 Simplexi müra

2001.aastal esitles Ken Perlin Simplexi müra, mis on uuem ja parem version Perlini mürast ning selle eelised on:

- Algoritmi keerukus on $O(N^2)$ ja Perlini müral on $O(2^N)$, mis tähendab seda, et 4D-st suuremast dimensioonist, on Simplex palju kiirem [8].
- Simplexi müra on lihtne riistvaras implementeerida [8].

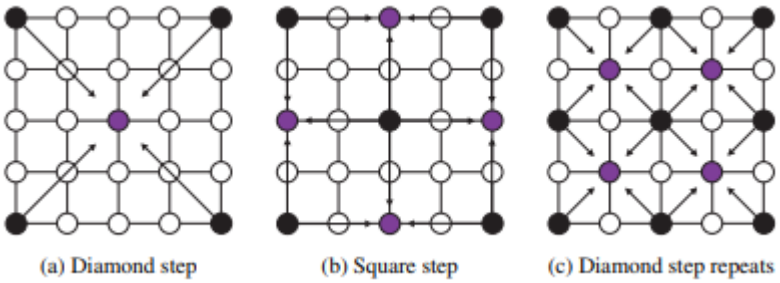
2.1.6 Teemant ruudu algoritm

Gradientmüral on siiski ebaloomulik aspekt: maastik on konstantsel sagedusel, mis on võre punktivahemiku jaoks valitud sagedus. Reaalne maastik on rohkem varieeruvam, näiteks tasanditelt tõuseb mäkke. Maastikku saab genereerida mitmete meetmete abil, mõned neist baseeruvad fraktsiooni matemaatikal ja mõned genereerivad sarnast efekti lihtsamalt [9].

Teemant ruudu viis luua maastikku on võtta 4 erinevat nurka ja määrata neile juhuslikud väärtused. Järgmisena leitakse punkt ruudu keskel, arvestades nelja erinevat nurka. See on “teemant“ samm. Järgmiseks leitakse 4 punkti, mis jäävad kahe nurga ja keskmise punkti vahele ja arvutatakse välja nende keskmine ja juhuslik väärtus. See on “ruut” samm [9]. Joonis 5 on välja toodud kolm vajalikku sammu.

Magnituud juhuslikust väärtustest nimetatakse konarlikkuseks ehk *roughness*, sest suuremad väärtused teevad konarlikumat maastikku. Nende kahe sammu tegemisel jagati originaalne ruut neljaks ruuduks. Nüüd tuleb vähendada konarlikkuse väärtust ja

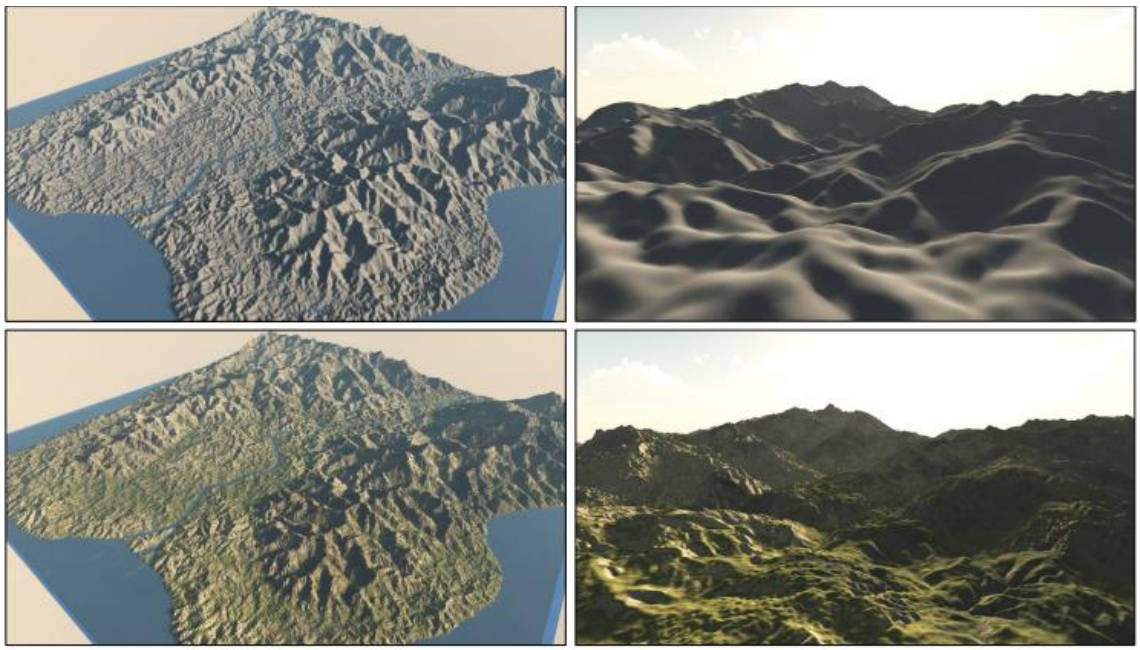
korrata kahte sammu, et täita väiksemad ruudud. Tüüpiliselt protsess kordab senikaua, kuni on täpsustatud maksimaalne number iteratsioone tehtud [9].



Joonis 5. Teemant ruudu samm. [9]

2.2 Erosiooniga maastiku genereerimine

Erosiooniga maastik genereeritakse kahe suurema protsessi järel, termiline erosioon või hüdrauliline erosioon. Erosioonimudelid lisavad kõrget kvaliteeti olemasolevatele mudelitele, aga nad kannatavad pikkade arvutusaegade ja madala kontrollitavuse all [10].



Joonis 6. Erosiooniga genereeritud maastik [10].

2.3 Online või offline genereerimine

Eristatakse kahte erinevat viisi. Online on mängimise ajal maastiku genereerimine ja offline on enne mängu tööle panemist. Praktilises osas kasutatakse offline genereerimist. Offline genereerimine on kasulik, kui algoritm on liiga aeglane, et seda onlines teha.

3 Kasutatud tehnoloogiad

3.1 Javascript

Javascript on veebi programmeerimiskeel, mis võimaldab dünaamiliselt uuendada sisu, kontrollida multimeediat, animeerida sisu. Võimalus säilitada väärtuseid muutujates, käivitada koodi teatud veebilehe sündmustele. Javascript käivitatakse tema mootori poolt, peale HTML-i ja CSS-i [11].

3.2 HTML

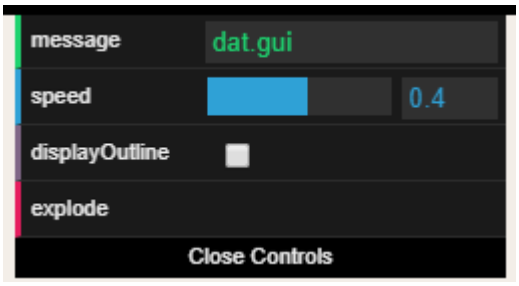
HTML on hüperteksti märgistuskeel, millega struktuurida veebilehte ja sisu. HTML-i kasutatakse veebibrauserites nii arvutis kui ka mobiilil. HTML5 on uusim versioon [13].

3.3 Three.js

Three.js on raamistik, mis pakub reaalajas 3D graafikat, mis on kirjutatud javascriptis ja on mõeldud kasutamiseks javascripti keskkonnas. Tavaliselt töötab kliendi poolel aga on võimalik töötama panna ka serveri poolel [13].

3.4 Dat.gui.js

Dat.gui.js on raamistik, mis annab väärtuste muutmiseks kasutajaliidese, välja arendatud Google-s, näidis Joonis 7 [14].



Joonis 7. Dat.gui.js raamistik [14]

3.5 WebGL

WebGL on veebi graafika raamistik, mis joonistab punktid, jooned ja kolmnurgad esitatud koodil. WebGL jookseb GPU peal, millele on vaja ette anda kaks funktsiooni: tipu vari ehk *vertex shader*, mille ülesanne on arvutada tipu positsioone ja fragmenti vari ehk *fragment shader*, mille ülesandeks on arvutada iga piksli värvus [15].

4 Veebirakenduse arendusprotsess

Antud bakalaureusetöö eesmärk oli luua veebirakendus, millega saab luua erinevate algoritmidega maastikku. Loodavas programmis tuli rõhku panna ka visuaalsele poolele. Veebirakenduse arenduse saab jaotada konkreetseteks etappideks, mis on vaja läbida.

4.1 Rakenduse nõuded

Funktsionaalsed nõuded, mis peavad olema loodaval veebirakendusel.

1. Kasutaja saab kaameraga liikuda ja maastikku sisse ning välja suumida.
2. Kasutaja saab genereerida Perlini modifitseeritud müra.
3. Kasutaja saab muuta Perlini müra parameetreid.
4. Kasutaja saab genereerida teemant ruudu algoritmiga tehtud maastikku.
5. Kasutaja saab muuta teemant ruudu algoritmil parameetreid.
6. Kasutaja saab muuta maastiku kõrgust genereerimisel.
7. Kasutaja saab muuta maastiku kõrgust mudelil.
8. Kasutaja saab valida võre ehk *wireframe* või selle välja lülitada.
9. Kasutaja saab genereerida ruudukujulist maastiku kõrguskaardiga.
10. Kasutaja saab muuta maastiku suurust.
11. Kasutaja saab valida mitmest ruudukujulisest tasapinnast maastik koosneb.
12. Kasutaja saab muuta limiteeritult materjalide asukohta.
13. Kasutaja saab vee taset tõsta või langetada.

Lisaks funktsionaalsetele nõutele peavad kehtima ka järgmised mittefunktsionaalsed nõuded:

1. Loodav veebirakendus peab olema hea disainiga.
2. Veebirakenduse kasutajaliides on lihtne ja mugav.
3. Rakendust saab kasutada enim kasutatavate brauseritega.
4. Rakendus on inglise keeles.
5. Rakendus on optimeeritud, maastiku genereerimine on kiire.

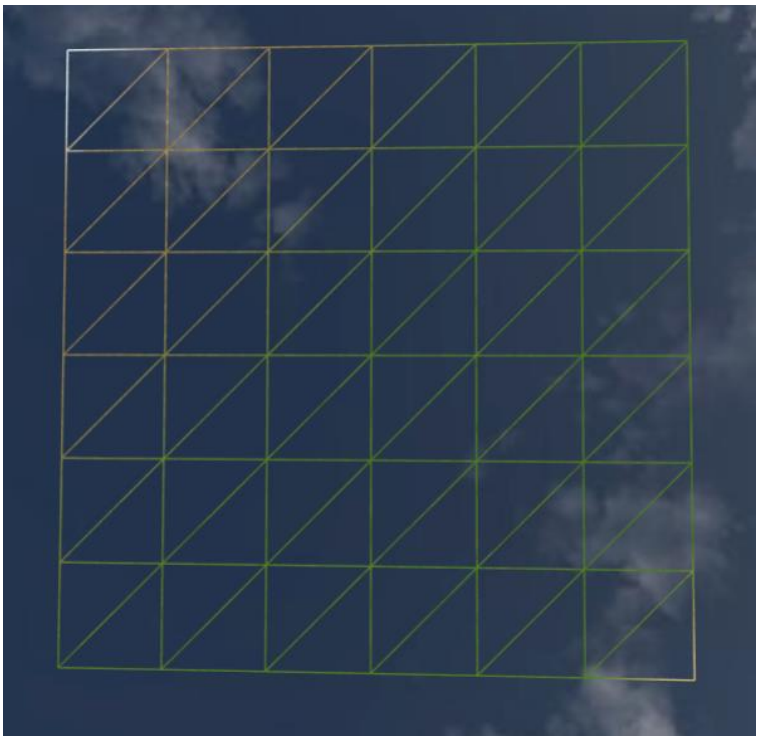
4.2 Stseeni loomine

Stseen koosneb kolmest objektist: genereeritud maastikust, vee tasapinnast, taevakastist ehk *skybox*. Valgustust pole vaja, sest piisav valgustus on stseenil endal. Stseenil on kaamera.

4.3 Tasapinna võre loomine

Töös kasutatakse ruutudest võre, ehk *wireframe*, mis omakorda koosnevad kahest kolmnurgast, Joonis 8. See on levinum viis maastiku tasapinna seadmiseks. Kasutatakse kahemõõtmelist xy tasapinda kolmemõõtmelises ruumis, kus z – telg on suunaga üles.

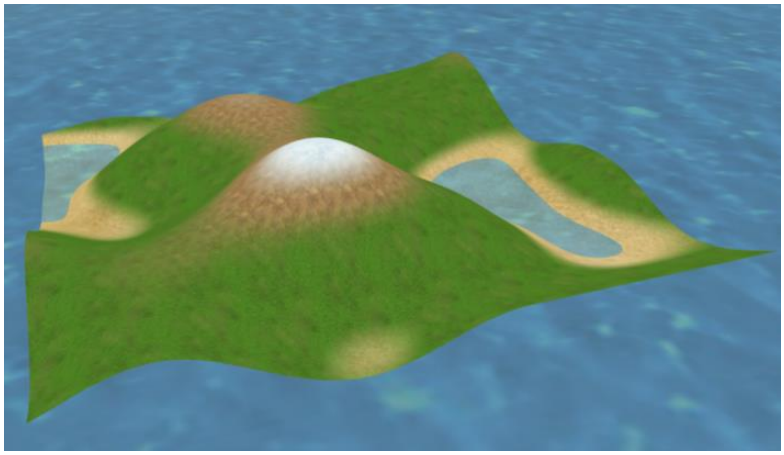
Tasapinna võrgustiku loomiseks luuakse three.js puhver geomeetria objekt, mis koosneb atribuudidest. Igal tipul on xyz positsioon, normaalvektor ja värvus, tippude vaheliste joonte määramiseks peab määrama indeksid. Tekstuuri lisamiseks, tuleb määrata uv positsioonid. Uv positsioonid asuvad vahemikus [0,1], kus 0 on tasapinna algus ja 1 on tasapinna lõpp [16].



Joonis 8. *Wireframe* on ruutudest koosnev võre.

4.4 Modifitseeritud Perlini müra lisamine

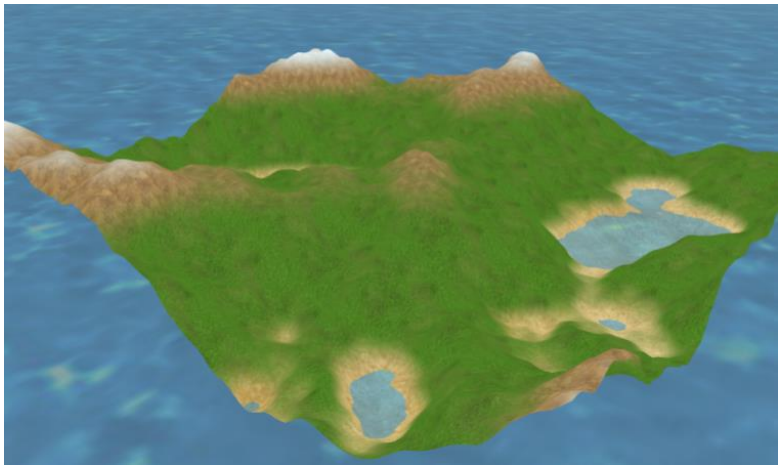
Töös kasutatakse Stefan Gustavsoni tehtud Perlini müra javascripti implementatsiooni [17]. Maastiku genereerimiseks võetakse z indeksi väärtus x ja y positsioonil. Z väärtuse muutmiseks kasutatakse perlin2() funktsiooni. Perlini müra tagastab väärtuse vahemikus [-1,1], kõrguse muutmiseks mudelil, salvestatakse z väärtused eraldi massiivi. Perlini mürale tuleb anda ette juhuslik arv ehk *random seed*, vastasel juhul on müra samasugune. Kuna Stefani tehtud kood toetab seede kuni 65536-ni, siis töös leitakse Math.random() funktsiooniga seed vahemikus [1,65536], mis on piisav arv, et saada erinevad väärtuseid [17]. Stefani kood teeb Perlini müra modifitseerimata kujul, mis jätab liiga ebanaturaalse tulemuse, vaata Joonis 9.



Joonis 9. Perlini müra modifitseerimata kujul.

Lisades oktaavi juurde Perlini mürale, saab reaalsema maastiku, Joonis 10. Negatiivne on see, et iga oktaaviga tõuseb koodi täitmisaeg. Püsivus ehk *persistence* muudab amplituudi. Täitevus ehk *lacunarity* muudab sagedust ja määrab ära selle, kui palju detaili on lisatud või kustutatud iga oktaaviga [7].

Mõlemal joonisel on x ja y sagedus 0,02 ja võre ruutude arv ehk segments 128. Modifitseeritud Perlini müraga on oktaavide arv 3 püsivus 0.24 ja täitevus 3.57, vaata Joonis 10.



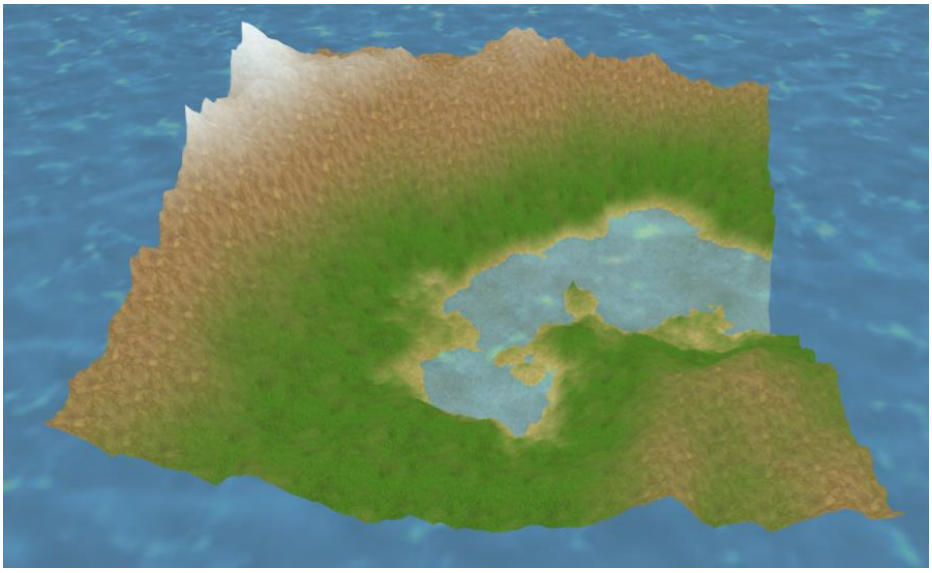
Joonis 10. Modifitseeritud Perlini müra.

4.5 Teemant ruudu algoritm

Teemant ruudu algoritmi ehk *diamond-square* algoritmi jaoks kasutasin Githubis olemasolevat javascripti implementatsiooni [18]. Kasutades teemant ruudu algoritmi, tuleb terve maailm luua korraga ja tippude arv ühel küljel on järgmise valemi järgi:

$$f(x) = 2^n + 1$$

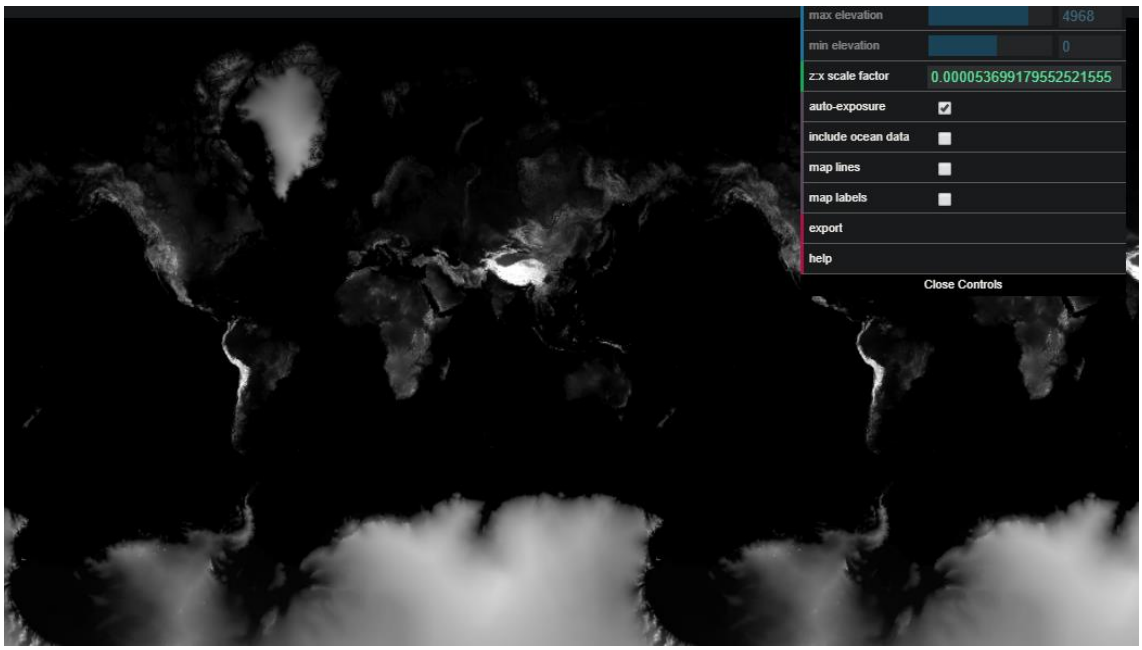
Teemant massiiv on 2D ruutude massiiv. Teemant ruudu algoritm teeb maastikke, mis sobivad alal, kus on mäed, genereeritud maastik on Joonis 11.



Joonis 11. Teemant ruudu algoritmiga loodud maastik.

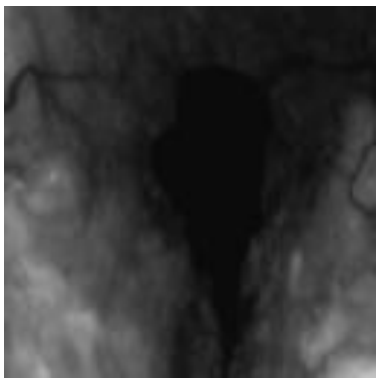
4.6 Kõrguskaardiga maailma tegemine

Kõrguskaardi ehk *heightmapiga* käib maastiku genereerimine ilma müra algoritmita, laadides rakendusse must-valge pildi ja piksli värvi tumeduse järgi saab otsustada kui kõrgel asuvad z indeksid. Kõrguskaart tuleb võtta internetist [19] ja peab olema ruudu kujuline, vaata Joonis 12.



Joonis 12. Kuvatõmmis heightmapi valimiseks [30].

Näidiseks võeti pilt Võrtsjärvest, mille suuruseks on 128x128 pikslit.



Joonis 13.Autori valitud pilt.

Veebirakenduses tehtud mudel on Joonis 14.

Joonis 14. Mudel rakenduses.

4.7 Materiali lisamine

Töös kasutati tekstuure, sest need on visuaalselt ilusamad.

Materiali lisamine igale tasapinnale eraldi jätab sakilised üleminekud nagu on näha Joonis 15, lahenduseks kasutatakse WebGLi tekstuuride segunemiseks.

Joonis 15. Igale ruudule omaenda materiali lisamine.

Tipu vari ehk v*ertex shader* käivitub esimesena, mis omastab atribuudid, kalkuleerides iga individuaalse tipu asukoha ja edastab lisainformatsiooni edasi fragmendi varjule ehk *fragment shader*ile, mis käivitub teisena, värvides igat renderdatud pikslit eraldi [20]. Üleminekud materjalide vahel on näha Joonis 11.

3 erinevat tüüpi muutujat on varjudes:

- *Uniformid* on muutujad, millel on sama väärtus kõigil tippudel, kus hoitakse valgust, varje. Uniforme saab omastada endale nii tipu vari kui ka fragmendi vari. Neid kasutatakse selleks, et tuua javascriptist sisse mingi muutuja [21].

- Atribuutidid on muutujad, mida seostatakse iga tipuga, näiteks tipu positsioon, normaalvektor. Atribuute saab muuta ainult tipu varjuga [21].

- Varieerujad ehk *varyings* on muutujad, mida edastatakse tipu varjult fragmendi varjule. Iga fragmendi kohta, iga varieeruja väärtus interpoleeritakse sujuvalt kõrvalolevate tippude põhjal [21].

4.8 GLSL koodi lisamine

Töös tuleb kasutada ka kirjutada GLSL-i, materjalide segunemiseks.

Kõrguse massiivi normaliseerimiseks vahemikust [-in_min, in_max], vahemikku [0, 1] leidmiseks kasutan valemit:

$$f(x) = (z – in\_min)\ (in\_max – in\_mix),$$

kus x on kõrgus, in_min on väiksem z väärtus ja in_max on suurim z väärtus. Joonis 16 määran ära main meetodis positiooni z, mille annan edasi fragmendi varjule.

```
<script id="vertexShader" type="x-shader/x-vertex">
	uniform float in_min;
	uniform float in_max;
	varying vec2 vUV;
	varying float height;
	void main(){
		vUV = uv;
		height = (position.z - in_min) / (in_max - in_min);
		 gl_Position = projectionMatrix * modelViewMatrix
		* vec4( position, 1.0 );
	}
</script>
```

Joonis 16. *Vertex shaderi* programmikood.

Fragment shaderis on muutujateks rohu tekstuur, kus *smoothstep* teeb Hermite interpoleerimist kahe väärtuse vahel [22]. Koodi on näite mõttes lühendatud, reaalselt kasutatakse rohkem tekstuure. Joonis 17 on koodi näidis, kus kõrguse väärtus *height* on tipu varjult saadud väärtus vahemikus [0,1].

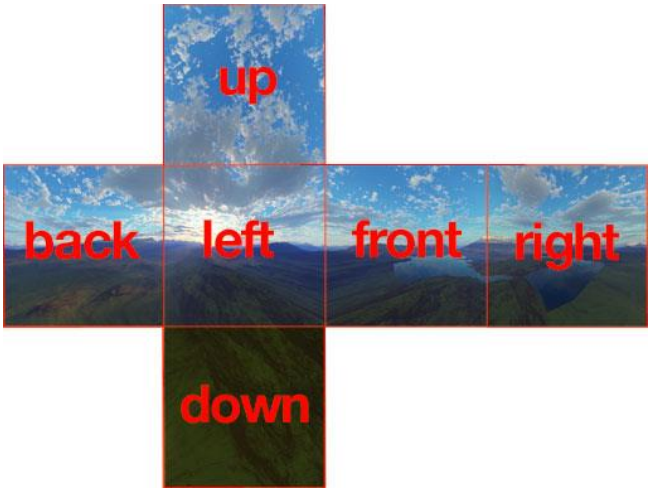
```
<script id="fragmentShader" type="x-shader/x-vertex">
	uniform sampler2D grassTexture;
	varying vec2 vUV;
	varying float height;

	void main(){
		vec4 grass = (a - b) * texture2D( grassTexture, vUV * 10.0);
		gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0) + grass;
	}
</script>
```

Joonis 17. Fragment shaderi kood

Üleminekutel kasutatakse järgmist valemit: (a – b) * textureColor, kus a on tekstuuri algus ja b on tekstuuri lõpp.

4.9 Visuaalse poole parandamine

Töös kasutatakse taeva kasti ehk *skyboxi*, mille eesmärgiks on lisada realismi. Selleks tehakse kuup ja lisatakse igale küljele tekstuuri nii, et seest oleks igas suunas pilt, nagu on näidatud Joonis 18. *Skybox* on lihtne viis lisada realismi stseeni, minimaalse koormusega riistvarale. Töös on kasutatud ka tasapinda, millele on lisatud vee tekstuur, et tekitada vee efekti.



Joonis 18. *Skyboxi* 6 erinevat vaadet [23]

5 Maastiku genereerimise kiiruse testimine

Testimises võrreldakse Perlini müra, modifitseeritud Perlini müra, ja teemant ruudu algoritmi kiiruseid. Maastiku testitakse segmentide arvuga $f(x) = 2^n$.

5.1 Algoritmide kiirused

Testin algoritmide kiiruseid, arvutades aritmeetilise keskmisega kolm paremat tulemust, peale kolme esialgset genereerimist, sest siis on kõige stabiilsemad arvud. Modifitseeritud Perlini müras kasutatakse ühte oktaavi. Kiiruseid testiti console.time ja console.timeEndi funktsioonidega. Tabel 1 võrreldakse algoritmide kiiruseid.

Tabel 1. Kiiruste võrdlus algoritmil.

Kiirus/ms	**Modifititseeritud Perlini müra**	**Teemant ruudu algoritm**	**Kõrguskaart**
128x128	35.1	12.6	13.2
256x256	146.5	62.5	69.1
512x512	650.7	260.8	308.4

Võrdlen Perlini modifitseeritud müra oktaavide tõusuga kaasnenud aeglustamist. Tabel 2 on näha oktaavide tõusuga kaasnevat algoritmi aeglustumist. Iga oktaaviga on lineaarne tõus [7], mida on näha ka tabelilt.Tabel 2. Modifitseeritud müra oktaavide tõusuga kaasnenud algoritmi aeglustumine.

Tabel 2. Modifitseeritud müra oktaavide tõusuga kaasnenud algoritmi aeglustumine.

Kiirus/ms	**2 oktaavi**	**3 oktaavi**	**4 oktaavi**
128x128	63	87.8	115.4
256x256	255.8	360.3	464.3
512x512	1406.2	2059.2	2601.1

Testisin järgnevate spetsifikatsioonidega:

- I7 – 4700MQ CPU @ 2.40GHz
- 8GB RAM
- AMD Radeon HD 8600/8700M

5.2 Järeldus

Kõrguskaardi ehk *heightmapiga* on sobiv luua keskkondi, kus on võetakse reaalselt olemasolevast kohast kaardilt mudel. Kiiruselt jäi algoritm teisele kohale. Kõrguskaardiga maastiku realiseerimiseks on vaja ise selleks kood kirjutada.

Modifitseeritud Perlini müra sobib hästi maastiku loomiseks, olles üks aeglasemaid algoritme, on ta üks kõige paindlikumaid. Perlini müra saab kasutada kui musta kasti ja seda pole vaja paljudes teistes programmeerimiskeeltes realiseerida, vaid see on olemas funktsioonina, lisaks saab sama maastikku uuesti genereerida. Võimalus erinevaid maastiku luua on palju, sest on olemas erinevaid parameetreid maastiku muutmiseks.

Kõige kiirema tulemuse saamiseks tuleks kasutada teemant ruudu algoritmi. Mängude loomiseks tuleb algoritmi lahendus leida internetist või realiseerida ise. Parameetritest saab muuta ainult maastiku künklikust.

6 Võimalikud edasiarendused

6.1 Visuaalne pool

Jagada maa ära mitmeks erinevaks kohaks ehk bioomideks, mõndades kohtades on ainult lume tekstuurid, teistes kõrbe tekstuurid. Lisada puid ja taimi. Lisada juurde liikuv veetekstuur, graafiline pool teha paremini. Kuna GLSL-i on suure õppimiskurviga, jäi see osa realiseerimata. Töö maht jääb 200 tunni juurde.

6.2 Edasi mänguks arendamine

Kaardil liikumine ei ole suur töö juurde lisada, aga mänguks tegemine on suur töö, sest mängust nauditava kogemuse saamiseks on vaja lisada juurde väga palju erinevaid funktsioone, mille töö maht võib jääda 300 tunni juurde.

6.3 Mudelite eksportimine

Kasutades three.js võimalust eksportida mudeleid kas JSON või binaarvormis, et kasutada teistes programmides. Töö maht on autori arvates 30 tundi.

7 Kokkuvõte

Käesolevas töös uuriti erinevaid maastiku genereerimise võimalusi. Protseduuriliselt genereeritud maastiku loomisel on olulisel kohal mürad, millega saab genereerida piisavalt paindlikke ja efektiivseid maastikke. Leiti, et tulemuste paremaks saavutamiseks tuleb kasutada erinevaid meetodeid ja igal meetodil on oma positiivne ja negatiivne pool. Võrreldi meetodite kiiruseid ja kasutusmugavust. Veebis olemasolevad maastiku genereerimise programme on vähe ja nad on minimaalse funktsionaalsusega, seetõttu töötati välja lihtsustatud funktsionaalsusega lahendus.

Töö eesmärk sai täidetud, selle käigus valmis kasutuskõlblik veebirakendus, mis võimaldab kasutajatel võrrelda erinevaid algoritme ja teha kindlaks, milline meetod vastab tema nõudmistele kõige rohkem.

Rakenduse loomisel pöörati tähelepanu erinevatele funktsionaalsustele ja parameetritele, mis oleksid muudetavad ja lihtsasti arusaadavad. Samuti pöörati rõhku ka visuaalsele poolele. Töös saab genereerida kolmel eri viisil maastikku. Rakenduses üritati leida piisavalt optimaalseid võimalusi erinevate maastikkude genereerimiseks.

Rakendus valmis prototüübina, mida on võimalik edasi täiendada mängu arenduseks. Rakendus on mõeldud kasutamiseks arvutis aga seda saab kasutada ka telefonis.

Töö käigus omandas autor uusi programmeerimisoskuseid. Kõige suuremaks probleemiks oli three.js ja GLSL programeerimiskeele õppimise kurv, sest autor ei olnud enne nendega kokku puutunud ja graafika programmeerimisega tegeleti koolis vähe. Kõige meeldivamaks osaks oli three.js dokumentatsioon.

Kasutatud kirjandus

[1] Noor Shaker, Julian Togelius, Mark J. Nelson “Procedural Content Generation in Games: A Textbook and an Overview of Current Research." Springer. 2016, pp. 1-4

[2] “Rogue” http://pcg.wikidot.com/pcg-games:rogue [Online] (14.05.2019)

[3] Wesley Yin-Poole “How many weapons are in Borderlands 2?” http://www.levitylab.com/blog/2011/02/brief-history-of-spore/ [Online] (14.05.2019)

[4] Jon Fingas“Here’s how ‘Minecraft’ creates its gigantic worlds” https://www.engadget.com/2015/03/04/how-minecraft-worlds-are-made/ [Online] (14.05.2019)

[5] “Procedural Noise Categories ” http://physbam.stanford.edu/cs448x/old/Procedural_Noise(2f)Categories.html [Online] (15.05.2019)

[6] “Noise and Turbulence“ https://mrl.nyu.edu/~perlin/doc/oscar.html [Online] (20.05.2019)

[7] Flafla2, “Understanding Perlin Noise” [Online] https://flafla2.github.io/2014/08/09/perlinnoise.html (12.05.2019)

[8] “Simple noise demystified” http://weber.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf [Online] (15.05.2019)

[9] Noor Shaker, Julian Togelius, Mark J. Nelson “Procedural Content Generation in Games: A Textbook and an Overview of Current Research." Springer. 2016, pp. 62-64

[10] “Large Scale Terrain Generation from Tectonic Uplift and Fluvial Erosion” https://hal.inria.fr/hal-01262376/document [Online] (19.05.2019)

[11] “What is JavaScript?” [Online] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (13.05.2019)

[12] Margaret Rouse, “HTML (Hypertext Markup Language).” [Online] https://www.theserverside.com/definition/HTML-Hypertext-Markup-Language (11.05.2019)

[13] Dusan Bosnjak, “What is three.js?” [Online] https://medium.com/@pailhead011/what-is-three-js-7a03d84d9489 (12.05.2019)

[14] Dat.GUI [Online] https://github.com/dataarts/dat.gui (12.05.2019)

[15] “WebGL Fundamentals”https://webglfundamentals.org/webgl/lessons/webgl-fundamentals.html [Online] (14.05.2019)

[16] “BufferGeometry” https://threejs.org/docs/#api/en/core/BufferGeometry [Online] (19.05.2019)

[17] Stefan Gustavson, Peter Eastman, ”Noisejs.” [Online] https://github.com/josephg/noisejs (12.05.2019)

[18] Ischlessinger1, “FractalTerrainGeneration” [Online] https://github.com/lschlessinger1/fractalTerrainGeneration (13.05.2019)

[19] “Heightmap” https://tangrams.github.io/heightmapper [Online] (19.05.2019)

[20] “WebGL 3d - Textures” https://webglfundamentals.org/webgl/lessons/webgl-3d-textures.html [Online] (14.05.2019)

[21] “WebGL Fundamentals”https://webglfundamentals.org/webgl/lessons/webgl-fundamentals.html [Online] (14.05.2019)

[22] “Smoothstep” https://thebookofshaders.com/glossary/?search=smoothstep [Online] (19.05.2019)

[23] “Skybox” https://en.wikipedia.org/wiki/Skybox_(video_games)#/media/File:Skybox_example.png [Online] (19.05.2019)